DETEKSI DAN PENGENDALIAN KEMACETAN JARINGAN MELALUI PROTOKOL ROUTING *OPEN SHORTEST PATH FIRST*

TESIS

PROGRAM MAGISTER TEKNIK ELEKTRO
MINAT SISTEM KOMUNIKASI DAN INFORMATIKA

Ditujukan untuk memenuhi persyaratan
memperoleh gelar Magister Teknik

FRANSISKA SISILIA MUKTI
NIM 136060300111031

UNIVERSITAS BRAWIJAYA
FAKULTAS TEKNIK
MALANG
2017

TESIS

DETEKSI DAN PENGENDALIAN KEMACETAN JARINGAN MELALUI PROTOKOL ROUTING OPEN SHORTEST PATH FIRST

Fransiska Sisilia Mukti
NIM. 136060300111031

telah dipertahankan didepan penguji
Pada tanggal 10 Januari 2018
dinyatakan telah memenuhi syarat
untuk memperoleh gelar Magister Teknik

Komisi Pembimbing,

Pembimbing I,	Pembimbing II,
Achmad Basuki, S.T., M.MG., Ph.D.	Dr-Ing. Onny Setyawati, S.T., M.T., M.Sc.
NIP. 19741118 200312 1 002	NIP. 19740417 200003 2 007

Malang, 10 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 : Deteksi Dan Pengendalian Kemacetan Jaringan Melalui Protokol Routing Open Shortest Path First

Nama Mahasiswa : Fransiska Sisilia Mukti
NIM : 136060300111031
Program Studi : Magister Teknik Elektro
Minat : Sistem Komunikasi dan Informatika

KOMISI PEMBIMBING
Ketua : Achmad Basuki, S.T., M.MG., Ph.D
Anggota : Dr-Ing. Onny Setyawati, S.T., M.T., M.Sc

TIM DOSEN PENGUJI
Dosen Penguji 1 : Dr. Ir. Muhammad Aswin, M.T.
Dosen Penguji 2 : Hadi Suyono, S.T., M.T., Ph.D.

Tanggal Ujian : 10 Januari 2018
SK Penguji :

Karya ini ku persembahkan untuk

Papa (alm.) dan

Mama tercinta

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, November 2017

Mahasiswa,

Fransiska Sisilia Mukti
NIM. 136060300111031

RIWAYAT HIDUP

Fransiska Sisilia Mukti, Malang, 28 September 1990, anak ketiga dari Mukayat (Almarhum) dan Tri Redjeki, SD Katolik Santa Maria I Malang dan SMP Negeri 3 Malang, SMK Negeri 8 Malang lulus tahun 2008. Studi S1 Teknik Komputer Institut Teknologi Nasional Malang pada tahun 2009. Pengalaman kerja sebagai staff pengajar pada Universitas Islam Raden Rahmat, Politeknik Kota Malang dan STMIK Asia Malang sampai sekarang. Melanjutkan studi program Magister (S2) di Program Magister Teknik Elektro Jurusan Elektro Fakultas Teknik Universitas Brawijaya pada tahun 2013-2017.

Malang, November 2017

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-Ing. Onny Setyawati, S.T., M.T., M.Sc. 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. Mama tercinta yang selalu setia mendampingi dan mendoakan, dan almarhum Papa yang selalu jadi inspirasi semangat.
5. Kedua saudara terkasih Johana Eka Mukti dan Josafat Dirham Setiawan. Kakak ipar Adi Suwito, dan ponakan tersayang Sarah Shine Soteria.
6. Opa dan Oma Tangka yang selalu mendukung di dalam doa dan semangat.
7. Teman-teman Angkatan 2013 Program Magister Teknik Elektro SKI-1.
8. Sahabat dan rekan kerja STMIK Asia Malang.
9. LPDP, yang telah memberikan Beasiswa Penulisan Ilmiah (BPI) Tesis Tahun 2016, untuk mendukung terselesaikannya penelitian.

Malang, November 2017

Penulis

RINGKASAN

Fransiska Sisilia Mukti, Jurusan Teknik Elektro, Fakultas Teknik Universitas Brawijaya, November 2017, *Deteksi dan Pengendalian Kemacetan Jaringan Melalui Protokol Routing Open Shortest Path First*, Dosen Pembimbing: Achmad Basuki dan Onny Setyawati.

Open Shortest Path First (OSPF) merupakan salah satu protokol *routing* dinamis berbasis *link-state* yang menggunakan algoritme Dijkstra dalam melakukan komputasi jalur terbaiknya. Dijkstra menggunakan nilai *cost metric* sebagai satu-satunya faktor untuk menghitung rute terpendek antara node sumber dan node tujuan. Jika nilai *cost* yang ditetapkan tidak optimal, maka trafik dalam jaringan bisa saja terpusat pada salah satu jalur dan dapat menyebabkan peningkatan rasio kemacetan dalam jaringan. *Single-path routing* yang dihasilkan oleh Dijkstra menjadi tidak efektif apabila kemacetan terjadi pada jalur tersebut. Kemacetan akan menyebabkan data yang sudah ada ataupun yang akan melalui jalur tersebut tidak akan sampai di node tujuan dengan utuh, atau akan terjadi waktu tunda atau paket dibuang begitu saja oleh *router*.

Penggabungan mekanisme deteksi dan pengendalian kemacetan jaringan menggunakan *multipath routing* (ECMP) pada OSPF, terbukti efektif dalam mengurangi rasio kemacetan dan meningkatkan performansi jaringan. Hal ini dicapai dengan mendeteksi dan mengendalikan kemacetan jaringan yang terjadi melalui pengalihan jalur dan pemerataan beban pada jaringan. Setiap *link* yang menghubungkan antar *router* diberikan nilai *threshold*. Kemacetan diidentifikasi berdasarkan jumlah trafik yang melebihi nilai *threshold* yang telah ditetapkan di setiap *outbond interface router*, sedangkan modifikasi *routing cost metric* dilakukan pada *outbond interface router* dengan tingkat utilisasi *bandwidth* yang rendah. Pemodifikasian ini dilakukan dengan cara menyamakan nilai *cost metric* dengan jalur utama, yang bertujuan untuk membentuk jalur *multipath*. ECMP membantu mendefinisikan beberapa jalur yang memiliki nilai *cost* yang sama, sedangkan Dijsktra berperan dalam melakukan komputasi jalur yang terpendek.

Pengujian dilakukan dengan mengirimkan 4 aliran trafik sekaligus secara bersamaan. Hasil pengujian menunjukkan pada OSPF konvensional, terjadi penurunan *throughput* hingga 50% (2,4Mbps), sehingga menyebabkan terjadinya *packet loss* hingga 39%. Sedangkan pada OSPF *Multipath*, nilai *throughput* yang dihasilkan adalah sesuai dengan pengujian alirantrafik yang dikirim (4Mbps) sehingga tidak terjadi adanya *packet loss (0%)*. Dengan nilai *throughput* yang konstan, menjadikan OSPF *Multipath* memiliki waktu pendistribusian data yang lebih cepat, dengan nilai *jitter* mencapai 0,3ms, atau sepertiga dari waktu yang dihasilkan pada OSPF konvensional (1,02ms)

Kata kunci : *kemacetan jaringan, multipath routing, OSPF, utilisasi bandwidth*

SUMMARY

Fransiska Sisilia Mukti, *Department of Electrical Engineering, Faculty of Engineering, Unversity of Brawijaya, July 2017, Detect and Control Network Congestion with Open Shortest Path First, Academic Supervisor:* Achmad Basuki dan Onny Setyawati.

Open Shortest Path First (OSPF) is a dynamic routing protokol based on link-state that uses the Dijkstra's algorithm to compute its best path. Dijkstra uses cost metric values as the only factor to calculate the shortest route between the source node and destination node. If the set cost is not optimal, then the traffic within the network may be centered on one of the paths and may cause an increase in the congestion ratio in the network. Single-path routing generated by Dijkstra becomes ineffective when congestion occurs on the path. Congestion will cause existing data or who are going through these pathways will not arrive at the destination node with intact, or it could have happened a time delay or packets discarded by the router.

Merging the detection mechanism and control of network congestion using multipath routing (ECMP) in OSPF, has proven effective in reducing congestion ratio and improves network performance. This is achieved by detecting and controlling network congestion that occurs through line redirection and load equalization on the network. Each link that connects between routers is given a threshold value. Congestion is identified based on the amount of traffic that exceeds the threshold value that has been set in each outbound router interface, while the modification of routing cost metrics is performed on outbound router interfaces with low bandwidth utilization rates. This modification is done by equating the value of cost metric with the main line, which aims to establish lines of multipath. ECMP help define some paths have the same cost value, while Dijsktra role in computing the shortest path.

The simulation is done by sending 4 aliranof traffic simultaneously The test results reveal on the conventional OSPF, there is a decrease in throughput up to 50% (2,4Mbps), resulting in packet loss up to 39%. While in OSPF Multipath, the resulting throughput value is in accordance with the sent traffic alirantest (4Mbps) so there is no packet loss (0%). With a constant throughput value, OSPF Multipath has faster data distribution time, with jitter value reaching 0,3ms, or one-third of the time generated in conventional OSPF (1,02ms)

Keywords : *bandwidth utilization, multipath routing, network congestion, OSPF*

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-Ing. Onny Setyawati, S.T., M.T., M.Sc., 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 2013 Universitas Brawijaya Malang, yang telah memberikan dukungan moril kepada penulis.
4. Rekan-rekan kerja di STMIK ASIA 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, November 2017

Penulis

DAFTAR ISI

Halaman

DAFTAR TABEL

No. Judul Halaman

DAFTAR GAMBAR

No. Judul Halaman

DAFTAR LAMPIRAN

No. Judul Halaman

BAB I
PENDAHULUAN

1.1 Latar Belakang

Congestion atau kemacetan jaringan dapat diartikan sebagai keadaan jaringan pada saat jumlah permintaan trafik dari sumber daya yang diperlukan *user* telah melebihi jumlah kapasitas yang tersedia (D. Papadimitriou, 2011). Hal ini dapat terjadi disebabkan oleh beberapa faktor, antara lain akibat adanya perubahan dan perkembangan jaringan secara simultan (Kushwaha and Gupta, 2014), kurangnya kepekaan protokol jaringan dalam merespon kemacetan yang baru saja terjadi (*incipient congestion*) (Flannagan *et al*., 2001), maupun kesalahan dalam proses pemilihan jalur (*routing*) pendistribusian data (Nakahodo, Naito and Oki, 2014) .

Mekanisme *routing* yang tepat akan memperkecil rasio terjadinya kemacetan dalam jaringan, sehingga aturan *router* dalam melakukan proses *routing* (protokol *routing*), baik statis maupun dinamis, harus didesain seefisien mungkin (Moonlight dan Suhardi, 2012). *Routing* dinamis lebih banyak digunakan karena semua informasi yang terdapat dalam tabel *routing* akan dibentuk, dipelihara dan di*update* secara dinamis oleh protokol *routing* yang dijalankan (Balchunas, 2007). Salah satu protokol *routing* dinamis yang banyak diimplementasikan pada jaringan berskala besar adalah *Open Shortest Path First* (OSPF), karena kemampuannya dalam menentukan jalur atau rute menuju ke sebuah tujuan dengan cepat dan tepat (Susitaival and Aalto, 2004).

Untuk menentukan jalur terbaik ke semua node dalam jaringan, OSPF melakukan komputasi dengan menggunakan Algoritme Dijkstra (Solichin and Oktoviana, 2013). Dijkstra akan menggunakan nilai satuan beban (*cost metric*) yang merepresentasikan rasio kapasitas *bandwidth* yang tersedia pada setiap *interface* (Thomas, 2003). Setiap jalur akan memiliki jumlah nilai *cost* yang berbeda – beda, sekalipun memiliki jumlah lompatan *hop* yang sama. Nilai *cost* yang dihitung oleh Dijkstra akan berbanding terbalik dengan kapasitas *bandwidth* pada setiap interface. Semakin besar kapasitas *bandwidth* sebuah *link*, maka nilai *cost* yang dihasilkan akan semakin kecil (Kurose dan Ross, 2013).

Setiap node pada jaringan OSPF akan saling mengirimkan pesan yang disebut *Link State Advertisement* (LSA) untuk melakukan pertukaran informasi status node dan jalur-jalur yang ada. Informasi ini akan disimpan ke dalam *database* dan digunakan sebagai dasar untuk memulai perhitungan Dijkstra (Nakahodo, Naito dan Oki, 2014).

Dijkstra akan melakukan perhitungan *single-path routing* yang didasarkan pada rute terpendek, yaitu dengan jumlah nilai *cost* terkecil dari node sumber ke node tujuan. Setiap *link* akan memiliki bobot yang tetap, dan trafik akan dibawa melalui jalur terpendek, yaitu jalur dengan jumlah *cost* terendah yang telah didefinisikan oleh Dijkstra (Aprian dan Novandi, 2007).

Jika nilai *cost* yang ditetapkan tidak optimal, maka trafik dalam jaringan bisa saja terpusat pada salah satu jalur dan dapat menyebabkan peningkatan rasio kemacetan dalam jaringan. *Single-path routing* yang dihitung oleh Dijkstra akan menjadi tidak efektif apabila kemacetan terjadi pada jalur tersebut (Nakahodo, Naito dan Oki, 2014). Kemacetan akan menyebabkan data yang sudah ada ataupun yang akan melalui jalur tersebut tidak akan sampai di node tujuan dengan utuh, atau bisa saja terjadi waktu tunda atau paket dibuang begitu saja oleh *router* (Susitaival dan Aalto, 2004).

Kemacetan menjadi faktor utama yang penting untuk diketahui lebih dini untuk menghindari adanya kegagalan dalam proses *routing*. Mengoptimalkan penentuan bobot jalur berarti juga melakukan optimasi terhadap proses *routing* berdasarkan jalur terpendek. Tujuan utama dari optimasi *routing* adalah untuk mendistribusikan trafik dengan menggunakan sumber daya yang ada untuk menghindari adanya kemacetan jaringan (Nakahodo, Naito dan Oki, 2014). Oleh karena itu, setiap jaringan dituntut untuk memiliki mekanisme *routing* yang adaptif, yang berarti *router* harus mampu menentukan dan mengubah jalur pendistribusian data berdasarkan kondisi terkini dalam jaringan (Nugroho, Lamaida dan Ahsana, 2006).

Routing adaptif pada protokol *routing* OSPF didasarkan pada konsep deteksi dan pengendalian kemacetan jaringan, yang dibangun dengan menggunakan *Open Source Routing Software Quagga*. Kemacetan akan dideteksi berdasarkan persentase utilisasi *bandwidth* pada setiap *interface*, yang akan digunakan sebagai dasar dalam melakukan pengendalian kemacetan jaringan, yaitu dengan cara melakukan modifikasi nilai *cost metric*, dan mendistribusikan kembali data berdasarkan rute terbaru secara *multipathing*.

1.2 Rumusan Masalah

Berdasarkan uraian dari latar belakang, maka dapat dirumuskan suatu permasalahan sebagai berikut.

a. Bagaimana melakukan deteksi kemacetan jaringan pada setiap *interface router*.
b. Bagaimana melakukan modifikasi *routing cost metric* untuk mengendalikan kemacetan jaringan pada protokol *routing* OSPF.
c. Bagaimana performa jaringan OSPF dengan diterapkannya mekanisme *multipath routing*.

1.3 Batasan Masalah

Batasan masalah yang diberikan berdasarkan permasalahan yang telah dikemukakan di atas adalah sebagai berikut:

a. Protokol *routing* OSPF bersifat adaptif dalam menentukan dan mengubah jalur pendistribusian trafik berdasarkan kondisi jaringan terkini pada masing – masing *router.*
b. Penentuan kondisi jaringan terkini didasarkan pada utilisasi trafik pada setiap *interface* jaringan.
c. Nilai *cost* dimodifikasi berdasarkan status atau level kemacetan pada setiap *interface.*
d. Lingkungan uji coba (*testbed*) protokol *routing* OSPF dilakukan dengan menggunakan *Open Source Routing Software Quagga,* dengan menggunakan *daemon* zebra dan ospfd.
e. Pengujian dilakukan melalui protokol TCP dan UDP dengan menggunakan *tool iperf* dan *bwm-ng*.

1.4 Tujuan Penelitian

Penelitian ini bertujuan untuk membangun suatu *routing* adaptif melalui protokol *routing* OSPF untuk mendeteksi kemacetan yang terjadi dan melakukan modifikasi atau perubahan nilai *cost* untuk mengendalikan kemacetan, untuk kemudian mendistribusikan trafik berdasarkan rute yang baru.

1.5 Manfaat Penelitian

Hasil dari penelitian ini bermanfaat membantu *router* pada protokol *routing* OSPF untuk menentukan dan mengubah rute pendistribusian data secara dinamis, berdasarkan pertimbangan parameter *cost* dan kemacetan jaringan.

BAB II
TINJAUAN PUSTAKA

1.1 Penelitian Terkait

Nakahodo et al. menyajikan sebuah optimisasi *routing* OSPF untuk menghindari terjadinya *congestion* yang dinamakan *Smart-OSPF* (S-OSPF). Penemuan ini mengadopsi dari penelitian yang telah dikerjakan sebelumnya oleh Mishra et al. yang mengembangkan S-OSPF untuk mengurangi rasio *congestion* dengan cara mendistribusikan trafik hanya pada *edge node,* sedangkan *intermediate node* akan menerapkan mekanisme konvensional OSPF (Mishra & Sahoo, 2007). Hasil penelitian ini menunjukkan bahwa S-OSPF mencapai hasil rasio kemacetan jaringan yang lebih rendah dibandingkan dengan OSPF konvensional. Namun terjadi permasalahan pada saat akan menerapkan penambahan fungsi S-OSPF pada *router* yang saat ini sedang digunakan karena tidak semua *vendor* kompatible dengan fungsi ini. Untuk menangani masalah tersebut digunakan teknik *Hybrid Software Defined Networking* (H-SDN) yang diterapkan pada *edge node*. Konsep H-SDN adalah mengimplementasikan sistem SDN dan jaringan OSPF konvensional secara bersama – sama (Nakahodo, Naito and Oki, 2014)

Fenomena terjadinya *overload* dan kegagalan dalam proses *routing* dalam protokol *link-state*, membuat Jerry Ash et al. (Ash et al., 2002) mengajukan sebuah mekanisme yang bertujuan untuk mencegah terjadinya kemacetan, merespon secara baik apabila terjadi kegagalan dalam proses *routing*, dan melakukan perbaikan (*recovery*) terhadap dampak yang ditimbulkan dari kehilangan informasi dalam topology database. Mekanisme pengendalian dan pencegahan kemacetan jaringan ini diterapkan dalam protokol *routing* OSPF. Hal serupa dilakukan oleh G. Choudury (G. Choudhury, 2005) melalui sebuah penelitian yang bertujuan untuk meningkatkan skalabilitas dan stabilitas jaringan berskala besar yang menggunakan protokol *routing* OSPFv2. Mekanisme yang diperkenalkan dalam penelitian ini adalah memberikan prioritas tertinggi pada paket *hello* dan *Link-State Advertisement* (LSA) untuk mencegah terjadinya kemacetan (*congestion avoidance*).

Rasio terjadinya kemacetan dalam jaringan dapat diperkecil melalui pengontrolan trafik, melakukan mekanisme pencegahan maupun pengendalian kemacetan jaringan. Pengontrolan

trafik telah dilakukan oleh Borovina dan Kreso untuk membangun sebuah *routing adaptif* pada OSPF. Tujuan utama dari *routing adaptif* adalah untuk menyediakan proses pendistribusian trafik dalam jaringan secara optimal tanpa harus mengalami *packet drop*. Mekanisme ini dilakukan dengan mengumpulkan parameter yang berkaitan untuk melakukan perubahan nilai *cost* pada LSDB dengan menggunakan logika Fuzzy. Hasil penelitian menunjukkan bahwa terjadi peningkatan secara signifikan dalam hal efisiensi jalur dan berkurangnya jumlah paket yang hilang selama proses pendistribusian data pada *routing* adaptif OSPF dibandingkan dengan OSPF konvensional (Borovina and Kreso, 2005). Sedangkan mekanisme yang dapat dilakukan untuk mengendalikan kemacetan jaringan secara umum dibedakan menjadi 2 langkah, yaitu *detection* dan *action*. *Detection* menuntut jaringan untuk dapat mengetahui secara otomatis (mendeteksi) apakah jaringan tersebut sedang mengalami kemacetan atau beresiko terjadi kemacetan, sedangkan *action* merupakan langkah yang harus dilakukan oleh jaringan untuk merespon kemacetan yang telah terjadi (Medhi and Ramasamy, 2007).

1.2 *Routing*

Devais jaringan yang termasuk di dalam Layer *Network* , yaitu *switch* dan *router* memiliki pengetahuan yang cukup mengenai topologi jaringan yang ditangani sehingga keduanya dapat memilih *port* mana yang akan digunakan sebagai *output* untuk melewatkan paket. Dalam jaringan IP, *routing* merupakan salah satu factor penting bagi suatu paket. *Switch* atau *router* harus dapat melihat alamat tujuan untuk kemudian menentukan *port output* terbaik untuk paket yang akan menuju ke alamat tersebut. Keduanya (*switch* dan *router*) menentukan keputusan *routing* ini dengan melihat *forwarding table*, sehingga permasalahan mendasar dari sebuah *routing* adalah bagaimana cara *switch* dan *router* mendapatkan informasi dari *forwarding table*.

Dalam proses *routing* yang terjadi pada sebuah perangkat jaringan, terdapat dua buah table yang digunakan secara bersama, yaitu *forwarding table* dan *routing table*. *Forwarding table* digunakan pada saat paket akan diteruskan, dan harus berisi informasi yang cukup untuk menyelesaikan fungsi dari *forwarding* itu sendiri. Artinya, setiap abris dalam *forwarding table* berisi *mapping* atau pemetaan dari sebuah prefix jaringan untuk *outgoing interface*, serta beberapa informasi MAC seperti alamat *Ethernet* untuk lompatan (*hop*) berikutnya. Sedangkan *routing*

table, merupakan table yang dibangun oleh perhitungan algoritme *routing* untuk membangun *forwarding table*. Tabel ini akan berisi pemetaan dari prefix jaringan ke lompatan selanjutnya. Juga akan berisi bagaimana informasi pemetaan ini akan dipelajari, sehingga *router* dapat memutuskan kapan harus membuang beberapa informasi yang diterima.

Pada kasus-kasus tertentu, terdapat pemisahan struktur data antara tabel *forwarding* dan table *routing* pada implementasinya. Sebagai contoh, table *forwarding* disusun untuk mengoptimalkan proses pencarian alamat pada saat meneruskan paket, sedangkan table *routing* digunakan untuk melakukan perhitungan rute setiap kali terjadi perubahan topologi. Pada beberapa kasus, table *forwarding* diimplementasikan pada perangkat keras khusus. Gambar 2-1 menunjukkan contoh salah satu baris informasi yang terdapat dalam table *routing* dan table *forwarding*. Pada contoh ini, informasi pada table *routing* menginformasikan kepada kita bahwa jaringan dengan prefix 18/8 dicapai melalui *hop router* berikutnya dengan alamat IP 171.69.245.10, sedangkan table *forwarding* memberikan informasi bagaimana cara meneruskan paket melalui *hop* tersebut seperti berikut: "Kirimkan paket tersebut melalui *interface* nomor 0 dengan alamat MAC 8:0:2b:e4:b:1:2". Informasi ini akan disediakan oleh Protokol ARP (*Address Resolution Protokol*) (Peterson and Davie, 2012).

(a)

Prefix/Length	Next Hop
18/8	171.69.245.10

(b)

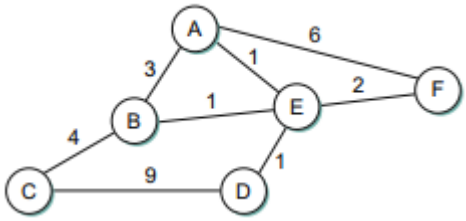
Prefix/Length	Interface	MAC Address
18/8	if0	8:0:2b:e4:b:1:2

Gambar 1-1. Contoh dari (a) Routing Table (b) Forwarding Table
Sumber: (Peterson and Davie, 2012)

1.2.1 Jaringan Sebagai Graf

Routing, pada dasarnya adalah sebuah permasalahan dari teori graf. Gambar 2-2 menunjukkan representasi graf dari sebuah jaringan. Node yang terdapat pada graf, dengan label A sampai F, digambarkan sebagai *host*, *switch*, *router* ataupun jaringan. Dalam studi pustaka ini, node akan difokuskan sebagai *router*. Setiap node akan saling terhubung dan

membentuk *link* yang masing-masing memiliki nilai *cost*, yang akan berdampak kepada pertimbangan dalam melakukan pengiriman trafik.

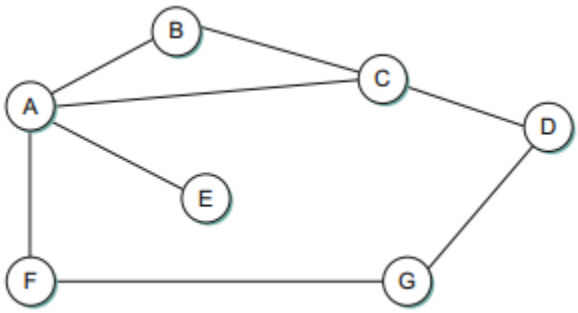


Gambar 1-2. Representasi Jaringan sebagai Graf
Sumber: (Peterson and Davie, 2012)

Permasalahan jaringan yang harus diselesaikan dalam proses *routing* adalah bagaimana cara menemukan jalur antar dua node dengan nilai *cost* yang terendah, dimana nilai *cost* sebuah jalur merupakan jumlah nilai *cost* dari semua node yang membentuk jalur. Untuk menyelesaikan ini, *routing* dicapai dengan menjalankan protokol *routing* antar node. Protokol ini menyediakan cara dinamis untuk mencari jalur dengan nilai *cost* terendah dengan memperhatikan adanya kegagalan *link* dan perubahan nilai *cost*. Secara umum, protokol *routing* digunakan pada semua jaringan terdistribusi. Dua kelas utama protokol *routing* yang digunakan saat ini antara lain *distance-vector* dan *link-state* (Peterson and Davie, 2012).

1.2.2 *Distance-Vector*

Setiap node menyusun array satu dimensi (vektor) yang berisi "jarak" (*cost*) ke semua node lain dan mendistribusikan vektor itu ke tetangga terdekatnya. Asumsi awal untuk *routing distance-vector* adalah setiap node mengetahui biaya link (*cost*) ke masing-masing secara langsung tetangga yang terhubung. Nilai ini bisa diberikan saat *router* dikonfigurasi oleh manajer jaringan. Sebuah link yang *down*, nilai *cost* akan didefinisikan tak terbatas (∞).



Gambar 1-3. Contoh Kasus Algoritme *Distance-Vector*
Sumber: (Peterson and Davie, 2012)

Untuk memahami bagaimana algoritme *routing idstance-vector* bekerja, kita melihat sebuah contoh pada Gambar 2-3. Pada contoh ini, nilai *cost* setiap *link* diatur nilai 1, jadi jalur dengan nilai *cost* terkecil adalah *hop* berikutnya. Representasi informasi mengenai jarak untuk semua node tertera pada Tabel 2-1.

Tabel 1-1. Jarak Awal yang Disimpan di Setiap Node (Tampilan Global)

Information Stored at Node	Distance to Reach Node						
	A	B	C	D	E	F	G
A	0	1	1	∞	1	1	∞
B	1	0	1	∞	∞	∞	∞
C	1	1	0	1	∞	∞	∞
D	∞	∞	1	0	∞	∞	1
E	1	∞	∞	∞	0	∞	∞
F	1	∞	∞	∞	∞	0	1
G	∞	∞	∞	1	∞	1	0

Awalnya, setiap node akan mengatur nilai *cost* 1 untuk setiap *router* terdekatnya, dan nilai ∞ untuk semua node. Sebagai contoh, node A dapat mencapai node B dalam satu lompatan, sedangkan node D adalah *unreachable*. Sehingga, table *routing* yang disimpan pada node A terlihat pada Tabel 2-2.

Tabel 1-2. Tabel Routing Awal Node A

Destination	Cost	NextHop
B	1	B
C	1	C
D	2	C
E	1	E
F	1	F
G	2	F

Langkah selanjutnya dalam *routing distance-vector* adalah setiap node mengirimkan sebuah pesan ke tetangga terdekat yang terhubung secara langsung. Sebagai contoh, node F akan memberitahu node A, bahwa untuk mencapai node F bisa silakukan melalui node G dengan *cost* 1; A juga tahu itu bisa mencapai F dengan *cost* 1, sehingga untuk mencapai node G total nilai *cost* yang dibangun adalah 2. Demikian proses akan terus berlangsung untuk node-node yang lain,sehingga pada akhirnya node A akan melakukan *update* terhadap *table routing* nya untuk semua node yang terdapat pada jaringan. Dengan tidak adanya perubahan topologi, hanya perlu beberapa pertukaran informasi antara tetangga sebelum masing-masing node memiliki tabel routing yang lengkap. Proses mendapatkan informasi routing yang konsisten ke semua node disebut konvergensi. Tabel 2-3 menunjukkan table *routing* final dari setiap node ke semua node setelah *routing* telah berhasil terkonvergensi (Peterson and Davie, 2012).

Tabel 1-3. Jarak Akhir yang Disimpan di Setiap Node (Tampilan Global)

Information Stored at Node	Distance to Reach Node						
	A	B	C	D	E	F	G
A	0	1	1	2	1	1	2
B	1	0	1	2	2	2	3
C	1	1	0	1	2	2	2
D	2	2	1	0	3	2	1
E	1	2	2	3	0	2	3
F	1	2	2	2	2	0	1
G	2	3	2	1	3	1	0

1.2.3 *Link-State*

Link-State merupakan algoritme *routing* kedua dalam protokol *routing*. Ide dasar di balik protokol *link-state* sangat sederhana: Setiap node tahu bagaimana cara mencapai tetangga yang terhubung langsung, dan jika kita memastikan bahwa keseluruhan pengetahuan ini disebarluaskan ke setiap node, maka setiap node akan memiliki cukup pengetahuan tentang jaringan untuk membangun peta lengkap jaringan. Ini jelas kondisi yang cukup (walaupun tidak perlu) untuk menemukan jalur terpendek ke titik manapun dalam jaringan. Dengan demikian, protokol routing *link-state* mengandalkan dua mekanisme: penyebaran informasi *link-state* yang andal, dan perhitungan rute dari jumlah semua pengetahuan *link-state* yang terakumulasi (Peterson and Davie, 2012).

1. *Flooding*

Flooding adalah proses untuk memastikan bahwa semua node yang berpartisipasi dalam protokol *routing* mendapatkan salinan informasi *link-state* dari semua node lainnya. Ide dasarnya adalah untuk sebuah node untuk mengirim informasi *link-state*-nya ke semua link yang terhubung langsung; setiap node yang menerima informasi ini kemudian meneruskannya keluar pada semua *link* nya. Proses ini berlanjut sampai informasi diterima semua node dalam jaringan.

Lebih tepatnya, setiap node membuat *update packet*, disebut juga *linkstate packet* (LSP), yang berisi informasi berikut:

- ID dari node yang menciptakan LSP
- Daftar tetangga yang terhubung langsung dengan simpul tersebut, dengan biaya link ke masing-masing
- Nomor urut
- *Time to Live* untuk paket ini

Dua poin pertama dibutuhkan untuk memungkinkan perhitungan rute; dua poin yang terakhir digunakan untuk membuat proses *flooding* paket ke semua node. Setiap node menghasilkan LSP dalam dua keadaan. Timer periodik yang kadaluarsa atau perubahan

topologi dapat menyebabkan node menghasilkan LSP baru. Namun, satu-satunya alasan berbasis topologi untuk sebuah node untuk menghasilkan LSP adalah jika salah satu dari tautan yang terhubung langsung atau tetangga yang berdekatan *down*. Kegagalan suatu *link* dapat dideteksi dalam beberapa kasus oleh protokol layer *link*. Kematian tetangga atau kerugian konektivitas ke tetangga tersebut dapat dideteksi dengan menggunakan paket "halo" secara periodik. Setiap node mengirimkan ini ke tetangga terdekatnya pada interval yang ditentukan. Jika waktu yang cukup lama berlalu tanpa menerima "halo" dari tetangga, tautan ke tetangga tersebut akan dinyatakan *down*, dan LSP baru akan dihasilkan.

Salah satu tujuan perancangan penting dari mekanisme *flooding* pada protokol *link-state* adalah bahwa informasi terbaru harus diinformasikan ke semua node secepat mungkin, sementara informasi lama harus dikeluarkan dari jaringan dan tidak diizinkan untuk beredar. Selain itu, untuk meminimalkan jumlah total lalu lintas perutean yang dikirim di sekitar jaringan (Peterson and Davie, 2012).

2. *Perhitungan Rute*

Setelah simpul tertentu memiliki salinan LSP dari setiap simpul lainnya, ia dapat melakukannya hitung peta lengkap untuk topologi jaringan, dan dari peta ini ia dapat menentukan rute terbaik untuk setiap tujuan. Pertanyaannya adalah bagaimana menghitung rute dari informasi ini. Solusinya didasarkan pada algoritme yang terkenal dari teori graf-algoritme jalur terpendek Dijkstra. Penjelasan mengenai algoritme ini terdapat di Sub-Bab 2.4 (Peterson and Davie, 2012).

1.3 *Open Shortest Path First* (OSPF)

Open Shortest Path First (OSPF) merupakan salah satu *Interior Gateway Routing Protokol* (IGRP), berbasis teknologi *Shortest Path First* (SPF) atau *link-state*. OSPF didesain secara spesifik untuk jaringan TCP/IP dan mendukung beberapa fitur seperti autentikasi *routing update*, penandaan jalur yang berasal dari luar jaringan, menanggapi perubahan topologi jaringan secara cepat, dan pembagian sumber daya pada jaringan terpusat. Dalam protokol routing SPF, masing –

masing *router* akan membuat sebuah *database* yang mendeskripsikan topologi *Autonomous System* (AS). Masing – masing *router* memiliki *database* yang identik, yang mendeskripsikan kondisi jaringan pada sebuah, yaitu informasi mengenai *interface* yang tersedia, *neighbor router*, dan informasi lain berkaitan dengan jalur pendistribusian data (Telesis, 2005).

Selain itu, OSPF juga merupakan protokol *routing* yang berstandar terbuka. Maksudnya adalah protokol routing ini bukan ciptaan dari *vendor* manapun. Dengan demikian, siapapun dapat menggunakannya, perangkat manapun dapat kompatibel dengannya, dan dimanapun protokol *routing* ini dapat diimplementasikan. OSPF merupakan protokol *routing* yang menggunakan konsep hirarki *routing*, artinya OSPF membagi-bagi jaringan menjadi beberapa tingkatan. Tingkatan-tingkatan ini diwujudkan dengan menggunakan sistem pengelompokan area (Thomas, 2003).

Sebagai salah satu protokol *routing link-state* yang paling banyak digunakan, OSPF menambahkan cukup banyak fitur algoritme dasar *link-state*, antara lain sebagai berikut.

1. Autentikasi pesan routing

   Salah satu fitur dari algoritme routing terdistribusi adalah bahwa penyebaran informasi dari satu node ke banyak node lainnya, dan keseluruhan jaringan dapat dipengaruhi oleh informasi buruk dari satu node. Untuk alasan ini, ada baiknya untuk memastikan bahwa semua node yang mengambil bagian dalam protokol dapat dipercaya. Autentikasi pesan *routing* akan membantu mencapai hal ini. Versi awal OSPF menggunakan kata sandi 8 byte sederhana untuk autentikasi. Ini bukanlah bentuk autentikasi yang cukup kuat untuk mencegah adanya kejahatan dalam jaringan, namun ini meringankan beberapa masalah yang disebabkan oleh kesalahan konfigurasi atau serangan biasa (Peterson and Davie, 2012).

2. *Routing* hirarki

   Dengan menggunakan konsep hirarki *routing* ini sistem penyebaran informasinya menjadi lebih teratur dan tersegmentasi, tidak menyebar dengan sembarangan. Efek dari keteraturan distribusi *routing* ini adalah jaringan yang penggunaan *bandwidth*-nya lebih efisien, lebih cepat mencapai konvergensi, dan lebih presisi dalam menentukan rute-rute terbaik menuju ke

sebuah lokasi. Hal ini membuat protokol *routing* OSPF menjadi sangat cocok untuk terus dikembangkan menjadi jaringan berskala besar (Telesis, 2005).

3. *Load Balancing*

OSPF mengijinkan adanya penggunaan beberapa rute yang memiliki node tujuan yang sama dengan nilai *cost* yang sama. Hal ini akan menyebabkan trafik akan didistribusikan secara merata pada rute tersebut, sehingga dapat memanfaatkan kapasitas jaringan yang ada dengan lebih baik (Peterson and Davie, 2012).

1.3.1 Konsep *Routing* dalam OSPF

Secara garis besar, proses yang dilakukan protokol *routing* OSPF mulai dari awal hingga dapat saling bertukar informasi ada lima langkah. Berikut ini adalah langkah-langkahnya (Kurose and Ross, 2013):

1. Membentuk *Adjacency Router*

   Tahapan pertama dalam OSPF adalah membentuk *adjacency*, yaitu membangun hubungan dengan *router* terdekat (tetangga), melalui hello protokol yang dikiriman melalui alamat *multicast* sebagai tujuan dan disertai field yang berisi *neighbour ID*. Yang kemudian dilanjutkan pertukaran informasi sesuai media yang digunakan.

2. Memilih DR dan BDR (jika diperlukan)

   Pada media *broadcast multi-access* (BMA), pemilihan DR sangat diperlukan sebagai penyebar informasi tentang keadaan semua *router* yang ada dalam jaringan, semua paket yang dikirim dalam jaringan akan disebar melalui DR atau BDR. Oleh karena itu, pemilihan DR atau BDR merupakan proses penting. Penentuannya berdasarkan nilai prioritas yang dimiliki tiap *router* dengan nilai prioritas 0-255. Posisi DR dan BDR ini tidak akan berubah sampai terjadi kerusakan pada *router* (DR dan BDR) tersebut, walaupan pada satu waktu ada *router* yang memiliki nilai prioritas lebih tinggi. sedangkan jika ada 2 buah *router* memiliki nilai priritas yang sama, maka yang terpilih yang memiliki router ID tertinggi.

3. Mengumpulkan *State-state* dalam Jaringan

Setelah terbentuk hubungan antar *router*, berikutnya akan melakukan pertukaran informasi *state* dan jalu-jalur yang ada di dalam jaringan. Berikutnya akan ditentukan mana yang sebagai *master* (mengirim data terlebih dulu) dan mana sebagai *slave* (mendengarkan lebih dulu) . Setelah itu, dilakukan fase *Exchange* dimana dilakukan pertukaran *Database Description Packet*, yang berisi tentang seluruh ringkasan status untuk seluruh media yang ada dalam jaringan. Jika informasi yang diperlukan masih belum ada di basis data , maka pengirim akan melakukan pengiriman semua informasi *state* yang ada dalam jaringan (fase *Loading State*). Dan router siap untuk melakukan penyampaian data.

4. Memilih Rute Terbaik untuk Digunakan

Setelah informasi seluruh jaringan berada dalam *database*, maka kini saatnya untuk memilih rute terbaik untuk dimasukkan ke dalam *routing table*. Jika sebuah rute telah masuk ke dalam *routing table*, maka rute tersebut akan terus digunakan. Untuk memilih rute-rute terbaik, parameter yang digunakan oleh OSPF adalah *Cost*. Metrik *Cost* biasanya akan menggambarkan seberapa dekat dan cepatnya sebuah rute. Router OSPF akan menghitung semua *cost* yang ada dan akan menjalankan algoritme *Shortest Path First* untuk memilih rute terbaiknya. Setelah selesai, maka rute tersebut langsung dimasukkan dalam *routing table* dan siap digunakan untuk *forwarding data* (Kurose and Ross, 2013).

5. Menjaga Informasi Routing Tetap *Up-to-da**te***

Ketika sebuah rute sudah masuk ke dalam *routing table*, router tersebut harus juga me-*maintain state database*-nya. Hal ini bertujuan apabila ada sebuah rute yang sudah tidak *valid*, maka *router* harus tahu dan tidak boleh lagi menggunakannya. Ketika ada perubahan *link-state* dalam jaringan, OSPF *router* akan melakukan *flooding* terhadap perubahan ini. Tujuannya adalah agar seluruh *router* dalam jaringan mengetahui perubahan tersebut. Sampai di sini semua proses OSPF akan terus berulang-ulang. Mekanisme seperti ini membuat informasi rute-rute yang ada dalam jaringan terdistribusi dengan baik, terpilih dengan baik dan dapat digunakan dengan baik pula (Kurose and Ross, 2013).

1.4 Algoritme Djikstra

Algoritme Dijkstra ditemukan oleh Edsger W. Dikstra dan dipublikasikan pada tahun 1959 pada sebuah jurnal *Numerische Mathematik* yang berjudul *A Note on Two Problems in Connexion with Graphs* (Devetak and Kapoor, 2017). Algoritme Dijkstra merupakan salah satu jenis algoritme *Shortest Path Computation* (SPF) yang paling banyak digunakan di berbagai aplikasi kehidupan. Algoritme ini digunakan untuk mencari lintasan terpendek pada sebuah graf berarah dengan menggunakan nilai satuan bobot (*cost*). Cara kerja algoritme Dijkstra memakai stategi *greedy*, dimana pada setiap langkah dipilih sisi dengan bobot terkecil yang menghubungkan sebuah simpul yang sudah terpilih dengan simpul lain yang belum terpilih (Nugroho, Lamaida and Ahsana, 2006).

Kontribusi algoritme Djikstra dalam dunia komputer secara khusus dalam penentuan jalur terpendek antara 2 titik atau lebih pada kumpulan *node* atau yang sering diimplementasikan sebagai *router* dalam dunia *routing*. OSPF merupakan salah satu protokol *routing* yang menggunakan algoritme Dijkstra dalam proses perhitungan jalurnya. Masing – masing *router* dalam OSPF akan secara otomatis mengatur nilai *cost* berdasarkan alokasi *bandwidth* yang tersedia pada masing- masing *interface*, atau dapat juga diatur secara manual oleh *administrator* jaringan. Nilai *cost* dalam sebuah *interface* memiliki nilai berbanding terbalik dengan nilai *bandwidth* pada *interface* tersebut. Nilai *bandwidth* tertinggi mengindikasikan nilai *cost* terkecil. Persamaan (2-1) menunjukkan rumus yang digunakan untuk menghitung nilai *cost* secara otomatis (Thomas, 2003). Sedangkan pada Gambar 2.4 menunjukkan parameter *bandwidth* pada berbagai jenis *interface* yang digunakan untuk menentukan nilai *cost* secara otomatis.

$$Cost = \frac{Reference\_Bandwidth}{Interface\_Bandwidth} \qquad (2\text{-}1)$$

di mana, *Reference_Bandwidth* = 10^8 in bps

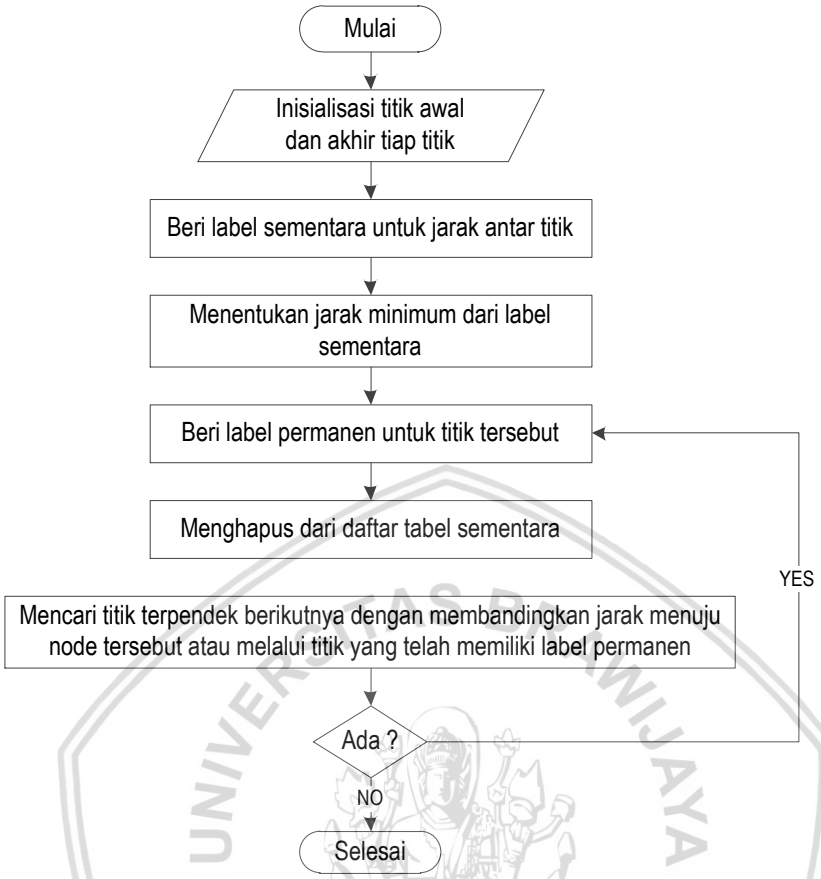
Interface Type	10^8/ bps = Cost
Fast Ethernet and faster	10^8/ 100,000,000 bps = 1
Ethernet	10^8/ 10,000,000 bps = 10
E1	10^8/ 2,048,000 bps = 48
T1	10^8/ 1,544,000 bps = 64
128 kbps	10^8/ 128,000 bps = 781
64 kbps	10^8/ 64,000 bps = 1562
56 kbps	10^8/ 56,000 bps = 1785

Gambar 1-4 Parameter Bandwidth *Nilai* Cost
Sumber: (Kurose and Ross, 2013)

Kecepatan pengiriman data pada masing-masing interface dapat berubah secara dinamis, sehingga hal tersebut akan berdampak pada nilai cost pada interface yang dikonfigurasi secara otomatis. Sedangkan pada manual cost, pengiriman data akan didistribusikan pada interface dengan nilai cost terkecil tanpa terpengaruh dengan adanya perubahan kecepatan pada masing – masing interface (Telesis, 2005).

Setiap akan melakukan proses komputasi jalur, Djisktra akan melakukan inisialisasi titik awal dan jarak untuk setiap titik dalam topologi jaringan. Hal ini kemudian akan dilanjutkan dengan pemberian nilai jarak semetara untuk menentukan jarak minimum setiap titik secara sementara. Jika telah memenuhi prosedur di atas, maka Djikstra akan memberikan label secara permanen terhadap setiap titik, untuk kemudian akan mencari jalur terpendek. Pencarian jalur terpendek dilakukan dengan cara melakukan perbandingan jarak setiap titik yang terhubung langsung dengan titik awal. Apabila telah selesai melakukan perbandingan, maka Djikstra akan meneruskan paket data berdasarkan jalur terpendek yang telah dipilih.

Diagram alur mekanisme pencarian rute terpendek pada algoritme Dijkstra diberikan pada Gambar 2-5.



Gambar 1-5 Diagram Alur Algoritme Djikstra
Sumber: (Thomas, 2003)

Algoritme Dijkstra melakukan komputasi jalur dengan nilai *cost* terkecil dari satu node (node sumber, dengan notasi *u*) ke semua node dalam jaringan. Dijkstra merupakan algoritme iteratif dan memiliki sifat bahwa setelah iterasi ke - *k* dari proses algoritme, jalur dengan nilai *cost* terkecil diketahui oleh *k* node tujuan, dan di antara semua jalur, *k* memiliki jalur dengan nilai *cost* terkecil. Berikut ini merupakan definisi notasi yang digunakan dalam Dijkstra.

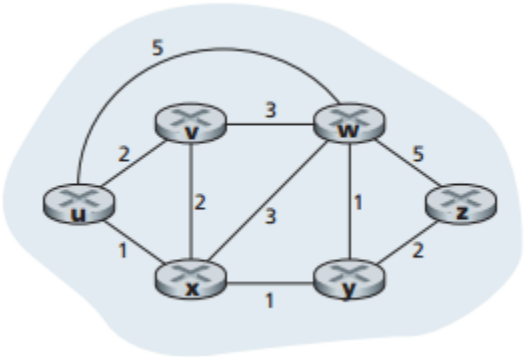
- *D(v)* : nilai *cost* terkecil dari node sumber ke node tujuan *v* sebagai hasil iterasi algoritme
- *p(v)* : node sebelumnya (tetangga dari *v*) sepanjang jalur terpendek dari node sumber ke *v*
- *N* : subset dari node; *v* merupakan salah satu bagian *N* jika jalur terpendek dari node sumber ke *v* telah diketahui.

Algoritme *routing* secara global terdiri dari sebuah inisialisasi langkah yang dilakukan secara berulang. Jumlah perulangan yang dieksekusi adalah sama dengan jumlah node yang terdapat pada jaringan. Setelah perulangan berhenti, algoritme akan melakukan perhitungan jalur terpendek dari node sumber *u* ke semua node yang terdapat pada jaringan. Gambar 2-6 menunjukkan inisialisasi algoritme Dijkstra dalam melakukan komputasi jalur terpendek.

```
1  Initialization:
2    N' = {u}
3    for all nodes v
4      if v is a neighbor of u
5        then D(v) = c(u,v)
6      else D(v) = ∞
7
8  Loop
9    find w not in N' such that D(w) is a minimum
10   add w to N'
11   update D(v) for each neighbor v of w and not in N':
12         D(v) = min( D(v), D(w) + c(w,v) )
13   /* new cost to v is either old cost to v or known
14    least path cost to w plus cost from w to v */
15 until N'= N
```

Gambar 1-6. Algoritme Dijkstra dari Node Sumber *u*
Sumber: (Kurose dan Ross, 2013)

Contoh operasi perhitungan dan penentuan rute menggunakan Algoritme Djikstra dalam aplikasi jaringan sederhana diberikan pada Gambar 2-7. Jaringan computer dimodelkan seperti graf berarah, Algoritme Dijkstra akan melakukan komputasi jalur dengan nilai *cost* terkecil dari node u ke semua node destinasi yang memungkinkan. Sebuah kesimpulan secara tabulasi dari perhitungan algoritme ini ditunjukkan pada Tabel 2-4 , dimana setiap baris yang terdapat pada tabel menunjukkan nilai dari setiap variable algoritme dari setiap iterasi perhitungan.



Gambar 1-7. Pemodelan Graf Sebuah Jaringan Komputer

Sumber: (Kurose and Ross, 2013)

Berikut ini merupakan langkah yang dilakukan dalam komputasi Dijkstra.

1. Pada langkah inisialisasi awal, jalur dengan nilai *cost* terendah yang diketahui adalah jalur dari node u ke tetangga terdekatnya yang langsung terhubung, yaitu node v, w, dan x. Ketiganya diinisialisasikan nilai *cost* masing-masing 2, 1, dan 5. Sedangkan, nilai *cost* menuju node y dan z ditentukan dengan nilai *infinity*, karena mereka tidak langsung terhubung ke node u.
2. Pada iterasi yang pertama, terlihat node yang belum ditambahkan ke himpunan N dan menemukan node dengan *cost* terendah pada iterasi sebelumnya. Node tersebut adalah x, dengan nilai *cost* 1, kemudian tambahkan x ke dalam himpunan N. Baris ke-12 dari algoritme Dijkstra pada Gambar 2-6 dilakukan untuk memperbarui informasi pada D(v) untuk semua node v, yang menghasilkan informasi di baris kedua pada Tabel 2-4. Nilai *cost* menuju ke node v tidak berubah. Nilai *cost* menuju node w (memiliki nilai 5 pada akhir inisialisasi) melalui node x diketahui adalah 4. Sehingga, jalur ini yang dipilih berdasarkan nilai *cost* terendah, yaitu untuk menuju ke node w dipilih melalui node x. Demikian pula, nilai *cost* menuju node y (melalui x) dihitung nilainya adalah 2, dan tabel akan di-*update* sesuai dengan perhitungan tersebut.
3. Pada iterasi kedua, node v dan node y ditemukan memiliki nilai *cost* terendah, menambahkan node y ke himpunan N, sehingga sekarang berisi u,x dan y. Nilai *cost* untuk node yang tersisa di himpunan N antara lain node v, w, dan z. Proses pembaharuan ini dilakukan pada langkah ke -12 dari algoritme Dijkstra yang tercantum pada Gambar 2-6, dan menghasilkan perhitungan yang ditunjukkan pada baris ketiga dari Tabel 2-4.
4. Begitu seterusnya dilakukan, sampai didapatkan hasil bahwa masing-masing node memiliki rute menuju ke seluruh node dalam jaringan.

Tabel 1-4. Perhitungan Algoritme Dijkstra

step	N′	D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4	uxyvw					4,y
5	uxyvwz					

1.5 Utilisasi Trafik Jaringan (*Bandwidth Utilization*)

Utilisasi merupakan parameter yang menunjukkan seberapa besar prosentase suatu sumber daya yang digunakan. Dalam hal ini sumber daya yang dimaksud adalah *bandwidth* suatu *link* yang menghubungkan antara kedua sisi yaitu sisi pengirim dan penerima (*end-to-end system*). Utilisasi bandwidth menunjukan rasio antara ukuran bandwidth total yang terpakai dengan bandwidth yang tersedia pada jalur tersebut. Persamaan (2-2) menunjukkan rumus yang digunakan untuk menghitung nilai utilisasi *bandwidth* dari sebuah *link*.

$$Utilisasi(\%) = \frac{Data\ Throughput\ Terukur}{Kapasitas\ Bandwidth\ yang\ Tersedia} x100\% \qquad (2\text{-}2)$$

Sumber: (Muslim, 2007)

Besar nilai dari presentase utilisasi yang diperoleh menunjukkan kemungkinan terjadinya kemacetan lalu lintas (*traffic congestion*) dalam suatu jaringan. Semakin besar nilai presentase utilisasi, menunjukkan tingkat kemacetan dalam jaringan semakin besar, hal ini akan mengakibatkan penurunan kinerja dalam jaringan (Borovina and Kreso, 2005).

1.6 Performa Jaringan

Jaringan Internet dapat dipandang sebagai infrastruktur yang menyediakan layanan untuk aplikasi terdistribusi yang berjalan pada *end system.* Secara ideal, *user* menginginkan adanya layanan Internet yang dapat mentransmisikan data antara dua *end system* secara instan, tanpa

adanya kehilangan data. Sayangnya, hal ini tidak selalu dapat dicapai pada dunia nyata. Sebagai gantinya, jaringan computer harus dapat membatasi nilai *throughput* (jumlah data per detik yang dapat dikirim) antara *end system*, mempertimbangkan adanya *delay* antar *end system*, dan adanya *packet loss*. Berikut ini akan dijelaskan pengukuran terhadap performansi jaringan computer yang meliputi nilai *delay, packet loss* dan *throughput*.

1.6.1 *Delay*

Suatu pengiriman paket memiliki alur sebagai berikut: dikirimkan dari sebuah *host* (sumber) → melewati serangkaian *router* → dan mengakhiri perjalanannya di *host* lain (tujuan). Sebagai satu mekanisme perjalanan dari satu node (*host* atau *router*) menuju ke node berikutnya (*host* atau *router*), paket akan mengalami beberapa jenis penundaan (*delay*) pada setiap node. Performansi dari berbagai aplikasi Internet, seperti *search-engine, Web browsing, e-mail, maps, instant messaging* dan VoIP, sangatlah dipengaruhi oleh *delay* jaringan. Untuk mendapatkan pemahaman yang mendalam mengenai *packet-switching* dan jaringan computer, kita harus memahami sifat dan pentingnya nilai *delay* dalam pentransmisian data. Nilai-nilai *delay* yang dimaksud meliputi *processing delay, transmission delay*, dan *propagation delay*; yang ketiganya diakumulasikan untuk menghasilkan nilai *nodal delay* (Kurose and Ross, 2013).

1.6.2 *Packet Loss*

Setiap *intermediary node* terdapat kapasitas antrian yang dapat ditangani. Pada kenyataannya antrian yang mendahului sebuah link memiliki kapasitas yang terbatas, walaupun kapasitas antrian sangat bergantung pada desain dan biaya router. Karena kapasitas antrian terbatas, paket bisa sampai menemukan antrian penuh. Dengan tidak adanya tempat untuk menyimpan paket seperti itu, router akan menjatuhkan paket itu; Artinya, paketnya akan hilang (*packet loss*). Dilihat dari sudut pandang sistem akhir (*end system*), *packet loss* diartikan paket yang ada telah ditransmisikan ke jaringan namun tidak pernah muncul di node tujuan. Fraksi paket yang hilang meningkat seiring dengan meningkatnya intensitas lalu lintas. Oleh karena itu,

kinerja pada node sering diukur tidak hanya dalam hal *delay*, tapi juga dalam hal probabilitas *packet loss* (Kurose and Ross, 2013).

1.6.3 *Throughput*

Selain *delay* dan *packet loss*, ukuran kinerja penting lainnya di jaringan 19rotocol adalah *end-to-end throughput*. Untuk menentukan *throughput*, dimisalkan seperti mentransfer file besar dari Host A ke Host B di jaringan 19rotocol. Hasil yang ditampilkan saat itu adalah kecepatan (dalam bit / detik) di mana Host B menerima file tersebut. Jika file terdiri dari bit F dan transfer membutuhkan T detik untuk Host B untuk menerima semua F bit, maka *throughput* rata-rata transfer file adalah F / T bits / sec. Untuk beberapa Aplikasi, seperti telepon Internet, sangat diharapkan untuk memiliki *delay* yang rendah dan *throughput* sesaat secara konsisten di atas beberapa ambang batas (misalnya, berakhir 24 kbps untuk beberapa aplikasi *telephony internet* dan lebih dari 256 kbps untuk beberapa aplikasi video *realtime*). Untuk aplikasi lain, termasuk yang melibatkan file Transfer, keterlambatan tidak penting, tapi sangat diharapkan untuk memiliki yang setinggi mungkin throughput.

Bila digunakan dalam konteks jaringan komunikasi, seperti *Ethernet* atau packet radio, *throughput* adalah tingkat pengiriman pesan yang berhasil melalui saluran komunikasi. Data pesan-pesan ini dapat dikirim melalui tautan fisik atau logis, atau bisa melewati simpul jaringan tertentu. *Throughput* biasanya diukur dalam bit per detik (bit / s atau bps), dan terkadang dalam paket data per detik (p / s atau pps) atau paket data per slot waktu.

Throughput sistem adalah jumlah dari kecepatan data yang dikirimkan ke semua terminal dalam sebuah jaringan. *Throughput* pada dasarnya 19rotoco dengan konsumsi *bandwidth* digital; Hal ini dapat dianalisis secara matematis dengan menerapkan teori antrian, dimana beban dalam paket per satuan waktu dilambangkan sebagai tingkat kedatangan (λ), dan *throughput*, di mana penurunan paket per satuan waktu, dilambangkan sebagai tingkat keberangkatan (μ). *Throughput* sistem komunikasi dapat dipengaruhi oleh berbagai 19rotoc, termasuk keterbatasan media fisik analog yang mendasarinya, tersedia kekuatan pemrosesan komponen sistem, dan perilaku pengguna akhir (Kurose and Ross, 2013).

1.6.4 *Jitter*

Jitter didefinisikan sebagai variasi dalam penundaan paket yang diterima. Sisi pengirim mentransmisikan paket dalam aliran terus menerus dan memberi ruang secara merata. Karena kemacetan jaringan, antrian yang tidak semestinya, atau kesalahan konfigurasi, hal-hal ini dapat menyebabkan terjadinya penundaan antar paket bisa bervariasi, bukan konstan (Cisco Systems, 2005).

1.7 Kemacetan Jaringan

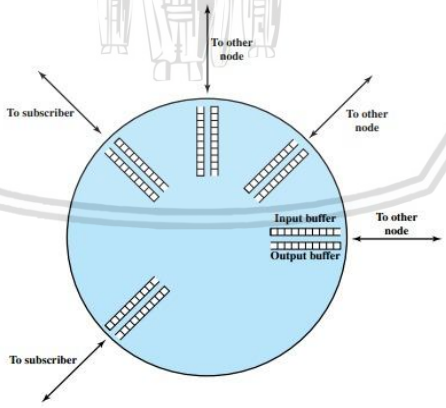
Kemacetan terjadi ketika jumlah paket yang ditransmisikan melalui jaringan mulai mendekati kapasitas penanganan paket pada sebuah jaringan. Pada intinya, sebuah jaringan data atau internet merupakan sebuah antrian jaringan. Setiap node (baik itu berupa *switch* atau *router*) pasti akan didapati sebuah antrian paket-paket untuk setiap kanal keluarannya. Jika jumlah paket yang datang dan antrian yang ada telah melebihi kapasitas paket yang dapat ditransmisikan, maka ukuran antrian akan semakin besar dan meningkatkan *delay* pada selama pengiriman paket berlangsung. Bahkan, jika kapasitas paket yang datang kurang dari kapasitas transmisi, maka antrian juga akan meningkat secara signifikan sampai kapasitas paket yang masuk mencapai kapasitas transmisi. Sebagaimana aturan yang telah ditetapkan, ketika jalur yang dilewati oleh paket mengalami antrian hingga lebih dari 80% kapasitasnya, maka panjang antrian ini berada pada tingkat yang mengkhawatirkan. Terbatasnya ukuran antrian yang dapat ditangani dan pertumbuhan panjang antrian memiliki dampak kepada peningkatan *delay* pada paket (Stallings, 2007).

Berbagai mekanisme TCP secara khusus dikembangkan untuk menyediakan layanan transfer data yang handal dalam menghadapi adanya *packet loss*. Pada dasarnya, *loss* terjadi sebagai akibat adanya meluapnya kapasitas *buffer* pada *router* pada saat jaringan mengalami kepadatan. Adanya *packet retransmission* merupakan salah satu gejala terjadinya kemacetan jaringan, yang bisa saja terjadi akibat trafik dari berbagai sumber mengirimkan data dengan kapasitas yang tinggi ke destinasi yang sama. Untuk menangani kemacetan diperlukan mekanisme khusus untuk

mengurangi atau menutup pengiriman dari *sender* selama kemacetan jaringan berlangsung (Kurose and Ross, 2013).

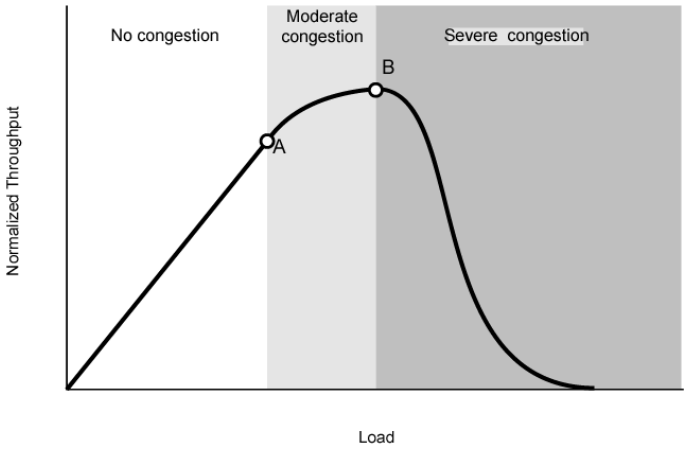
1.7.1 Dampak Kemacetan Jaringan

Situasi antrian pada sebuah *packet switch* atau *router* diilustrasikan pada Gambar 2-5. Setiap node memiliki sejumlah port I/O sebagai berikut: satu atau lebih untuk ke node lain, dan nol atau lebih untuk ke *end-system*. Pada setiap *port*, paket akan datang dan keluar. Artinya, terdapat dua *buffer* atau antrian pada setiap *port*, satu untuk menerima paket yang datang, dan yang lain untuk menahan paket yang menunggu untuk dikirimkan. Setiap paket yang datang akan disimpan ke dalam *buffer input* sesuai dengan *port*-nya. Node akan memeriksa setiap paket datang, membuat keputusan *routing*, dan kemudian memindahkan paket tersebut ke *buffer output* yang sesuai. Paket yang mengantri pada *buffer output* akan ditransmisikan secepat mungkin (dengan menggunakan pembagian waktu statistic *multiplexing*). Jika paket yang datang terlalu cepat bagi node untuk melakukan proses *routing*, atau lebih cepat dibandingkan waktu yang diperlukan untuk menghapus paket pada *buffer output*, maka pada akhirnya paket akan tiba tanpa menyimpan *memory*.



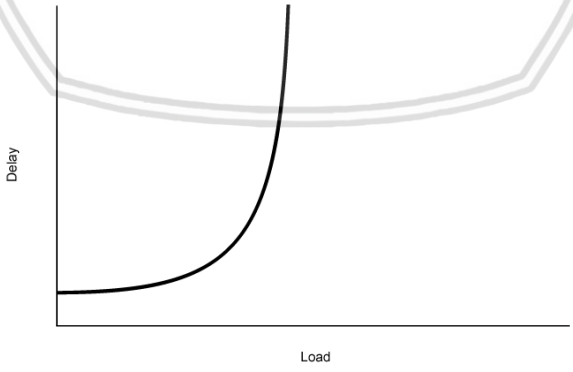
Gambar 1-8. Antrian *input-output* pada sebuah node
Sumber: (Stallings, 2007)

Beberapa dampak yang ditimbulkan akibat kemacetan yang terjadi jika ditinjau berdasarkan performa jaringan itu sendiri, disimpulkan dalam 3 jenis, yaitu dari sisi nilai *throughput, delay,* dan jumlah paket yang tidak berhasil dikirimkan (*packet loss*).



Gambar 1-9. Performa Jaringan Akibat Terjadinya Kemacetan (*Throughput*)
Sumber: (Thomas, 2003)

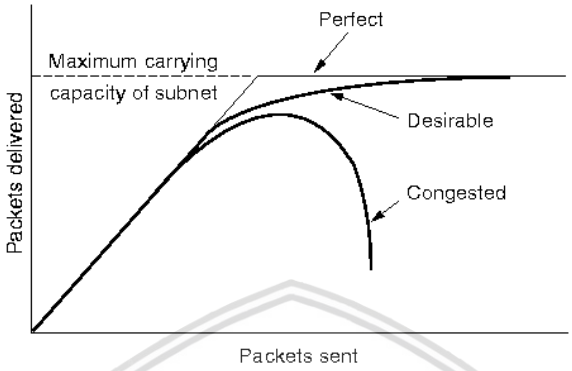
Gambar 2-10 menunjukkan performa jaringan ditinjau dari sisi nilai *throughput* yang dihasilkan. Terlihat pada grafik yang dihasilkan, pada saat tidak terjadi kemacetan, maka nilai *throughput* yang dihasilkan akan mengalami kenaikan secara linier seiring dengan meingkatnya beban trafik yang dikirimkan. Selanjutnya, saat kemacetan mulai terjadi (tingkat kemacetan sedang), nilai *throughput* yang dihasilkan tidak menunjukkan kenaikan secara signifikan seperti yang terjadi pada saat jaringan normal, melainkan mulai mengalami perlambatan kecepatan. Sebaliknya, saat kemacetan sudah berada pada puncaknya, nilai *throughput* akan mengalami penurunan secara cepat bahkan sampai mendekati nilai 0, artinya tidak ada trafik yang dilewatkan.



Gambar 1-10. Performa Jaringan Akibat Terjadinya Kemacetan (*Delay*)
Sumber: (Thomas, 2003)

Dampak kedua yang ditimbulkan karena kemacetan jaringan adalah *delay*¸yaitu waktu tunda pengiriman data. Terlihat pada Gambar 2-11, bahwa semakin besar beban yang

dikirimkan, maka nilai *delay* akan semakin meningkat apabila terjadi kemacetan. Dampaknya adalah data akan semakin lama ditransmisikan ke node tujuan.



Gambar 1-11. Performa Jaringan Akibat Terjadinya Kemacetan (*Packet*)
Sumber: (Thomas, 2003)

Dengan menurunnya nilai *throughput* dan meningkatnya *delay* dalam proses pengiriman data, maka hal tersebut secara signifikan akan mengakibatkan banyak data yang tidak terkirimkan secara utuh. Jumlah paket yang diterima tidak sama dengan jumlah paket yang dikirim. Terlihat pada Gambar 2-12, saat terjadi kemacetan dalam jaringan, maka grafik jumlah paket yang diterima akan menurun secara drastis.

1.7.2 Pengendalian Kemacetan Protokol TCP

Beragam penelitian telah dilakukan untuk menciptakan algoritme yang bertujuan untuk mengendalikan kemacetan dalam jaringan. *Network Layer* merupakan salah satu layer dalam 7 OSI Layer yang memiliki kemampuan untuk melakukan manajemen kemacetan jaringan hingga batasan tertentu. Layer ini akan bekerja sama dengan *Transport Layer* yang bertanggungjawab terhadap proses pendistribusian data dalam sebuah koneksi. Saat ini, yang menjadi prioritas utama dalam pengendalian kemacetan jaringan adalah bagaimana cara untuk mencegah sebelum kemacetan yang sesungguhnya terjadi. Salah satu hal yang dapat dilakukan adalah dengan cara menurunkan kecepatan data, dan hal ini telah dikerjakan dengan baik oleh protokol *Transmission Control Protokol* (TCP). Tujuan utama dari metode ini adalah untuk menghentikan sementara paket baru yang akan masuk, sampai paket sebelumnya telah berhasil

dikirimkan. TCP mencoba mencapai tujuan tersebut dengan cara melakukan manipulasi ukuran *congestion window* secara dinamis (Roy, 2006).

Pada kondisi nyata, TCP akan meningkatkan beban trafik secara simultan sebagai salah satu cara untuk dapat mengetahui titik di mana terjadi kemacetan. Namun hal ini menjadi suatu kerugian karena TCP akan menghabiskan kapasitas *bandwidth* yang tersedia dalam jaringan (Peterson and Davie, 2012)

Beberapa jenis algoritme yang telah dikembangkan dengan menggunakan protokol TCP ditujukan untuk membantu TCP dalam mengatasi kemacetan. Namun keseluruhan algoritme yang dikembangkan tersebut memiliki asumsi kemacetan yang sama, yaitu terjadinya *timeout* yang disebabkan oleh *packet loss* (Roy, 2006). Kerugian yang ditimbulkan akibat mekanisme tersebut adalah adanya pengiriman ulang data yang mengalami *packet loss*, sehingga jaringan akan mengalami *stuck* sampai TCP dapat mendeteksi lokasi kemacetan dan mengirimkan kembali data tersebut. Kondisi ini tentunya akan semakin menyebabkan adanya penumpukan data pada beberapa jalur.

1.7.3 Pencegahan Kemacetan Jaringan pada Protokol TCP

Sebagai upaya untuk mencegah kemacetan jaringan, protokol TCP secara khusus membedakan mekanisme ini menjadi 2 pendekatan, yaitu *End-to-end approach* dan *Network-assisted approach.*. Pada *End-to-end approach,* layer jaringan tidak menyediakan bantuan eksplisit secara langsung kepada layer transportasi. Kehadiran kemacetan dalam jaringan akan dideteksi oleh *end system* dengan berdasarkan kepada pengamatan tingkah laku jaringan, sebagai contoh adanya *packet loss* atau *delay*. Sedangkan pada *Network-assisted approach,* komponen layer jaringan (dalam hal ini *router*) menyediakan *feedback* secara eksplisit kepada *router* pengirim mengenai kondisi kemacetan yang sedang terjadi dalam jaringan. *Feedback* ini dilakukan dengan dua cara sederhana, secara langsung (*direct feedback*) maupun secara tidak langsung (Kurose and Ross, 2013).

Additive Increase Multiplicative Decrease (AIMD), *TCP Slow-Start*, dan *DECBit* merupakan beberapa algoritme yang paling banyak diimplementasikan pada protokol TCP.

AIMD merupakan algoritme yang dikembangkan pertama kali dengan mengadopsi konsep pengendalian kemacetan jaringan yang dilakukan oleh TCP. Setiap *TCP Source* akan mendefinisikan kapasitas yang tersedia dalam jaringan untuk menentukan ukuran paket data yang dapat ditransmisikan secara aman pada jalur tersebut, dan variabel yang digunakan untuk menjalankan mekanisme ini adalah *CongestionWindow (cwnd)*. Nilai variabel *cwnd* akan diatur berdasarkan level kemacetan yang terjadi. Ukuran *cwnd* akan diturunkan pada saat level kemacetan naik, demikian pula sebaliknya ukuran *cwnd* akan dinaikkan pada saat level kemacetan menurun. Mekanisme inilah yang dilakukan secara bersamaan pada algoritme AIMD. Algoritme ini akan menjadi model pendekatan yang baik digunakan ketika *TCP Source* bekerja mendekati batas kapasitas yang tersedia dalam jaringan, namun akan membutuhkan waktu yang cukup lama pada saat akan memulai kembali koneksi di tempat terjadinya kemacetan.TCP kembali menyediakan mekanisme kedua untuk mengatasi kekurangan AIMD, yaitu dengan menggunakan algoritme *Slow Start*. Algoritme ini akan menaikkan nilai *cwnd* dengan cepat secara eksponensial dibandingkan dengan algoritme AIMD (Peterson and Davie, 2012)

Jika AIMD dan *Slow Start* merupakan algoritme yang dikembangkan pada protokol TCP, maka *DECBit* merupakan algoritme yang dapat diimplementasikan pada jaringan berbasis TCP dan IP. Ide utama dari *DECBit* adalah konsep penanganan kemacetan yang terbagi sama rata antara *router* dan *end node*, sedangkan pada AIMD dan *Slow Start* hanya difokuskan pada *TCP Source* saja. Setiap *router* akan melakukan *monitoring* pada beban trafik dan memberikan notifikasi secara eksplisit (berupa nilai *binary*) kepada *end node* apabila terjadi kemacetan, sehingga pihak pengirim data dapat mengurangi *transmission rate* untuk menghindari kemacetan yang lebih parah. Namun sekali lagi kemacetan hanya akan ditanggulangi apabila telah dideteksi terjadi kemacetan, bukan merupakan suatu tindakan pencegahan sebelum kemacetan terjadi (Peterson and Davie, 2012)

Konsep *end-to-end approach* dilakukan antar *end node*, yaitu antara pihak pengirim dan penerima data. Namun pada konsep *network-assisted approach*, mekanisme pencegahan kemacetan akan dilakukan sepenuhnya oleh pihak *router*. *Random Early Detection* (RED) dan *Explicit Congestion Notification* (ECN) merupakan 2 algoritme independen yang seringkali diaplikasikan bersama-sama guna meningkatkan performansi jaringan. RED biasanya

digunakan untuk *gateway/router backbone* dengan tingkat trafik yang sangat tinggi. RED mengendalikan trafik jaringan sehingga terhindar dari kemacetan pada saat trafik tinggi berdasarkan pemantauan perubahan nilai antrian minimum dan maksimum. RED akan menghitung antrian dalam *bytes* pada setiap paket untuk menentukan level kemacetan jalur, dan RED akan menghitung probabilitas *gateway* memberikan tanda pada setiap paket untuk memberitahukan kepada *router sender* bahwa sedang terjadi kemacetan pada jalur tersebut (Floyd and Jacobson, 1993).

Notifikasi kemacetan pada *end-system* secara umum dilakukan setelah *router* membuang paket pada saat *buffer* telah penuh. *Explicit Congestion Notification* (ECN) diajukan sebagai salah satu mekanisme terpisah dari *router* yang digunakan untuk memberikan peringatan secara eksplisit mengenai kemacetan tanpa harus membuang paket. Mekanisme ini akan berjalan lebih baik pada saat diimplementasikan bersama-sama dengan algoritme RED. Namun sebagaimana konsep pada TCP, demikian pula dengan algoritme RED dan ECN yang akan menangani kemacetan jaringan hanya setelah dideteksi adanya paket yang dibuang atau pada saat kemacetan itu sendiri telah terjadi (Khosroshahy, 2009).

1.8 *Quagga*

Quagga yang terintegrasi dengan sebuah sistem/mesin akan berlaku sebagai *dedicated router*. Dengan menggunakan *Quagga*, *host* akan saling berinteraksi dan bertukar informasi *routing* dengan *router* lain melalui protokol *routing*. *Quagga* akan menggunakan informasi ini untuk melakukan *update* tabel *routing* sehingga data dapat ditransmisikan secara tepat dan benar. Konfigurasi juga dapat diubah secara dinamis dan informasi tabel *routing* dapat ditampilkan melalui terminal *Quagga* (Ishiguro, 2011).

Sebagai salah satu perangkat lunak *routing* berbasis *open-source*, *Quagga* menyediakan implementasi OSPF, RIP dan BGP untuk platform Linux. Quagga memiliki *library* pengembangan yang cukup signifikan untuk memfasilitasi implementasi analisis performansi protokol *routing* . Quagga memiliki arsitektur *daemon* (servis layanan) terpusat yang dinamakan *zebra*. Setiap *daemon* dapat dikonfigurasi secara fleksibel melalui jaringan yang disebut CLI (*Command Line*

Interface). Keseluruhan konfigurasi *daemon* yang dilakukan oleh *administrator* dapat dikelola dalam satu lokasi melalui 'vtysh'.

Daemon **ripd** akan menangani protokol RIP, daemon **ospfd** akan mendukung protokol *routing* OSPFv2, sedangkan **bgpd** merupakan daemon untuk protokol *routing* BGP. Konfigurasi *routing* dilakukan pada masing-masing protokol sesuai dengan kebutuhan. Selain itu, *user* dapat menjalankan daemon secara khusus dan mengirimkan *routing report* ke konsol *routing* terpusat, yaitu pada daemon *zebra* (Ishiguro, 2011).

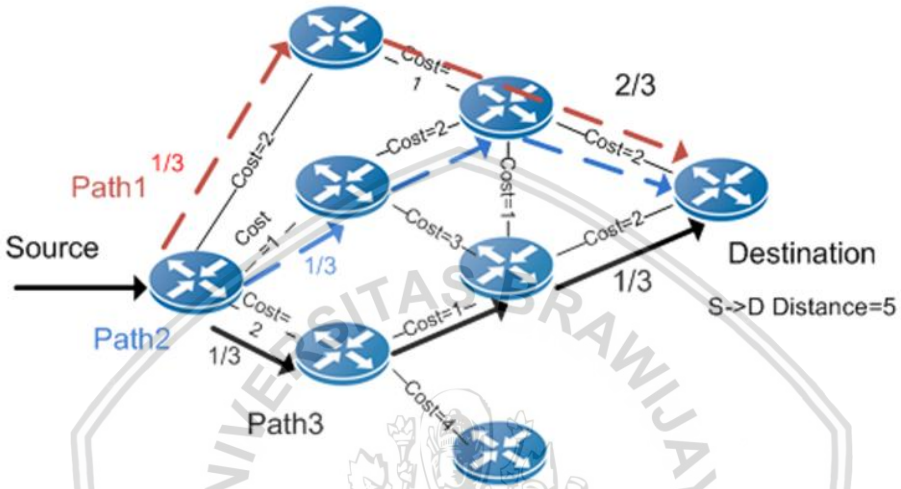
1.9 *Equal Cost Multipath (ECMP)*

Protokol link-state seperti OSPF dan IS-IS merupakan protokol *routing* yang berbasiskan algoritme *Shortest Path First* (SPF) untuk menghitung *single shortest path* (jalur terpendek tunggal) dari *node* sumber ke *node* tujuan. *Equal Cost Multipath* (ECMP) merupakan sebuah teknik yang memungkinkan penggunaan beberapa jalur yang memiliki nilai *cost* yang sama dalam perutean. Fitur ini bukan hanya akan membantu mendistribusikan lalu lintas lebih merata, tetapi juga sebagai metode proteksi atau keamanan dalam jaringan. Dengan menggunakan ECMP, jalur yang memiliki nilai *cost* yang sama akan dipasang ke dalam table *load balancing* di lapisan *forwarding* pada *router*. Pada saat *router* mendeteksi adanya kegagalan *link*, trafik akan didistribusikan secara otomatis kepada jalur kedua dengan tanpa adanya kehilangan trafik yang cukup signifkan.

ECMP tidak memerlukan konfigurasi secara khusus, sebab algoritme SPF akan menghitung secara otomatis jalur-jalur dengan nilai *cost metric* yang sama, untuk kemudian melakukan *advertising* kepada layer *forwarding* pada *router*. Jumlah jalur ECMP bergantung kepada algoritme *load balancing* yang mendukung. Umumnya, dikonfigurasikan antara 1 sampai 8 dan 16 jalur, bergantung kepada nilai maksimum dari jalur yang ada.

Contoh penggunaan *ECMP Load Balancing* terlihat pada Gambar 2-5. Trafik akan tersebar secara merata ke seluruh jalur yang ada. Ketiga jalur ECMp tersebut akan menjadi *back-up* satu sama lain. Jika salah satu jalur terdeteksi gagal, maka trafik akan terbagi ke dua alur yang lain. Hanya ada satu *node* dan satu jalur saja yang akan berbagi lebih dari satu jalur. Dalam scenario

ini, hanya *router* sumber yang perlu mendukung fasilitas ECMP. Sekalipun *router* lain hanya akan mendukung *single path routing*, ECMP masih akan bekerja denganc ara yang sama. Tetapi hanya *router* sumber yang bisa membagi traffic dan menyediakan perlindungan. Untuk bisa mengkombinasikan *single path routing* dan ECMP, sangat penting untuk menggunakan konfigurasi bobot yang tepat untuk membentuk jalur ECMP dengan benar (Lappetel, 2011).



Gambar 1-12. *ECMP Load Balancing*
Sumber: (Lappetel, 2011)

1.10 *Queueing Disciplines*

Ketika *sender* melakukan transmisi data lebih cepat atau lebih besar dibandingkan dengan yang dapat diterima oleh pihak *receiver*, maka paket secara otomatis akan dibuang. Dengan menggunakan algoritme standar TCP, *packet loss* ini dipastikan akan terjadi, sehingga secara bertahap akan menurunkan kinerja jaringan (Bowden, Crawford and Limit, 2006).

Sebagai bagian dari mekanisme alokasi sumber daya, setiap *router* harus menerapkan beberapa mekanisme antrian (*queueing discipline*) yang akan mengatur bagaimana sebuah paket ditahan 28rotoc menunggu untuk ditransmisikan. Berbagai macam disiplin antrian dapat digunakan untuk mengendalikan paket mana yang dapat langsung ditransmisikan (*bandwidth allocation*) dan paket mana yang harus dibuang (*buffer space*). Adanya disiplin antrian ini akan

memberikan dampak terhadap nilai *latency* sebuah paket, dengan menentukan berapa lama paket harus menunggu untuk ditransmisikan.

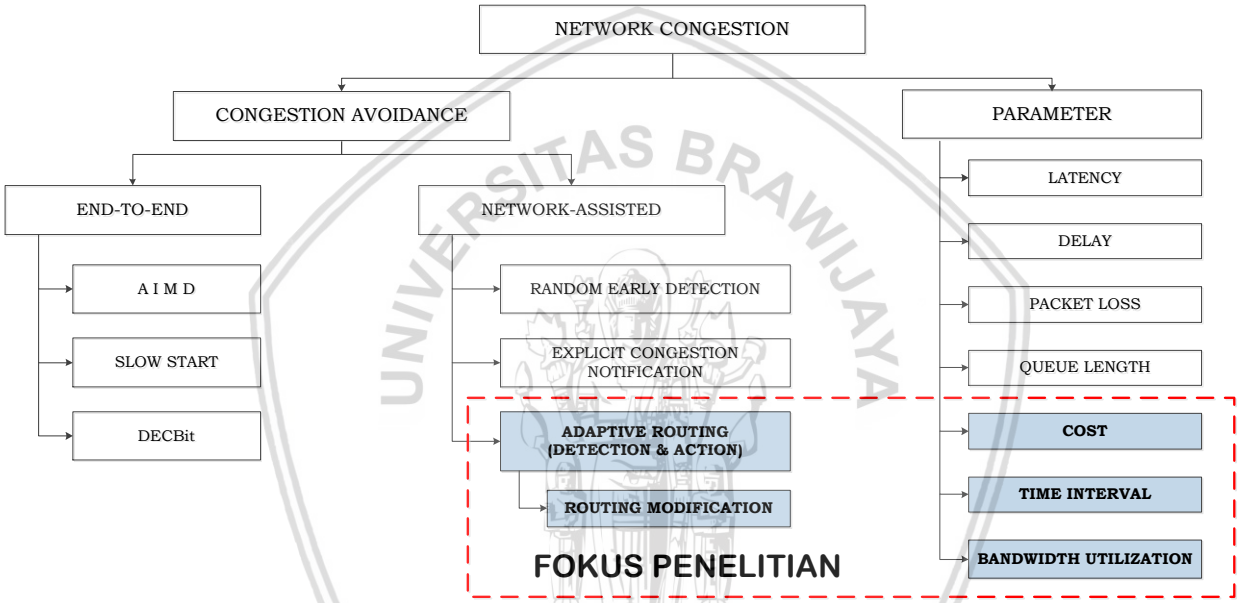
Salah satu mekanisme penggunaan *queueing disciplines* adalah melakukan pembatasan trafik *(traffic shaping*). *Traffic Shaping* merupakan usaha untuk mengendalikan trafik dengan cara memprioritaskan sumber daya jaringan dan menjamin nilai *bandwidth* berdasarkan peraturan kebijakan yang telah ditetapkan. *Traffic Shaping* membantu dalam melakukan kendali terhadap layanan jaringan, melakukan limitasi terhadap *bandwidth*, dan menjamin nilai QoS. Penentuan metode *traffic shaping* yang tepat akan meningkatkan *latency* jaringan, ketersediaan layanan, dan utilisasi *bandwidth* (Media, 2016).

Setiap devais jaringan akan memiliki *root* dimana *queue discipline* (qdisc) akan dikonfigurasikan. *Root* memiliki nilai *fq_codel* sebagai nilai *default*-nya. Terdapat dua jenis disiplin antrian, *classfull* dan *classless*. *Classfull qdisc* mengijinkan adanya pembentukan kelas, yang akan bekerja seperti ranting pada sebuah pohon.Aturan pada dispilin ini digunakan untuk melakukan penyaringan paket yang masuk ke dalam kelas-kelas. Sedangkan, *classless qdisc* tidak mengijinkan adanya penambahan kelas *qdisc* di dalamnya. Antrian yang dilakukan pada jenis disiplin ini adalah dengen memanajemen trafik secara dasar, yaitu melakukan *reordering, slowing* atau *dropping paket* (ArchLinux, 2017).

Classfull qdisc sangat berguna jika trafik yang akan dikendalikan memiliki perbedaan perlakuan. *Classfull qdisc* mengijinkan adanya cabang, yang disebut dengan *classes*. Setiap kelas akan diberi nama dengan menggunakan parameter *classid*. Sedangkan parameter *parent* akan mengindikasikan induk dari kelas tersebut. Salah satu jenis *classfull qdisc* yang banyak digunakan adalah *Hiearchial Token Bucket* (HTB). HTB akan melakukan kendali dari penggunaan *outbond bandwidth* pada sebuah jalur. Beberapa hal yang dapat dilakukan dengan HTB antara lain *link sharing, sharing hierarchy,* dan *priorizing bandwith* (Cohen, 2014).

BAB III
KERANGKA KONSEP PENELITIAN

Dengan mengacu kepada hasil studi empirik ringkasan pustaka dan penelitian terdahulu yang telah dijelaskan sebelumnya, maka pada bab ini akan dipaparkan kerangka konsep penelitian yang dilakukan. Hasil kerangka konsep penelitian diberikan pada Gambar 3.1.



Gambar 1-1 Kerangka Konsep Penelitian

Sebagai upaya untuk mencegah kemacetan jaringan, protokol TCP secara khusus membedakan mekanisme ini menjadi 2 pendekatan, yaitu *End-to-end approach* dan *Network-assisted approach.*. Pada *End-to-end approach,* layer jaringan tidak menyediakan bantuan eksplisit secara langsung kepada layer transportasi. Kehadiran kemacetan dalam jaringan akan dideteksi oleh *end system* dengan berdasarkan kepada pengamatan tingkah laku jaringan, sebagai contoh adanya *packet loss* atau *delay.* Sedangkan pada *Network-assisted approach,* komponen layer jaringan (dalam hal ini *router*) menyediakan *feedback* secara eksplisit kepada *router* pengirim mengenai kondisi kemacetan yang sedang terjadi dalam jaringan. *Feedback* ini dilakukan dengan dua cara sederhana, secara langsung (*direct feedback*) maupun secara tidak langsung (Kurose and Ross, 2013).

Konsep *end-to-end approach* dilakukan antar *end node*, yaitu antara pihak pengirim dan penerima data. Namun pada konsep *network-assisted approach*, mekanisme pencegahan kemacetan akan dilakukan sepenuhnya oleh pihak *router*. *Random Early Detection* (RED) dan *Explicit Congestion Notification* (ECN) merupakan 2 algoritme independen yang seringkali diaplikasikan bersama-sama guna meningkatkan performansi jaringan. RED biasanya digunakan untuk *gateway/router backbone* dengan tingkat trafik yang sangat tinggi. RED mengendalikan trafik jaringan sehingga terhindar dari kemacetan pada saat trafik tinggi berdasarkan pemantauan perubahan nilai antrian minimum dan maksimum. RED akan menghitung antrian dalam *bytes* pada setiap paket untuk menentukan level kemacetan jalur, dan RED akan menghitung probabilitas *gateway* memberikan tanda pada setiap paket untuk memberitahukan kepada *router sender* bahwa sedang terjadi kemacetan pada jalur tersebut (Floyd and Jacobson, 1993).

Notifikasi kemacetan pada *end-system* secara umum dilakukan setelah *router* membuang paket pada saat *buffer* telah penuh. *Explicit Congestion Notification* (ECN) diajukan sebagai salah satu mekanisme terpisah dari *router* yang digunakan untuk memberikan peringatan secara eksplisit mengenai kemacetan tanpa harus membuang paket. Mekanisme ini akan berjalan lebih baik pada saat diimplementasikan bersama-sama dengan algoritme RED. Namun sebagaimana konsep pada TCP, demikian pula dengan algoritme RED dan ECN yang akan menangani kemacetan jaringan hanya setelah dideteksi adanya paket yang dibuang atau pada saat kemacetan itu sendiri telah terjadi (Khosroshahy, 2009).

Alternatif solusi yang dapat diusulkan untuk melengkapi kelemahan mekanisme pencegahan kemacetan jaringan pada protokol TCP dan IP adalah memprediksikan kapan dan di mana kemacetan akan terjadi, sebelum adanya paket yang dibuang atau terjadi *timeout* dan kemudian melakukan pengendalian kemacetan untuk menentukan jalur terbaik pendistribusian data. Penelitian ini akan mengadopsi mekanisme *congestion avoidance* dengan menggunakan pendekatan *network-assisted* karena pengubahan akan dilakukan pada setiap *interface* jaringan (*router*), dengan memperhatikan nilai *cost*, *time interval* serta *bandwidth utilization* sebagai parameter utama bahan pertimbangan modifikasi *routing*.

1.1 Kerangka Pemikiran

Kerangka pemikiran pada penelitian ini dibuat untuk menjelaskan secara garis besar alur logika dari penelitian yang dilakukan. Kerangka ini dibuat berdasarkan pertanyaan penelitian dan merepresentasikan himpunan konsep yang saling berkaitan, yang dibedakan menjadi 2 bagian, yaitu analisis permasalahan yang akan diteliti, dan konsep solusi yang ditawarkan.

1.1.1 Analisis Masalah

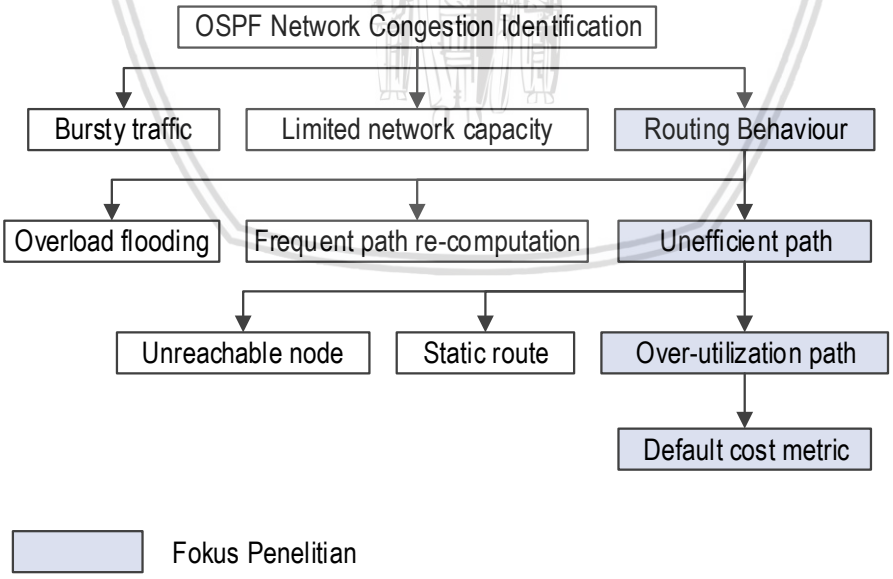
Penelitian ini dilandasi oleh pemikiran bahwa suatu jaringan yang baik adalah jaringan yang dapat menangani proses pendistribusian data dengan efektif dan efisien. Efektif memiliki arti tepat guna, sedangkan efisien berarti penggunaan sumber daya secara minimum namun sanggup memberikan hasil yang optimum. Prosedur pendistribusian data ini menjadi tanggung jawab dari protokol *routing*. Tujuan utama protokol *routing* adalah untuk mendefinisikan bentuk atau topologi jaringan dan menentukan rute terbaik menuju ke sebuah destinasi.

Untuk dapat melakukan analisis permasalahan terhadap fenomena yang terjadi pada sebuah protokol *routing*, maka terlebih dahulu dilakukan identifikasi terhadap tingkah laku node-node dalam jaringan yang mengimplementasikan protokol *routing* di dalamnya, dalam hal ini adalah *router*.

OSPF, merupakan salah satu protokol *routing* dinamis berbasis *link-state* yang melakukan proses *flooding* dengan tujuan untuk menyebarkan informasi *routing* kepada seluruh node (*router*) dalam jaringan. Informasi *routing* yang disebarkan ini termasuk di antaranya adalah informasi mengenai jalur-jalur dari setiap node menuju ke semua node dalam jaringan pada setiap *router*. Penentuan jalur pada protokol *routing* OSPF menggunakan algoritme Dijkstra dilakukan hanya berpedoman pada nilai *cost metric*, yang ditentukan berdasarkan alokasi *bandwidth* yang tersedia pada setiap *interface router*. Untuk menuju ke sebuah prefix dalam jaringan, Dijkstra akan mendefinisikan satu jalur utama (*single-path routing*) yaitu jalur terpendek dengan jumlah nilai *cost* yang terkecil dari semua jalur yang tersedia antara node sumber dan node tujuan.

Berdasarkan *behavior* yang dikerjakan oleh *router* OSPF tersebut, terdapat beberapa permasalahan yang mungkin saja terjadi pada saat-saat tertentu. Adanya perubahan topologi, baik karena adanya pengembangan jaringan ataupun karena adanya node yang mati (*down*) dalam jaringan, dapat menyebabkan *router* OSPF melakukan perhitungan atau komputasi jalur ulang secara berkala. Selain itu, *flooding* akan terus menerus dilakukan sekalipun terdapat beberapa node yang *down*, sehingga dapat terjadi adanya *overload flooding*. Selanjutnya, pada saat jaringan memiliki intensitas trafik yang cenderung tinggi, atau terjadi *bursty traffic* secara tiba-tiba dalam jaringan, jalur utama yang dihasilkan oleh Dijkstra (*single-path routing*) tentunya akan mengalami penumpukan antrian trafik. Keadaan ini akan menyebabkan terjadinya kemacetan dalam jaringan, mengingat terbatasnya kapasitas sumber daya pada sebuah *interface*. Kemacetan pada salah satu *interface* akan berdampak kepada ketidakkonsistenan jalur, sehingga paket data tidak dapat didistribusikan dengan tepat dan cepat.

Analisis permasalahan dalam penelitian ini dirangkum secara singkat dalam bentuk skema seperti yang diberikan pada Gambar 3.2.



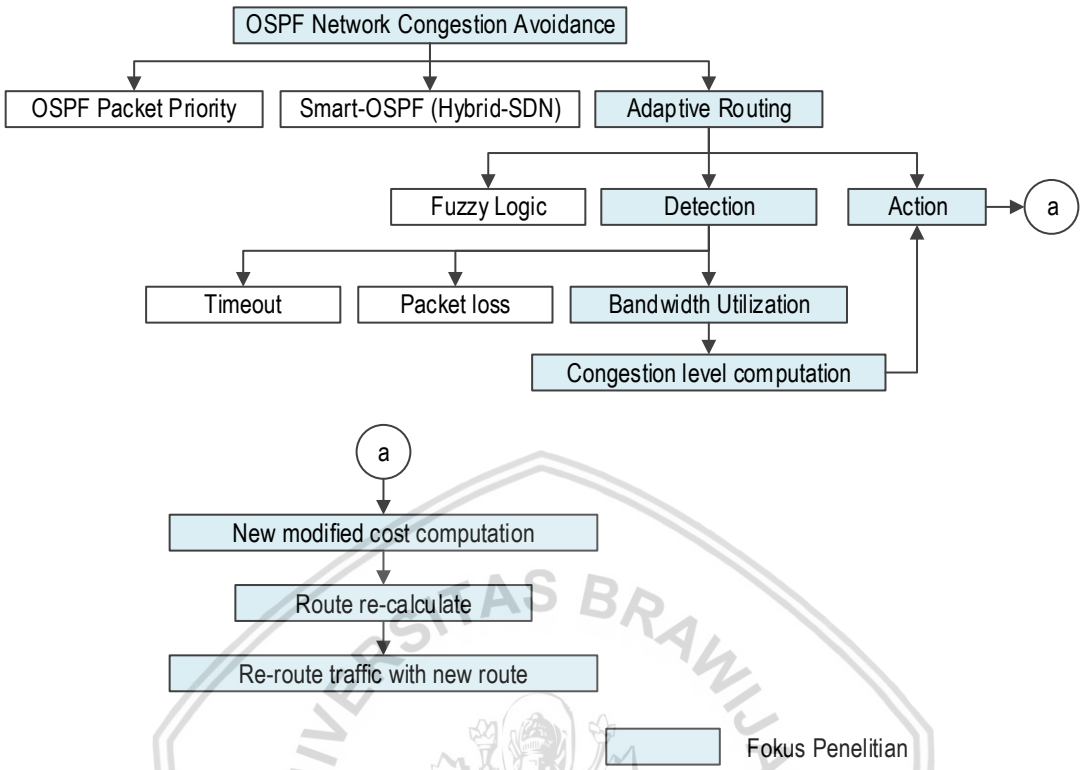
Gambar 1-2 Kerangka Identifikasi Kemacetan Jaringan di OSPF

1.1.2 Konsep Solusi

Fenomena terjadinya kemacetan dalam jaringan menjadi faktor utama untuk melakukan analisis terhadap tingkah laku trafik data dalam jaringan OSPF. Konsep solusi yang dilakukan terkait dengan permasalahan yang diangkat dalam penelitian ini bermula dari pengamatan terhadap jalur atau rute pendistribusian data yang ditentukan oleh Algoritme Djikstra secara *default*, dengan menggunakan akumulasi nilai *cost metric*. Beberapa parameter yang digunakan dalam melakukan pengamatan tingkah laku jaringan OSPF meliputi nilai *cost metric*, informasi rute pendistribusian trafik, waktu pendistribusian data, hingga tingkat penggunaan sumber daya. Keempat parameter ini akan menjadi bahan pertimbangan yang menunjukkan apakah jaringan mengalami kemacetan atau tidak. Jika sebuah *interface* didapati memiliki nilai utilisasi *bandwidth* yang cenderung tinggi, maka dapat ditarik sebuah kesimpulan bahwa jalur tersebut rentan terhadap kemacetan. Hasil analisis inilah yang akan menjadi pertimbangan utama bagi jaringan OSPF dalam melakukan proses *re-routing* dengan tujuan melakukan pengalihan jalur distribusi data ke jalur alternative lain yang tidak mengalami kemacetan. Konsep ini secara ringkas disajikan pada Gambar 3-3 dalam bentuk kerangka konsep solusi penelitian.

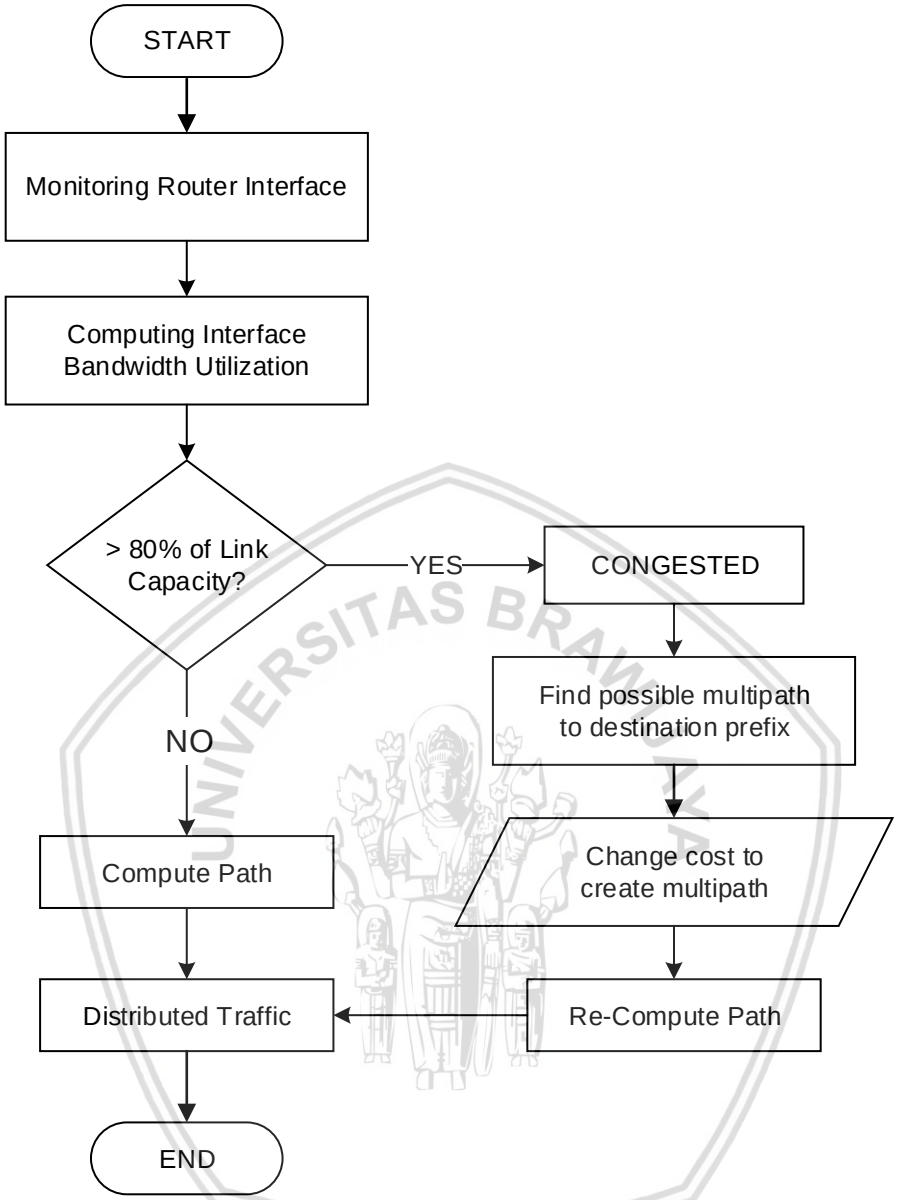
Routing adaptif menjadi salah satu konsep solusi yang dapat dilakukan untuk mencegah terjadinya kemacetan dalam jaringan OSPF. Gagasan utama dari penelitian ini adalah membangun protokol *routing* OSPF yang bersifat adaptif, dimana jaringan secara dinamis dapat melakukan perubahan, baik dalam hal perubahan nilai *cost metric* maupun perubahan jalur, dengan berpedoman kepada tingkat kemacetan yang telah dideteksi. *Routing* adaptif juga mencerminkan bahwa prosedur pengubahan jalur maupun nilai *cost metric* akan didasarkan kepada kondisi terkini jaringan dalam kurun waktu tertentu.

Kemacetan jaringan dideteksi berdasarkan utilisasi *bandwidth* pada setiap *interface router*. Jika jumlah trafik yang harus dilewatkan melalui *outbond interface router* telah mencapai 80% atau bahkan melebihi kapasitas yang dapat ditangani (*inbond traffic* > *outbond capacity*), maka jalur tersebut (jalur utama yang ditentukan oleh algoritme Dijsktra) dinyatakan mengalami kemacetan. Dengan demikian, maka dilakukan pemodifikasi nilai *cost metric* pada *interface* dengan utilisasi *bandwidth* yang lebih rendah untuk membentuk jalur *multipath*. Komputasi jalur ulang dilakukan oleh Dijkstra dan trafik akan kembali dirutekan melalui jalur yang baru.



Gambar 1-3 Kerangka Konsep Solusi Penelitian

Kesimpulan usulan solusi deteksi dan pengendalian kemacetan jaringan pada protocol *routing* OSPF diberikan pada Gambar 3-4 dalam bentuk diagram alur penelitian. Saat trafik akan dilewatkan melalui sebuah node (*router*), setiap *router* akan mengidentifikasi terlebih dahulu, apakah jumlah trafik yang masuk memiliki kapasitas yang lebih besar atau lebih kecil, apakah jalur yang sedang digunakan sedang penuh atau tidak, untuk menganalisis apakah jalur yang akan dilewati trafik tersebut rentan terhadap kemacetan atau tidak. Jika jalur utama yang telah didefinisikan secara *default* terdeteksi macet, maka *router* akan mencari alternative jalur lain yang dapat dilewati untuk menuju ke prefix tujuan. Setelah mendefinisikan jalur alternative tersebut, maka dilakukan perubahan nilai *cost metric*, dengan tujuan melakukan *multipathing*, yaitu pentransmisian trafik melalui beberapa jalur dengan jumlah nilai *cost metric* yang sama. Namun sebaliknya, apabila dideteksi jalur yang dilewatkan sedang tidak penuh, atau jumlah trafik yang akan dilewatkan lebih kecil dibandingkan kapasitas yang dapat ditangani, maka trafik akan tetap ditransmisikan melalui jalur utama, sesuai dengan *default* rute yang telah dihitung oleh Dijkstra.



Gambar 1-4 Diagram Alur Konsep Routing Adaptif pada OSPF

1.2 Hipotesis Penelitian

Berdasarkan kajian empiris atas berbagai pengaruh antar variable serta dukungan teori dan hasil penelitian terdahulu yang telah dipaparkan sebelumnya, maka hipotesis yang akan diuji dalam penelitian ini adalah apabila kalkulasi perhitungan *routing* berdasarkan nilai *cost metric* pada protokol *routing* OSPF dipengaruhi oleh tingkat kepadatan trafik pada sebuah jalur, maka hasil pengalihan jalur yang dilakukan secara dinamis akan membuat performansi protokol *routing* OSPF

menjadi lebih baik, dengan tingkat akurasi ketepatan dan kecepatan penditribusian data di atas 90%, dibandingkan dengan mekanisme OSPF konvensional.

BAB IV
METODE PENELITIAN

Perancangan konsep deteksi dan pengendalian kemacetan jaringan melalui protokol *routing* OSPF dilakukan melalui beberapa tahapan metode penelitian seperti yang disajikan pada Gambar 4.1, dengan berpedoman kepada kerangka pemikiran yang telah dijelaskan pada Bab 3.

1.1 Observasi

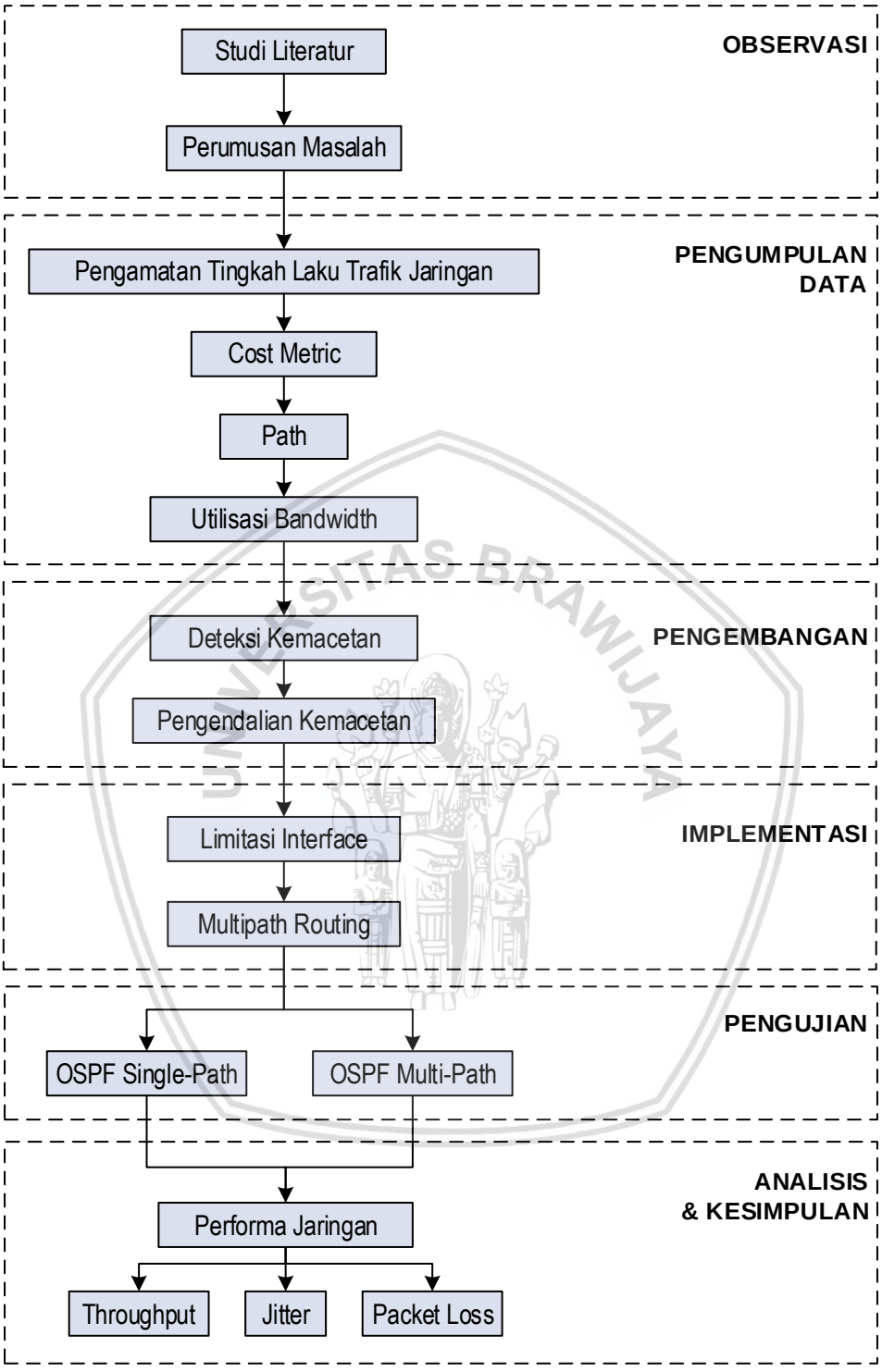
Observasi merupakan tahapan penelitian yang isinya adalah mengkaji permasalahan yang akan diangkat dalam penelitian untuk menentukan hal–hal yang penting sebagai dasar penyelesaian permasalahan melalui analisis kebutuhan, perancangan dan implementasi konsep *routing adaptif* pada protokol OSPF. Hal ini dilakukan dengan terlebih dahulu mempelajari penelitian terkait untuk menggali ide konsep penelitian yang dilakukan.

1.2 Pengumpulan Data

1.2.1 Pengamatan Tingkah Laku Trafik Jaringan

Tingkah laku trafik dalam jaringan menjadi dasar penting untuk dapat melakukan pengkajian terhadap permasalahan yang akan diangkat dalam penelitian. Dengan demikian dapat menentukan hal–hal yang penting sebagai dasar penyelesaian permasalahan melalui analisis kebutuhan, perancangan dan implementasi konsep *routing adaptif* pada protokol OSPF.

Kasus yang sering terjadi berkaitan dengan kemacetan jaringan adalah kemungkinan terjadinya konflik dalam proses *routing*, beberapa trafik dari sumber yang berbeda namun memiliki tujuan yang sama, sehingga menyebabkan *overutilization* pada sebagian *path* pada jaringan. Hal ini juga sering terjadi pada jaringan yang menggunakan protokol *routing* dinamis, salah satunya adalah OSPF. Algoritme Dijkstra pada OSPF secara otomatis melakukan komputasi *shortest-path* berdasarkan nilai *cost* metric. Algoritme ini akan menghasilkan *single-path routing*, yang memungkinkan terjadinya kemacetan pada saat -



Gambar 1-1 Bagan Metode Penelitian

saat tertentu. Hal ini bisa saja diselesaikan dengan cara mengurangi *rate* pada *node* sumber dengan mengaktifkan mekanisme *Congestion Control* (CC) yang sesuai, namun sebagai

dampaknya adalah adanya degradasi atau penurunan *throughtput* dalam jaringan (Fang *et al.*, 2013).

Beberapa hal yang menjadi pokok permasalahan yang melandasi penelitian ini antara lain sebagai berikut.

a. OSPF menggunakan nilai *cost metric* sebagai satu-satunya variabel penentu jalur pendistribusian data
b. Jalur dengan nilai *cost metric* terkecil dianggap sebagai jalur terbaik
c. Paket data dengan tujuan yang sama akan mengalami penumpukan pada salah satu jalur dengan nilai *cost metric* terkecil
d. Penumpukan data pada salah satu jalur menyebabkan kemacetan jaringan

1.2.2 Parameter Penelitian

Parameter penelitian menunjuk kepada beberapa indikator yang akan digunakan untuk melakukan identifikasi permasalahan dan prosedur penyelesaian permasalahan. Terdapat 3 indikator utama yang akan digunakan, yaitu nilai *cost metric*, utilisasi *bandwidth* setiap *interface*, serta pengukuran tingkat kemacetan jaringan. Secara khusus, ketiga indikator ini akan dijelaskan lebih detail pada Tabel 4.1.

Tabel 1-1 Tabel Definisi Operasional dan Pengukuran Variabel

VARIABEL	PERSAMAAN
Bandwidth Utilization Nilai presentase penggunaan sumber daya (Muslim, 2007)	• Perhitungan Utilisasi Bandwidth $Utilisasi(\%) = \frac{Data\ Throughput\ Terukur}{Kapasitas\ Bandwidth\ yang\ Tersedia} x100\%$
	• *Threshold* Batasan nilai kapasitas atau utilisasi *bandwidth* yang digunakan untuk memantau kondisi *interface router* yang berpotensi terjadi kemacetan (Jose, 2016). Nilai *threshold* menjadi pedoman utama untuk penentuan level kemacetan jaringan. Limitasi nilai *threshold* yang diberikan pada setiap *interface router* masing-masing sebesar **80%** dari jumlah kapasitas *bandwidth link* yang tersedia.

VARIABEL	PERSAMAAN
Network Congestion Parameter Definisi tingkat kemacetan jaringan sebagai bahan pertimbangan prosedur modifikasi *routing* dan parameter *input*	• *Low Congestion* Jika nilai utilisasi *bandwidth* < *threshold*, tidak dilakukan perubahan nilai *cost*, data langsung dikirim sesuai dengan jalur yang terdefinisi pada *routing table* • *High Congestion* Jika nilai utilisasi *bandwidth* > *threshold*, dilakukan perubahan nilai *cost* menggunakan perhitungan OSPF MULTI-PATH, data dikirim dengan menggunakan nilai *cost* dan *routing table* yang baru.
Cost Metric Nilai satuan beban yang digunakan oleh *router* untuk menentukan jalur terbaik menuju ke sebuah destinasi jaringan (Thomas, 2003)	• Perhitungan OSPF Konvensional $Cost = \frac{Reference\_Bandwidth}{Interface\_Bandwidth}$ **Reference_Bandwidth* = 10^8 in bps
	• Perhitungan OSPF *Multipath* Membuat jumlah nilai *cost* pada jalur utama **sama dengan** jumlah nilai *cost* pada jalur alternatif (skema *symmetric cost ECMP*)

1.3 Pengembangan

Perancangan sistem dibuat berdasarkan kesimpulan dari pengamatan terhadap tingkah laku trafik dalam jaringan, yang diperkuat dengan adanya studi literatur terhadap referensi yang berkaitan dengan topik, selanjutnya penelitian dikembangkan dengan mengasumsikan kemungkinan solusi yang dapat dilakukan. Beberapa kesimpulan peluang penyelesaian masalah yang dapat dilakukan antara lain sebagai berikut.

a. Kepadatan jalur mengindikasikan tingkat kemacetan jaringan.
b. Tingkat kemacetan jaringan didasarkan pada nilai utilisasi *bandwidth* pada setiap jalur.
c. Pengubahan nilai *cost metric* dilakukan secara dinamis berdasarkan tingkat kemacetan jaringan
d. Jalur terbaik merupakan jalur dengan tingkat kemacetan rendah namun memiliki waktu pendistribusian data tercepat

Salah satu upaya untuk menjawab peluang solusi tersebut di atas adalah dengan memanfaatkan kemampuan *Multi Path* (MP) *routing* untuk mengalihkan beberapa arus trafik

dalam jaringan ke jalur lain, dengan demikian menjamin *loss-free transmission* dan nilai *throughput* yang tinggi (Fang *et al.*, 2013). *Multipath routing* sangatlah penting, tidak hanya mempengaruhi *throughput*, tetapi juga dalam hal kehandalan dan keamanan. Dalam *multipath routing,* peningkatan performa dicapai dengan cara memanfaatkan beberapa jalur (Devetak and Kapoor, 2007). Selanjutnya mengintegrasikannya dengan mekanisme CC, untuk melakukan deteksi kemacetan dengan mengindikasikannya berdasarkan utilisasi *link* yang menghubungkan antar *intermediary nodes* dalam jaringan. Konsep inilah yang akhirnya dirumuskan dalam jenis pendekatan penyelesaian masalah menggunakan konsep *Routing* adaptif OSPF.

Konsep Routing Adaptif OSPF

Pertama-tama, setiap *router* akan melakukan pendefinisian rute utama melalui komputasi algoritme Dijkstra. Sebagai inisialisasi, OSPF akan melakukan *broadcast Link-State Advertisements* (LSA) untuk menunjukkan topologi dan menghitung rute yang tersedia bagi setiap *router*. Hasil komputasi ini akan ditampilkan di dalam table routing.

Pada protokol routing yang berbasis *link*-state, setiap jalur akan dihitung oleh Algoritme Dijkstra untuk menentukan satu jalur terbaik saja dari *node* sumber ke *node* tujuan. Setiap *link* akan memiliki bobot yang tetap, dan trafik akan dibawa melalui jalur terbaik, dalam hal ini adalah jalur terpendek. Jalur ini biasanya dianggap memiliki *hop* terkecil dan selanjutnya meminimalisasi total pemakaian *bandwidth* pada *link* (Susitaival and Aalto, 2004).

Komputasi Algoritme Dijkstra hanya akan menentukan *single-path* routing saja. Jika jalur utama ini mengalami kemacetan, maka data yang sudah ada ataupun yang akan melalui jalur tersebut tidak akan sampai di *node* tujuan dengan utuh, bisa saja terjadi waktu tunda atau paket dibuang begitu saja oleh *router*.

Konsep adaptif routing pada protokol OSPF memiliki tujuan untuk meminimalisasi *delay*, mencegah adanya kehilangan data (*packet* loss), dan memaksimalkan nilai *throughput* selama proses transmisi data berlangsung. Dua mekanisme utama yang digunakan meliputi sistem deteksi kemacetan dan sistem pengendalian kemacetan jaringan.

1.3.1 Deteksi Kemacetan Jaringan

Setiap *interface* pada *router* akan dipantau dalam kurun waktu tertentu (*time interval*), untuk mendapatkan hasil penggunaan sumber daya, atau yang disebut sebagai utilisasi *bandwidth*. Hasil observasi terhadap tingkah laku trafik dalam jaringan digunakan untuk menarik kesimpulan dalam menentukan nilai batasan utilisasi *bandwidth*, yang disebut sebagai nilai *threshold.* Nilai ini akan menjadi pedoman, apakah jalur ini rentan terhadap kemacetan, ataukah tidak.

Pendefinisian *threshold* dilakukan dengan cara memberikan limitasi *bandwidth* yang diperbolehkan dalam sebuah jalur. Limitasi diberikan pada setiap *interface router* yang terhubung langsung dengan *router* lain, sedangkan antar *router* dan *client* tidak dilakukan limitasi. Berdasarkan data pada Tabel 4-1, maka nilai *threshold* yang ditetapkan untuk masing-masing *interface* adalah sebesar 80% dari total kapasitas yang tersedia. Pada saat jumlah paket data yang akan dikirimkan menuju *router* lain mencapai nilai *threshold* tersebut, maka *router* akan mendeteksinya sebagai suatu perintah untuk memindahkan jalur pengiriman data.

1.3.2 Pengendalian Kemacetan Jaringan

Utilisasi *bandwidth* yang buruk cenderung akan menyebabkan kemacetan di inti jaringan (*core network),* dalam hal ini *router* yang menerima maupun mengirim trafik gabungan dari semua *node* pada jaringan (Fang *et al.*, 2013). Sehingga perlu adanya suatu mekanisme yang memanfaatkan semua sumber daya *bandwidth* yang tersedia, tanpa harus tersalurkan pada satu jalur saja.

Routing adaptif pada OSPF akan dilakukan dengan cara mencari dan mendefinisikan jalur lain yang memungkinkan data dapat sampai di *node* tujuan. Hal ini dicapai dengan menggunakan teknik *multipath routing*, yaitu pendefinisian beberapa jalur ke *node* tujuan. Pada saat *router* mendeteksi bahwa kapasitas *bandwidth* telah mencapai limitasi (*threshold*), maka *router* akan mengalihkan jalur pengiriman data berdasarkan *list* jalur yang telah didefinisikan sebelumnya dengan teknik *multipath routing*.

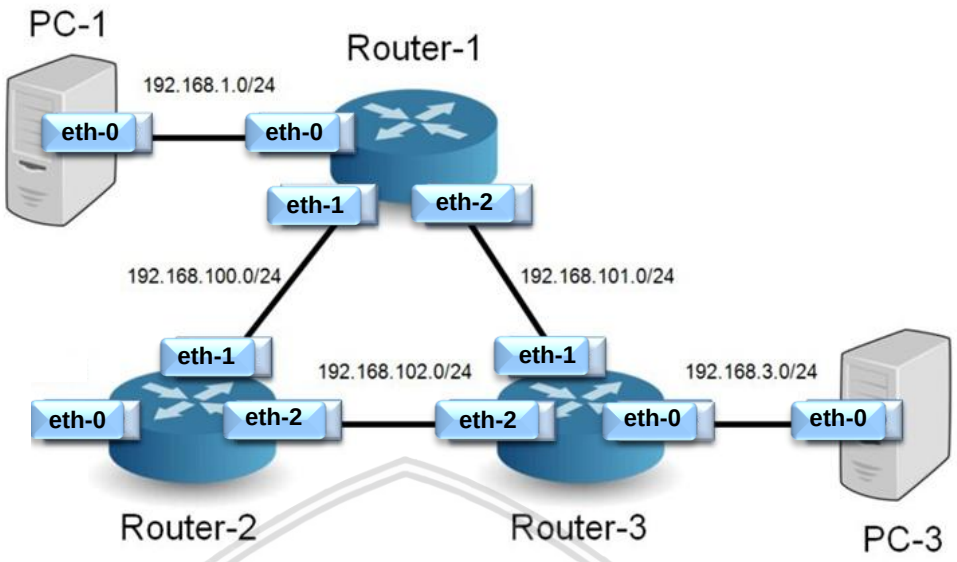
Equal Cost Multi Path (ECMP) merupakan salah satu teknik *multipath routing* yang memungkinkan penggunaan beberapa jalur dengan nilai *cost* metric yang sama dari *node* sumber ke *node* tujuan. Di setiap node, dalam hal ini adalah *interface router*, trafik yang masuk dengan destinasi yang sama akan dikumpulkan untuk kemudian dibagi ke dalam beberapa jalur yang telah didefinisikan oleh ECMP.

1.4 Implementasi Sistem

Sistem adaptif routing pada OSPF dibangun dengan menggunakan Quagga, berbasis *virtual environment*. Quagga merupakan perangkat lunak *routing* berbasis *open source* yang dapat digunakan untuk mengubah Linux menjadi sebuah *router* lengkap yang mendukung protokol *routing* utama, seperti RIP, OSPF, BGP, dan IS-IS. Quagga juga memiliki dukungan secara penuh untuk pengalamatan IPv4 dan IPv6.

Di dalam Quagga, terdapat beberapa *daemon* yang bekerja bersama-sama. *Daemon* yang digunakan untuk membangun sistem ini meliputi **zebra** dan **ospfd.** Zebra merupakan *daemon* utama, yang bertanggungjawab sebagai antarmuka kernel, sedangkan ospfd merupakan *daemon* yang mengijinkan konfigurasi protokol *routing* berbasis OSPF.

Sistem dibangun berdasarkan topologi yang digambarkan pada Gambar 4-3. Terdapat 6 buah mesin *virtual*, yang terdiri dari 3 *router* yang menggunakan Linux CentOS 7 (kernel 3.10), dan 3 buah PC yang menggunakan Linux Ubuntu 16.04 LTS (kernel 4.4). Setiap *virtual machine* dikonfigurasikan untuk saling terhubung satu sama lain. Untuk setiap *router*, memiliki 4 buah adapter jaringan, dengan rincian 2 buah adapter untuk terhubung dengan *router* tetangga, 1 buah adapter untuk LAN Network (PC Client), dan 1 buah adapter sebagai *Host-Only Adapter* untuk melakukan *remote* dari *guest-machine* melalui MobaXterm. Sedangkan untuk *virtual machine* PC, hanya terdapat 2 adapter saja, yaitu 1 buah adapter untuk terhubung ke *router*, dan adapter lainnya untuk terhubung dengan *guest-machine* Windows.



Gambar 1-2. Topologi Pengujian Sistem

1.4.1 Konfigurasi Router

Quagga hanya diinstal dan dikonfigurasikan pada *router* saja, sedangkan pada PC-Client, hanya dikonfigurasikan sistem operasi Ubuntu secara umum. Konfigurasi Quagga pada *router* dilakukan melalui terminal "vtysh" atau denganc ara membuat file konfigurasi daemon *quagga*. Untuk melakukan instalasi Quagga beserta *package*-nya pada Linux CentOS 7 menggunakan perintah berikut ini.

```
#yum install quagga quagga-doc
```

Untuk melakukan konfigurasi Quagga pada Linux CentOS 7, maka fitur SELinux harus dinonaktifkan terlebih dahulu. Hal ini penting untuk dilakukan agar *daemon* pada Quagga dapat menyimpan dan menjalankan hasil konfigurasi yang dikerjakan.

```
#setsebool -P zebra_write_config 1
#systemctl stop firewalld
#systemctl disable firewalld
#vi /etc/selinux/config → SELINUX=disabled
```

Secara *default*, setiap kernel Linux akan menonaktifkan fitur *IP Forwarding*. Fitur ini harus diaktifkan jika kita membangun sebuah *router* atau *gateway* Linux. Untuk mengaktifkan fitur *ip forwarding* secara permanen dapat dilakukan melalui file **/etc/sysctl.conf**.

```
#echo "net.ipv4.ip_forward = 1" >> /etc/sysctl.conf
#sysctl -p /etc/sysctl.conf
```

Setelah Quagga terinstall, maka selanjutnya adalah mengaktifkan *daemon* yang akan digunakan. Sebagaimana telah disebutkan sebelumnya, bahwa *daemon* yang diaktifkan dalam sistem ini adalah **zebra** dan **ospfd.** Konfigurasi *daemon* Quagga dilakukan dengan melakukan pengeditan file **/etc/quagga/daemons**. Pengaktifan **zebra** dan **ospfd** dilakukan dengan mengubah status *daemon* menjadi **yes.**

1. *Konfigurasi Daemon*

Sebelum melakukan konfigurasi *daemon*¸terlebih dahulu dibuat file konfigurasinya dengan cara mengkopi file konfigurasi *default* yang telah disediakan oleh Quagga.

```
#cp/usr/share/doc/quagga-XXXXX/zebra.conf.sample  \
 /etc/quagga/zebra.conf
#service zebra start
#chkconfig zebra on
```

Setelah mengaktifkan zebra, maka selanjutnya adalah menyimpan semua file konfigurasi zebra ke dalam log file Quagga

```
#vtysh
router-1#log file /var/log/quagga/quagga.log
```

Setelah daemon zebra telah diaktifkan dan dijalankan, maka selanjutnya adalah melakukan konfigurasi daemon ospfd. Sebelum melakukan konfigurasi *daemon*¸terlebih dahulu dibuat file konfigurasinya dengan cara mengkopi file konfigurasi *default* yang telah disediakan oleh Quagga.

```
#cp/usr/share/doc/quagga-XXXXX/ospfd.conf.sample  \
```

```
 /etc/quagga/ospfd.conf
#service ospfd start
#chkconfig ospfd on
```

2. *Konfigurasi Interface*

Konfigurasi setiap *interface*, baik pada *router* maupun pada PC-Client dilakukan melalui file konfigurasi **/etc/sysconfig/network-scripts/ifcfg-ethX**. Hal ini dilakukan agar pada saat node tersebut mengalami *reboot*, maka konfigurasi IP yang dilakukan tidak hilang. Adapter jaringan yang terhubung dengan *guest-machine* Windows dikonfigurasikan secara DHCP, sedangkan adapter jaringan yang terhubung dengan *virtual-machine* lain dikonfigurasikan secara statis. Tabel 4-2 menunjukkan detail alamat IP yang dikonfigurasikan pada masing-masing *interface router* beserta deskripsinya.

Tabel 1-2. Konfigurasi Alamat IPv4 Router

Node	Interface	IP Address	Keterangan
Router-1	eth-0	192.168.1.1	to-LAN
	eth-1	192.168.100.1	to-R2
	eth-2	192.168.101.1	to-R3
Router-2	eth-1	192.168.100.2	to-R1
	eth-2	192.168.102.2	to-R3
Router-3	eth-0	192.168.3.1	to-LAN
	eth-1	192168.101.3	to-R1
	eth-2	192.168.102.3	to-R2

3. *Konfigurasi OSPF*

Konfigurasi *routing* OSPF dilakukan melalui terminal ‘vtysh’. Setiap *router* yang dikonfigurasikan menggunakan OSPF akan mendefinisikan jaringan mana saja yang terhubung langsung dengannya. Dalam sistem yang dibangun ini, area OSPF yang digunakan adalah area *backbone* atau area 0. Berikut ini merupakan salah satu konfigurasi OSPF yang dilakukan pada Router-1.

```
#vtysh
router-1#configure terminal
router-1(config)#router ospf
```

```
router-1(config-router)#network 192.168.1.0/24 area 0
router-1(config-router)#network 192.168.100.0/24 area 0
router-1(config-router)#network 192.168.101.0/24 area 0
router-1(config-router)#end
router-1#write
router-1#exit
```

Perintah **write** digunakan untuk menyimpan hasil konfigurasi *routing* OSPF pada file konfigurasi **/etc/quagga/ospfd.conf.**

1.4.2 Konfigurasi Client

Konfigurasi yang dilakukan pada *client* adalah konfigurasi alamat IP dan alamat *router* yang terhubung dengannya dan berperan sebagai *gateway*. Seperti halnya melakukan konfigurasi *nterface* pada *router*, konfigurasi alamat IP dan *gateway* pada *client* dilakukan pada file konfigurasi **/etc/sysconfig/network-scripts/ifcfg-ethX.** Dengan konfigurasi *bootproto=static* pada eth-0, dan konfigurasi *bootproto=dhcp* pada eth-1. Tabel 4-3 menunjukkan alamat IP yang dikonfigurasikan pada masing-masing PC.

Tabel 1-3. Konfigurasi Alamat IPv4 Router

Node	Interface	IP Address	Keterangan
PC-1	eth-0	192.168.1.10	to-R1
	eth-3	DHCP	Host-Only
PC-3	eth-0	192.168.3.30	to-R3
	eth-3	DHCP	Host-Only

1.4.3 Verifikasi Sistem

Verifikasi sistem dilakukan untuk menguji keberhasilan konfigurasi sistem yang telah dikerjakan. Verifikasi yang dilakukan meliputi beberapa poin, antara lain melakukan tes dengan menggunakan *ping command*, pengecekan table *routing*, serta verifikasi jalur OSPF dan OSPF *neighbors*.

4. ***Uji konektivitas node***

Ping merupakan salah satu *tool* jaringan yang digunakan untuk mengecek konektivitas jaringan antara dua *host*. *Ping* menggunakan protokol ICMP yang dibuat untuk mengecek konektivitas IP dan mendapatkan informasi mengenai mesin lainnya dalam sebuah jaringan IP. Gambar 4-4 menunjukkan hasil uji konektivitas antara PC-1 dan PC-3 dengan menggunakan perintah *ping*.

```
fransiska@host-1:~$ sudo su -
root@host-1:~# ping 192.168.3.30
PING 192.168.3.30 (192.168.3.30) 56(84) bytes of data.
64 bytes from 192.168.3.30: icmp_seq=1 ttl=61 time=3.25 ms
64 bytes from 192.168.3.30: icmp_seq=2 ttl=61 time=2.26 ms
64 bytes from 192.168.3.30: icmp_seq=3 ttl=61 time=1.92 ms
64 bytes from 192.168.3.30: icmp_seq=4 ttl=61 time=5.06 ms
64 bytes from 192.168.3.30: icmp_seq=5 ttl=61 time=2.03 ms
64 bytes from 192.168.3.30: icmp_seq=6 ttl=61 time=1.92 ms

--- 192.168.3.30 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5010ms
rtt min/avg/max/mdev = 1.920/2.744/5.063/1.135 ms
root@host-1:~#
```

Gambar 1-3. Verifikasi *ping* dari PC-1 ke PC-3

Hasil perintah *ping* pada Gambar 4-5 menunjukkan bahwa PC-1 dan PC-3 terkoneksi dengan baik. Ditunjukkan bahwa terdapat 4 paket ICMP yang dikirimkan dan 4 paket yang berhasil diterima, dan tidak ada *packet loss*, dengan rata-rata nilai *latency* yang dihasilkan adalah sebesar 2.794ms.

5. ***Verifikasi table routing***

Dalam jaringan computer, table *routing* merupakan data berbentuk tabel yang tersimpan di dalam *router*, yang berisi daftar rute ke beberapa destinasi jaringan tertentu. Tabel *routing* yang ada di dalam OSPF berisi juga jarak, atau nilai *cost* untuk menuju ke prefix tujuan. Tabel *routing* ini juga berisi informasi mengenai topologi secara keseluruhan yang ada dalam jaringan tersebut. Verifikasi table *routing* dilakukan untuk melakukan pengecekan apakah konfigurasi yang dilakukan sudah benar, sehingga menghasilkan rute atau jalur yang tepat. Pengecekan table *routing* dapat dilakukan melalui kernel Linux dengan perintah **ip route** (lihat Gambar 4-5)**,** atau melalui Quagga dengan mengetikkan perintah **show ip route**. Kedua perintah tersebut akan menghasilkan isi table routing yang sama, hanya perbedaannya pada Quagga akan terdapat kode **O** yang menandakan bahwa *router*

tersebut dikonfigurasikan menggunakan protokol *routing* OSPF, seperti yang terlihat pada Gambar 4-6.

```
[fransiska@router-1 ~]$ sudo su -
Last login: Wed Jan 10 09:28:41 WIB 2018 on pts/0
[root@router-1 ~]# ip ro
169.254.0.0/16 dev eth0 scope link metric 1002
169.254.0.0/16 dev eth1 scope link metric 1003
169.254.0.0/16 dev eth2 scope link metric 1004
169.254.0.0/16 dev eth3 scope link metric 1005
192.168.1.0/24 dev eth0 proto kernel scope link src 192.168.1.1
192.168.2.0/24 via 192.168.100.2 dev eth1 proto zebra metric 15
192.168.3.0/24 proto zebra metric 20
        nexthop via 192.168.101.3 dev eth2 weight 1
        nexthop via 192.168.100.2 dev eth1 weight 1
192.168.56.0/24 dev eth3 proto kernel scope link src 192.168.56.110
192.168.100.0/24 dev eth1 proto kernel scope link src 192.168.100.1
192.168.101.0/24 dev eth2 proto kernel scope link src 192.168.101.1
192.168.102.0/24 via 192.168.100.2 dev eth1 proto zebra metric 10
[root@router-1 ~]#
```

Gambar 1-4. Tabel Routing Router-1 (kernel Linux)

```
router-1# sh ip ro
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, A - Babel,
       > - selected route, * - FIB route

C>* 127.0.0.0/8 is directly connected, lo
K>* 169.254.0.0/16 is directly connected, eth3
O   192.168.1.0/24 [110/10] is directly connected, eth0, 00:27:21
C>* 192.168.1.0/24 is directly connected, eth0
O>* 192.168.2.0/24 [110/15] via 192.168.100.2, eth1, 00:25:36
O>* 192.168.3.0/24 [110/20] via 192.168.101.3, eth2, 00:24:39
  *                         via 192.168.100.2, eth1, 00:24:39
C>* 192.168.56.0/24 is directly connected, eth3
O   192.168.100.0/24 [110/5] is directly connected, eth1, 00:27:21
C>* 192.168.100.0/24 is directly connected, eth1
O   192.168.101.0/24 [110/10] is directly connected, eth2, 00:27:21
C>* 192.168.101.0/24 is directly connected, eth2
O>* 192.168.102.0/24 [110/10] via 192.168.100.2, eth1, 00:25:36
router-1#
```

Gambar 1-5. Tabel Routing Router-1 (Quagga)

6. *Verifikasi OSPF neighbor and route*

Setiap r*outer* pada OSPF akan membangun hubungan dengan *router* tetangganya untuk melakukan pertukaran informasi *routing*. OSPF *neighbor* akan ditemukan secara dinamis dengan cara mengirimkan paket *hello.* Jika status yang ditampilkan pada kolom *State* adalah FULL, menandakan bahwa masing-masing *router* telah melakukan sinkronisasi database dan siap melakukan pengiriman data. Verifikasi terhadap OSPF *neighbor* dilakukan melalui terminal 'vtysh' dengan mengetikkan perintah **show ip ospf neighbor**. Beberapa informasi akan ditampilkan, termasuk di dalamnya adalah informasi *interface* yang menghubungkan dengan *router* lain. Selanjutnya untuk mengetahui rute yang

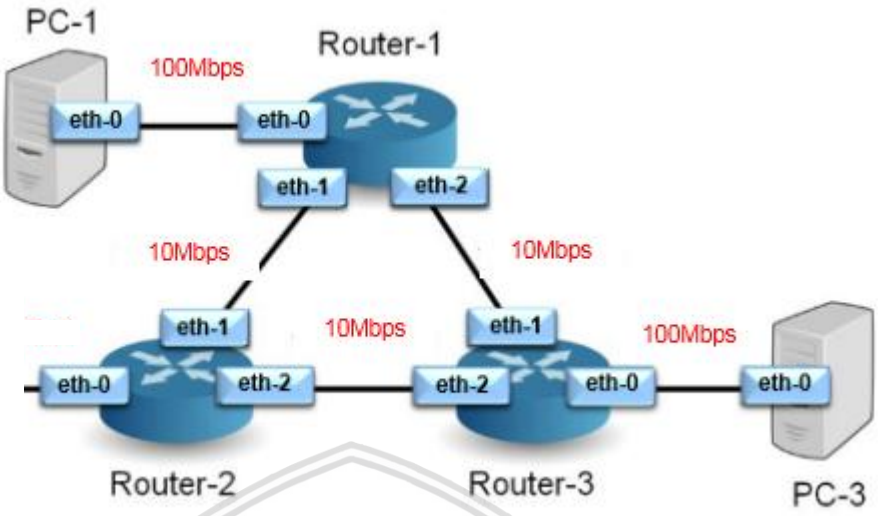
dibangun oleh OSPF, dapat dilakukan dengan cara mengetikkan perintah **show ip ospf route**. Di sana akan ditampilkan table *routing* yang telah dibangun setelah proses *adjacency* berhasil terbentuk. Gambar 4-7 menunjukkan tampilan OSPF *Neighbor* dan rute pada Router-1.

```
router-1# sh ip ospf  neighbor

    Neighbor ID Pri State           Dead Time Address         Interface            RXmtL RqstL DBsmL
192.168.2.1       1 Full/Backup       39.770s 192.168.100.2   eth1:192.168.100.1       0     0     0
192.168.3.1       1 Full/Backup       35.326s 192.168.101.3   eth2:192.168.101.1       0     0     0
router-1# sh ip ospf  route
============ OSPF network routing table ============
N    192.168.1.0/24        [10] area: 0.0.0.0
                           directly attached to eth0
N    192.168.2.0/24        [15] area: 0.0.0.0
                           via 192.168.100.2, eth1
N    192.168.3.0/24        [20] area: 0.0.0.0
                           via 192.168.101.3, eth2
                           via 192.168.100.2, eth1
N    192.168.100.0/24      [5] area: 0.0.0.0
                           directly attached to eth1
N    192.168.101.0/24      [10] area: 0.0.0.0
                           directly attached to eth2
N    192.168.102.0/24      [10] area: 0.0.0.0
                           via 192.168.100.2, eth1

============ OSPF router routing table =============

============ OSPF external routing table ===========
```

Gambar 1-6. Verifikasi OSPF Neighbor & Routes

1.4.4 Limitasi *Interface*

Untuk memvisualisasikan terjadinya kemacetan pada jaringan, maka limitasi *interface router* dilakukan dengan menggunakan *tool tc* dan metode antrian yang digunakan adalah *Hierarchial Token Buckets (HTB)*. Dalam penelitian ini, hanya *interface* yang menghubungkan antara *router* diberikan *threshold* sebesar 10Mbps, sedangkan *link* yang menghubungkan antara *router* dan PC tidak diberi limitasi. Pendefinisian *threshold* ini dilakukan dengan menggunakan aturan antrian HTB dengan destinasi aliranberdasarkan ip address tujuan. Gambar 4-8 menunjukkan konfigurasi pemberian nilai *threshold* pada setiap *interface*.



Gambar 1-7. Limitasi *Interface* pada Sistem

Konfigurasi *tc rules* pada CentOS 7 disimpan pada file konfigurasi **/etc/rc.d/rc.local**. Untuk mengaktifkan konfigurasi yang telah dibuat, maka file konfigurasi tersebut terlebih dahulu diberikan *permission* dengan mengetikkan perintah **chmod +x /etc/rc.d/rc/local.** Gambar 4-9 merupakan potongan *script* penerapan *HTB tc rules* yang diimplementasikan pada *interface eth-1* dan *eth-2* di Router-1.

```
touch /var/lock/subsys/local

tc qdisc add dev eth1 root handle 1:0 htb
tc class add dev eth1 parent 1:0 classid 1:1 htb rate 10Mbit
tc filter add dev eth1 protocol ip parent 1:0 prio 0 u32 match ip dst 192.168.3.0/24 flowid 1:1

tc qdisc add dev eth2 root handle 1:0 htb
tc class add dev eth2 parent 1:0 classid 1:1 htb rate 10Mbit
tc filter add dev eth2 protocol ip parent 1:0 prio 0 u32 match ip dst 192.168.3.0/24 flowid 1:1
```

Gambar 1-8. Penerapan *HTB TC Rules* pada Router-1

Penjelasan script HTB tc rules

`tc qdisc add dev eth1 root handle 1:0 htb`

➔ Pembuatan *root* antrian "qdisc" 1:0 untuk interface eth1 dengan menggunakan disiplin antrian HTB.

`tc class add dev eth1 parent 1:0 classid 1:1 htb rate 10Mbit`

➔ Pembuatan kelas antrian HTB dengan id 1:1 di bawah aturan *root* 1:0. Setiap trafik yang dilewatkan eth-1 akan mendapatkan nilai *maximum rate* sebesar 10Mbps. HTB memastikan bahwa jumlah

layanan yang diberikan untuk setiap kelas yang terdefinisi adalah sesuai jumlah yang telah ditetapkan.

```
tc filter add dev eth1 protokol ip parent 1:0 prio 0 u32 match
ip dst 192.168.3.0/24 flowid 1:1
```

➔ Setiap paket yang melewati *interface* eth1 dan memiliki destinasi ke jaringan dengan prefix 192.168.3.0/24 akan diklasifikasikan untuk mengikuti aturan kelas antrian dengan id 1:1.

1.4.5 *Routing* Adaptif OSPF

Routing adaptif pada OSPF dikembangkan untuk mengurangi kemacetan jaringan dengan cara mendeteksi kemacetan yang terjadi dan mencari jalur lain yang bebas dari kemacetan (tingkat utilisasi *bandwidth* rendah) antara node sumber dan node tujuan. Konsepnya adalah sebagai berikut.

Ketika sebuah node sumber harus mentransmisikan paket data ke sebuah destinasi, protocol *routing* adaptif OSPF pertama-tama akan membangun sebuah kumpulan jalur yang bebas kemacetan (*Congestion Free Set-CFS*) yang menghubungkan antara satu *hop* dengan *hop* yang lain. Daftar jalur ini didapatkan dari informasi *routing table* pada *router*. Kemudian node sumber akan memulai prosedur penemuan rute menggunakan CFS untuk mengidentifikasi jalur bebas kemacetan ke tujuan.

Proses penemuan rute dengan CFS ini hanya dapat dilakukan apabila kemacetan sudah mulai terjadi pada jalur utama. Setelah jalur utama ditetapkan oleh Dijkstra, maka pengiriman paket data akan terus berlanjut. Sehingga tugas utama dari protocol *routing* adaptif OSPF ini adalah untuk menemukan jalur yang bebas dari kemacetan (tingkat utilisasi *bandwidth* rendah) antara node sumber dan node tujuan. Hal ini dicapai dengan terlebih dahulu melakukan pengamatan terhadap tingkat utilisasi *bandwidth* pada *interface router*, menghitung tingkat kemacetan, dan mengendalikan kemacetan yang terjadi dengan mengarahkan trafik ke jalur lain.

1.4.5.1 Deteksi Kemacetan Jaringan

Kemacetan ditandai dimana sebuah node pada interval waktu tertentu mengalami kepadatan dan mulai terjadi adanya *packet loss*. Beberapa parameter tersedia untuk memantau status kemacetan yang terjadi pada setiap node (*router*). Dalam protocol *routing* adaptif OSPF, parameter yang digunakan untuk memantau status kemacetan adalah berdasarkan tingkat utilisasi *bandwidth*. Dalam kurun waktu tertentu, *router* memeriksa secara langsung tingkat utilisasi *bandwidth* pada setiap *interface* yang menghubungkannya dengan *router* lain.

Parameter yang digunakan dalam menentukan status kemacetan jaringan antara lain *Minth*, *Maxth* dan w_q. *Minth* dan *Maxth* merupakan batas utilisasi yang menyajikan status terkini dari tingkat antrian paket yang ada pada *inbond interface*, sedangkan w_q merupakan parameter bobot antrian yang merepresentasikan tingkat utilisasi pada saat itu.

Kinerja deteksi kemacetan pada protokol *routing* adaptif OSPF berpedoman pada ketiga parameter tersebut. Jika ambang batasnya kecil, maka utilisasi *link* akan sangat rendah. Sebaliknya, jika ambang batas terlalu tinggi, maka kemacetan mungkin terjadi sebelum node diberi informasi untuk mengatasi hal tersebut. Oleh karena itu, diusulkan strategi pemilihan ambang batas yang efektif.

Berikut ini merupakan pengaturan parameter untuk proses deteksi kemacetan jaringan pada protokol *routing* adaptif OSPF.

*Minth = 20% * Queue_size*
*Maxth = 4 * Minth*

Berdasarkan referensi aturan antrian yang digunakan dalam algoritme RED (Floyd and Jacobson, 1993), maka parameter *Maxth* diatur dengan nilai minimal 2 kali lipat dari nilai *Minth*. Sehingga memberikan batasan nilai maksimum utilisasi *bandwidth* (*threshold*) untuk menentukan kemacetan adalah sebesar 80%, yaitu nilai *Maxth.*

Jika nilai w_q lebih kecil dibandingkan *Minth,* menunjukkan bahwa *link* berada dalam tingkat kemacetan rendah atau tidak terjadi kemacetann (zona aman). Jika nilai w_q berada di antara nilai *Minth* dan nilai *Maxth,* maka *link* diidentifikasikan berada di zona tengah, yaitu

rentan terjadi kemacetan. Jika nilai w_q lebih besar daripada *Maxth*, menunjukkan bahwa *link* mengalami kemacetan (zona kemacetan). Berikut ini merupakan *pseudocode* konsep deteksi kemacetan yang dilakukan oleh protokol *routing* adaptif OSPF.

Algorithm I: Congestion Detection

```
//initialization
Queue_size = 0;
Link_utilization = 0;
Minth = 0.2 * link_utilization;
Maxth = 4 * Minth;
//for each arriving packet in the queue
Queue_size ++
     If the queue is not empty, then
          Wq = link_utilization;
//Calculate average link utilization
     If Wq < Minth then
          Begin
          Link_status = "safe";
     Else if Wq > Minth && Wq < Maxth then
          Begin
          Link_Status = "likely to be congested";
     Else
          Link_Status = "congested"
     End
//for each departing packet in the queue
Queue_Size --
```

1.4.5.2 Pengendalian Kemacetan Jaringan

Merujuk pada Gambar 3-4, maka pengendalian kemacetan dibedakan menjadi 2 fase. Fase pertama, adalah melakukan pencarian terhadap jalur alternatif dari node sumber ke node tujuan. Fase kedua, adalah melakukan pemodifikasian nilai *cost* pada jalur dengan tingkat utilisasi *bandwidth* yang rendah.

Proses pencarian jalur alternatif dari node sumber ke node tujuan dilakukan apabila nilai w_q berada di antara nilai *Minth* dan *Maxth*, atau nilai w_q lebih besar dibandingkan nilai *Maxth*.

Kesimpulannya, pencarian jalur dilakukan apabila kondisi jaringan terkini memiliki utilisasi *link* yang dinyatakan pada zona rentan kemacetan sampai zona macet.

Keseluruhan jalur didefinisikan pada *routing table.* Tetapi, hanya satu jalur saja yang akan ditampilkan sebagai jalur terbaik untuk menuju ke sebuah destinasi. Jalur alternatif, merupakan jalur ke node tujuan dengan jumlah *hop* yang lebih banyak, atau memiliki panjang *path* yang lebih panjang dibandingkan jalur utama yang didefinisikan Dijkstra. Hal ini bisa diidentifikasi berdasarkan topologi yang terbangun, dengan melihat isi tabel *routing*.

Fase kedua adalah pemodifikasian nilai *cost metric* pada jalur alternatif. Pemodifikasian *cost metric* ini dilakukan dengan cara mengubah nilai *cost* dari node sumber ke node tujuan pada jalur alternatif, untuk memiliki nilai yang *cost* sama (secara total keseluruhan jalur) dengan jalur utama yang didefinisikan Dijkstra. Pengubahan nilai *cost* ini dilakukan pada *link* yang menghubungkan antar *router* dari node sumber ke node tujuan. Berikut ini merupakan *pseudocode* konsep pengendalian kemacetan jaringan pada protokol *routing* adaptif OSPF.

Algorithm II: Congestion Control

```
// initialization
    main_path = 0;
    alt_path = 0;
//for each arriving packet
   find all path to destination prefix
   N' = {u}
      for all nodes v
            if v is a neighbor of u
                  then D(v) = c(u,v)
            else D(v) = ∞
//loop
     Find w not in N' such that D(w) is a minimum
     Add w to N'
     Update D(v) for each neighbor v of w and not in N':
            D(v) = min ( D(v), D(w) + c(w,v) )
            /* new cost to v is either old cost to vv or known least path
               cost to w plus cost from w to v */
Until N' = N
//calculate best path (single-path)
```

```
main_path = D(v);

//activate multi-path
    If Link_status = "likely to be congested"
    or Link_status = "congested"
          Begin
                Find alternative path
  //initiate alternative path (multi-path)
    Calculate hops in main_path
          main_hops = a;
    Calculate hops in alternative_path
          alt_hops = b;
    Compute total cost in main path
          main_cost = x;
    Compute total cost in alternative path
          alt_cost = y;
    find all path to destination
          if b > a
                begin path_status = "alternative path"
          else
                path_status = "main path"
  //alternative path consideration
    If path_status = "alternative path"
          Begin
             Call procedure COST
  //cost modification
    Procedure COST (input: main_cost; output: new_cost)
    Begin
          If alt_path in routing_table, then
            Changing cost in alternative path
                new_cost = alt_cost / alt_hops
            Set alt_path = TRUE
                main_cost=x
                new_cost = y
                y = x
          Else
            Don't change the cost
          End
```

```
    End
End
```

Berikut ini merupakan definisi notasi yang digunakan.

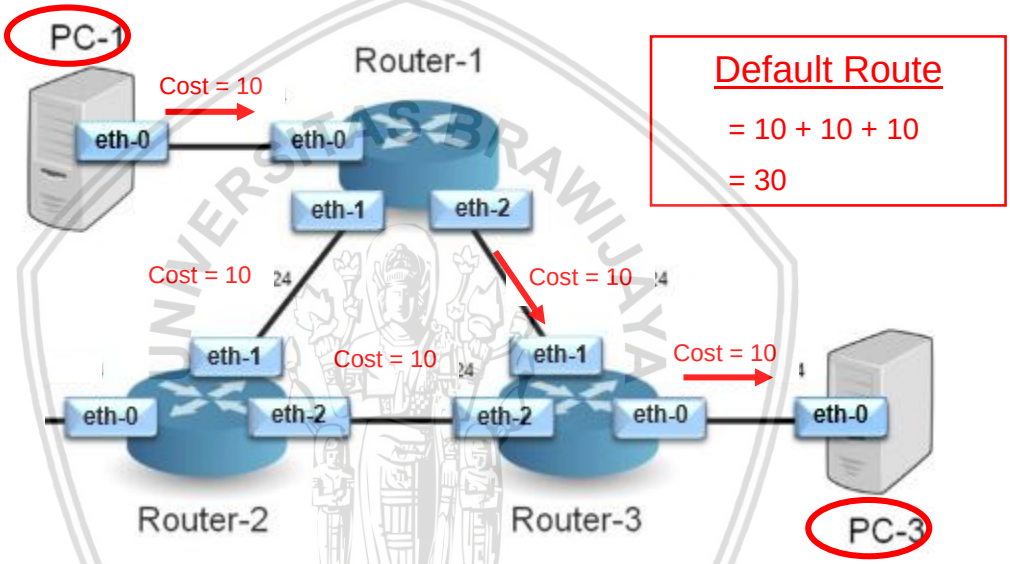
- *u* : node sumber
- *D(v)* : nilai *cost* terkecil dari node sumber ke node tujuan *v* sebagai hasil iterasi algoritme
- *p(v)* : node sebelumnya (tetangga dari *v*) sepanjang jalur terpendek dari node sumber ke *v*
- *N* : subset dari node; *v* merupakan salah satu bagian *N* jika jalur terpendek dari node sumber ke *v* telah diketahui.
- *main_path* : jalur utama yang didefinisikan oleh Algoritme Dijkstra untuk proses *routing*
- *alt_path* : jalur alternative yang didefinisikan melalui proses pertukaran informasi LSA pada *router* OSPF
- *main_hops* : kumpulan *hop* yang digunakan oleh jalur perutean utama
- *alt_hops* : kumpulan *hop* yang digunakan oleh jalur alternatif yang didefinisikan oleh protokol *routing* adaptif OSPF
- *main_cost* : jumlah total nilai *cost* dari node sumber ke node tujuan berdasarkan perhitungan *single-path routing* algoritme Dijkstra
- *alt_cost* : jumlah total nilai *cost* dari node sumber ke node tujuan dari jalur alternatif
- *new_cost* : nilai *cost* yang didefinisikan untuk menyamakan jumlah nilai total *cost* pada jalur utama dengan jumlah nilai total *cost* pada jalur alternatif

Berdasarkan algorima di atas, maka berikut ini merupakan uraian prosedur pengendalian kemacetan pada protokol *routing* adaptif OSPF sesuai skenario pada Gambar 4-3. Setiap *link* yang menghubungkan antar *node* pada Gambar 4-3 menggunakan kabel *Ethernet*, yang memiliki kapasitas masing-masing sebesar 100Mbps (pengecekan kecepatan dan kapasitas pada kernel Linux dilakukan dengan menggunakan perintah *lshw –class network*), sehingga nilai *cost metric* untuk kabel *Ethernet* adalah sebesar 10 (merupakan nilai *cost* standar untuk kabel Ethernet).

Untuk dapat menerapkan teknik *multipath routing* dengan menggunakan ECMP, maka perlu dilakukan penyamaan nilai *cost metric* jalur alternatif dan jalur utama, dari node sumber ke node tujuan. Dalam hal ini, yang menjadi node sumber adalah PC-1 (bertindak sebagai

client), yang akan mengirimkan sejumlah alirantrafik ke PC-3 sebagai node tujuan (bertindak sebagai *server*).

Berdasarkan komputasi yang dilakukan oleh Algoritme Dijkstra, maka jalur utama yang digunakan untuk mengirimkan trafik pada OSPF konvensional dari PC-1 ke PC-3 adalah melalui *interface* eth2 pada Router-1, dengan alamat IP 192.168.101.3 (lihar Gambar 4-6). Dengan demikian, jumlah nilai *cost metric* yang terdefinisi adalah 30. Gambar 4-10 menunjukkan jalur utama pengiriman trafik dari PC-1 ke PC-3 beserta dengan nilai *cost metric* yang didefinisikan oleh OSPF.



Gambar 1-9. Jalur Utama PC-1 → PC-3

Konsep *routing* adaptif didapatkan dengan cara mendefinisikan beberapa jalur yang tersedia dari node sumber ke node tujuan, yang dicapai dengan teknik *multipath routing* menggunakan ECMP. ECMP dikonfigurasikan dengan melakukan penyamaan nilai *cost metric* antara node sumber dengan node tujuan.

Jika melihat topologi jaringan pada Gambar 4-9, maka terdapat jalur lain dari PC-1 menuju ke PC-3, yaitu melalui Router-2, tetapi memiliki akumulasi nilai *cost metric* yang lebih besar, yaitu 40. Untuk menerapkan skema ECMP pada sistem, maka pemodifikasian nilai *cost metric* dilakukan pada Router-1, Router-2 dan Router-3 dengan tujuan membangun

skema ECMP simetrik. Tabel 4-4 menunjukkan pendefinisian nilai *symmetric cost metric* ECMP pada masing-masing *router*.

Tabel 1-4. Symmetric Cost ECMP

Router	Interface	Cost
Router-1	eth-1	5
Router-2	eth-1	5
	eth-2	5
Router-3	eth-2	5

Konfigurasi *cost metric* OSPF pada Quagga dilakukan melalui terminal 'vtysh'. Berdasarkan konfigurasi nilai *cost metric* yang tercantum pada Tabel 4-4, maka pada Router-1, hanya interface eth-1 saja yang diubah nilai *cost*-nya menjadi 5. Hal ini dilakukan dengan cara mendefinisikan nilai *cost* pada konfigurasi Quagga di Router-1, dengan menggunakan perintah sebagai berikut.

```
#vtysh
router-1#configure terminal
router-1(config)#interface eth1
router-1(config-if)#ip ospf cost 5
```

Gambar 4-11(a) menunjukkan cara mengkonfigurasi nilai cost metric pada Router-1 melalui Quagga, sedangkan Gambar 4-11(b) menunjukkan hasil konfigurasi cost metric yang telah dilakukan, dengan mengetikkan *sh run*.

```
router-1# configure  terminal
router-1(config)# interface  eth1
router-1(config-if)# ip ospf  cost  5
router-1(config-if)# end
router-1# wr
Building Configuration...
Configuration saved to /etc/quagga/zebra.conf
Configuration saved to /etc/quagga/ospfd.conf
[OK]
router-1#
```

(a)

```
hostname router-1
hostname ospfd@router-1
log stdout
!
password zebra
!
interface eth0
 description to-LAN
 ip address 192.168.1.1/24
 ipv6 nd suppress-ra
!
interface eth1
 description to-R2
 ip address 192.168.100.1/24
 ip ospf cost 5
 ipv6 nd suppress-ra
!
interface eth2
 description to-R3
 ip address 192.168.101.1/24
 ipv6 nd suppress-ra
!
interface eth3
 ipv6 nd suppress-ra
!
interface lo
!
router ospf
 ospf router-id 192.168.1.1
 network 192.168.1.0/24 area 0.0.0.0
 network 192.168.100.0/24 area 0.0.0.0
 network 192.168.101.0/24 area 0.0.0.0
!
ip forwarding
```

(b)

Gambar 1-10. Konfigurasi *Cost Metric* Router-1

Dengan mendefinisikan *symmetric cost* ECMP, maka algoritme Dijkstra akan melakukan komputasi ulang terhadap semua jalur pada jaringan. Sehingga, table *routing* akan diupdate kembali berdasarkan komputasi terbaru. Gambar 4-12 menunjukkan perbedaan antara table *routing* antara OSPF konvensional dan OSPF *Multipath*.

```
[root@router-1 ~]# ip ro
169.254.0.0/16 dev eth0 scope link metric 1002
169.254.0.0/16 dev eth1 scope link metric 1003
169.254.0.0/16 dev eth2 scope link metric 1004
169.254.0.0/16 dev eth3 scope link metric 1005
192.168.1.0/24 dev eth0 proto kernel scope link src 192.168.1.1
192.168.2.0/24 via 192.168.100.2 dev eth1 proto zebra metric 20
192.168.3.0/24 via 192.168.101.3 dev eth2 proto zebra metric 20
192.168.56.0/24 dev eth3 proto kernel scope link src 192.168.56.110
192.168.100.0/24 dev eth1 proto kernel scope link src 192.168.100.1
192.168.101.0/24 dev eth2 proto kernel scope link src 192.168.101.1
192.168.102.0/24 proto zebra metric 15
        nexthop via 192.168.100.2 dev eth1 weight 1
        nexthop via 192.168.101.3 dev eth2 weight 1
[root@router-1 ~]#
```

(a)

```
[fransiska@router-1 ~]$ sudo su -
Last login: Wed Jan 10 09:28:41 WIB 2018 on pts/0
[root@router-1 ~]# ip ro
169.254.0.0/16 dev eth0 scope link metric 1002
169.254.0.0/16 dev eth1 scope link metric 1003
169.254.0.0/16 dev eth2 scope link metric 1004
169.254.0.0/16 dev eth3 scope link metric 1005
192.168.1.0/24 dev eth0 proto kernel scope link src 192.168.1.1
192.168.2.0/24 via 192.168.100.2 dev eth1 proto zebra metric 15
192.168.3.0/24 proto zebra metric 20
        nexthop via 192.168.101.3 dev eth2 weight 1
        nexthop via 192.168.100.2 dev eth1 weight 1
192.168.56.0/24 dev eth3 proto kernel scope link src 192.168.56.110
192.168.100.0/24 dev eth1 proto kernel scope link src 192.168.100.1
192.168.101.0/24 dev eth2 proto kernel scope link src 192.168.101.1
192.168.102.0/24 via 192.168.100.2 dev eth1 proto zebra metric 10
[root@router-1 ~]#
```

(b)

Gambar 1-11. Perbedaan Tabel Routing Setelah Konfigurasi ECMP

Gambar 4-12(a) menunjukkan table routing dari Router-1 menggunakan OSPF konvensional, yaitu hanya terdapat satu jalur saja (*single-path*) dari PC-1 ke PC-3 (dengan prefix 192.168.3.0/24). Sedangkan pada Gambar 4-12(b) menunjukkan hasil *update* table *routing* dari Router-1 setelah dikonfigurasikan ECMP, sehingga terdapat dua jalur (*multi-path*) dari PC-1 ke PC-3 (dengan prefix 192.168.3.0/24).

BAB V
HASIL DAN PEMBAHASAN

Bab ini menyediakan informasi mengenai parameter dan konfigurasi yang diperlukan selama proses pengujian. Implementasi metode yang diusulkan dilakukan pada protokol *routing* OSPF secara pengujian dengan menggunakan *Quagga Software Routing Suite*. Prosedur implementasi merupakan jembatan penghubung untuk melakukan pengujian terhadap keberhasilan metode yang ditawarkan.

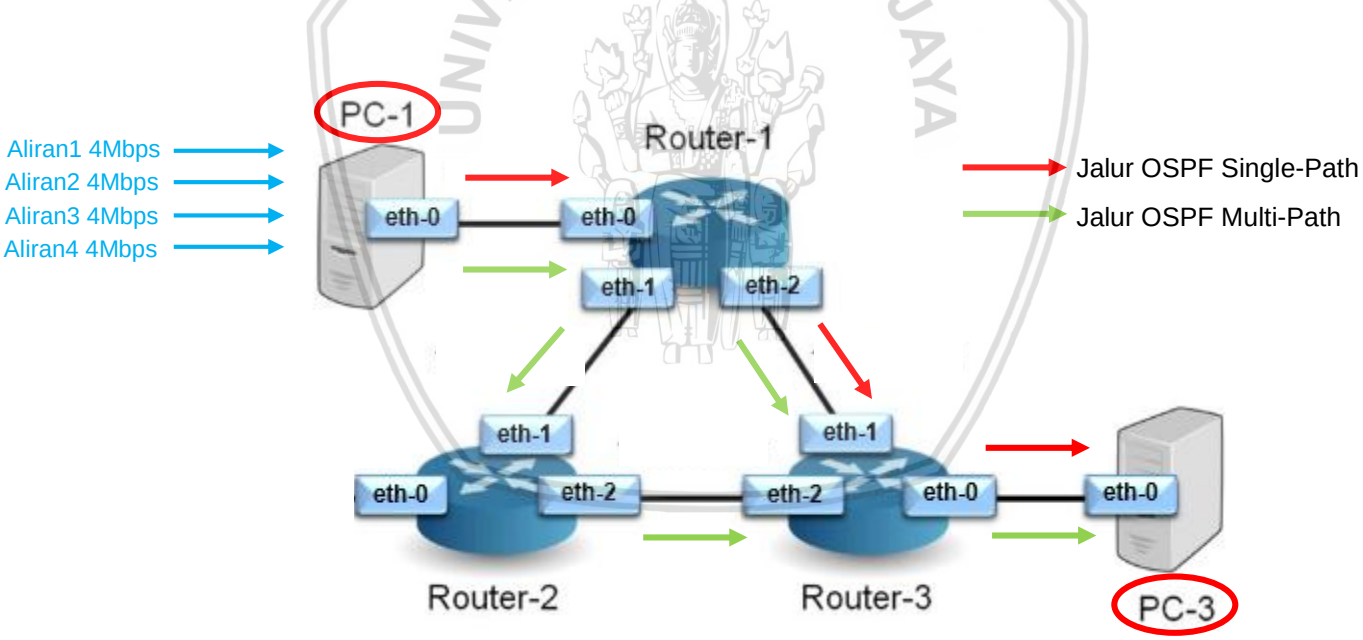
1.1 Pengujian

Konsep *routing* adaptif pada protokol OSPF dibangun dengan menggunakan *Open Source Routing Software Quagga.* Quagga memiliki *library* pengembangan yang cukup signifikan untuk memfasilitasi implementasi analisis performansi protokol *routing* OSPF. Quagga memiliki arsitektur *daemon* (servis layanan) terpusat yang dinamakan *zebra*. Setiap *daemon* dapat dikonfigurasi secara fleksibel melalui jaringan yang disebut CLI (*Command Line Interface)*. Keseluruhan konfigurasi *daemon* yang dilakukan oleh *administrator* dapat dikelola dalam satu lokasi melalui 'vtysh' (Ishiguro, 2011).

Pengujian terhadap sistem dilakukan dengan membangun topologi seperti yang ditunjukkan pada Gambar 4-3. Sistem dibuat dengan menghubungkan 3 buah PC yang dibangun sebagai Linux-Router. Ketiga *router* dikonfigurasikan untuk saling terhubung dengan menggunakan protokol *routing* OSPF dengan menggunakan Quagga, dimana setiap *router* didefinisikan sebagai *area 0* atau *backbone area*. Masing-masing router memilikii *link* yang terhubung dengan *LAN Network*, yang diwakilkan dengan keberadaan PC-1 untuk *LAN Network* Router-1 dan PC-3 untuk *LAN Network* Router-3. Setiap *link* yang menghubungkan antar *node* menggunakan *Ethernet*, yang memiliki kapasitas masing-masing sebesar 100Mbps (pengecekan kecepatan dan kapasitas pada kernel Linux dilakukan dengan menggunakan perintah *lshw –class network*), sehingga nilai *cost metric* untuk kabel *Ethernet* adalah sebesar 10 (merupakan nilai *cost* standar untuk kabel Ethernet, lihat Gambar 2-1).

Skenario pengujian sistem dilakukan sebagai berikut.

1. Setiap *link* yang menghubungkan antar *router* akan diberikan limitasi (*threshold*) sebesar 10Mbps, dengan menggunakan *tool tc rules (htb).*
2. Trafik akan dikirim dari PC-1 ke PC-3, dengan 4 alirantrafik masing-masing 4Mbps, dengan menggunakan *tool iperf* untuk protokol UDP dan *iperf3* untuk protokol TCP.
3. Beban link atau utilisasi *bandwidth* pada setiap *link* akan diukur dalam kurun waktu tertentu, yaitu 50s, 100s, 300s dan 600s.
4. Informasi utilisasi *bandwidth* pada setiap *router* akan dipantau selama pengujian pengiriman trafik berlangsung dengan menggunakan *tool bwm-ng*.
5. Pengujian pertama dilakukan dengan menggunakan jalur standar yang telah ditetapkan oleh OSPF konvensional.
6. Pengujian kedua dilakukan dengan menggunakan pemodifikasian OSPF *Multipath,* dengan mengkonfigurasikan jalur tambah dengan metode ECMP.
7. Hasil pengujian pertama dan pengujian kedua akan dianalisis dan diambil kesimpulan.



Gambar 1-1. Skenario Pengujian

Untuk melakukan komparasi hasil penelitian dengan performansi dari standar OSPF, dua set pengujian *routing* dijalankan. Pengujian pertama, menggunakan OSPF konvensional *single-path routing,* dan pengujian kedua, menjalankan OSPF Multi-Path, yaitu OSPF yang telah diintegrasikan dengan mekanisme *congestion control* dan *multipath routing*. Kedua

pengujian ini dijalankan dengan menggunakan komponen yang sama, topologi jaringan yang sama, dan parameter konfigurasi jaringan yang sama (Gambar 5-1).

OSPF konvensional dikonfigurasikan secara standar konfigurasi OSPF pada umumnya, sedangkan OSPF Multi-Path dikonfigurasikan dengan terlebih dahulu mendefinisikan nilai *cost metric* pada setiap *router*.

Berdasarkan skenario pengujian yang dijalankan, yaitu pengiriman 4 buah trafik dari PC-1 menuju ke PC-3, maka secara *default tabel routing* OSPF konvensional, akan mengirimkan seluruh trafik hanya melalui satu jalur saja, yakni dari Router-1 → Router-3, dengan jumlah nilai *cost* berdasarkan perhitungan Algoritme Dijkstra adalah *PC-1 to Router-1 (cost=10)* → *Router-1 to Router-3 (cost=10)* → *Router-3 to PC-3 (cost=10)* = 10 + 10 + 10 = 30.

Sedangkan, terdapat jalur lain yang dapat digunakan untuk mengirimkan trafik dari PC-1 ke PC-3, yaitu melalui Router-1, dengan jalur Router-1 → Router-2 → Router-3. Untuk dapat menerapkan teknik *multipath routing*, maka jumlah nilai *cost* pada jalur utama (PC-1 → Router-1 → Router-3 → PC-3) akan disamakan dengan jumlah nilai *cost* pada jalur alternatif (PC-1 → Router-1 → Router-2 → Router-3 → PC-3).

Dengan menggunakan *Open Source Quagga* pada kernel Linux 3.10, fitur *multipath routing* sudah secara otomatis aktif. Sehingga, dengan mendefinisikan nilai *cost metric* yang sama, *router* akan menampilkan jalur-jalur alternatif di dalam table *routing*. Tabel 5-1 menunjukkan perbedaan tabel routing antara OSPF konvensional dan OSPF Multi-Path berdasarkan pengujian yang dilakukan.

Tabel 1-1. Perbedaan Tabel Routing Pengujian OSPF dan OSPF Multi-Path

Source	**Destination**	**Route**	
		OSPF	**OSPF Multi-Path**
PC-1	PC-3	$R_1 \rightarrow R_3$	$R_1 \rightarrow R_3$
			$R_1 \rightarrow R_2 \rightarrow R_3$
PC-3	PC-1	$R_3 \rightarrow R_1$	$R_3 \rightarrow R_1$
			$R_3 \rightarrow R_2 \rightarrow R_1$

Tabel 5-2, Tabel 5-3 dan Tabel 5-4 memperlengkapi keterangan Tabel 5-2. Ketiga tabel ini mendefinisikan jalur-jalur yang tersimpan dalam tabel *routing* masing-masing *router* yang telah dikonfigurasikan dengan menggunakan OSPF dan ECMP, beserta dengan nilai *cost metric* yang telah ditentukan (*simetric cost ECMP*).

Tabel 1-2. Tabel Routing Router-1

Destination	Port	Hops	Cost
192.168.1.0/24	eth0	192.168.1.1	directly connected
192.168.2.0/24	eth1	192.168.100.2	15
192.168.3.0/24	eth2	192.168.101.3	20
	eth1	192.168.100.2	
192.168.100.0/24	eth1	192.168.100.1	directly connected
192.168.101.0/24	eth2	192.168.101.1	directly connected
192.168.102.0/24	eth1	192.168.100.2	directly connected

Tabel 1-3. Tabel Routing Router-2

Destination	Port	Hops	Cost
192.168.1.0/24	eth1	192.168.10.1	15
192.168.2.0/24	eth0	192.168.2.1	directly connected
192.168.3.0/24	eth2	192.168.102.3	15
192.168.100.0/24	eth1	192.168.100.2	directly connected
192.168.101.0/24	eth1	192.168.100.1	15
	eth2	192.168.102.3	
192.168.102.0/24	eth2	192.168.100.2	directly connected

Tabel 1-4. Tabel Routing Router-3

Destination	Port	Hops	Cost
192.168.1.0/24	eth1	192.168.101.1	20
	eth2	192.168.102.2	
192.168.2.0/24	eth2	192.168.102.2	15
192.168.3.0/24	eth0	192.168.3.1	directly connected
192.168.100.0/24	eth2	192.168.102.2	10
192.168.101.0/24	eth1	192.168.101.3	directly connected
192.168.102.0/24	eth2	192.168.102.3	directly connected

Terlihat pada tabel 5-2, untuk menuju ke jaringan 192.168.3.0/24, Router-1 mendefinisikan dua jalur, yaitu melalui *port* eth2 (*link* ke Router-3) dan melalui *port* eth1 (*link* ke Router-2) , dengan masing-masing nilai *cost metric* nya adalah 10. Artinya, penyamaan nilai *cost* antara jalur Router-1 → Router-3 dan jalur Router-1 → Router-2 → Router-3, telah berhasil dilakukan. Verifikasi keberhasilan dua jalur juga dapat dilakukan dengan menggunakan perintah *traceroute*. *Traceroute* merupakan perintah yang digunakan untuk menunjukkan rute yang dilewati oleh paket untuk mencapai tujuan. Rute yang ditampilkan adalah daftar *interface router* (yang paling dekat dengan *host*), yang terdapat pada jalur antara *host* dan tujuan. Gambar 5-2 menunjukkan hasil *traceroute* yang dilakukan dari PC-1 menuju ke PC-3 dengan terlebih dahulu dilakukan *ping* untuk mengecek konektivitas antara kedua node.

```
root@host-1:~# ping 192.168.3.30
PING 192.168.3.30 (192.168.3.30) 56(84) bytes of data.
64 bytes from 192.168.3.30: icmp_seq=1 ttl=61 time=12.4 ms
64 bytes from 192.168.3.30: icmp_seq=2 ttl=61 time=2.52 ms
64 bytes from 192.168.3.30: icmp_seq=3 ttl=61 time=10.4 ms
64 bytes from 192.168.3.30: icmp_seq=4 ttl=61 time=1.99 ms

--- 192.168.3.30 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3005ms
rtt min/avg/max/mdev = 1.995/6.851/12.480/4.655 ms
root@host-1:~# traceroute 192.168.3.30
traceroute to 192.168.3.30 (192.168.3.30), 30 hops max, 60 byte packets
 1  192.168.1.1 (192.168.1.1)  3.032 ms  2.967 ms  2.954 ms
 2  192.168.100.2 (192.168.100.2)  2.943 ms  2.932 ms  2.921 ms
 3  192.168.102.3 (192.168.102.3)  6.750 ms  6.747 ms  6.736 ms
 4  192.168.3.30 (192.168.3.30)  9.615 ms  9.566 ms  9.554 ms
root@host-1:~#
```

Gambar 1-2. *Traceroute* Jalur OSPF Multipath dari PC-1 ke PC-3

Tujuan penyamaan nilai *cost metric* ini adalah agar beban trafik yang dikirimkan dari PC-1 menuju ke PC-3 akan dibagi secara merata, sehingga kemacetan jaringan pun dapat dihindari. Dengan demikian, lalu lintas dapat tersebar ke seluruh *path* yang telah terdefinisi dan efektif mengurangi kemacetan dalam jaringan dan meningkatkan nilai *throughput* (Chiesa, Kindler and Schapira, 2014).

1.2 Analisa Performa Jaringan

1.2.1 Pengujian Protokol TCP

Pengujian pertama dilakukan dengan menggunakan *tool iperf3* melalui protokol TCP, untuk mengukur nilai *throughput* yang dilihat dari sisi *client*. Pengujian dilakukan dengan mengirimkan 4 alirantrafik sekaligus secara bersamaan. Setiap *link* antar *router* diberikan *threshold* sebesar 10Mbps, sehingga jika terdapat 4 alirantrafik @4Mbps (total trafik = 4x4Mbps = 16Mbps) yang dikirimkan melalui jalur tersebut, akan menimbulkan kemacetan jaringan, karena nilai *input traffic* (16Mbps) > *output capacity* (10Mbps).

Pengujian dilakukan sebanyak 4 kali, dimana PC-3 bertindak sebagai *server* , dan PC-1 bertindak sebagai *client*. Untuk meng-*generate* trafik, terlebih dahulu *shell command* dijalankan pada PC-3 yang bertindak sebagai *server*.

```
#iperf3 -s -B 192.168.3.31
```

Perintah **–s** menunjukkan PC-3 menjalankan *tool* iperf3 sebagai server, dan perintah **–B** menunjukkan bahwa *bind-ing* ke *interface* dengan alamat tertentu, yaitu 192.168.3.31.

Selanjutnya menjalankan iperf3 pada sisi *client*, yaitu pada PC-1 untuk mulai melakukan generate trafik, dengan perintah sebagai berikut.

```
#iperf3 –B 192.168.1.11 –c 192.168.3.31 –b 4M –t 50
```

Perintah **–c** menunjukkan PC-1 menjalankan *tool* iperf3 sebagai *client*. Perintah **–B** menunjukkan *interface* pada PC-1 dengan alamat IP 192.168.1.11 melakukan *generate* trafik pada *server* PC-3 dengan alamat IP 192.168.3.31. Perintah **–b 4M**, menunjukkan bahwa trafik yang dikirimkan adalah sebesar 4Mbps, dengan kurun waktu selama 50s (**-t 50**).

Tabel 5-5 menunjukkan alamat IP *interface* yang digunakan untuk pengujian pengiriman trafik antara PC-1 (*client*) dan PC-3 (*server*).

Tabel 1-5. Alamat IP Pengujian iperf Server dan Client

No	Server	Client
1	192.168.3.31	192.168. 1.11
2	192.168.3.32	192.168. 1.12
3	192.168.3.33	192.168. 1.13
4	192.168.3.34	192.168. 1.14

Pengujian dilakukan selama 4 kali pengujian untuk masing-masing *routing*, dengan interval waktu 50s, 100s, 300s, dan 600s. Tabel 5-6 menunjukkan perbandingan nilai *throughput* yang dihasilkan antara OSPF Single-Path (konvensional) dan OSPF Multi-Path, pada pengujian pasangan *server* dan *client* no. 1 (Tabel 5-5) dengan interval waktu 50s.

Hasil akhir pada setiap pengujian akan disimpulkan seperti yang terlihat pada Gambar 5-3. Pada salah satu pengujian yang dilakukan, terlihat perbedaan yang cukup signifikan antara *throughput* yang dihasilkan OSPF Single-Path (Gambar 5-3a) dan OSPF Multi-Path (Gambar 5-3b).

Interval			Transfer	Bandwidth	Retr	
0	-	50	14.6 Mbytes	2.45 Mbits/sec	1	sender
0	-	50	14.6 Mbytes	2.43 Mbits/sec		receiver

(a)

Interval			Transfer	Bandwidth	Retr	
0	-	50	23.8 Mbytes	4 Mbits/sec	5	sender
0	-	50	23.8 Mbytes	4 Mbits/sec		receiver

(b)

Gambar 1-3. Hasil Akhir Pengujian iperf3

Tabel 1-6. Perbandingan Hasil Throughput (Interval Waktu=50s)

Interval (s)	Bandwidth (Mbits/sec)		Retransmission	
	OSPF Single-Path	OSPF Multi-Path	Single	Multi
0 - 1	3.84	3.84	1	2
1 - 2	4.19	4.19	0	1
2 - 3	4.2	4.2	0	0
3 - 4	3.15	4.19	0	0
4 - 5	4.19	4.19	0	0
5 - 6	2.1	3.14	0	0
6 - 7	1.05	4.19	0	0
7 - 8	4.19	4.2	0	0
8 - 9	2.1	4.2	0	0
9 - 10	1.05	4.19	0	0
10 - 11	3.15	3.14	0	0
11 - 12	1.05	4.2	0	0
12 - 13	3.14	4.19	0	0
13 - 14	3.15	4.19	0	0
14 - 15	0	4.19	0	0
15 - 16	4.19	3.14	0	0
16 - 17	2.1	4.2	0	0
17 - 18	0	4.19	0	0
18 - 19	6.3	4.19	0	0
19 - 20	0	4.19	0	0
20 - 21	1.05	4.19	0	0
21 - 22	5.24	3.14	0	0
22 - 23	0	4.21	0	0
23 - 24	2.1	4.19	0	0
24 - 25	4.2	4.19	0	0
25 - 26	0	4.19	0	1
26 - 27	4.19	3.15	0	0
27 - 28	2.1	4.2	0	0
28 - 29	0	4.19	0	0
29 - 30	4.19	4.19	0	0
30 - 31	2.1	4.19	0	0
31 - 32	0	4.19	0	0
32 - 33	4.19	3.14	0	1
33 - 34	2.1	4.19	0	0
34 - 35	0	4.2	0	0
35 - 36	6.29	4.19	0	0
36 - 37	0	4.2	0	0
37 - 38	0	3.15	0	0
38 - 39	6.29	4.19	0	0
39 - 40	0	4.19	0	0
40 - 41	1.05	4.19	0	0
41 - 42	5.24	4.19	0	0
42 - 43	0	3.15	0	0
43 - 44	4.19	4.19	0	0
44 - 45	2.1	4.19	0	0
45 - 46	0	4.2	0	0
46 - 47	4.19	4.19	0	0
47 - 48	2.1	4.19	0	0
48 - 49	0	3.14	0	0
49 - 50	6.29	4.19	0	0

Tabel 1-7. Perbandingan Hasil Throughput (Interval Waktu=100s)

Interval (s)	Bandwidth (Mbits/sec)		Retransmission	
	OSPF Single-Path	OSPF Multi-Path	Single	Multi
0 - 1	3.84	3.84	1	2
1 - 2	4.2	4.19	0	1
2 - 3	4.19	4.2	0	0
3 - 4	4.19	4.19	0	0
4 - 5	4.19	4.19	0	0
5 - 6	3.15	3.14	0	0
6 - 7	4.19	4.19	0	0
7 - 8	2.1	4.2	0	0
8 - 9	3.14	4.2	0	0
9 - 10	3.15	4.19	0	0
10 - 11	2.1	3.14	0	0
11 - 12	2.1	4.2	0	0
12 - 13	3.15	4.19	0	0
13 - 14	2.1	4.19	0	0
14 - 15	3.15	4.19	0	0
15 - 16	3.15	3.14	0	0
16 - 17	1.05	4.2	0	0
17 - 18	2.1	4.19	0	0
18 - 19	3.15	4.19	0	0
19 - 20	1.05	4.19	0	0
20 - 21	3.14	4.19	0	0
21 - 22	3.15	3.14	0	0
22 - 23	2.1	4.21	0	0
23 - 24	3.14	4.19	0	0
24 - 25	1.05	4.19	0	0
. - .	.	.	.	.
. - .	.	.	.	.
. - .	.	.	.	.
. - .	.	.	.	.
. - .	.	.	.	.
. - .	.	.	.	.
. - .	.	.	.	.
. - .	.	.	.	.
. - .	.	.	.	.
84 - 85	3.14	4.19	0	0
85 - 86	1.05	4.19	0	0
86 - 87	3.14	3.15	0	0
87 - 88	2.1	4.18	0	0
88 - 89	2.1	4.21	0	0
89 - 90	3.14	4.19	0	0
90 - 91	2.1	4.19	0	0
91 - 92	2.1	3.13	0	0
92 - 93	3.15	4.21	0	0
93 - 94	1.05	4.2	0	0
94 - 95	2.1	4.19	0	0
95 - 96	3.15	4.21	0	0
96 - 97	1.05	3.14	0	0
97 - 98	2.1	4.2	0	0
98 - 99	3.15	4.19	0	0
99 - 100	2.1	4.19	0	0

Tabel 1-8. Perbandingan Hasil Throughput (Interval Waktu=300s)

Interval (s)	Bandwidth (Mbits/sec)		Retransmission	
	OSPF Single-Path	OSPF Multi-Path	Single	Multi
0 - 1	3.84	3.83	0	2
1 - 2	4.2	4.2	0	0
2 - 3	4.19	4.19	0	0
3 - 4	4.19	4.19	0	0
4 - 5	4.19	4.19	0	0
5 - 6	3.15	3.15	0	0
6 - 7	4.19	4.19	0	0
7 - 8	2.1	4.19	0	0
8 - 9	3.14	4.2	0	0
9 - 10	3.15	4.19	0	0
10 - 11	2.1	3.15	0	0
11 - 12	2.1	4.19	0	0
12 - 13	3.15	4.19	0	0
13 - 14	2.1	4.19	0	0
14 - 15	3.15	4.19	0	0
15 - 16	3.15	3.14	0	0
16 - 17	1.05	4.18	0	0
17 - 18	2.1	4.21	0	0
18 - 19	3.15	4.2	0	0
19 - 20	1.05	4.19	0	0
20 - 21	3.14	4.19	0	0
21 - 22	3.15	3.15	0	0
22 - 23	2.1	4.19	0	0
23 - 24	3.14	4.2	0	0
24 - 25	1.05	4.19	0	0
. - .	.	.	.	.
. - .	.	.	.	.
. - .	.	.	.	.
. - .	.	.	.	.
. - .	.	.	.	.
. - .	.	.	.	.
. - .	.	.	.	.
. - .	.	.	.	.
. - .	.	.	.	.
284 - 285	3.15	4.19	0	0
285 - 286	2.1	3.15	0	0
286 - 287	2.1	4.19	0	0
287 - 288	2.1	4.19	0	0
288 - 289	2.1	4.19	0	0
289 - 290	2.1	4.19	0	0
290 - 291	2.1	4.19	0	0
291 - 292	2.1	3.15	0	0
292 - 293	3.15	4.19	0	0
293 - 294	1.05	4.19	0	0
294 - 295	2.1	4.19	0	0
295 - 296	4.19	4.19	0	0
296 - 297	0	3.15	0	0
297 - 298	2.1	4.19	0	0
298 - 299	4.19	4.2	0	0
299 - 300	1.05	4.2	0	0

Tabel 1-9. Perbandingan Hasil Throughput (Interval Waktu=600s)

Interval (s)	Bandwidth (Mbits/sec)		Retransmission	
	OSPF Single-Path	OSPF Multi-Path	Single	Multi
0 - 1	3.84	3.84	1	0
1 - 2	4.19	4.18	0	0
2 - 3	4.2	4.21	0	0
3 - 4	4.2	4.19	0	0
4 - 5	4.19	4.19	0	0
5 - 6	3.15	3.14	0	0
6 - 7	4.19	4.19	0	0
7 - 8	2.1	4.2	0	0
8 - 9	3.15	4.19	0	0
9 - 10	3.15	4.19	0	1
10 - 11	2.1	3.15	0	0
11 - 12	3.15	4.19	0	0
12 - 13	2.1	4.19	0	0
13 - 14	3.15	4.19	0	0
14 - 15	3.14	4.19	0	1
15 - 16	1.05	3.14	0	0
16 - 17	2.1	4.2	0	0
17 - 18	3.15	4.19	0	0
18 - 19	1.05	4.2	0	0
19 - 20	3.15	4.19	0	0
20 - 21	2.1	4.2	0	0
21 - 22	1.05	3.14	0	0
22 - 23	3.15	4.2	0	0
23 - 24	3.14	4.19	0	1
24 - 25	2.1	4.19	0	0
. - .	.	.	.	.
. - .	.	.	.	.
. - .	.	.	.	.
. - .	.	.	.	.
. - .	.	.	.	.
. - .	.	.	.	.
. - .	.	.	.	.
. - .	.	.	.	.
. - .	.	.	.	.
584 - 585	2.1	4.19	0	0
585 - 586	2.1	4.19	0	0
586 - 587	3.14	4.2	0	0
587 - 588	2.1	4.19	0	0
588 - 589	3.15	3.14	0	0
589 - 590	2.1	4.19	0	0
590 - 591	2.1	4.2	0	0
591 - 592	2.1	4.19	0	0
592 - 593	2.1	4.2	0	0
593 - 594	2.1	3.14	0	0
594 - 595	3.15	4.2	0	0
595 - 596	2.1	4.2	0	0
596 - 597	3.14	4.19	0	0
597 - 598	2.1	4.19	0	0
598 - 599	2.1	3.15	0	0
599 - 600	2.1	4.19	0	0

Keempat tabel hasil pengujian di atas memperlihatkan bahwa terdapat perbedaan secara signifikan terhadap nilai *throughput* yang dihasilkan pada OSPF Single-Path dan OSPF Multi-Path. Dengan lama waktu pengujian mulai dari 50s, 100s, 300s sampai 600s, nilai *throughput* yang dihasilkan oleh keduanya tidak memberikan perubahan secara spesifik. Perbedaan ini dihitung secara rata-rata dan ditampilkan pada Tabel 5-7.

Tabel 1-10. Perbandingan Rata-rata Throughput OSPF dan OSPF Multi-Path

Time	**Throughput**	
	Single-Path	**Multi-Path**
50s	2.38300 Mbits/s	3.99705 Mbits/s
100s	2.35965 Mbits/s	4.00065 Mbits/s
300s	2.32595 Mbits/s	3.99963 Mbits/s
600s	2.343275 Mbits/s	3.99867 Mbits/s

Rata-rata nilai *throughput* yang dihasilkan pada OSPF Single-Path hanya berkisar 2.3Mbps, artinya hanya bernilai setengah dari nilai *bandwidth* trafik yang dikirimkan. Hal ini terjadi karena trafik yang dikirimkan hanya melewati satu jalur saja, dengan nilai batasan maksimal yang dapat dikirimkan hanya sebatas 10Mbps. Jika keempat trafik ini digabungkan, maka jumlah nilai *throughput* yang dihasilkan dalam sekali pengiriman adalah sebesar 2.3Mbps x 4aliran≈ 9.2Mbps. Padahal, jumlah trafik yang harus dikirimkan adalah sebesar 4.00Mbps x 4aliran≈ 16Mbps, sehingga terdapat penurunan nilai *throughput* sebesar 42,5%. Penurunan nilai *throughput* ini menandakan bahwa terjadi kemacetan pada jalur tersebut, sehingga data yang dikirimkan tidak dapat tersampaikan seutuhnya (pembuktian ini terlihat dari hasil pengujian protokol UDP pada sub-bab 5.3).

Sedangkan pada OSPF Multi-Path, nilai *throughput* yang dihasilkan adalah 3.9Mbps, atau hampir setara dengan nilai *bandwidth* yang dikirimkan. Artinya, tidak terjadi penurunan nilai *throughput*, sebaliknya *throughput* yang dihasilkan adalah konstan atau stabil. Hal ini terjadi karena trafik yang dikirimkan akan dibagi secara merata melalui dua port pada Router-1, yakni *port* eth2 dan *port* eth1. Terjadi keseimbangan beban jaringan yang dikirimkan pada kedua port, sehingga jalur yang digunakan tidak mengalami kemacetan dan data yang dikirimkan dapat diterima secara utuh (tidak ada *packet loss*).

1.2.2 Pengujian Protokol UDP

Pengujian kedua dilakukan dengan menggunakan *tool iperf* melalui protokol UDP, untuk mengukur beberapa nilai, di antaranya jumlah *transfer file*, nilai *throughput*, nilai *jitter*, dan prosentase *packet loss* yang terjadi. Pengujian yang dilakukan sama dengan pengujian pada protokol TCP, yaitu mengirimkan 4 alirantrafik sekaligus secara bersamaan.

Pengujian dilakukan sebanyak 4 kali, dimana PC-3 bertindak sebagai *server* , dan PC-1 bertindak sebagai *client*. Alamat IP yang digunakan selama pengujian protokol UDP sama dengan alamat IP yang digunakan pada pengujian protokol TCP (lihat Tabel 5-5). Untuk meng-*generate* trafik, terlebih dahulu *shell command* dijalankan pada PC-3 yang bertindak sebagai *server*.

```
#iperf –u –s –B 192.168.3.31
```

Perintah **–u** menunjukkan bahwa trafik yang dikirim adalah melalui protokol UDP. Perintah **–s** menunjukkan PC-3 menjalankan *tool* iperf sebagai server, dan perintah **–B** menunjukkan bahwa *bind-ing* ke *interface* dengan alamat tertentu, yaitu 192.168.3.31.

Selanjutnya menjalankan iperf pada sisi *client*, yaitu pada PC-1 untuk mulai melakukan generate trafik, dengan perintah sebagai berikut.

```
#iperf –u –B 192.168.1.11 –c 192.168.3.31 –b 4M –t 600
```

Perintah **–u –c** menunjukkan PC-1 menjalankan *tool* iperf sebagai *client* melalui protokol UDP. Perintah **–B** menunjukkan *interface* pada PC-1 dengan alamat IP 192.168.1.11 melakukan *generate* trafik pada *server* PC-3 dengan alamat IP 192.168.3.31. Perintah **–b 4M**, menunjukkan bahwa trafik yang dikirimkan adalah sebesar 4Mbps, dengan kurun waktu selama 600s (**-t 600**).

Pada percobaan yang pertama, dilakukan pengiriman trafik dengan kurun waktu selama 50s. Sehingga, jumlah file yang dikirimkan adalah sebesar 23.8Mbytes. Hasil pengujian *iperf* untuk protokol UDP selama 50s tertera pada Tabel 5-12.

Tabel 1-11. Percobaan 1 (Jumlah trafik=4, Besar trafik@4Mbps, Waktu=50s, Total Transfer=23.8MBytes)

No	Parameter	Single-Path					Multi-Path				
		1	2	3	4	Avg	1	2	3	4	Avg
1	Transfer File (Mbytes)	9.83	18.3	17.3	13.9	14.8325	23.8	23.8	23.8	23.8	23.8
2	Throughput (Mbits/sec)	1.61	2.99	2.83	2.28	2.4275	4.00	4.00	4.00	4.00	4.00
3	Jitter (ms)	0.907	1.174	1.131	0.884	1.024	0.614	0.22	0.389	0.083	0.3265
4	Packet Loss (%)	59	23	27	42	37.75	0	0	0	0	0

Tabel 1-12. Percobaan 2 (Jumlah trafik=4, Besar trafik@4Mbps, Waktu=100s, Total Transfer=47.7MBytes)

No	Parameter	Single-Path					Multi-Path				
		1	2	3	4	Avg	1	2	3	4	Avg
1	Transfer File (Mbytes)	14.6	33.3	25.8	43.4	29.275	47.7	47.7	47.7	47.7	47.7
2	Throughput (Mbits/sec)	1.21	2.76	2.14	3.6	2.4275	4.00	4.00	4.00	4.00	4.00
3	Jitter (ms)	0.701	0.715	0.979	0.604	0.74975	0.402	0.396	0.58	1.066	0.611
4	Packet Loss (%)	69	30	46	8.9	38.475	0	0	0	0	0

Tabel 1-13. Percobaan 3 (Jumlah trafik=4, Besar trafik@4Mbps, Waktu=300s, Total Transfer=143MBytes)

No	Parameter	Single-Path					Multi-Path				
		1	2	3	4	Avg	1	2	3	4	Avg
1	Transfer File (Mbytes)	79.6	84.3	98.6	86.1	87.15	143	143	143	143	143
2	Throughput (Mbits/sec)	2.22	2.35	2.75	2.4	2.43	4.00	4.00	4.00	4.00	4.00
3	Jitter (ms)	1.021	0.676	1.284	0.671	0.913	0.702	0.398	0.649	0.199	0.487
4	Packet Loss (%)	44	41	31	40	39	0	0	0	0	0

Tabel 1-14. Percobaan 4 (Jumlah trafik=4, Besar trafik@4Mbps, Waktu=600s, Total Transfer=286MBytes)

No	Parameter	Single-Path					Multi-Path				
		1	2	3	4	Avg	1	2	3	4	Avg
1	Transfer File (Mbytes)	174	120	157	246	174.25	286	286	286	286	286
2	Throughput (Mbits/sec)	2.42	1.67	2.19	3.43	2.4275	4.00	4.00	4.00	4.00	4.00
3	Jitter (ms)	0.808	0.678	0.605	1.364	0.86375	0.314	0.65	0.463	0.346	0.44325
4	Packet Loss (%)	39	58	45	14	39	0	0	0	0	0

Empat parameter yang diamati selama percobaan pertama menunjukkan perbedaan yang cukup menonjol antar OSPF Single-Path dan OSPF Multi-Path. Adanya kemacetan jalur selama proses pengiriman trafik berlangsung pada OSPF Single-Path menyebabkan nilai *throughput* yang dihasilkan hanya berkisar setengah dari *bandwidth* yang seharusnya, sehingga hal ini berdampak kepada jumlah *transfer file* (rata-rata 14,8325 Mbytes) yang dilakukan tidak dapat mencapai *transfer file* yang seharusnya (23,8 Mbytes). Tentunya hal ini akan mempengaruhi nilai *packet loss* yang dominan cukup besar.

Sebaliknya, pada OSPF Multi-Path, *throughput* yang dihasilkan adalah setara dengan nilai *bandwidth* yang dikirimkan, sehingga jumlah *transfer file* yang tercatat adalah sama dengan besar file yang dikirim, yaitu 23,8Mbytes. Tidak adanya *packet loss* selama pengiriman berlangsung karena kestabilan nilai *throughput* yang dihasilkan, menyebabkan pengiriman data dengan menggunakan OSPF Multi-Path lebih cepat dibandingkan OSPF Single-Path, hal ini dibuktikan dengan nilai *jitter* yang dihasilkan OSPF Multi-Path (0,3265ms) adalah sepertiga dari nilai *jitter* yang dihasilkan oleh OSPF Single-Path (1,024ms).

Percobaan kedua dilakukan pengiriman trafik dengan kurun waktu selama 100s. Sehingga, jumlah file yang dikirimkan adalah sebesar 47,7Mbytes. Hasil pengujian *iperf* untuk protokol UDP selama 100s tertera pada Tabel 5-12.

Nilai *throughput* yang dihasilkan oleh OSPF Single-Path hampir memiliki persamaan dengan percobaan pertama, yakni berkisar setengah dari nilai *bandwidth*. Dampaknya, *transfer file* yang dilakukan akan mengalami *losses* hingga mencapai 38,475%. Sebaliknya, pada OSPF Multi-Path, *throughput* yang dihasilkan adalah setara dengan nilai *bandwidth* yang dikirimkan, sehingga jumlah *transfer file* yang tercatat adalah sama dengan besar file yang dikirim, yaitu 47,7Mbytes. Hanya terdapat sedikit perbedaan nilai *jitter* antara OSPF Single-Path dan OSPF Multi-Path, yakni 0,13875.

Percobaan ketiga dilakukan pengiriman trafik dengan kurun waktu selama 300s. Sehingga, jumlah file yang dikirimkan adalah sebesar 143Mbytes. Hasil pengujian *iperf* untuk protokol UDP selama 100s tertera pada Tabel 5-13.

Nilai *throughput* yang dihasilkan oleh OSPF Single-Path hampir memiliki persamaan dengan percobaan pertama dan kedua, yakni berkisar setengah dari nilai *bandwidth*. Dampaknya, *transfer file* yang dilakukan akan mengalami *losses* hingga mencapai 44%. Sebaliknya, pada OSPF Multi-Path, *throughput* yang dihasilkan adalah setara dengan nilai *bandwidth* yang dikirimkan, sehingga jumlah *transfer file* yang tercatat adalah sama dengan besar file yang dikirim, yaitu 143Mbytes. Pada percobaan ini, nilai *jitter* pada OSPF Multi-Path tetap lebih baik dibandingkan dengan OSPF Single-Path.

Percobaan keempat dilakukan pengiriman trafik dengan kurun waktu selama 600s. Sehingga, jumlah file yang dikirimkan adalah sebesar 286Mbytes. Hasil pengujian *iperf* untuk protokol UDP selama 100s tertera pada Tabel 5-14.

Nilai *throughput* yang dihasilkan oleh OSPF Single-Path mengalami penurunan bahkan sampai mencapa 1,67Mbps. Dampaknya, *transfer file* yang dilakukan akan mengalami *losses* terbesar hingga mencapai 58%. Sebaliknya, pada OSPF Multi-Path, *throughput* yang dihasilkan adalah setara dengan nilai *bandwidth* yang dikirimkan, sehingga jumlah *transfer file* yang tercatat adalah sama dengan besar file yang dikirim, yaitu 286Mbytes, yang menandakan tidak adanya *packet loss* selama pengiriman berlangsung. Nilai *jitter* yang dihasilkan oleh OSPF Multi-Path adalah setengah dari nilai *jitter* yang dihasilkan oleh OSPF Single-Path.

Dari keempat percobaan protokol UDP yang telah dilakukan, dapat disimpulkan bahwa OSPF Multi-Path memiliki performansi yang 50% lebih baik dibandingkan dengan OSPF Single-Path, ditinjau dari parameter *throughput*, *jitter* dan *transfer file*. Hal ini semakin diperkuat dengan prosentase OSPF Multi-Path yang stabil pada prosentase 0% selama pengiriman trafik berlangsung.

1.2.3 *Log File Router (bwm-ng)*

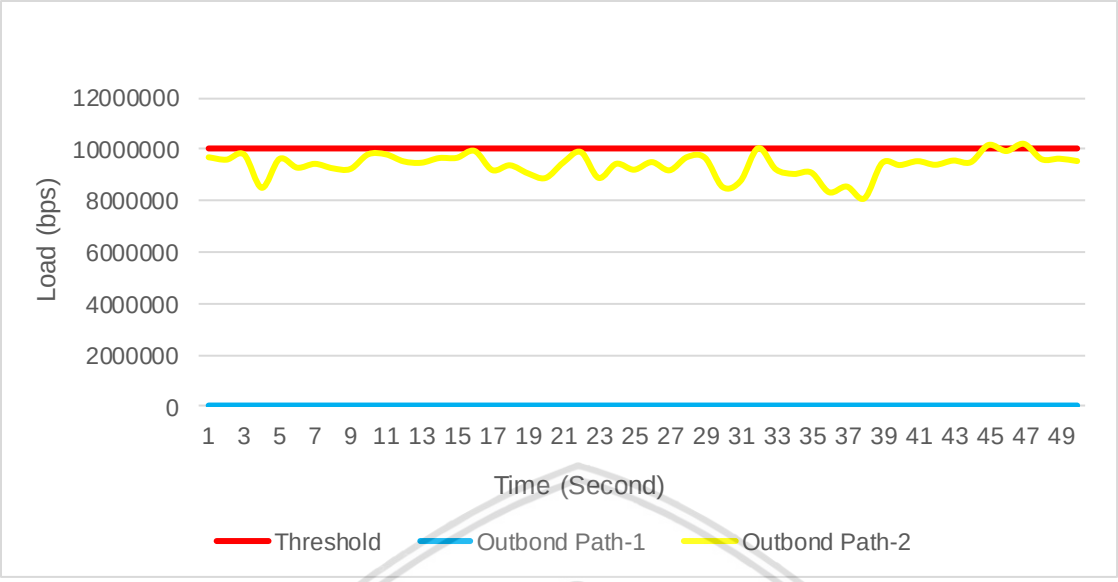
Pengujian dengan menggunakan *tool iperf* pada protokol UDP dan *tool iperf3* pada protokol TCP dilakukan dengan mengamati hasil pada sisi *client* (*end-to-end communications)*. Untuk mendukung kebenaran hasil pengujian, dilakukan pengamatan juga pada sisi *router* dengan menggunakan *tool bwm-ng*. Selama proses pengujian protokol UDP maupun protokol

TCP, *bwm-ng* dijalankan pada masing-masing *router* untuk mengetahui berapakah jumlah trafik yang dikirimkan pada setiap *interface*. Selain itu, untuk memantau apakah trafik yang dikirimkan melalui *link* antar *router*, melebihi ataukah di bawah nilai *threshold* yang telah didefinisikan.

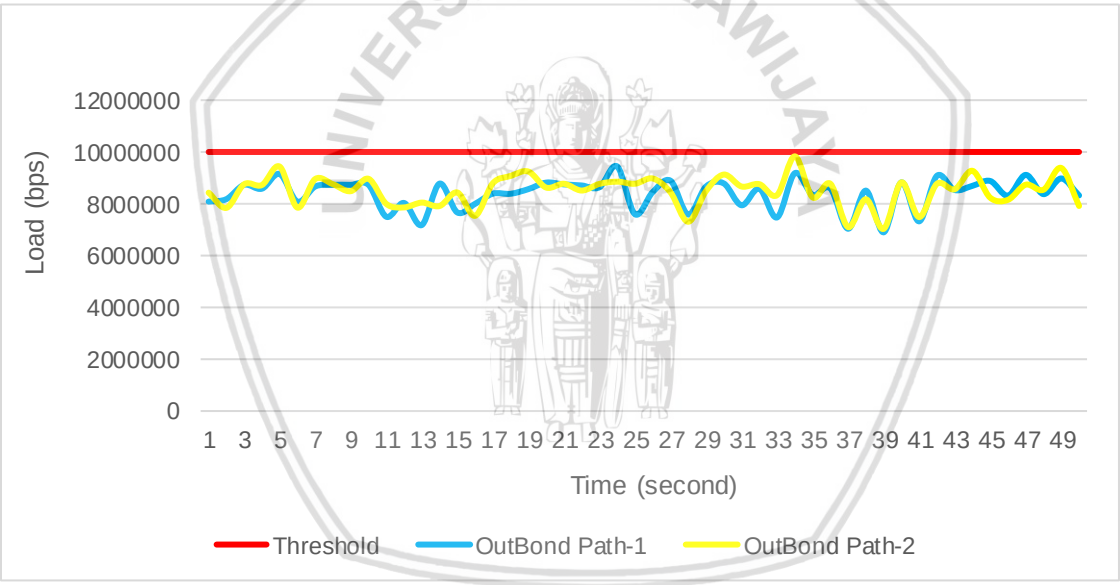
Hasil pengamatan *router* menggunakan *bwm-ng* disimpan dalam bentuk *log file csv* yang kemudian data tersebut akan ditransformasikan ke dalam bentuk tabel untuk diubah dalam bentuk grafik seperti yang terlihat pada Gambar 5-4 sampai Gambar 5-7. Grafik yang ditampilkan hanya dari sisi Router-1, sebagai *router* yang mengimplementasikan ECMP. Pengamatan dilakukan pada *interface* eth-1 dan eth-2 yang berperan sebagai *outbond interface* selama pengujian sistem.

Hasil *log file bwm-ng* dilakukan selama kurun waktu pengujian 50s, 100s, 300s dan 600s. Setiap percobaan yang dilakukan akan menghasilkan dua buah grafik yang menunjukkan hasil pengamatan *outbond path* di Router-1 pada masing-masing routing. Garis berwarna merah menunjukkan nilai *threshold* yang dikonfigurasikan pada setiap *interface router*. Garis berwarna biru merupakan beban jaringan yang dilewatkan pada *outbond path 1* (eth-1), dan garis berwarna kuning merupakan beban jaringan yang dilewatkan pada *outbond path 2* (eth-2).

Gambar 5-4(a) menunjukkan hasil pengamatan terhadap Router-1 selama kurun waktu 50s pada OSPF Single-Path. Dari hasil grafik, terlihat jelas bahwa trafik hanya dilewatkan pada *Path-1* saja, sedangkan pada *Path-2* tidak didapati adanya aktifitas pengiriman data. Sehingga, trafik yang dikirimkan hanya berasal dari *interface* eth-1 saja. Sedangkan pada Gambar 5-4(b), ditunjukkan adanya aktifitas pengiriman data yang terjadi, baik pada *Path-1* maupun *Path*-2 pada OSPF Multi-Path. Grafik yang ditunjukkan pada Router-1 terlihat jelas bahwa terdapat keseimbangan trafik yang diterima pada *interface* eth-1 dan eth-2. Sehingga, trafik yang dikirimkan menuju ke PC-3, merupakan akumulasi dari *interface* eth-1 dan eth-2.



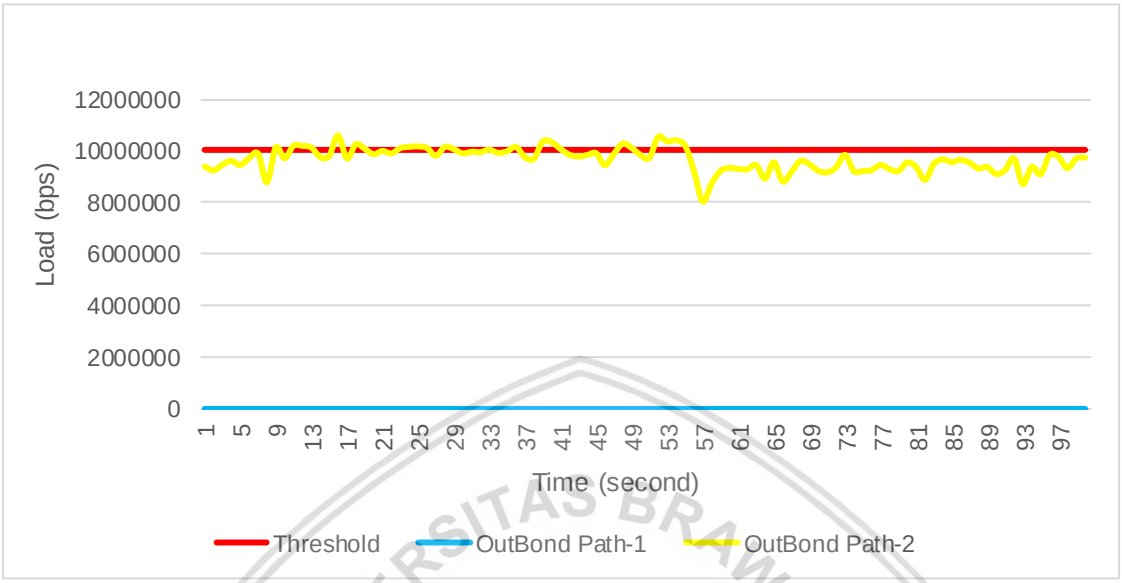
(a)



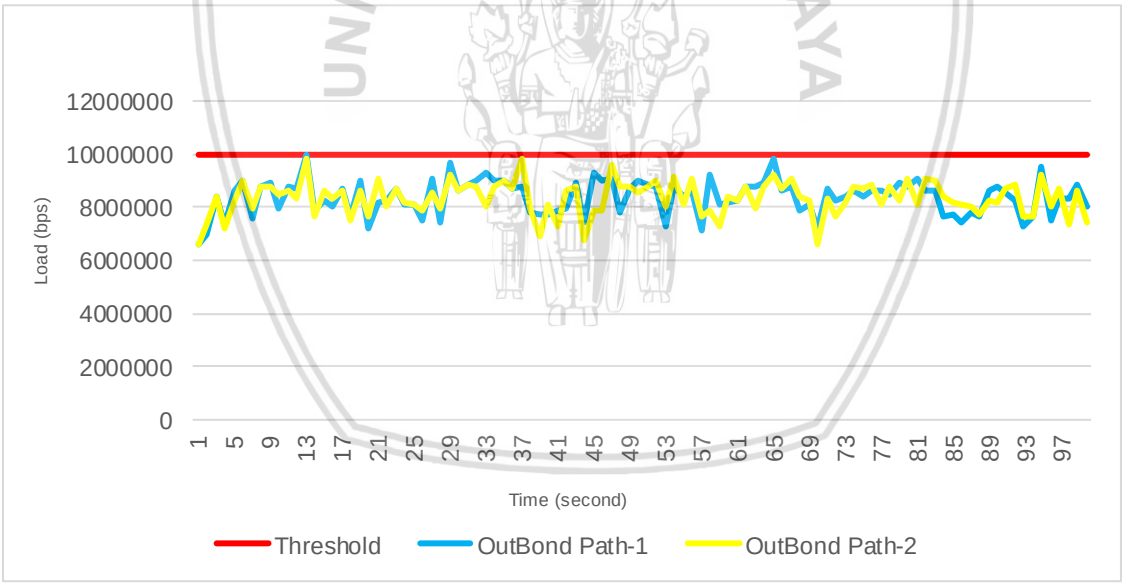
(b)

Gambar 1-4. Perbandingan Beban Trafik Hasil Log File Router-1 (Interval Waktu=50s)

Gambar 5-5(a) menunjukkan hasil pengamatan terhadap Router-1 selama kurun waktu 100s pada OSPF Single-Path. Seperti pada pengujian pertama, tidak ada trafik yang dilewatkan pada *Path*-2, sehingga PC-3 hanya dapat mentransmisikan trafik yang berasal dari Path-1 saja. Sedangkan pada Gambar 5-5(b), terdapat aktifitas trafik yang masuk baik pada Path-1 dan Path-2 pada OSPF Multi-Path, sehingga Router-1 mentransmisikan trafik secara utuh (tanpa adanya *packet loss*), sesuai dengan hasil pengujian pada protokol UDP.



(a)

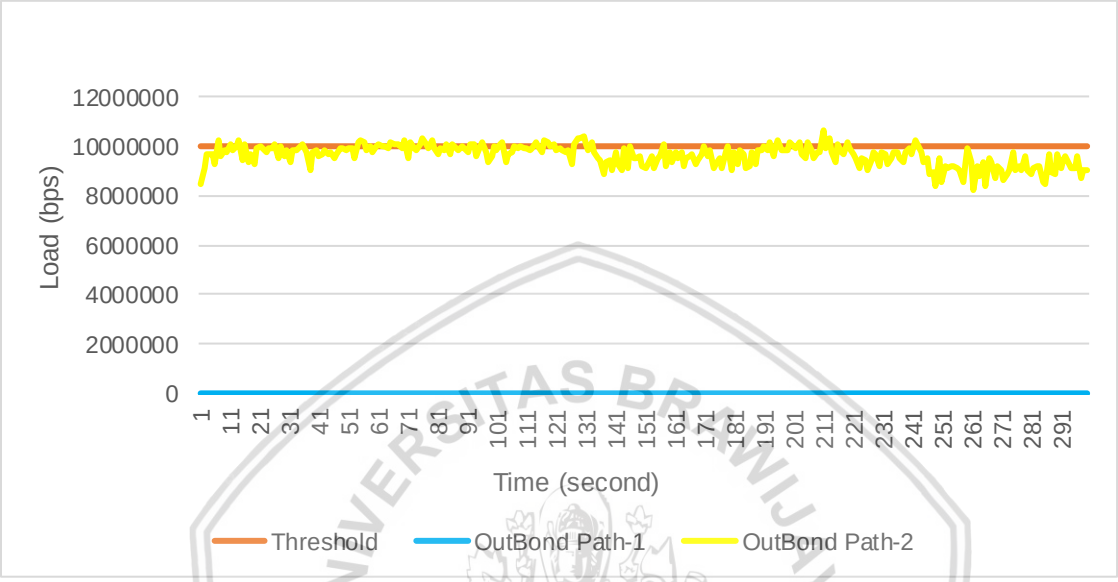


(b)

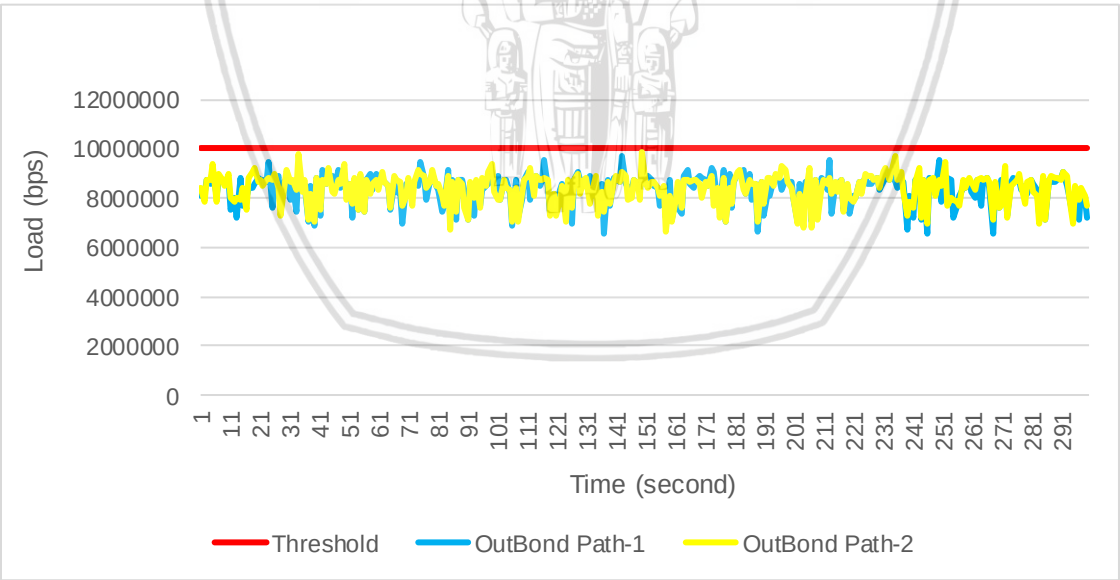
Gambar 1-5. Perbandingan Beban Trafik Hasil Log File Router-1 (Interval Waktu=100s)

Gambar 5-6(a) menunjukkan hasil pengamatan terhadap Router-1dengan kurun waktu lebih lama, yaitu 300s. Pada OSPF Single-Path, terlihat dari hasil grafik Path-1 menunjukkan nilai *throughput* yang tidak stabil, menunjukkan bahwa adanya *packet loss* yang terjadi, karena

Path-2 tidak menunjukkan aktifitas pengiriman trafik. Sedangkan pada Gambar 5-6(b), ditunjukkan adanya aktifitas pengiriman data yang terjadi, baik pada Path-1 maupun Path-2 pada OSPF Multi-Path. Grafik yang ditunjukkan pada Router-1 terlihat jelas bahwa terdapat keseimbangan trafik yang diterima pada *interface* eth-1 dan eth-2.



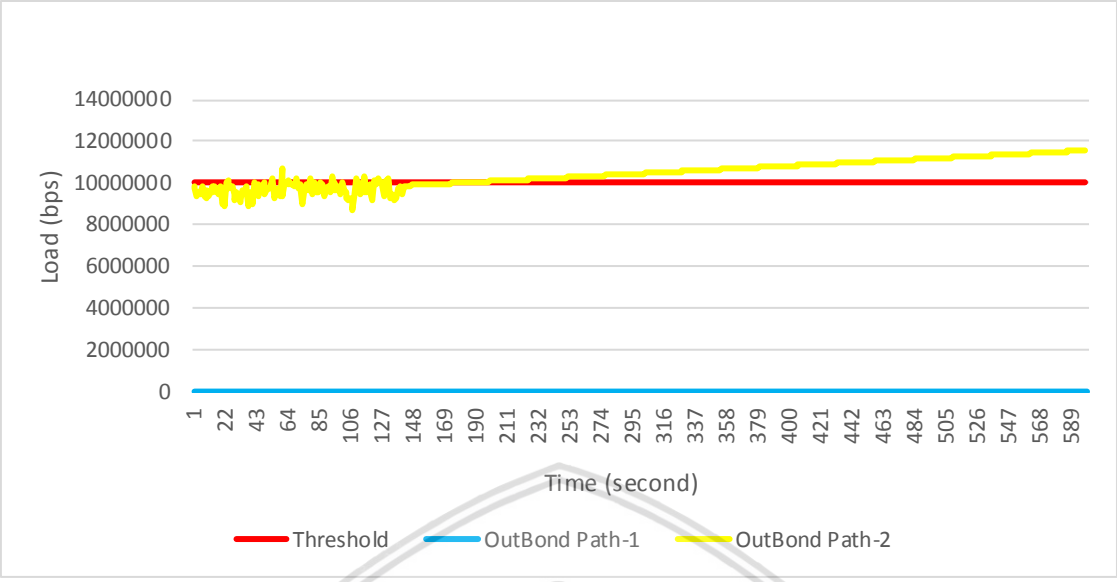
(a)



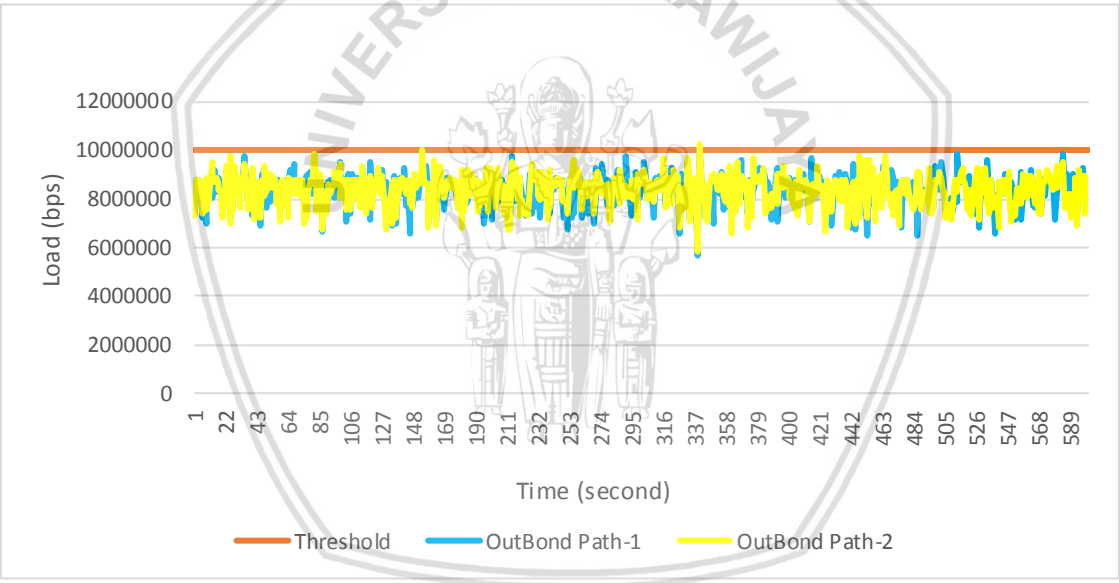
(b)

Gambar 1-6. Perbandingan Beban Trafik Hasil Log File Router-1 (Interval Waktu=300s)

Gambar 5-7(a) menunjukkan hasil pengamatan terhadap Router-1 selama kurun waktu 600s pada OSPF Single-Path. Hasil pengamatan yang dilakukan pada Path-1, terlihat bahwa trafik cenderung naik mulai dari detik ke-150 sampai detik ke-600, hingga melebihi nilai *threshold* yang telah ditetapkan. Namun sebaliknya, tidak ada trafik yang dilewatkan pada Path-2. Sebaliknya, Gambar 5-7(b) menyatakan hasil pengamatan pada OSPF Multi-Path menunjukkan keseimbangan trafik yang diterima pada *interface* eth-1 dan eth-2, tanpa melewati nilai *threshold* yang telah ditetapkan. Sehingga, trafik yang dikirimkan melalui kedua *outbond path* mengalami kestabilan dan tidak ada *packet loss* yang terjadi.



(a)



(b)

Gambar 1-7. Perbandingan Beban Trafik Hasil Log File Router-1 (Interval Waktu=600s)

BAB VI
KESIMPULAN DAN SARAN

1.1 Kesimpulan

Dalam penelitian ini, telah dibangun sebuah sistem routing adaptif melalui protokol routing OSPF, yang dapat melakukan deteksi dan pengendalian terhadap kemacetan jaringan. Pengujian sistem telah dilakukan melalui prosedur pengujian dan analisisnya diberikan berdasarkan nilai-nilai performa sebuah jaringan. Berdasarkan semua hasil dan pembahasan yang telah dijelaskan pada bab sebelumnya, maka dapat disimpulkan sebagai berikut :

1. Deteksi kemacetan jaringan dilakukan dengan mendefinisikan *threshold* di setiap *link* yang menghubungkan antar *router*. Kemacetan diidentifikasi berdasarkan jumlah trafik yang melebihi nilai *threshold* yang telah ditetapkan di setiap *outbond interface router*.
2. Modifikasi *routing cost metric* dilakukan pada *outbond interface router* dengan tingkat utilisasi *bandwidth* yang rendah. Pemodifikasian ini dilakukan dengan cara menyamakan nilai *cost metric* dengan jalur utama, yang bertujuan untuk membentuk jalur *multipath.*
3. Penggabungan antara mekanisme deteksi kemacetan dan teknik *multipath routing* menggunakan ECMP pada OSPF, terbukti efektif dalam mengurangi rasio kemacetan dan meningkatkan performansi jaringan. Hal ini dicapai dengan mengendalikan kemacetan jaringan melalui pengalihan jalur dan pemerataan beban pada jaringan. ECMP membantu mendefinisikan beberapa jalur yang memiliki nilai *cost* yang sama, sedangkan Dijsktra berperan dalam melakukan komputasi jalur yang terpendek.
4. Pengujian dilakukan dengan mengirimkan 4 alirantrafik sekaligus secara bersamaan melalui protokol TCP dan UDP. Analisis pengujian dilakukan melalui komparasi performa antara OSPF konvensional dan OSPF Multi-Path, selama kurun waktu 50s, 100s, 300s dan 600s untuk masing-masing *routing* menggunakan *iperf* dan *bwm-ng*.
5. Hasil pengujian menunjukkan pada OSPF konvensional, terjadi penurunan *throughput* hingga 50% (2,4Mbps), sehingga menyebabkan terjadinya *packet loss* hingga 39%. Sedangkan pada OSPF *Multipath*, nilai *throughput* yang dihasilkan adalah sesuai dengan pengujian alirantrafik yang dikirim (4Mbps) sehingga tidak terjadi adanya *packet loss (0%)*.

Dengan nilai *throughput* yang konstan, menjadikan OSPF *Multipath* memiliki waktu pendistribusian data yang lebih cepat, dengan nilai jitter mencapai 0,3ms, atau sepertiga dari waktu yang dihasilkan pada OSPF konvensional (1,02ms).

1.2 Saran

Perlu adanya pengembangan penelitian kemacetan dengan sistem kontrol terpusat dengan menggunakan *agent* pada router, sehingga saat dideteksi adanya kemacetan pada salah satu *node*, *controller* akan menginformasikan kepada *agent* untuk mengaktifkan mekanisme pengontrolan trafik . Sistem kontrol terpusat yang dibangun dapat mengadopsi konsep yang digunakan pada penelitian ini, untuk dibuat secara *programmable*, sehingga pendefinisian nilai *cost* dapat dilakukan secara dinamis berdasarkan utilisasi *bandwidth* yang dideteksi.

DAFTAR PUSTAKA

Aprian, R. and Novandi, D. (2007) 'Studi Implementasi Algoritma Dijkstra Pada Protokol Perutean Open Shortest Path First (OSPF)', *Teknik Informatika ITB*.

ArchLinux (2017) *Advanced traffic control*. Available at: https://wiki.archlinux.org/index.php/advanced_traffic_control (Accessed: 11 November 2017).

Balchunas, A. (2007) *Static vs . Dynamic Routing*. Available at: www.routeralley.com/guides/static_dynamic_routing.pdf (Accessed: 30 July 2015).

Borovina, N. and Kreso, S. (2005) 'OSPF-Based Model of Adaptive Routing and Possibility for Stable Network Operations', in *Proceedings of the 5th WSEAS International COnference on Applied Informatics ang COmmunications*. USA: World Scientific and Engineering Academy and Society (WSEAS), pp. 104–109.

Bowden, M., Crawford, M. and Limit, W. R. (2006) 'Rate Limiting in Linux'.

Chiesa, M., Kindler, G. and Schapira, M. (2014) 'Traffic Engineering with {ECMP}: An Algorithmic Perspective', *Proc. IEEE INFOCOM*, 25(2), pp. 1590–1598. doi: 10.1109/INFOCOM.2014.6848095.

Cisco Systems (2005) 'Jitter', pp. 50–60.

Cohen, D. (2014) 'HTB Linux Queuing Discipline Manual', pp. 1–10.

D. Papadimitriou, E. (2011) *Open Research Issues in Internet Congestion Control*, *Internet Research Task Force (IRTF)*. Available at: https://tools.ietf.org/html/rfc6077 (Accessed: 20 October 2015).

Devetak, F. and Kapoor, S. (2007) *Dynamic Multipath Routing with Congestion Control*.

Devetak, F. and Kapoor, S. (2017) *Dynamic MultiPath Routing*, *Internet Research Task Force (IRTF)*. Available at: https://tools.ietf.org/html/draft-kapoor-rtgwg-dynamic-multipath-routing-00 (Accessed: 30 October 2017).

Fang, S. *et al.* (2013) 'A Loss-Free Multipathing Solution for Data Center Network Using Software-Defined Networking Approach', 49(6), pp. 2723–2730.

Flannagan, M. E. *et al.* (2001) *Administering Cisco QoS in IP Networks: Including CallManager 3.0, QoS, and uOne*. United States of America: Syngress Publishing, Inc. Available at: https://books.google.com/books?id=K65BSbGf4qsC&pgis=1.

Floyd, S. and Jacobson, V. (1993) 'Random Early Detection ({RED}) Gateways for Congestion Avoidance', *IEEE/ACM Transactions on Networking*, 1(4), pp. 397–413.

G. Choudhury, E. (2005) *Prioritized Treatment of Specific OSPF Version 2 Packets and Congestion Avoidance, Internet Research Task Force (IRTF)*. Available at: https://tools.ietf.org/html/rfc4222 (Accessed: 10 August 2015).

Ishiguro, K. (2011) 'Quagga, a routing software package for TCP/IP networks', (September).

Jose, S. (2016) 'Thresholding Configuration Guide , StarOS Release 20', (6387).

Khosroshahy, M. (2009) *Congestion avoidance and control, Tech. Report*. doi: 10.1145/52325.52356.

Kurose, J. . and Ross, K. . (2013) *A Top-Down Approach 6th Edition*. United States: PEarson Education, Inc.

Kushwaha, V. and Gupta, R. (2014) 'Congestion Control for High-Speed Wired Network: A Systematic Literature Review', *Journal of Network and Computer Applications*, pp. 62–78.

Lady Silk Moonlight and Suhardi, S. (2012) 'Pengaruh Model Jaringan Terhadap Optimasi Routing Open Shortest Path First (Ospf)', *Teknologi*, 1(2), pp. 68–80. Available at: http://www.journal.unipdu.ac.id/index.php/teknologi/article/view/56.

Lappetel, A. (2011) 'Equal Cost Multipath Routing in IP Networks'.

Medhi, D. and Ramasamy, K. (2007) *Network Routing: Algorithms, Protocols, and Architectures*. San Fransisco: Elsevier.

Media, B. (2016) *Traffic Shaping, Bandwidth Shaping, Packet Shaping with Linux tc htb*. Available at: https://www.iplocation.net/traffic-control (Accessed: 8 November 2017).

Muslim, M. A. (2007) 'Analisis Codec dan Payload pada Micronet dan CISCO Pada Jaringan VPN - MPLS', XII(2), pp. 109–121.

Nakahodo, Y., Naito, T. and Oki, E. (2014) 'Implementation of Smart-OSPF in Hybrid Software-Defined Network', in *IEEE Proceedings of IC-NIDC 2014*, pp. 0–4.

Nugroho, G. A., Lamaida, A. S. and Ahsana, Y. (2006) 'Analisis Algoritma Pencarian Rute Terpendek Dengan Algoritma Dijkstra dan Bellman - Ford', *Teknik Informatika ITB*.

Peterson, L. L. and Davie, B. S. (2012) *Computer Networks : A Systems Approach*. 5th Editio. Elsevier, Inc.

Roy, A. (2006) 'Modification of Congestion Control Algorithm for TCP and Its Extension to Explicit Rate Adjusment Algorithm', *Indian Institute of Technology*, (May).

Solichin, R. and Oktoviana, M. Y. L. T. (2013) ‘Implementasi Algoritma Dijkstra dalam Pencarian Lintasan Terpendek Lokasi Rumah Sakit, Hotel dan Terminal Kota Malang’, pp. 1–7.

Stallings, W. (2007) *Data and Computer Communication (Eight Edition)*.

Susitaival, R. and Aalto, S. (2004) ‘Adaptive load balancing with OSPF’, (January 2004), pp. 1–10. Available at: http://books.google.com/books?hl=en&lr=&id=fy9gjdEeiVAC&oi=fnd&pg=PA85&dq=Adaptive+load+balancing+with+OSPF&ots=KHRvuyqSe2&sig=LQYWkMGJl-nhKIIwyZoBWICZgXw.

Telesis, A. (2005) ‘Open Shortest Path First (OSPF)’, *AR400 Series Router Software Reference*, p. Chapter 23.

Thomas, T. M. (2003) *OSPF network design solutions, Cisco Press Publications*. Edited by Cisco Systems Inc. Cisco Press. Available at: http://www.lavoisier.fr/livre/notice.asp?id=RASWLOAAO23OWF.

LAMPIRAN

Lampiran 1. File Konfigurasi Limitasi Interface pada Router-1

```
#!/bin/bash
# THIS FILE IS ADDED FOR COMPATIBILITY PURPOSES
#
# It is highly advisable to create own systemd services or udev
rules
# to run scripts during boot instead of using this file.
#
# In contrast to previous versions due to parallel execution
during boot
# this script will NOT be run after all other services.
#
# Please note that you must run 'chmod +x /etc/rc.d/rc.local' to
ensure
# that this script will be executed during boot.

touch /var/lock/subsys/local

tc qdisc add dev eth1 root handle 1:0 htb
tc class add dev eth1 parent 1:0 classid 1:1 htb rate 10Mbit
tc filter add dev eth1 protokol ip parent 1:0 prio 0 u32 match
ip dst 192.168.3.0/24 flowid 1:1

tc qdisc add dev eth2 root handle 1:0 htb
tc class add dev eth2 parent 1:0 classid 1:1 htb rate 10Mbit
tc filter add dev eth2 protokol ip parent 1:0 prio 0 u32 match
ip dst 192.168.3.0/24 flowid 1:1
```

Lampiran 2. File Konfigurasi zebra.conf pada OSPF Multi-Path (Router-1)

```
!
! Zebra configuration saved from vty
!   2018/01/08 19:37:56
!
hostname router-1
!
interface eth0
 description to-LAN
 ip address 192.168.1.1/24
 ipv6 nd suppress-ra
!
interface eth1
 description to-R2
 ip address 192.168.100.1/24
 ipv6 nd suppress-ra
!
interface eth2
 description to-R3
 ip address 192.168.101.1/24
 ipv6 nd suppress-ra
!
interface eth3
 ipv6 nd suppress-ra
!
interface lo
!
ip forwarding
!
!
line vty
```

Lampiran 3. File Konfigurasi ospfd.conf pada OSPF Multi-Path (Router-1)

```
!
! Zebra configuration saved from vty
!   2018/01/08 19:37:56
!
hostname ospfd@router-1
password zebra
log stdout
!
!
!
interface eth0
 description to-LAN
!
interface eth1
 description to-R2
 ip ospf cost 5
!
interface eth2
 description to-R3
!
interface eth3
!
interface lo
!
router ospf
 ospf router-id 192.168.1.1
 network 192.168.1.0/24 area 0.0.0.0
 network 192.168.100.0/24 area 0.0.0.0
 network 192.168.101.0/24 area 0.0.0.0
!
line vty
```

Lampiran 4. Hasil Konfigurasi Quagga pada OSPF Multi-Path (Router-1)

```
Building configuration...

Current configuration:
!
hostname router-1
hostname ospfd@router-1
log stdout
!
password zebra
!
interface eth0
 description to-LAN
 ip address 192.168.1.1/24
 ipv6 nd suppress-ra
!
interface eth1
 description to-R2
 ip address 192.168.100.1/24
 ip ospf cost 5
 ipv6 nd suppress-ra
!
interface eth2
 description to-R3
 ip address 192.168.101.1/24
 ipv6 nd suppress-ra
!
interface eth3
 ipv6 nd suppress-ra
!
interface lo
!
router ospf
 ospf router-id 192.168.1.1
 network 192.168.1.0/24 area 0.0.0.0
 network 192.168.100.0/24 area 0.0.0.0
 network 192.168.101.0/24 area 0.0.0.0
!
ip forwarding
!
line vty
!
end
```

Lampiran 5. File Konfigurasi Limitasi Interface pada Router-2

```
#!/bin/bash
# THIS FILE IS ADDED FOR COMPATIBILITY PURPOSES
#
# It is highly advisable to create own systemd services or udev
rules
# to run scripts during boot instead of using this file.
#
# In contrast to previous versions due to parallel execution during
boot
# this script will NOT be run after all other services.
#
# Please note that you must run 'chmod +x /etc/rc.d/rc.local' to
ensure
# that this script will be executed during boot.

touch /var/lock/subsys/local

tc qdisc add dev eth1 root handle 1:0 htb
tc class add dev eth1 parent 1:0 classid 1:1 htb rate 10Mbit
tc filter add dev eth1 protokol ip parent 1:0 prio 0 u32 match ip
dst 192.168.1.10 flowid 1:1

tc qdisc add dev eth2 root handle 1:0 htb
tc class add dev eth1 parent 1:0 classid 1:1 htb rate 10Mbit
tc filter add dev eth1 protokol ip parent 1:0 prio 0 u32 match ip
dst 192.168.3.30 flowid 1:1
```

Lampiran 6. File Konfigurasi zebra.conf pada OSPF Multi-Path (Router-2)

```
!
! Zebra configuration saved from vty
!   2018/01/05 09:34:15
!
hostname router-1
!
interface eth0
 description to-LAN
 ip address 192.168.2.1/24
 ipv6 nd suppress-ra
!
interface eth1
 description to-R1
 ip address 192.168.100.2/24
 ipv6 nd suppress-ra
!
interface eth2
 description to-R3
 ip address 192.168.102.2/24
 ipv6 nd suppress-ra
!
interface eth3
 ipv6 nd suppress-ra
!
interface lo
!
ip forwarding
!
!
line vty
```

Lampiran 7. File Konfigurasi ospfd.conf pada OSPF Multi-Path (Router-2)

```
!
! Zebra configuration saved from vty
!   2018/01/05 09:34:16
!
hostname ospfd@router-1
password zebra
log stdout
!
!
!
interface eth0
 description to-LAN
!
interface eth1
 description to-R1
 ip ospf cost 5
!
interface eth2
 description to-R3
 ip ospf cost 5
!
interface eth3
!
interface lo
!
router ospf
 ospf router-id 192.168.2.1
 network 192.168.2.0/24 area 0.0.0.0
 network 192.168.100.0/24 area 0.0.0.0
 network 192.168.102.0/24 area 0.0.0.0
!
line vty
```

Lampiran 8. Hasil Konfigurasi Quagga pada OSPF Multi-Path (Router-2)

```
Building configuration...

Current configuration:
!
hostname router-1
hostname ospfd@router-1
log stdout
!
password zebra
!
interface eth0
 description to-LAN
 ip address 192.168.2.1/24
 ipv6 nd suppress-ra
!
interface eth1
 description to-R1
 ip address 192.168.100.2/24
 ip ospf cost 5
 ipv6 nd suppress-ra
!
interface eth2
 description to-R3
 ip address 192.168.102.2/24
 ip ospf cost 5
 ipv6 nd suppress-ra
!
interface eth3
 ipv6 nd suppress-ra
!
interface lo
!
router ospf
 ospf router-id 192.168.2.1
 network 192.168.2.0/24 area 0.0.0.0
 network 192.168.100.0/24 area 0.0.0.0
 network 192.168.102.0/24 area 0.0.0.0
!
ip forwarding
!
line vty
!
end
```

Lampiran 9. File Konfigurasi Limitasi Interface pada Router-3

```
#!/bin/bash
# THIS FILE IS ADDED FOR COMPATIBILITY PURPOSES
#
# It is highly advisable to create own systemd services or udev
rules
# to run scripts during boot instead of using this file.
#
# In contrast to previous versions due to parallel execution
during boot
# this script will NOT be run after all other services.
#
# Please note that you must run 'chmod +x /etc/rc.d/rc.local' to
ensure
# that this script will be executed during boot.

touch /var/lock/subsys/local

tc qdisc add dev eth1 root handle 1:0 htb
tc class add dev eth1 parent 1:0 classid 1:1 htb rate 10Mbit
tc filter add dev eth1 protokol ip parent 1:0 prio 0 u32 match
ip dst 192.168.1.0/24 flowid 1:1

tc qdisc add dev eth2 root handle 1:0 htb
tc class add dev eth2 parent 1:0 classid 1:1 htb rate 10Mbit
tc filter add dev eth2 protokol ip parent 1:0 prio 0 u32 match
ip dst 192.168.1.0/24 flowid 1:1
```

Lampiran 10. File Konfigurasi zebra.conf pada OSPF Multi-Path (Router-3)

```
!
! Zebra configuration saved from vty
!   2018/01/05 09:36:23
!
hostname router-3
!
interface eth0
 description to-LAN
 ip address 192.168.3.1/24
 ipv6 nd suppress-ra
!
interface eth1
 description to-R1
 ip address 192.168.101.3/24
 ipv6 nd suppress-ra
!
interface eth2
 description to-R2
 ip address 192.168.102.3/24
 ipv6 nd suppress-ra
!
interface eth3
 ipv6 nd suppress-ra
!
interface lo
!
interface teql0
 ipv6 nd suppress-ra
!
ip forwarding
!
!
line vty
```

Lampiran 11. File Konfigurasi ospfd.conf pada OSPF Multi-Path (Router-3)

```
!
! Zebra configuration saved from vty
!   2018/01/05 09:36:23
!
hostname ospfd@router-3
password zebra
log stdout
!
!
!
interface eth0
 description to-LAN
!
interface eth1
 description to-R1
!
interface eth2
 description to-R2
 ip ospf cost 5
!
interface eth3
!
interface lo
!
interface teql0
!
router ospf
 ospf router-id 192.168.3.1
 network 192.168.3.0/24 area 0.0.0.0
 network 192.168.101.0/24 area 0.0.0.0
 network 192.168.102.0/24 area 0.0.0.0
!
line vty
```

Lampiran 12. Hasil Konfigurasi Quagga pada OSPF Multi-Path (Router-3)

```
Building configuration...

Current configuration:
!
hostname router-3
hostname ospfd@router-3
log stdout
!
password zebra
!
interface eth0
 description to-LAN
 ip address 192.168.3.1/24
 ipv6 nd suppress-ra
!
interface eth1
 description to-R1
 ip address 192.168.101.3/24
 ipv6 nd suppress-ra
!
interface eth2
 description to-R2
 ip address 192.168.102.3/24
 ip ospf cost 5
 ipv6 nd suppress-ra
!
interface eth3
 ipv6 nd suppress-ra
!
interface lo
!
interface teql0
 ipv6 nd suppress-ra
!
router ospf
 ospf router-id 192.168.3.1
 network 192.168.3.0/24 area 0.0.0.0
 network 192.168.101.0/24 area 0.0.0.0
 network 192.168.102.0/24 area 0.0.0.0
!
ip forwarding
!
line vty
!end
```