BAB II

LANDASAN TEORI

2.1. Routing Protocol

Menurut RFC 791 mengenai *Internet Protocol*, *routing protocol*merupakan protokol yang menspesifikasikan cara sebuah *router* melakukan komunikasi dengan *router* lain. Dengan menggunakan *routing protocol*, *router* menyebarkan informasi mengenai rute yang dimilikinya sehingga memungkinkan komunikasi 2 *node* pada sebuah sistem jaringan. Pemilihan rute dilakukan menggunakan algoritma dari masing-masing*routing protocol*.

Setiap *router* mempunyai informasi mengenai jaringan yang terhubung langsung dengannya.*Routing protocol* menyebarkan informasi ini ke *neighbor* terdekat terlebih dahulu sampai ke seluruh jaringan. Ada 2 kategori utama *routing protocol* yang digunakan saat ini, yaitu *distance-vector* dan *link-state routing protocol*.

Distance vector routing protocol (RFC 1058) merupakan salah satu dari dua jenis *routing protocol* utama yang digunakan saat ini. *Distance vector routing protocol* menggunakan algoritma Bellman-Ford, Ford-Fulkerson, atau DUAL FSM untuk menghitung dan menetukan rute yang akan digunakan. Sebuah *distance vector routing protocol* membutuhkan

agar *router* menginformasikan *neighbor* secara periodik ketika terjadi perubahan topologi pada jaringan. Pengertian *distance vector* mengartikan bahwa *router* menggunakan *vector* jarak untuk mencapai *router-router* lain. *Distance vector routing protocol* memiliki tingkat kompleksitas yang lebih sederhana dan memiliki *message overhead* yang lebih kecil. Beberapa contoh *distance vector routing protocol* antara lain: RIP, RIPv2, IGRP.

Link state routing protocol (RFC 3626) merupakan kelas utama *routing protocol* yang digunakan saat ini, bersama dengan *distance vector routing protocol*. *Link state protocol* dilakukan oleh setiap *router* yang akan mengirimkan paket data pada jaringan. Konsep dasar dari *link state routing protocol* adalah setiap *node* membentuk peta konektivitas jaringan berbentuk sebuah *graph* yang menggambarkan konektivitas antar *node*. Kemudian setiap *node* akan menghitung rute terbaik ke setiap *destination* pada jaringan. Apabila dibandingkan dengan *distance-vector routing protocol,* setiap *node* pada *link state routing protocol* hanya mengirimkan informasi yang berhubungan dengan kondisi jaringan.

2.2. OSPF

Open shortest path first/OSPF (RFC 2328, 5340) merupakan *routing protocol* pada jaringan IP yang menggunakan algoritma *link state routing* dan berada pada kelompok interior *routing protocol,* beroperasi

pada satu AS (*autonomous system*). Menurut (Cianfrani, Eramo, Listanti, Marazza, & Vittorini, An Energy Saving Routing Algorithm for a Green OSPF Protocol, 2011), OSPF merupakan intra-AS *routing protocol* yang paling banyak digunakan saat ini.

OSPF merupakan IGP (*Interior Gateway Protocol*) yang merutekan paket IP pada sebuah AS. OSPF mengambil informasi *link state* dari *router* pada jaringan dan membentuk peta topologi jaringan. Peta topologi yang dibentuk menentukan *routing table* yang akan digunakan oleh *internet layer* untuk menentukan *routing decision*. OSPF didesain untuk mendukung VLSM (*Variable Length Subnet Masking*) dan CIDR (*Classless Inter-Domain Routing*).

OSPF mendeteksi perubahan pada topologi jaringan (seperti misalnya gangguan pada *link*) secara cepat dan melakukan konvergensi peta *routing* yang bebas dari *loop* dalam hitungan detik. OSPF menghitung *shortest path tree* untuk setiap rute menggunakan metode yang didasarkan pada algoritma Dijkstra.Informasi *link-state* (yang biasa disebut LSA/*Link State Advertisement*) disimpan oleh setiap *router* sebagai *link-state database* yang diupdate secara periodik menggunakan informasi yang disebarkan oleh OSPF *router*.

OSPF membentuk *routing table* yang diatur berdasarkan *cost* dari setiap *interface*. *Cost* yang dimaksud dapat berupa *round trip time, link throughput, delay*,dst. OSPF versi 3 diperkenalkan pada tahun 2008 dan mendukung implementasi Ipv6 yang berjalan pada setiap *link,* bukan pada

subnet. Informasi IP *prefix*telah dihapus dari LSA dan *hello packet* sehingga OSPFv3 dapat dikatakan *routing protocol* yang independen.

2.3. Autonomous System (AS)

Pada sebuah jaringan internet, *autonomous system* (RFC 1771, 1930) merupakan kumpulan dari IP *routing prefix* yang diatur oleh satu atau lebih operator jaringan dan merepresentasikan *routing policy* internet yang umum dan jelas. Pada awalnya, AS harus dikelola oleh sebuah entitas besar (biasanya sebuah ISP atau perusahan besar dengan koneksi ke beberapa jaringan), akan tetapi dalam perkembangannya, perusahaan kemudian menjalankan BGP (*Border Gateway Protocol*) menggunakan AS number yang dimiliki oleh ISP untuk menghubungkan organisasi-organisasi tersebut ke internet(Chiaraviglio, Mellia, & Neri, Reducing Power Consumption in Backbone Networks, 2009) dan (Chiaraviglio, Mellia, & Neri, Energy-Aware Backbone Networks, 2009). Internet hanya melihat *routing policy* yang dimiliki oleh ISP, dan ISP harus memiliki AS number yang sudah terdaftar pada internet. (ASN)

Setiap AS mempunyai ASN yang unik untuk digunakan pada *routing* BGP.ASN merupakan komponen yang penting karena ASN mengidentifikasikan setiap jaringan secara unik dan individual pada internet.

Hingga tahun 2007, AS number didefinisikan sebagai *integer* 16 bit (sehingga jumlah maksimum AS number adalah 65536). Kemudian melalui RFP 4893, IANA (*Internet Assigned Numbers Authority*) memperkenalkan 32-bit AS number.RFC terbaru yang menjelaskan AS number terdapat pada RFC 5396. Beberapa AS number secara lengkap dapat dilihat pada situs IANA.

2.4. Algoritma Shortest Path Dijkstra

Algoritma Dijkstra ditemukan oleh peneliti komputer dari negara Belanda yang bernama Edsger Dijkstra pada tahun 1956.Algoritma ini merupakan algoritma pencarian *graph* yang menyelesaikan permasalahan *shortest path* untuk sebuah *graph* yang memiliki *link* dengan nilai non-negatif.

Hasil dari algoritma ini berupa *shortest path tree* yang mengambarkan rute terpendek dari 1 *node* menuju seluruh *node lainnya.* Algoritma ini banyak digunakan untuk menyelesaikan masalah *routing* pada jaringan dan masalah algoritma *graph* lainnya.

Diasumsikan terdapat sebuah *node* pada *graph*, algoritma *shortest path* akan mencari jalur/rute dengan *cost* terendah dari satu *node* ke seluruh *node* lainnya. Algoritma ini juga dapat digunakan untuk mencari *shortest path* dari satu *node* asal ke satu *node* tujuan dengan menghentikan algoritma ketika jalur *shortest path* menuju tujuan telah berhasil

diidentifikasi. Algoritma *shortest path* banyak digunakan pada *routing protocol* jaringan, khususnya untuk IS-IS dan OSPF.

Algoritma Dijkstra dapat dijabarkan sebagai berikut:

1. Diasumsikan bahwa *node* yang akan digunakan dinamakan *node* awal. Jarak *node* Y adalah jarak dari *node* awal ke Y.
2. Setiap *node* diberikan nilai jarak tentatif, dimana *node* awal akan diberi angka 0 dan *node* lain diberi angka tak terhingga.
3. Memberikan tanda pada setiap *node* bahwa node tersebut belum pernah dikunjungi (kecuali *node* awal). *Node* awal ditentukan sebagai *current*, kemudian membentuk himpunan *node* yang belum dikunjungi sebagai himpunan *unvisited node* (terdiri dari seluruh node kecuali *node* awal).
4. Untuk *node* awal, hitung jarak tentative ke seluruh *unvisited node*. Meskipun *node* awal telah menghitung jarak tentative ke setiap *unvisited node*, *unvisited node* belum diubah menjadi *visited node*.
5. Ketika telah selesai menghitung seluruh *neighbor* dari *node* awal, selanjutnya adalah menentukan himpunan *visited set*. Himpunan *visited node* merupakan kumpulan *node* yang sudah dikunjungi dan tidak akan diperiksa kembali.
6. Selanjutnya, *node* awal akan dipilih ulang berdasarkan *node* yang memiliki jarak paling kecil pada *unvisited set*.
7. Jika *unvisited set* sudah kosong, maka algoritma akan dihentikan (sudah selesai). Sebaliknya, apabila *unvisited set* belum kosong,

node dengan jarak paling kecil akan menjadi *node* awal dan langkah 3 akan diulangi.

Pseduocode algoritma ini adalah sebagai berikut:

1. **function** Dijkstra(*Graph*, *source*):
2. **for each** vertex *v* in *Graph*:
3. dist[*v*] := infinity ;
4. previous[*v*] := undefined ;
5. **end for** ;
6. dist[*source*] := 0 ;
7. *Q* := the set of all nodes in *Graph* ;
8. **while**_Q_**is not** empty:
9. *u* := vertex in *Q* with smallest distance in dist[] ;
10. **if** dist[*u*] = infinity:
11. **break** ;
12. **end if** ;
13. remove *u* from *Q* ;
14. **for each** neighbor *v* of *u*:
15. *alt* := dist[*u*] + dist_between(*u*, *v*) ;
16. **if**_alt_< dist[*v*]:
17. dist[*v*] := *alt* ;
18. previous[*v*] := *u* ;
19. decrease-key *v* in *Q*;
20. **end if** ;
21. **end for** ;
22. **end while** ;
23. **return** dist[] ;
24. **end** Dijkstra.

2.5. Router CPU Load

Pada dasarnya, *router* (RFC 1812) memiliki komposisi perangkat yang hampir sama dengan sebuah PC, *Router* memiliki *processor* untuk melakukan *processing* dan pengolahan data, *memory* (biasanya berbentuk *flash*) untuk menyimpan konfigurasi dan sistem operasi, UTP *ethernet port,* dst. Yang membedakan *router* dengan PC adalah *router* merupakan

perangkat jaringan yang dikhususkan untuk mengirim dan menerima paket Ipv4/Ipv6 dan merupakan perangkat utama telekomunikasi dan internet dunia.

Router memiliki beberapa metode *processing* data yang berbeda, *router* dapat melakukan *processing* data secara tersentralisasi maupun *slot-based processing*.*Router* CPU *load* merupakan tingkat utilisasi CPU pada sebuah *router*. Banyak hal yang mampu mempengaruhi *router* CPU *load,* diantaranya:

1. Kemampuan *forwarding* data
2. Kemampuan *switching*
3. Serangan eksternal, misalnya virus, *intrusion*
4. Gangguan pada *interface router*

Router dengan CPU *load* yang tinggi dapat mempengaruhi kinerja jaringan, seperti misalnya penambahan *delay* sewaktu melakukan *processing* paket data, paket data didrop oleh *router*,dst

2.6. Link utilization & throughput

Pada sebuah sistem jaringan, *link* atau *data link* merupakan hubungan data antara satu *node/router* ke *node* lain yang bertujuan untuk mengirim dan menerima informasi paket data. Ada setidaknya 3 jenis *data link* yang dapat digunakan:

1. *Simplex*

   Komunikasi yang berjalan 1 arah

2. *Half Duplex*

   Komunikasi dapat berjalan 2 arah, namun komunikasi 2 arah tidak bisa dilakukan secara bersamaan, misal: HT.

3. *Full Duplex*

   Komunikasi 2 arah secara bersamaan.

Pada saat ini, hampir seluruh komunikasi jaringan internet sudah menggunakan *full-duplex*.

Link utilization merupakan gambaran nilai utilisasi (biasanya berupa persentase) sebuah koneksi data antar 2 *node* pada satu periode waktu tertentu. Menurut(Cianfrani, Eramo, Listanti, Marazza, & Vittorini, An Energy Saving Routing Algorithm for a Green OSPF Protocol, 2011) dan (Bolla, Bruschi, Davoli, & Cucchietti, 2011), utilisasi *link* pada jaringan internet di waktu malam hari (*off-peak hour*) hanya berkisar diantara 10 sampai 20%.Sedangkan pada saat *peak-hour* utilisasi *traffic* jaringan bisa mencapai 60-70%.

Link throughput merupakan besarnya paket data yang berhasil dikirimkan melalui sebuah channel komunikasi. *Link throughput* biasanya merupakan ukuran *traffic* yang benar-benar melewati sebuah *link* pada jaringan. Biasanya *throughput* diukur dalam *bit per second* atau *packet per second.*

2.7. Bit Error Rate

Pada transmisi digital, *bit error* merupakan jumlah *bit* data yang diterima pada saat transmisi data, namun *bit* tersebut telah mengalami perubahan yang diakibatkan oleh *noise, interference, distortion,* atau *bit synchronization error*. *Bit error rate* merupakan persentasi jumlah *bit error* (jumlah *bit error* dibagi dengan jumlah total *bit* yang dikirimkan) pada sebuah komunikasi data. *Bit error rate* biasanya direpresentasikan menggunakan persentase. *Bit error probability* merupakan nilai ekspektasi dari *bit error rate*.

2.8. Network Delay

Network delay merupakan spesifikasi lama waktu yang dibutuhkan untuk mengirimkan *bit* data dari 1 *node* menuju ke *node* tujuan. Biasanya *delay* direpresentasikan dan diukur menggunakan perhitungan mili atau bahkan mikro second. *Delay* memiliki pemahaman dan konsep yang berbeda, tergantung dari lokasi *node* yang saling berkomunikasi. Beberapa bagian dari *delay* antara lain:

1. *Processing delay*

   Merupakan waktu yang dibutuhkan *router* untuk memproses *packet header*.

2. *Queuing delay*

   Merupakan waktu lama *packet* berada dalam urutan *routing*.

3. *Transmission delay*

   Waktu yang dibutuhkan untuk mengirimkan bit data dari paket menuju *link*.

4. *Propagation delay*

   Waktu yang dibutuhkan sebuah signal untuk mencapai tujuannya.

Ada beberapa level *delay minimum* yang mungkin dirasakan ketika melakukan transmisi data lewat jaringan (hal ini dikarenakan *router* membutuhkan waktu untuk mengirimkan *packet* ke sebuah *link*), dan apabila ditambahkan *delay* yang dirasakan karena *congestion* pada jaringan, *delay* pada jaringan IP dapat bervariasi antara beberapa milisecond sampai ratusan milisecond.

Menurut ITU (*International Telecomunication Union*), terdapat level *standard delay* untuk paket data suara. Adapun *standard delay* yang disarankan ITU dapat dilihat pada tabel berikut:

Table 2.1 Standard delay ITU

Range **(dalam ukuran milisecond)**	**Penjelasan**
0-150	*Delay* masih dalam taraf yang sewajarnya
150-400	*Delay* masih dalam kategori wajar, dengan catatan bahwa *network administrator*

	mengetahui pengaruhnya terhadap kualitas aplikasi yang dijalankan
Di atas 400	*Delay* sudah berada pada tingkat yang tidak disarankan, namun dapat digunakan pada kasus-kasus tertentu

2.9. Algoritma standard d-EAR

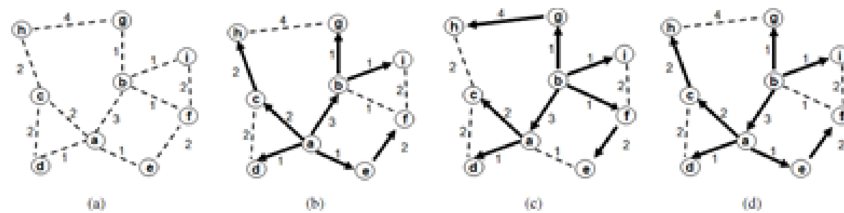
Referensi algoritma *standard* d-EAR didapat melalui jurnal (Cianfrani, Eramo, Listanti, Marazza, & Vittorini, An Energy Saving Routing Algorithm for a Green OSPF Protocol, 2011).

Tujuan dari algoritma *standard* d-EAR adalah meminimalisasi *active links* (jumlah *links* yang digunakan untuk merutekan *traffic*) pada jaringan. Tujuan ini dapat dipenuhi dengan memaksa sebagian *router* untuk menggunakan rute yang berbeda dengan rute pada SPTnya.Jenis *router* dibagi menjadi 3 yaitu *exporter*, *importer*, dan *neutral*.*Exporter router* (ER) diasosiasikan dengan beberapa *importer router* (IR), sedangkan 1 IR hanya diasosiasikan dengan 1 ER. Sebuah IR akan menghitung rutenya berdasarkan SPT yang telah dihitung ERnya. *Neutral router* bertindak sebagaimana routing protocol OSPF biasa.

Algoritma ini dapat dibagi menjadi 3 fase, dimana:

a. Fase 1 adalah fase “*ER Selection*”, dimana fase ini akandigunakan untuk menentukan ER.
b. Fase 2 adalah fase “*MPT Evaluation*”, dimana setiap IR menghitung MPTnya masing-masing dan menentukan link mana yang bisa dimatikan.
c. Fase 3, yaitu *routing path optimization*, adalah fase dimana setiap router melakukan evaluasi *routing path* pada topologi saat ini (yang didapat ketika menghilangkan link yang dimatikan pada fase sebelumnya)

Ketika menggunakan protocol OSPF, setiap *router* memiliki informasi penuh mengenai kondisi jaringan (berkat bantuan *database* LSA), sehingga setiap *router* dapat menjalankan algoritma EAR secara independen. Contoh aplikasi algoritma *standard* d-EAR dapat dilihat pada gambar 2.1 dibawah.



Gambar 2.1 Contoh Aplikasi algoritma *standard* d-EAR

Fase 1: *ER Selection*

a. Seperti yang telah dijelaskan sebelumnya, peran dari ER merupakan kunci dari algoritma EAR. ER menghitung SPT berdasarkan algoritma Dijkstra, *neighbor*dari ER akan menggunakan SPT dari ER untuk

memodifikasi *routing path* dan mengidentifikasi link yang akan dimatikan.

b. **Jumlah dari ER, yang diberi tanda R_e**, dan bagaimana cara dan kriteria pemilihannya akan sangat mempengaruhi performa dari algoritma EAR. Khususnya karena R_e sangat mempengaruhi jumlah link yang akan dimatikan sehingga nilai jumlah ER harus diperhitungkan dengan lebih matang untuk mencegah *active links* yang dimiliki menjadi *overloaded*.

c. Idealnya, ER dipilih untuk memaksimalkan jumlah *link* yang akan dimatikan, namun dengan **tetap memperhatikan load level di setiap *activelinks* dibawah dari batas yang telah ditentukan** untuk menjamin performa jaringan yang memuaskan. Hal ini sangat menyulitkan dan membutuhkan informasi mengenai *load* jaringan (yang belum dapat diberikan oleh *routingprotocol* saat ini).

d. Pada fase ini, setiap *router* membuat sebuah list yang dipilih dari LSA *database* berisi R_e*router* yang memiliki nilai dengan tingkatan tertinggi (*router* dengan jumlah *link* terbanyak/neighbor terbanyak). Apabila sebuah *router* sudah dipilih pada list ini, seluruh *neighborrouter* tersebut tidak bisa dianggap sebagai kandidat ER, sehingga mereka harus dihapus dari list (dengan kata lain *neighbor* dari sebuah ER **tidak bisa** menjadi sebuah ER).

e. Ide di balik prosedur diatas adalah, ketika sebuah *router* memiliki jumlah *neighbor* yang besar, *router* tersebut akan mengirimkan

SPTnya ke jumlah IR yang banyak juga, sehingga dapat meningkatkan kemungkinan untuk mematikan *link* dalam jumlah yang besar.

Fase 2: *MPTEvaluation*

a. Tujuan dari fase ini adalah mengidentifikasi/menentukan *link* yang akan dimatikan. Untuk mencapai tujuan ini, setiap IR harus melakukan modifikasi SPTnya untuk mendapat MPT.
b. Perhitungan jalur/path yang dilakukan setiap *router* untuk membentuk *routingtablenya* dipengaruhi oleh peran *router* pada jaringan:
    - Apabila *router* tersebut adalah ER atau bukan *neighbor* dari ER (NR), maka akan dilakukan algoritma dijkstra.
    - Sedangkan apabila *router* tersebut adalah IR, *router* harus melakukan perhitungan algoritma dijkstra yang telah dimodifikasi dengan menggunakan SPT yang dihitung oleh ER yang berhubungan.
c. Contoh kasus: Sebuah *router* IR (r_n) adalah *neighbor* dari r_e (ER). *Router* $r_n$akan menggunakan LSA *database* untuk melakukan eksekusi algoritma dijkstra dengan menganggap *router* r_e sebagai *rootnodenya*. Hasil dari operasi ini sama seperti SPT dari r_e (SPT(r_e)). SPT(r_e) dianggap rn sebagai *routingtreenya*.
d. Dengan kata lain, *routingtree* r_n secara topologi identik dengan SPT(r_e), namun dengan r_n sebagai *root* nodenya. Modifikasi ini dinamakan MPT(r_n,r_e). Untuk mencapai hasil ini, SPT(r_e) dapat dibagi menjadi 2 bagian, dimana:
    - Bagian dari *tree*/*subtree*, dengan r_n sebagai *rootnodenya*

- Sisa dari *tree*

Pada bagian pertama, MPT(r_n,r_e) identik dengan SPT(r_e), sedangkan pada bagian kedua, r_n akan memaksakan diri sebagai *rootnode* dan re sebagai *nexthoprouter* (mengganti arah *link* dari r_e->r_n menjadi r_e<- r_n).

e. Harus dijadikan catatan, bahwa perbedaan utama antara EAR dengan algoritma OSPF yang biasa digunakan terletak pada **IR yang menghitung SPT dimana *rootnodenya* adalah ER yang bersangkutan**, sehingga tingkat kompleksitasnya tidak jauh berbeda. Contoh perhitungan dapat dilihat pada gambar 2.1 diatas, dimana pada gambar 1(a), terdapat contoh *graph* jaringan (untuk kemudahan, diasumsikan kedua link unidirectional/1-arah yang dimiliki sepasang *router* memiliki weight yang sama). SPT yang dihitung menggunakan algoritma dijkstra classic oleh *router* a dan *router* b terdapat pada gambar 1(b) dan 1(c).

f. Apabila diasumsikan *router* a berperan sebagai ER, dan *router* b sebagai IR dari *router* a, *node* b akan melakukan fase evaluasi MPT. Pertama-tama *node* b akan melakukan komputasi SPT(a), dan melakukan modifikasi dengan menjadikan dirinya sebagai *rootnode* pada SPT(a) yang telah dihitung. Hasilnya adalah MPT(b,a) yang terdapat pada gambar 1(d). Jika dibandingkan antara solusi normal SPT(b) dengan solusi yang diujikan MPT(b,a), terdapat 1 *link* yang dapat dinonaktifkan yaitu *link* (b,f)

g. Setelah MPT telah berhasil dievaluasi secara keseluruhan, setiap IR dapat mematikan *links* yang tidak muncul pada MPT yang telah dihitung.

h. Aturan pengoperasian pada fase ini dapat disimpulkan menjadi pengamatan sbb:

   - Masing-masing *router* memiliki cara yang berbeda dalam melakukan evaluasi *routingpath* (ER/NR, IR).
   - ER dan NR melakukan perhitungan *minimumcostroutingpath*
   - Akan tetapi, *rute* yang dihitung pada fase kedua tidak bisa secara langsung diupdate pada *routingtable*, karena perhitungan ini tidak menjamin adanya sebuah looping. Secara gabungan, MPT yang dievaluasi oleh IR hanya ditujukan untuk menentukan *link* yang akan dimatikan, namun bukan merupakan strategi *routing* yang sebenarnya. Berdasarkan konsekuensi2 tersebut, EAR membutuhkan fase ketiga yang bertujuan untuk mencegah *looping* dan mendapat rute teroptimisasi yang dibasiskan pada operasi OSPF tambahan.

Fase 3: Routing path optimization

a. Seperti yang dijelaskan pada akhir fase 2 diatas, setiap IR telah mengidentifikasi link yang harus dimatikan. *Link-link* ini harus dihapus dari topologi supaya setiap *router* dapat menghitung *routingpath* yang sebenarnya dan melakukan update *routingtablenya* masing-masing.

b. Untuk mencapai tujuan ini, IR dengan minimal 1 *link* yang akan dimatikan harus melakukan update informasi topologi terupdate dengan menggunakan aturan pada *routingprotocol* OSPF (dengan menggunakan LSA *message*). Prosedur ini memungkinkan *router* untuk mengetahui topologi yang terupdate, *router*akan menghilangkan *link* yang telah dimatikan dan melakukan update*database* topologi.

c. Ketika prosedur diatas telah dilakukan dengan baik, setiap *router*akan melakukan perhitungan *routingpath* teroptimasi menggunakan algoritma dijkstra pada topologi yang telah terupdate (topologi yang *linknya* sudah dimatikan dan masing-masing *router* sudah melakukan update LSA *database*). Solusi ini memastikan setiap *router* melakukan perhitungan *minimumcost* berdasarkan referensi topologi yang sama (sekaligus menghilangkan kemungkinan looping).

2.10. Algoritma d-EAR *Max_Compatibility*

Ide dasar strategi perhitungan *path* yang dilakukan EAR adalah ***link* yang digunakan untuk merutekan traffic dapat dikurangi apabila hanya sebagian SPT saja yang digunakan, bukan keseluruhan SPT yang dihitung masing-masing *router* seperti pada kasus OSPF normal**(Cianfrani, Eramo, Listanti, & Polverini, An OSPF Enhancement for Energy Saving in IP Networks, 2011). Namun harus diperhatikan, dengan mengasumsikan bahwa setiap *router* memiliki pandangan yang

samamengenai topologi jaringan pada *database* LSAnya masing-masing, setiap *router* mampu memperhitungkan SPTnya sendiri dengan memanfaatkan algoritma Dijkstra. Di lain pihak, operasi EAR didasarkan pada konsep "SPT *exportation*", dimana sebagian *router*(yang dikatakan "*exporter*") akan memaksa *router* lain (yang disebut dengan "*importer*") untuk menggunakan SPTnya ketika *router* importer ini melakukan perhitungan path.

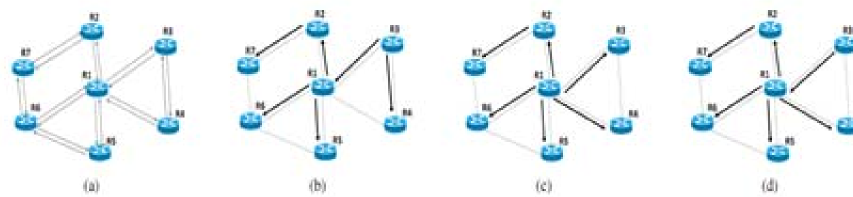
Peran dari *router* (*exporter*/importer) akan diberikan berdasarkan kumpulan *link* yang dinamakan *target links,* yaitu *link* yang menjadi kandidat untuk dinonaktifkan. Pendekatan seperti ini memungkinkan operator untuk dapat mengontrol performa jaringan dengan lebih baik dan memungkinkan implementasi strategi OSPF yang lebih *advanced.*

Diasumsikan T adalah kumpulan *set* dari target *link* (yaitu *link* yang menjadi kandidat untuk dimatikan). Untuk target *link* l ∈ T, metode *exportation* harus diidentifikasikan secara spesifik (didefinisikan *exporter* dan *importer* router yang akan dimanfaatkan untuk mematikan link). **Importer *router*adalah *router*Ri** , sebagai *source* dari *link* l. Hal ini sangat jelas, karena apabila target *link* l akan dinonaktifkan maka jalur *router* Ri harus dimodifikasi agar *link* l tidak digunakan oleh *router* Ri ini lagi. Identifikasi untuk *exporterrouter* lebih kompleks, sebuah kandidat untuk menjadi *exporterrouter* pada *link* l harus memenuhi 2 syarat berikut ini:

1. *Link* l tidak harus berada pada SPT *exporterrouter* ini.

2. *Exporterrouter* harus merupakan salah satu neighbors dari Ri/*importerrouter*

Alasan pertama cukup jelas, sedangkan alasan kedua digunakan untuk meminimalisasi *hop*/*path* lengths yang berjauhan.(Apabila *exporterrouter* bukan merupakan *neighbor* dari *importer*, maka bisa saja terjadi *hop* yang sangat jauh antara exporter/*importer)*. Berikut ini akan dijabarkan contoh implementasi dan cara kerja algoritma EAR dengan mengangap M(l) adalah kumpulan/set dari *exportation* yang harus dilakukan untuk mematikan *link* l.



Gambar 2.2 Contoh implementasi d-EAR 2

Gambar 2.2.a. diatas merupakan gambar topologi jaringan sederhana yang terdiri dari 7 *router* dan 18 *directlink*. Untuk kemudahan, setiap link memiliki OSPF *weight* yang sama. Apabila diasumsikan bahwa target link adalah (R3,R4). Router R3 adalah *importerrouter* (karena target *link* berasal dari R3), sedangkan *router* R1 dapat menjadi pilihan kandidat untuk exporter router karena R1 adalah *neighbor* dari R3 dan target *link* (R3,R4) tidak berada pada SPT dari R1 (gambar 2.2.c). MPT dari R3 yang dihasilkan dari mekanisme exportasi terlihat pada gambar 2.2.d. dan target *link* (R3,R4) tidak digunakan pada MPT ini dan dapat dinonaktifkan

Namun harus diperhatikan, bahwa kumpulan/set M(l) bisa saja menjadi kosong atau dengan kata lain, ada kemungkinan perhitungan

algoritma EAR tidak dapat menemukan *exporterrouter* yang mampu menonaktifkan target *link*. Hal ini dapat disebabkan oleh 2 alasan:

a. Tidak ada *neighborimporter* (kandidat *exporterrouter*) yang tidak memiliki target *link* dalam SPTnya (syarat 1/seluruh kandidat *exporterrouter* memiliki target *link* dalam SPTnya, sehingga target tidak bisa dinonaktifkan).
b. Mekanisme exportasi yang dilakukan sebelumnya telah membatasi langkah *exportasi* yang dapat dilakukan/dieksekusi selanjutnya.

Konsep Move dan Propertiesnya

Sebuah *network* dapat digambarkan dengan menggunakan *weighteddirectedgraph* (*graph* berarah yang memiliki beban di masing-masing *edge*) yang disimbolkan dengan G(V,E) dimana V adalah kumpulan dari *nodes* (yang menggambarkan *router*) dan E adalah set dari *directededges* (yang menggambarkan *link*). Untuk setiap directed *edge*/link berarah $e \in E$, S(e), E(e), dan w(e) merupakan simbol untuk *sourcenode*, *endnode*, dan *weight* dari *link*. N dan L akan dianggap sebagai *cardinalities* (jumlah elemen) dari V dan E ($N = \| V \|$ dan $L = \|E\|$), atau dengan kata lain N adalah jumlah *router* dan L adalah jumlah *link*. Set dari semua *shortestpath* dari *node* v ke *node-node* lain disimbolkan dengan SPT(v).

Sebuah *node* x dikatakan sejajar dengan *node* i apabila terdapat *direct* link e dari *node* x ke node i (S(e)=i, E(e)=x). Mekanisme exportasi dengan tujuan untuk mematikan target *link* l yang berasal dari *node* i dapat

dilakukan dengan menggunakan i sebagai *importer* dan x sebagai *exporter*, sehingga *node* i dapat menghitung MPT dengan menggunakan SPT(x) sebagai referensinya. Melalui perhitungan diatas, dapat disimpulkan bahwa *link* l merupakan *targetlink* dan *link* e menggambarkan langkah exportasi yang dibutuhkan untuk mematikan link l

Sangat dimungkinkan apabila terdapat beberapa cara untuk menonaktifkan*link* l, M(l) merupakan kumpulan mekanisme exportasi yang dapat digunakan untuk menonaktifkan*link* l. Pemilihan dan penggunaan salah satu dari himpunan ini (alternatif cara untuk mematikan *link* l) dapat menghasilkan sebuah status *network* yang spesifik (dengan topologi yang berbeda) dan menentukan hasil dari *energysaving* dan *performance* pada jaringan. Exportasi m ∈M(l) akan dinamakan dengan *move*.

Seperti halnya setiap *directlink* (contohnya e) mengidentifikasikan *sourcenode* sebagai *importer* (S(e) =i, i adalah importer) dan *destinationnode* sebagai *exporter* (E(e)=x, x adalah exporter), maka jumlah maksimum *move* yang dapat dilakukan pada sebuah jaringan G(V,E) adalah L. (L adalah jumlah elemen dari E/jumlah *link*). Langkah selanjutnya adalah mendefinisikan rule/aturan sederhana untuk menentukan apakah **2 *moves* dapat dilakukan secara berurutan atau bersifat *mutuallyexclusive*.** Konsep ini dapat disimpulkan dengan mendefinisikan “kedua *moves* dikatakan *compatible*” apabila:

- Kondisi *compatibility*: apabila terdapat 2 *moves*, m1 dan m2, move m2 dikatakan *compatible* dengan move m1 (simbol: m2 α m1)

apabila m2 masih dapat dilakukan setelah menjalankan eksekusi m1. Sebaliknya, *move* m2 dikatakan tidak *compatible* dengan m1 apabila m2 tidak bisa dilakukan setelah menjalankan eksekusi m1.

Namun harus diperhatikan bahwa *compatibility* bersifat simetris, dengan kata lain apabila m2 α m1, maka m1 α m2, yang berarti kedua *move* m1 dan m2 tidak memiliki urutan eksekusi yang pasti. Setelah mengetahui *move* m, langkah selanjutnya adalah mendefinisikan set $M_{NC}(m)$, yaitu kumpulan *moves* yang tidak *compatible* dengan m. Langkah pertamanya adalah menentukan kondisi-kondisi yang harus dipenuhi agar kedua *moves* dikatakan *compatible*, kedua syarat ini dapat dibagi menjadi *proceduralcondition* (prosedur-prosedur yang harus dipenuhi) dan *performance* yang harus dipenuhi.

Kondisi prosedural dapat diturunkan dari definisi proses exportasi itu sendiri, khususnya ketika ingin melakukan proses exportasi (dengan i sebagai *importer* dan x sebagai *exporter*), syarat (C1-C3) ini harus dipenuhi terlebih dahulu:

C1. i tidak boleh menjadi *importer* untuk proses exportasi selanjutnya.

C2. i tidak boleh menjadi *exporter* untuk proses exportasi selanjutnya.

C3. x tidak boleh menjadi *importer* untuk proses exportasi selanjutnya.

Persyaratan performance menunjukkan syarat yang dibuat untuk mengontrol efek perubahan *routingpath* pada *performance* jaringan, khususnya dengan melakukan penambahan rule 1 sbb:

- **Rule 1: SPT yang dimiliki *exporter* tidak boleh dimodifikasi oleh proses exportasi lain, dengan kata lain *routingpath* dari *exporternode* harus tetap merupakan *shortestpath*. (SPT dari *exporternode* tidak boleh dimodifikasi/diubah oleh proses exportasi lain).**

  Rule 1 ini menyebabkan efek berganda, yaitu:

i. Sebagian besar dari *networkpath* identik/mirip dengan *shortestpath*.

ii. *Reroutedpath*/jalur yang dirutekan ulang memiliki jarak/*hop* yang cukup dekat dengan *shortestpath*. Jika menggunakan rule 1, dapat dibuktikan bahwa sangat dimungkinkan pada jaringan yang memiliki *weight* yang sama, peningkatan network path (dalam kaitan jumlah hop) maksimum hanya 2 saja. Penggunaan rule 1, mengaktifkan 2 kondisi tambahan untuk compatibility:

C4. *Link* yang digunakan oleh SPT *exporter* tidak boleh dimatikan.

C5. *Node*/*router* yang jalurnya dimodifikasi oleh proses exportasi, tidak boleh berperan sebagai exporter router.

Apabila kondisi C1-C5 digunakan, maka sangat dimungkinkan untuk menunjukkan bahwa rute jaringan yang baru sudah *loop-free*.

Setelah mendefinisikan konsep *move* dan syarat *compatibility* antara 2 *moves*, maka permasalahan *energysaving* pada jaringan dapat diformulasikan sebagai “permasalahan untuk mencari kumpulan/set *compatiblemoves* yang mampu meminimalisasikan konsumsi energi pada *network* dengan tetap memberikan level QoS yang sesuai”.

Solusi EAR Problem

Diasumsikan terdapat IP *network*G(V,E). Lalu digambarkan *undirectedgraph* H={M,C}, setiap *node* m $\in$ M merepresentasikan sebuah *move* m, dan *edge*/garis c $\in$ C menggambarkan *compatibilitas* diantara *moves*. Jika terdapat *edge*/garis c antara *node* m1 dan m2, maka *move* m1 dan m2 dikatakan *compatible* Selain itu, setiap *node* m dikarakterisasikan dengan *weight*w(m) yang merepresentasikan jumlah *link* yang dapat dimatikan *move* m.

Pada H = {M,C}, set/kumpulan dari pasangan moves yang *compatible* direpresentasikan dengan *clique* $K_h(n)$, dimana n merupakan jumlah dari node pada *clique*. Sebuah *clique* adalah bagian/subgraph dari H dimana **setiap nodenya pasti memiliki pasangan adjacent/sejajarnya masing-masing**. (*clique* adalah sebuah teori *graph* dimana masing-masing 2 nodenya **pasti** dihubungkan dengan sebuah edge, dengan kata lain setiap *move* pada $K_h(n)$ dapat saling dijalankan secara acak tanpa ada urutan tertentu, karena setiap *movecompatible* dengan seluruh *move* lain).

Setiap *clique* $K_h(n)$ dikarakterisasikan dengan beban $W(K_h)$, yang merupakan total beban dari setiap *nodes* Pada kasus ini, karena 2 *compatiblemoves* tidak dapat mematikan link yang sama, $W(K_h)$ adalah jumlah *link* yang dapat dimatikan oleh *compatiblemoves* pada $K_h(n)$. Jadi, problem EAR sama halnya dengan mencari *subgraph* lengkap dari H yang dikarakterisasikan dengan *maximumweight*.

Versi sederhana dari persoalan ini didapat ketika setiap node memiliki *weight* yang sama, problem ini diketahui pada literatur sebagai

"***maximumcliqueproblem***" pada sebuah *graph* (*maximumcliqueproblem* adalah mencari jumlah pasangan node terbanyak pada sebuah *graph*, dimana setiap node harus terhubung dengan seluruh node lain).

Max Compatibility Heuristic

Untuk menjelaskan *Max_Compatibility_Heuristic*, diperlukan tambahan simbol/notasi. Setiap *move* $m_i$akan dikarakteristikkan dengan sebuah *vektor*/garis *compatibility* c_i yang merepresentasikan hubungan *compatibility* m_i dengan beberapa moves lain:

$C_i = \{c_{ij}, 1 \leq j \leq L\}$, dimana:

a. $c_{ij} = 0$ apabila $m_i \not\alpha\ m_j$

b. $c_{ij} = 1$ apabila $m_i \alpha\ m_j$

c. $c_{ij} = 0$ apabila $i = j$

Tingkat *compatibility* g_i dari move m_i merepresentasikan jumlah *move* yang *compatible* dengan m_i:

$Gi = \sum_{j=1}^{L} C_{ij}$

Salah satu alternatif solusi S_k yang memungkinkan untuk menonaktifkan kumpulan/set dari ip *interface* direpresentasikan dengan set *compatiblemoves* berikut:

$S_k = \{m_k, k \in K\}, c_{ij} = 1\ \forall i,j \in K$ dengan $i \neq j$

Ket: S_k adalah set dari m_k (sebuah move k), dimana k merupakan elemen dari *graph* K, setiap *compatibility* c_{ij} pada *graph* K = 1, dimana i dan j merupakan elemen dari K dan i tidak sama dengan j. Ini berarti setiap

move yang berada pada *graph* K memiliki *compatibility* antara satu dengan yang lain. K adalah *cliquegraph*

Selain itu, utility function $U(S_k)$ digunakan untuk membandingkan solusi yang berbeda:

$U(S_k) = \sum_{k \in K} w(m_k)$

Dimana $w(m_k)$ merupakan jumlah *interface* yang dapat dimatikan oleh *move* m_k. Dengan cara ini, solusi akan dikarakteristikkan dari kemampuannya untuk menghemat konsumsi energi.

Namun harus diperhatikan, bahwa apabila kita menggunakan *maximumcliqueproblem* (dengan mengasumsikan versi sederhana dari EAR dimana setiap weight dari graph sama), solusi S_k^+merupakan set of *moves* dengan *cardinality*/jumlah elemen terbanyak, jadi:

$||S_k^+|| = \max ||S_k|| \ \forall\ k$

Max_Compatibilityheuristic, yang dijelaskan pada *pseudocode* algoritma 1, sudah didefinisikan dengan memperhitungkan kedua problem (mencari jumlah *compatible* moves terbanyak dan mencari *bestsolution* berdasarkan *energysaving* dengan mengevaluasi fungsi $U(S_k)$)

Algorithm 1 Max_Compatibility Heuristic

1: **Find** m_M s.t. $g_M = \max_{j \in L} g_j$
2: $j = 1$
3: **for all** m_k t.c. $c_{Mk} = 1$ **do**
4: $\quad S_j = \{m_M, m_k\}$
5: $\quad j = j + 1$
6: $\quad M_j = \{m_i \text{ t.c } c_{iM} = 1 \text{ AND } c_{ik} = 1\}$
7: $\quad \mathbf{c_{S_j}} = \mathbf{c_M} \text{ AND } \mathbf{c_k}$
8: **end for**
9: **for** $j = 1$ to g_M **do**
10: $\quad$ **while** $M_j \neq \{\emptyset\}$ **do**
11: $\quad\quad m_l = \max_{m_k \in M_j}(\mathbf{c_{S_j}} \text{ AND } \mathbf{c_k})$
12: $\quad\quad S_j = S_j \cup \{m_l\}$
13: $\quad\quad M_j = M_j - \{m_l\} - \{m_i \in M_j \text{ s.t. } c_{il} = 0\}$
14: $\quad\quad \mathbf{c_{S_j}} = \mathbf{c_{S_j}} \text{ AND } \mathbf{c_l}$
15: $\quad$ **end while**
16: **end for**
17: $S_j^* = \max_{1 \leq j \leq g_i} U(S_j)$

Gambar 2.3 *Pseudocode* algoritma d-EAR *Max_Compatibility*

Secara umum, rumus heuristik diatas langsung mencari *maxcompatibilitymove* m_M yang memiliki *highestcompatibilitydegree* g_M (line 1). Dengan cara ini, solusi yang dibuat akan menambahkan *move* m_M. Langkah selanjutnya (line 3-8)mendefinisikan g_M kandidat solusi S_j yang akan dievaluasi pada saat eksekusi algoritma. Setiap set S_j awalnya terdiri dari *move* m_M dan *compatiblemove* mj. Selain itu, 2 struktur data tambahan juga akan ditambahkan, yaitu set M_j yang merupakan *moves* yang dapat disertakan pada S_j untuk langkah selanjutnya (dimana set M_j ini harus *compatible* dengan m_M dan m_j). Vector c_{sj} merepresentasikan *compatibilityvector* dari *moves* m_M dan m_j pada satu waktu yang sama.

Pada line 9-16, set S_j akan dipenuhi; untuk setiap solusi S_j, salah satu *move* pada M_j (yang merupakan set of *moves* dengan

highestcompatibility pada S_j) akan ditambahkan sampai tidak ada *compatiblemoves* lagi yang tersisa (M_j = {∅}); set M_j dan *vectorcompatibility* c_{sj} akan dihitung ulang setiap kali terdapat penambahan move pada S_j.

- Yang terakhir/line 17 akan menghitung solusi terbaik S_j^* dari sisi aspek *energysaving*.

2.11. Min_Used_Links Heuristics

Min_used_linksheuristic mengurutkan *link* berdasarkan jumlah *router* yang menggunakan mereka, dan *link* dengan jumlah penggunaan terkecil akan dimatikan(Cianfrani, Eramo, Listanti, & Polverini, An OSPF Enhancement for Energy Saving in IP Networks, 2011).*HeuristicMin_Used_Links* didasarkan pada pengetahuan dari SPT masing-masing *router*. Untuk menentukan utilisasi *link*, *heuristic*akan memberikan setiap link l_i sebuah *value* n_i, dimana n_i ini merupakan jumlah *router* yang memiliki *link* l_i di setiap SPTnya.

Setelah melakukan perhitungan SPT, *link* $l_i$akan diurutkan berdasarkan nilai n_i. Pada setiap langkah iterasi, *link* dengan nilai n_i terendah akan dihilangkan dari list, dan dicarikan *moves* yang dapat menghilangkan *link* ini. Setelah *link* dimatikan, SPT akan dihitung kembali dan table *link* dengan nilai n_i tersebut akan dihitung ulang.

Prosedur akan berakhir ketika tidak ada proses exportasi yang dapat dilakukan lagi

2.12. Rocketfuel Project

Rocketfuel project merupakan program penelitian universitas *Washington,* Amerika Serikat yang bertujuan untuk memberikan informasi mengenai topologi ISP di seluruh dunia.*Rocketfuel project* merupakan salah satu *tools*yang banyak digunakan ketika ingin melakukan penelitian mengenai kinerja dan performa jaringan, beberapa contohnya dapat dilihat pada jurnal (Spring, Mahajan, & Wetherall, 2002), (Spring, Mahajan, & Anderson, Quantifying the Causes of Path Inflation, 2003), (Barroso & Holzle, 2007), dan (Chiaraviglio, Mellia, & Neri, Reducing Power Consumption in Backbone Networks, 2009) Dengan menggunakan *rocketfuel project*, peneliti dapat mencari referensi mengenai topologi ISP di seluruh belahan dunia sebagai dasar penelitian.

2.13. OPNET

OPNET merupakan perangkat *software* bisnis yang menyediakan analisa performa pada jaringan komputer.Pada awalnya, OPNET dikhususkan bagi *network administrator* yang ingin melakukan simulasi terhadap performa jaringan yang dimiliki.Akan tetapi, seiring

perkembangannya, OPNET memberikan *software* simulasi OPNET IT Guru Academic Edition secara cuma-cuma dan dikhususkan untuk para peneliti yang ingin melakukan simulasi kinerja jaringan. (http://www.opnet.com/)

Software OPNET mampu melakukan simulasi beberapa *routing protocol* seperti BGP, OSPF, IS-IS, dst. Selain itu, OPNET juga dapat digunakan untuk meneliti kinerja performa jaringan *wireless.*Hasil dari simulasi menggunakan *software* OPNET adalah variabel kinerja jaringan seperti misalnya *delay, throughput, bit error rate,* dst.

Saat ini, versi OPNET komersil yang terbaru adalah OPNET IT Modeller 14.0, dan versi *software* simulasi OPNET yang dikhususkan untuk akademik, yaitu OPNET IT Guru Academic Edition adalah versi 9.1. OPNET merupakan salah satu *tools* simulasi jaringan yang banyak digunakan dalam penelitian jaringan seperti misalnya pada jurnal (Reviriego, Hernandez, Larrabeiti, & Maestro, 2009).