BAB 6
PENGUJIAN DAN ANALISIS

Pada bagian ini dijelaskan hasil pengujian terhadap implementasi yang telah dilakukan beserta analisisnya. Pengujian dilakukan untuk menilai apakah seluruh kebutuhan yang telah dispesifikasikan sebelumnya telah terpenuhi. Analisis dilakukan untuk menilai apakah sistem bekerja sesuai yang diharapkan. Pengujian akan berisi tentang pengujian *Yen algorithm,* algoritme DFS dan algoritme Djikstra.

6.1 Skenario Pengujian

Pada pengujian ini akan dilakukan dengan menggunakan topologi *Abilene* yang jumlah *switch*nya *10* dengan masing-masing *switch* memiliki 1 *host.* Skenario ini dilakukan karena jika *link* bertambah pada topologi akan membuat pemilihan jalur semakin banyak dan jika *switch* bertambah pada topologi akan menambah *cost* pada *link* tersebut. Dilakukan *setting* ukuran bandwidth pada tiap-tiap *link*. Bandwidth disetting pada ukuran 1000 Mbps. Pada pengujian ada beberapa hal yang harus diuji, berikut tujuan penelitian:

- Pengujian Pencarian jalur

Pengujian pencarian jalur bertujuan untuk mengetahui apakah sistem dapat mencari jalur k-terpendek.

- Pengujian *throughput*

Pengujian throughput bertujuan untuk mengukur performa jaringan antara *Yen algorithm*, algoritme DFS dan algoritme Djikstra.

- Pengujian *packet loss*

Pengujian Packet loss bertujuan untuk menunjukan jumlah total paket yang hilang. seluruh paket IP yang hilang dengan seluruh paket IP yang dikirimkan antara pada *source* dan *destination*. Salah satu penyebab *packet loss* adalah *Congestion*, disebabkan terjadinya antrian yang berlebihan dalam jaringan.

- Pengujian *multipath*

Pengujian *multipath* bertujuan untuk melihat keberhasilan algoritme untuk menggunakan semua jalur yang telah ditemukan. Data yang digunakan untuk pengujian ini adalah data transmisi *byte* dari tiap *path* yang telah ditemukan. Untuk melihat pergerakan transmisi data didalam topologi, maka penulis menggunakan aplikasi bernama bwm-ng.

Tabel 6.1 Syntax *running iperf*

Untuk *server*	Iperf -s
Untuk *client*	Iperf -c alamat_IPserver -i interval_waktu -t waktu_kirim

Berdasarkan tabel 6.1 di atas merupakan syntax untuk melakukan uji parameter throughput. Pada pengujian ini telah ditentukan 1 *host* yang menjadi *server* adalah *host* tujuan. *Host* yang bertindak sebagai *server* dijalankan terlebih dahulu lalu *host client*. Nilai throughput yang diambil adalah nilai rata-rata yang tercetak pada terminal *host client*. Pada pengujian ini menggunakan *interval* waktu 1 dan waktu kirim 20.

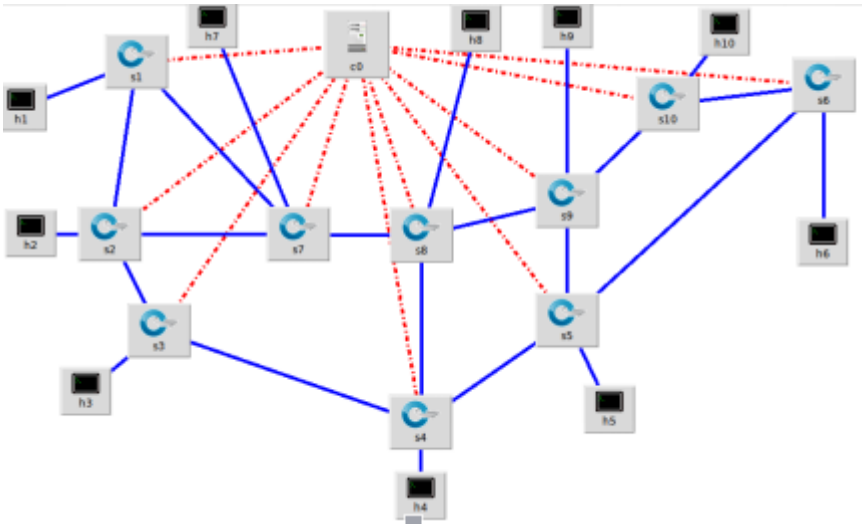
Tabel 6.2 Syntax *running ping*

ping -c banyaknya_paket_dikirim alamat_IP_host_tujuan

Tabel 6.2 merupakan syntax untuk melakukan uji parameter *packet loss*.

6.1.1 Pengujian Pencarian Jalur

Pengujian Pencarian jalur dilakukan dengan implementasi algoritme terhadap topologi yang telah dirancang sebelumnya. Topologi *Abilene* yang akan diuji untuk menilai keberhasilan *Yen Algorithm*.

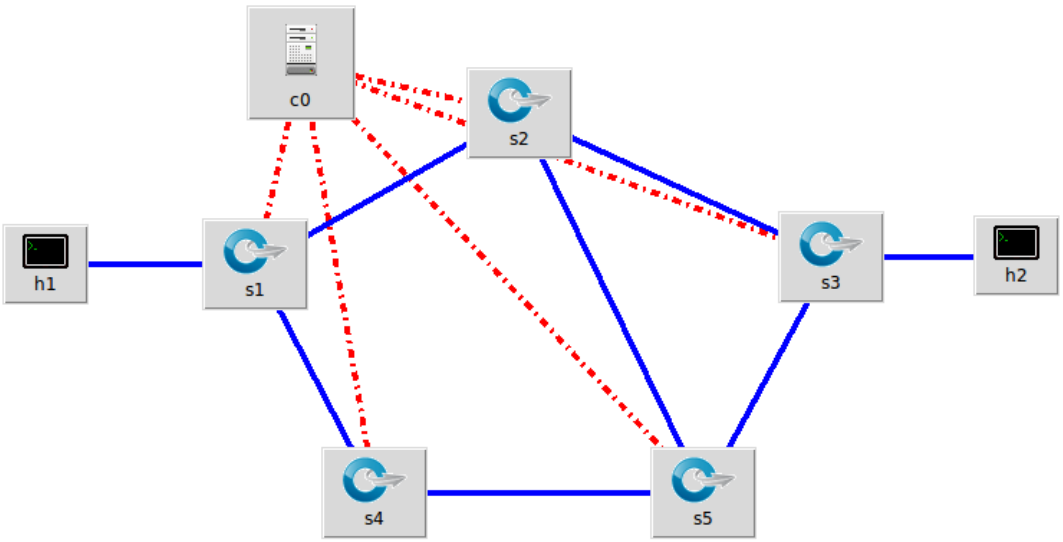


Gambar 6.1 Topologi 1 *Abilene*

Topologi *Abilene* untuk pengujian ditunjukkan pada gambar 6.1. topologi terdiri atas 10 *switch* dan 10 *host* yang berada di ujung topologi, h1 bertidak sebagai *server* dan h6 bertindak sebagai *client*. Di topologi tersebut

menggunakan *switch* yang sama. Seluruh topologi tersebut menggunakan *switch* dengan *link* bandwidth yang seragam, yaitu 1000 Megabit . Hal ini dikarenakan jika menggunakan *link* bandwith *default* (*unlimited*) maka bandwith dapat berubah secara acak.

Jumlah jalur yang didapatkan oleh algoritme akan digunakan untuk menilai apakah algoritme mampu mendapatkan sebanyak K-*path* yang diinginkan. Pengukuran *convergence time* dari *routing* yang dilakukan dibutuhkan untuk melihat seberapa cepat algoritme mampu mencari jalur-jalur tersebut. Untuk memulai pengujian, *client* akan melakukan *ping* menuju *host* 1 sebanyak 15 kali. C*ontroller* akan mencari sebanyak K-*path* yang diinginkan untuk mengirimkan paket data, dan selanjutkan akan dikalkulasi waktu yang dibutuhkan dalam pencarian jalur.



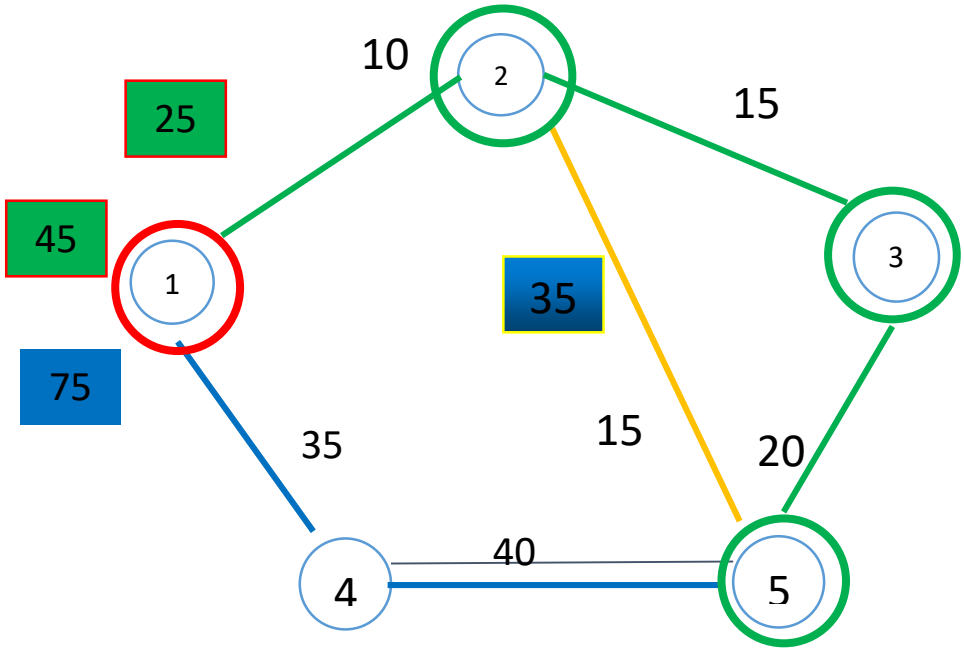
Gambar 6.2 Topologi 2 (T2)

Topologi 2 untuk pengujian multipath yen algorithm ditunjukkan pada gambar 6.2. topologi terdiri atas 5 *switch* dan 2 *host* yang berada di ujung topologi, h1 bertidak sebagai *server* dan h2 bertindak sebagai *client*. Di topologi tersebut menggunakan *switch* yang sama. Seluruh topologi tersebut menggunakan *switch* dengan *link* bandwidth yang seragam, yaitu 1000 Megabit . Hal ini dikarenakan jika menggunakan *link* bandwith *default* (*unlimited*) maka bandwidth dapat berubah secara acak.

```
carlo@carlo-X455LD: ~/ryu/ryu/app
Path installation finished in  0.00904583930969
YenKSP is called
src= 1  dst= 6  first_port= 3  final_port= 3  max_k=
this is called
node_spur= 1  path_root= [1]
node_spur= 2  path_root= [1, 2]
node_spur= 3  path_root= [1, 2, 3]
node_spur= 4  path_root= [1, 2, 3, 4]
node_spur= 5  path_root= [1, 2, 3, 4, 5]
node_spur= 1  path_root= [1]
0
YenKSP->
{'path': [1, 2, 3, 4, 5, 6], 'cost': 5}
{'path': [1, 7, 8, 4, 5, 6], 'cost': 5}
[(1, 5), (2, 5)]
[(2, 5)]
[(2, 5)]
[(2, 5)]
[(2, 5)]
[(2, 5)]
[(3, 5)]
[(2, 5)]
[(3, 5)]
Path installation finished in  0.00708508491516
```

Gambar 6.3 Contoh *output* pencarian jalur

Contoh *output* pencarian jalur ditampilkan pada Gambar 6.3. Terdapat 2 jalur K-terpendek yang ditemukan beserta *cost* dari masing-masing jalur yang telah ditemukan dan dibawah nya terdapat waktu yang dibutuhkan oleh algoritme untuk mencari jalur-jalur tersebut yang didefinisikan sebagai *convergence time*.



Gambar 6.4 T2 Pengujian Matematis Yen

Tabel 6.3 Hasil Pengujian Matematis T2

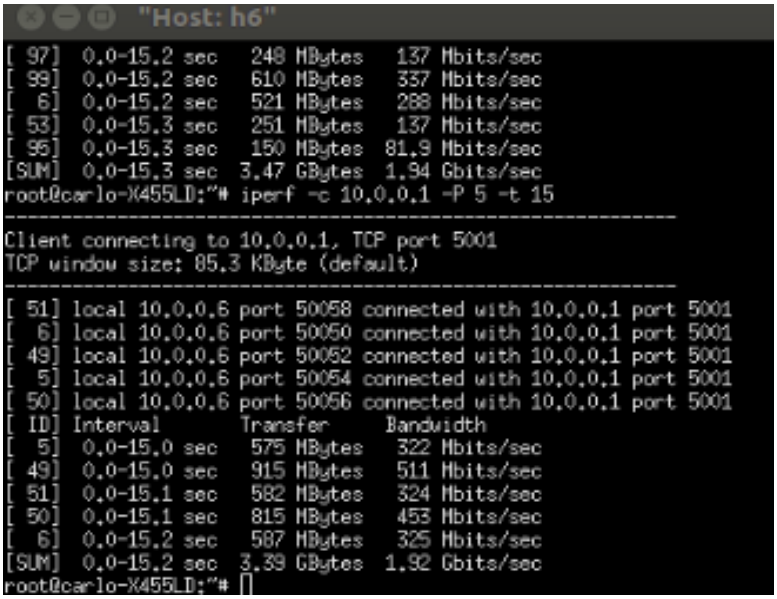
Titik terkunjungi	1	2	3	4	5
	∞	∞	∞	∞	∞
3	∞	**15**	0	∞	20
3,2	∞	**10**	0	∞	20
3,2,1	**25**		0	∞	

Titik terkunjungi	1	2	3	4	5
	∞	∞	∞	∞	∞
3	∞	-	0	∞	20
3,5	∞	-	0	40	**20**
3,5,2	**35**	-	0	40	
3,5,2,1	**45**	-	0	40	

Contoh dari pengujian pencarian Y*en algorithm* untuk menghitung 2 jalur dari *switch* 3 ke *switch* 1, disini algoritma dijkstra digunakan untuk menghitung jalur terbaik 3 ke 1 yang mana akan melewati *switch* 3 , *switch* 2 dan *switch* 1 dengan total cost 25, jalur ini ditambahkan ke jalur 1 dan menjadi jalur K-terpendek pertama. Node 3 akan menjadi simpul jalur dan untuk jalur tepi *switch* 3 dan *switch* 2 dihapus karena bertepatan dengan jalur yang sebelumnya yang sudah masuk dalam penjaluran. Kembali lagi Dijkstra digunakan untuk pencarian jalurnya untuk menghitung jalur terpendek dengan melewati 3,5,2, dan 1 dengan total cost 45 dan jalur 3,5,2 dan 1 ditambahkan ke jalur2 sebagai jalur K-terpendek potensial.

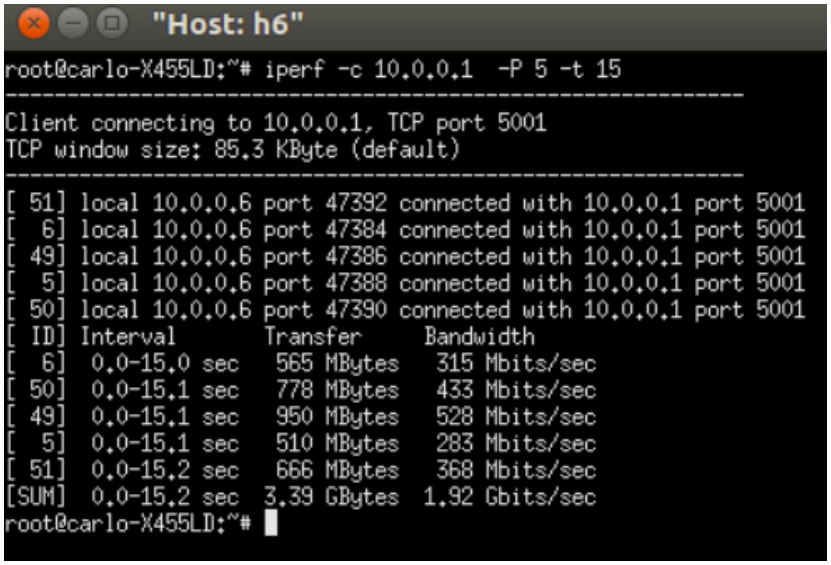
6.1.2 Pengujian Throughput

Throughput menyatakan jumlah *byte* yang berlalu pada suatu *link* per satuan waktu. Pengujian ini dilakukan untuk membandingkan throughput antara *Yen algorithm* dan algoritme *DFS* dalam pengiriman data. Pengujian Throughput pada *Yen algorithm* dilakukan dengan memberi nilai 3 pada variabel MAX_K sehingga dapat menghasilkan beberapa jalur untuk topologi *abilene* Untuk mengukur throughput, digunakan aplikasi *iperf* antara *host* 1 dan *host* 6. Pengujian throughput dilakukan selama 15 detik serta dilakukan peningkatan jumlah koneksi *client* secara parallel dari 5 hingga 25 koneksi secara bersamaan.



Gambar 6.5 Contoh pengujian throughput Yen 5 koneksi

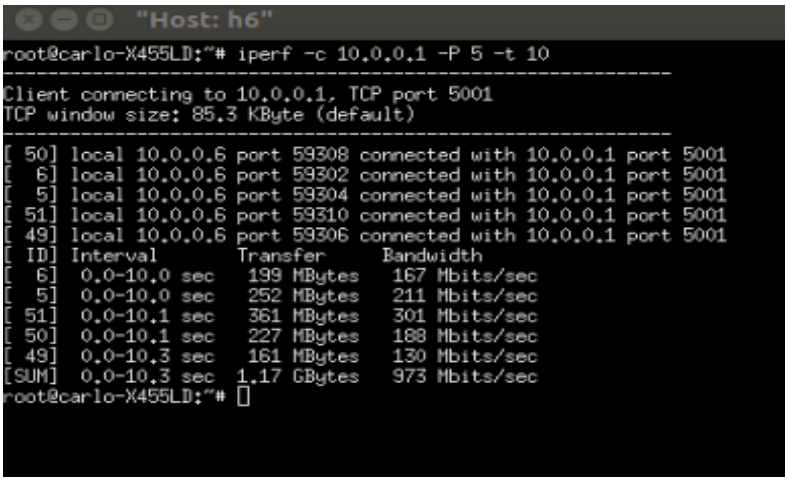
Contoh dari pengujian throughput pada (*host* 6) ditampilkan pada gambar 6.5. Pengujian throughput dilakukan dengan menggunakan 5 koneksi dan waktu selama 15 detik. ID 1 mencapi bandiwth 322 Mbits, ID 2 mencapai bandwidth 511 Mbits, ID 3 mencapai bandwith 324 Mbits, ID 4 mencapai bandwith 453 Mbits dan ID 5 mencapai bandwith 325 Mbits. Throughput SUM *Yen Algorithm* 5 koneksi sebesar 1.92 Gbits/sec.



Gambar 6.6 Contoh pengujian throughput DFS 5 koneksi

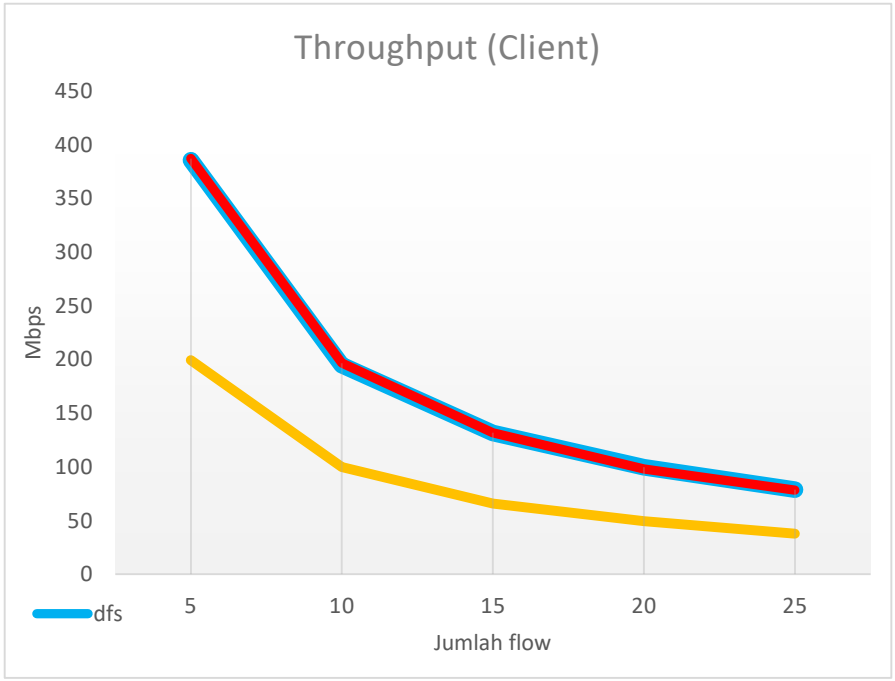
Contoh dari pengujian throughput pada (*host* 6) ditampilkan pada gambar 6.6. Pengujian throughput dilakukan dengan menggunakan 5 koneksi dan waktu

selama 20 detik. ID 1 mencapi bandiwith 315 Mbits, ID 2 mencapai bandwidth 433 Mbits, ID 3 mencapai bandwith 528 Mbits, ID 4 mencapai bandwith 283 Mbits dan ID 5 mencapai bandwith 368 Mbits. Throughput SUM DFS 5 koneksi sebesar 1.92 Gbits/sec.



Gambar 6.7 Contoh pengujian throughput Djikstra 5 koneksi

Contoh dari pengujian throughput pada (*host* 6) ditampilkan pada gambar 6.7. Pengujian throughput dilakukan dengan menggunakan 5 koneksi dan waktu selama 10 detik. ID 1 mencapi bandiwth 167 Mbits, ID 2 mencapai bandwidth 211 Mbits, ID 3 mencapai bandwith 301 Mbits, ID 4 mencapai bandwith 188 Mbits dan ID 5 mencapai bandwith 130 Mbits. Throughput SUM Djikstra 5 koneksi sebesar 974 Mbits/sec.

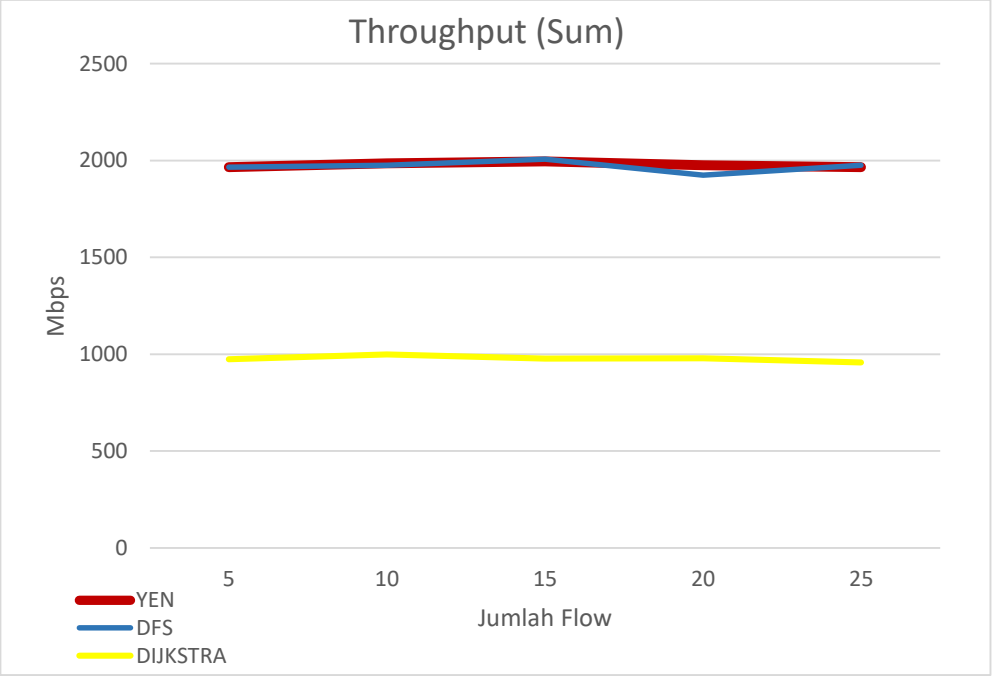


Gambar 6.8 Grafik pengujian throughput

Nilai throughput *client* dari *Yen algorithm,* algoritme DFS dan algoritme Dijkstra pada topologi *Abilene* ditujukkan pada gambar 6.8. Nilai rata-rata throughput *Yen algorithm* di 5 koneksi mencapai 387 Mbits untuk algoritme DFS di 5 koneksi mencapai 385.4 Mbits dan untuk algoritme Dijkstra di 5 koneksi mencapai 199.4 Mbits.

Tabel 6.4 Hasil pengujian throughput

Koneksi	THROUGHPUT		
	DFS	YEN	DIJKSTRA
5	385.4	387	199.4
10	194.67	196.69	99.92
15	131	131.81	65.9
20	99.7	98.13	49.57
25	78.9	78.48	37.7
Rata-rata	**177.934**	**178.422**	**90.498**



Gambar 6.9 Grafik pengujian throughput SUM

Nilai throughput SUM disini adalah total penjumlahan bandwidth dari dari *Yen algorithm,* DFS dan Dijkstra pada topologi *Abilene* ditujukkan pada

gambar 6.9. Nilai throughput *Yen algorithm* tidak jauh berbeda dengan algoritme DFS sedangkan pengujian algoritme Dijkstra untuk 5 sampai 25 *flow* rendah karena hanya melewati 1 jalur saja.

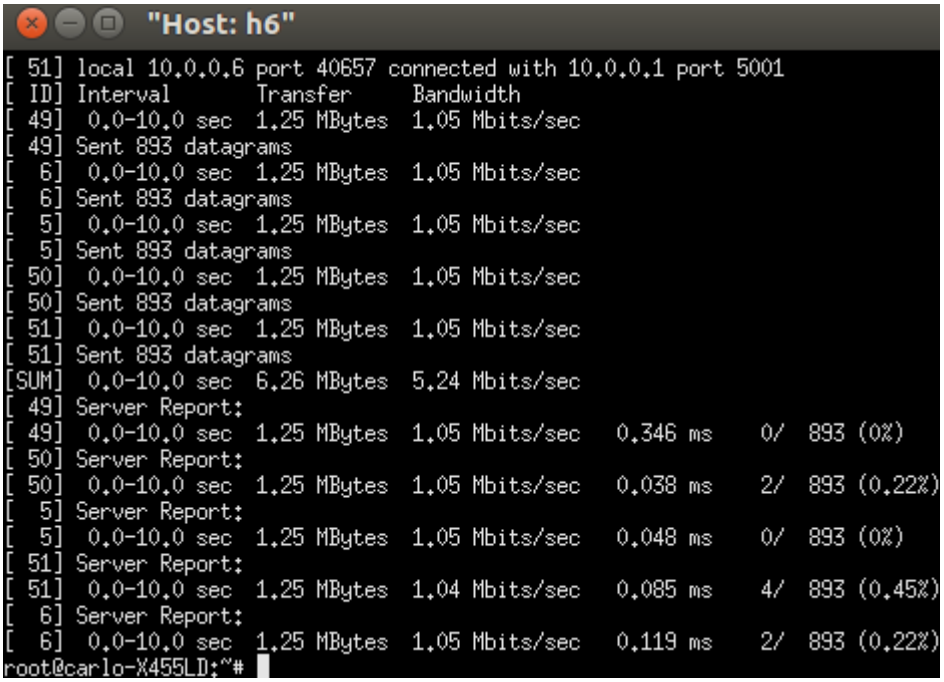
Tabel 6.5 Hasil pengujian throughput(SUM)

Koneksi	THROUGHPUT(SUM)		
	DFS	YEN	DIJKSTRA
5	1966	1966	973
10	1976	1986	999
15	2007	1996	977
20	1925	1976	979
25	1976	1966	958

6.1.3 Pengujian *Packet Loss*

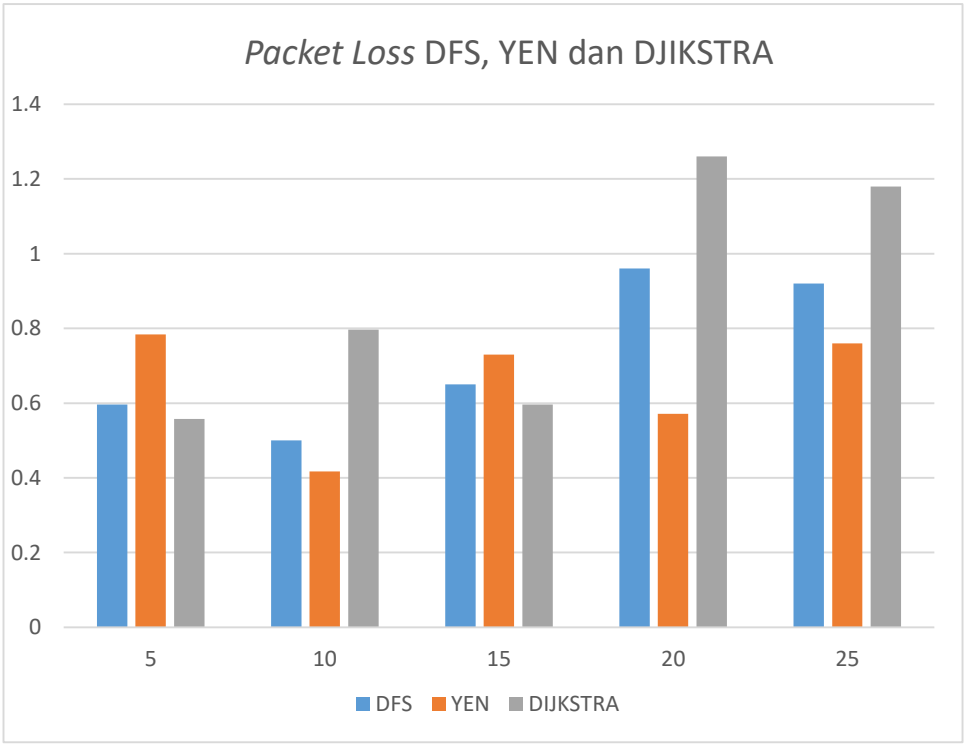
Pengujian *packet loss* dilakukan dengan memperhatikan persen *packet loss* di *server*. Setelah *client* mengirimkan paket UDP dan *server* melakukan *listen* server UDP maka akan muncul keluaran Transfer, Bandwith, Jitter, *Lost*/Total Datagrams serta persen *packet loss* nya. *Packet loss* merupakan parameter yang menggambarkan suatu kondisi yang menunjukan jumlah total paket yang hilang.

Untuk mengukur *packet loss* digunakan aplikasi *iperf* dari *client* menuju *server*. Pengujian *packet loss* dilakukan dengan beban transfer 1 MB untuk setiap koneksi. Pengujian *packet loss* dilakukan peningkatan jumlah koneksi *client* secara parallel dari 5 hingga 25 koneksi secara bersamaan.



Gambar 6.10 Contoh pengujian *Packet loss* 5 koneksi

Contoh dari pengujian *Packet loss* pada topologi *Abilene* ditunjukkan pada gambar 6.10 Pengujian dilakukan dengan menggunakan 5 koneksi. Pada gambar tersebut terdapat beberapa data, dan diantaranya ada *packet loss*.



Gambar 6.11 Grafik pengujian *packet loss*

Nilai *packet loss* dari *Yen algorithm* algoritme DFS dan algoritme Dijkstra pada topologi *Abilene* ditunjukkan pada gambar 6.11. Nilai *packet loss Yen algorithm* lebih kecil dan tidak berbeda jauh dengan dibandingkan algoritme DFS sedangkan untuk algoritme Dijkstra nilai packet loss besar di bandingkan dengan Yen dan DFS untuk 5 sampai 25 *flow* pada gambar tersebut, itu karena algoritme Dijkstra hanya melewati 1 jalur saja ini yang menyebabkan antrian (c*ongestion)*

Tabel 6.5 Hasil pengujian *packet loss*

Jumlah Koneksi	Persentase *Packet Loss* (%)		
	Yen Algorithm	**DFS**	**DJIKSTRA**
5	0.784	0.586	0.558
10	0.417	0.5	0.797
15	0.73	0.65	0.596
20	0.571	0.96	1.26
25	0.76	0.92	1.18
Rata-rata	**0.6524**	**0.7232**	**0.8782**

Nilai *packet loss* dari pengujian topologi ditampilkan pada tabel 6.5. *Packet loss* untuk *Yen algorithm* dan algoritme DFS lebih kecil di bandingkan dengan algoritme Djikstra. Perhitungan *packet loss* dilakukan untuk melihat persentase pengiriman *packet* yang gagal untuk setiap algoritme pada topologi.

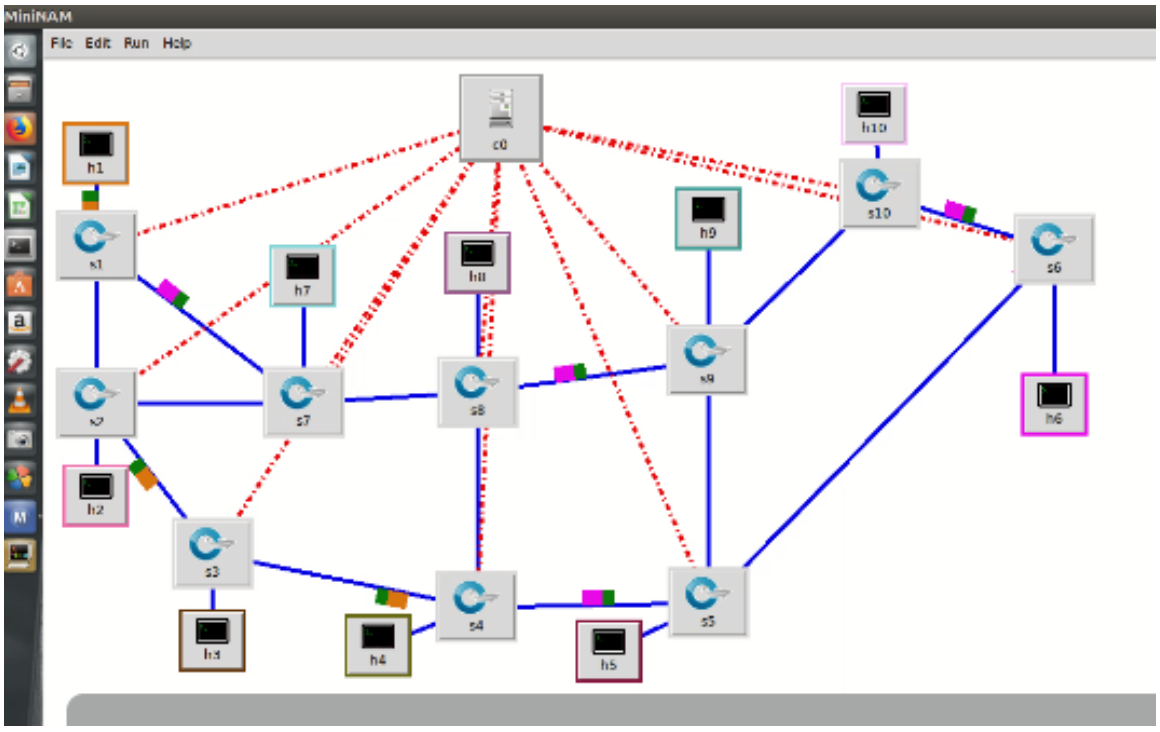
6.1.4 Pengujian *Multipath*

Pengujian *Multipath* dilakukan dengan memperhatikan penghitung *byte* transmisi dari *switch* dengan *outgoing traffic* atau *traffic* ke luar dari masing-masing *port* yang terdapat dalam skema *multipath.* Pengujian ini dilakukan dengan menggunakan aplikasi bwm-ng untuk *capture* bandwidth. Pengujian ini dilakukan untuk menilai apakah sistem telah berhasil melewati paket ke beberapa jalur (*multipath)* yang ada. Untuk itu, digunakan kembali aplikasi *iperf* yang dijalankan dengan *interval* waktu 20 detik dari *host* 1 ke *host* 6 dengan bandwith *link* antar *switch* adalah 1000 megabit. Dibawah ini merupakan hasil dari pengujian *multipath*. Pengujian *multipath* hanya menggunakan 1 *host* saja yaitu *S1-eth1:s2-eth1, s1-eth2:s7-eth1, s1-eth3:h1* untuk melihat keberhasilan implementasi *multipath*. *Port-port* penghubung dalam masing-masing topologi dapat dilihat pada gambar 6.11.

```
carlo@carlo-X455LD: ~
*** Starting CLI:
mininet> net
h5 h5-eth0:s5-eth4
h9 h9-eth0:s9-eth4
h3 h3-eth0:s3-eth3
h1 h1-eth0:s1-eth3
h10 h10-eth0:s10-eth3
h7 h7-eth0:s7-eth4
h6 h6-eth0:s6-eth3
h4 h4-eth0:s4-eth4
h8 h8-eth0:s8-eth4
h2 h2-eth0:s2-eth4
s6 lo:  s6-eth1:s5-eth2 s6-eth2:s10-eth2 s6-eth3:h6-eth0
s4 lo:  s4-eth1:s3-eth2 s4-eth2:s5-eth1 s4-eth3:s8-eth3 s4-eth4:h4-eth0
s5 lo:  s5-eth1:s4-eth2 s5-eth2:s6-eth1 s5-eth3:s9-eth3 s5-eth4:h5-eth0
s2 lo:  s2-eth1:s1-eth1 s2-eth2:s3-eth1 s2-eth3:s7-eth3 s2-eth4:h2-eth0
s10 lo:  s10-eth1:s9-eth2 s10-eth2:s6-eth2 s10-eth3:h10-eth0
s1 lo:  s1-eth1:s2-eth1 s1-eth2:s7-eth1 s1-eth3:h1-eth0
s8 lo:  s8-eth1:s7-eth2 s8-eth2:s9-eth1 s8-eth3:s4-eth3 s8-eth4:h8-eth0
s7 lo:  s7-eth1:s1-eth2 s7-eth2:s8-eth1 s7-eth3:s2-eth3 s7-eth4:h7-eth0
s9 lo:  s9-eth1:s8-eth2 s9-eth2:s10-eth1 s9-eth3:s5-eth3 s9-eth4:h9-eth0
s3 lo:  s3-eth1:s2-eth2 s3-eth2:s4-eth1 s3-eth3:h3-eth0
c0
mininet>
```

Gambar 6.12 *Port* penghubung pada topologi *Abilene*

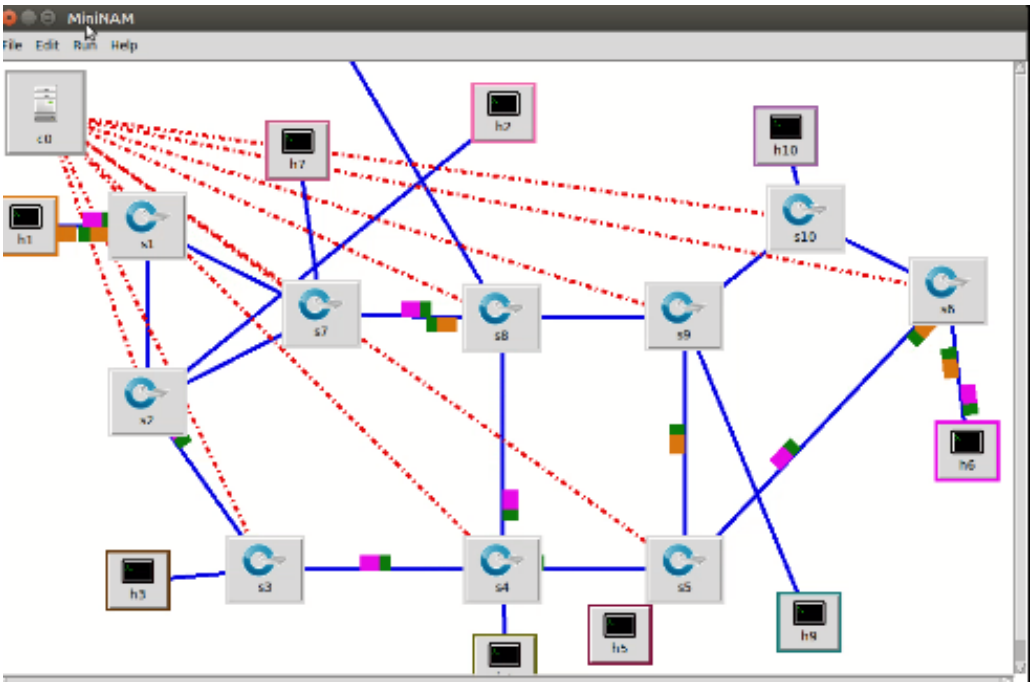
*Port-por*t yang menghubungkan antara *switch* satu dan *switch* yang lainnya, atau antara *switch* dan *host* pada topologi *Abilene* ditunjukkan pada gambar 6.12. Data ini dibutuhkan untuk melihat jalur data yang terjadi dalam topologi.



Gambar 6.13 Jalur yang digunakan menggunakan *Yen algorithm*

Sebagian besar protokol data *transfer* mengirimkan sebuah *acknowledgement message*, yang menjadi indikasi bahwa sebuah data yang terkirim telah diterima dengan baik.

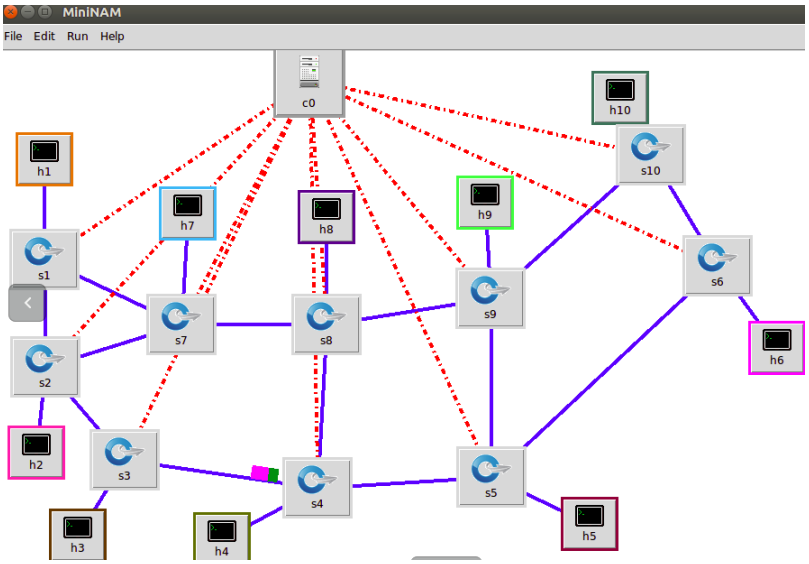
Jalur-jalur yang digunakan oleh *Yen algorithm* sebagai media pengiriman data pada topologi *Abilene* ditampilkan pada gambar 6.13. Terdapat 2 jalur yang digunakan oleh algoritme dari *host* 6 ke *host* 1 untuk mengirim data, hal ini karena pada algoritme yang digunakan diberi nilai MAX_K= 3, jadi setiap *host* dapat menggunakan 2 atau 3 jalur berbeda untuk melakukan pengiriman data. *Path* K-terpendek yang dilewati *Yen algorithm* adalah S6, S5 , S4 , S3, S2 dan S1 dan S6, S10, S9, S8, S7 dan S1.



Gambar 6.14 Jalur yang digunakan menggunakan algoritme DFS

Jalur-jalur yang digunakan oleh algoritme DFS sebagai media pengiriman data pada topologi *Abilene* ditampilkan pada gambar 6.14. *Available path host* 6 ke *host* 1 melewati Switch {[1, 7, 2, 3, 4, 5, 6], [1, 7, 2, 3, 4, 5, 9, 10, 6], [1, 7, 2, 3, 4, 8, 9, 5, 6], [1, 7, 2, 3, 4, 8, 9, 10, 6], [1, 7, 8, 4, 5, 6], [1, 7, 8, 4, 5, 9, 10, 6], [1, 7, 8, 9, 5, 6], [1, 7, 8, 9, 10, 6], [1, 2, 7, 8, 4, 5, 6], [1, 2, 7, 8, 4, 5, 9, 10, 6], [1, 2, 7, 8, 9, 5, 6], [1, 2, 7, 8, 9, 10, 6], [1, 2, 3, 4, 5, 6], [1, 2, 3, 4, 5, 9, 10, 6], [1, 2, 3, 4, 8, 9, 5, 6], [1, 2, 3, 4, 8, 9, 10, 6]} ini karena algoritme DFS Pencariannya dilakukan pada satu *node* dalam setiap level dari yang paling kiri. Jika pada level yang paling dalam, solusi belum ditemukan, maka pencarian dilanjutkan pada *node* sebelah kanan. *Node* yang kiri dapat dihapus dari memori. Jika pada level yang paling

dalam tidak ditemukan solusi, maka pencarian dilanjutkan pada level sebelumnya. Demikian seterusnya sampai ditemukan solusi. Jika solusi ditemukan maka tidak diperlukan proses *backtracking* (penelusuran balik untuk mendapatkan jalur yang dinginkan).



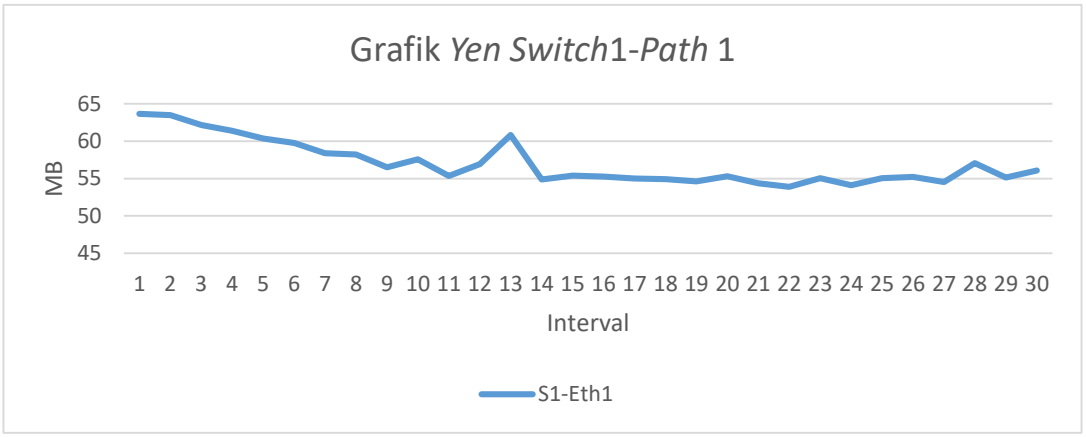
Gambar 6.15 Jalur yang digunakan menggunakan algoritme Djikstra

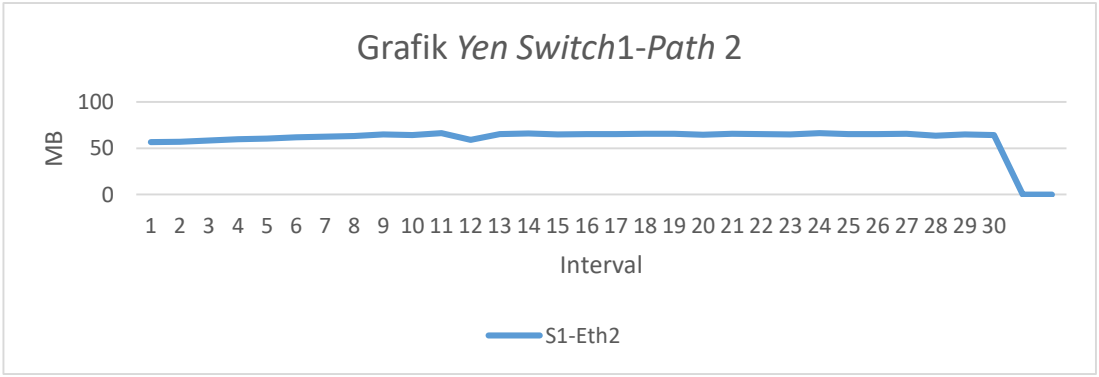
Jalur yang digunakan oleh algoritme Djikstra sebagai media pengiriman data pada topologi *Abilene* ditampilkan pada gambar 6.15. Terlihat bahwa 1 jalur saja yang digunakan *host* 6 ke *host* 1 melewati Switch [6, 5, 4, 3, 2, 1]. ini karena algoritme Dijkstra hanya mencari 1 jalur terbaik sebagai media pengiriman data.

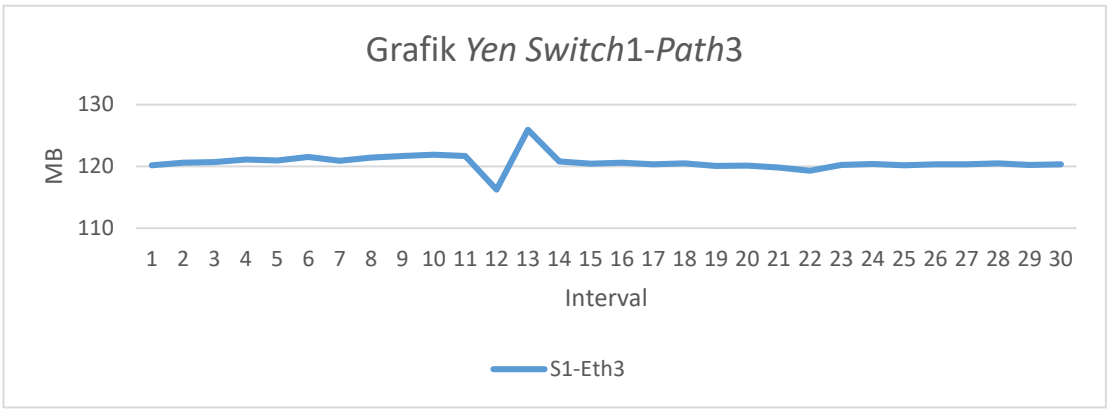
```
bwm-ng v0.6 (probing every 0.500s), press 'h' for help
input: /proc/net/dev type: rate
|         iface                   Rx                   Tx                Total
==============================================================================
       s10-eth2:        116715.01 KB/s             0.00 KB/s        116715.01 KB/s
        s3-eth1:           254.85 KB/s        116837.39 KB/s        117092.25 KB/s
        s3-eth3:             0.00 KB/s             0.00 KB/s             0.00 KB/s
        s1-eth2:        116842.27 KB/s           118.60 KB/s        116960.87 KB/s
        s4-eth2:        116837.39 KB/s           373.34 KB/s        117210.73 KB/s
        s2-eth1:           254.85 KB/s        116877.04 KB/s        117131.89 KB/s
        s7-eth2:        116842.27 KB/s           118.60 KB/s        116960.87 KB/s
        s4-eth4:             0.00 KB/s             0.00 KB/s             0.00 KB/s
        s5-eth1:           373.47 KB/s        116792.24 KB/s        117165.70 KB/s
        s2-eth3:             0.12 KB/s             0.12 KB/s             0.23 KB/s
              lo:             7.19 KB/s             7.19 KB/s            14.38 KB/s
        s7-eth4:             0.00 KB/s             0.12 KB/s             0.12 KB/s
        s8-eth1:           118.60 KB/s        116842.27 KB/s        116960.87 KB/s
        s5-eth3:             0.00 KB/s             0.00 KB/s             0.00 KB/s
        s8-eth3:             0.12 KB/s           118.47 KB/s           118.59 KB/s
        s6-eth2:             0.00 KB/s        116842.15 KB/s        116842.15 KB/s
        s9-eth2:        116842.15 KB/s             0.00 KB/s        116842.15 KB/s
        s9-eth4:             0.00 KB/s             0.00 KB/s             0.00 KB/s
       s10-eth3:             0.00 KB/s             0.00 KB/s             0.00 KB/s
          wlan0:             0.12 KB/s             0.00 KB/s             0.12 KB/s
        s3-eth2:        116877.04 KB/s           254.98 KB/s        117132.02 KB/s
        s1-eth1:        116964.66 KB/s           254.98 KB/s        117219.65 KB/s
        s4-eth1:           254.98 KB/s        116877.04 KB/s        117132.02 KB/s
        s1-eth3:           373.47 KB/s        233719.19 KB/s        234092.66 KB/s
        s7-eth1:           118.60 KB/s        116842.27 KB/s        116960.87 KB/s
        s4-eth3:           118.60 KB/s             0.12 KB/s           118.72 KB/s
        s2-eth2:        116877.04 KB/s           254.85 KB/s        117131.89 KB/s
        s7-eth3:             0.12 KB/s             0.12 KB/s             0.23 KB/s
        s5-eth2:        116837.39 KB/s           373.34 KB/s        117210.73 KB/s
        s2-eth4:             0.00 KB/s             0.00 KB/s             0.00 KB/s
        s8-eth2:        116842.15 KB/s             0.00 KB/s        116842.15 KB/s
        s5-eth4:             0.00 KB/s             0.00 KB/s             0.00 KB/s
        s6-eth1:           373.47 KB/s        116879.86 KB/s        117253.33 KB/s
        s8-eth4:             0.00 KB/s             0.12 KB/s             0.12 KB/s
        s9-eth1:             0.00 KB/s        116842.15 KB/s        116842.15 KB/s
        s6-eth3:        234019.14 KB/s           373.47 KB/s        234392.61 KB/s
       s10-eth1:             0.00 KB/s        116842.15 KB/s        116842.15 KB/s
```

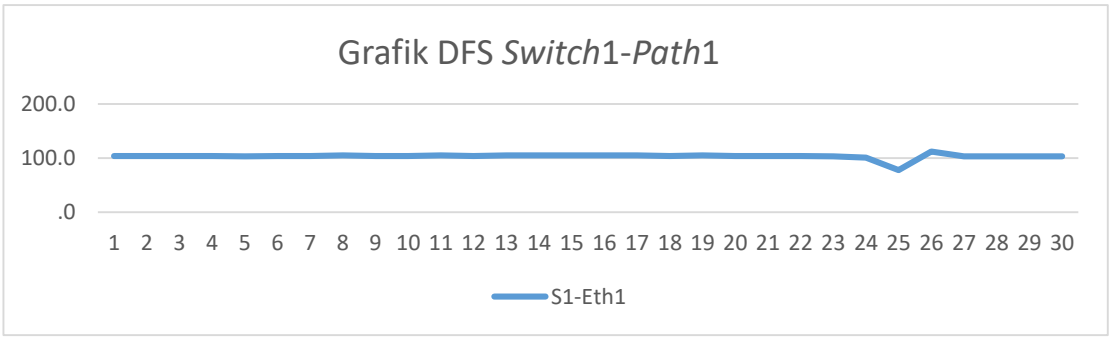
Gambar 6.16 Contoh *output* aplikasi bwm-ng

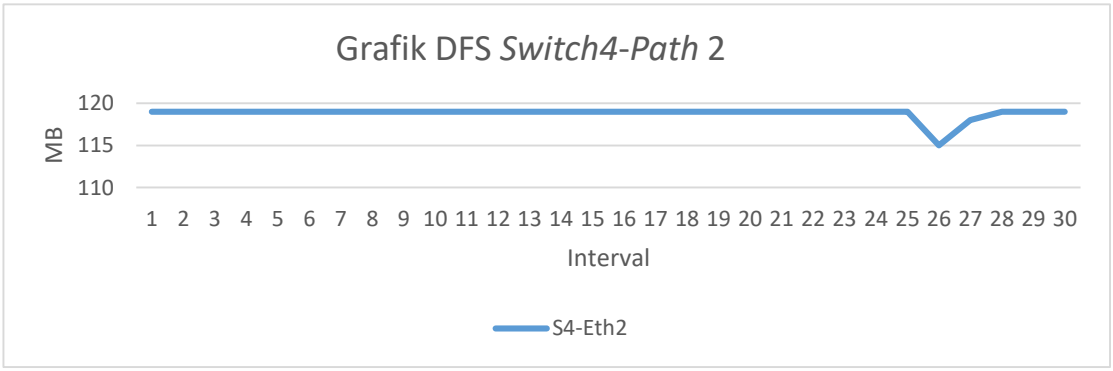
Contoh dari *output* aplikasi bwm-ng yang digunakan untuk pengujian *multipath* ditujukkan pada gambar 6.16. Terdapat data jumlah transmisi dan penerimaan paket di masing-masing *port*. Hasil dari pengujian *multipath* ini dilihat dari seberapa banyak *port* yang digunakan oleh sistem sebagai jalur transaksi data. Didalam pengujian ini saya menggunakan MAX_K sebanyak 3 jalur. *Port* yang memiliki beban *transfer* harus sesuai dengan *port-port* yang digunakan sebagai penghubung oleh jalur yang ditemukan agar pengujian ini berhasil.

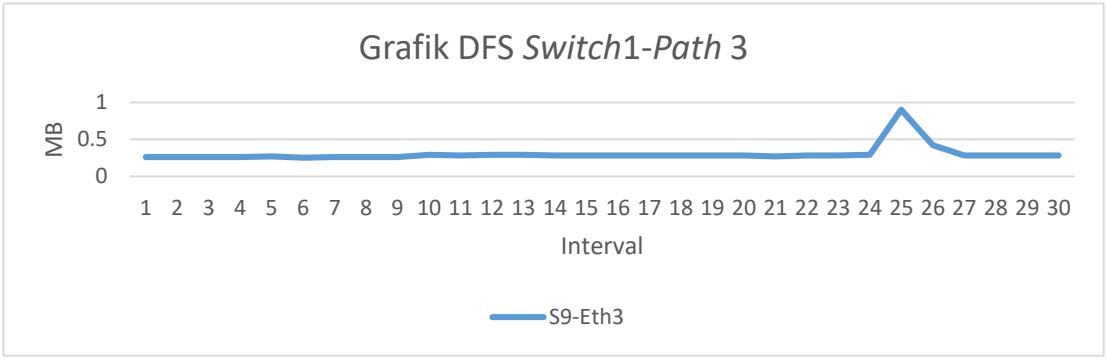


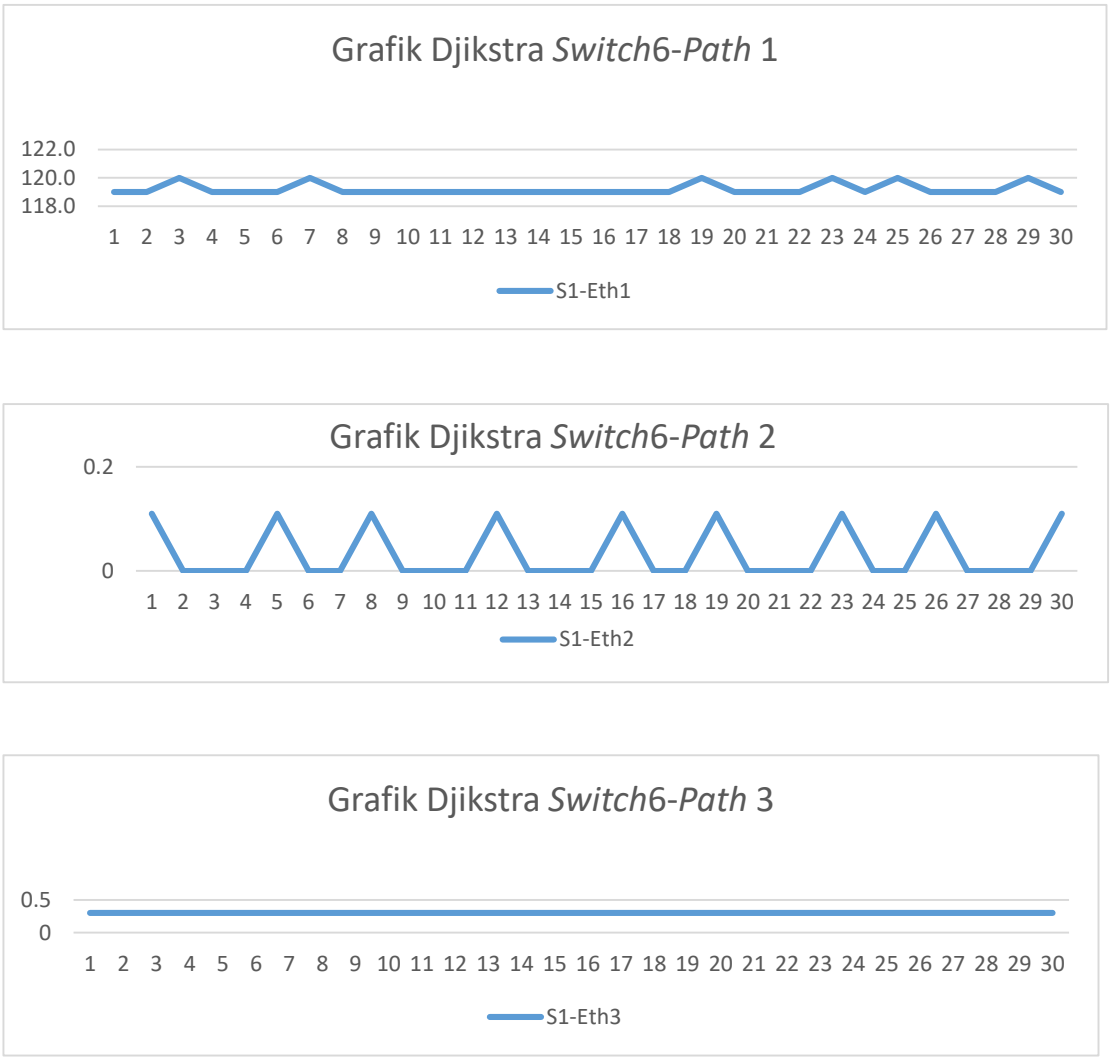



Grafik *Yen Switch*1-*Path*3
130
120
110
MB
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Interval
S1-Eth3


Grafik DFS *Switch*1-*Path*1
200.0
100.0
.0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
S1-Eth1

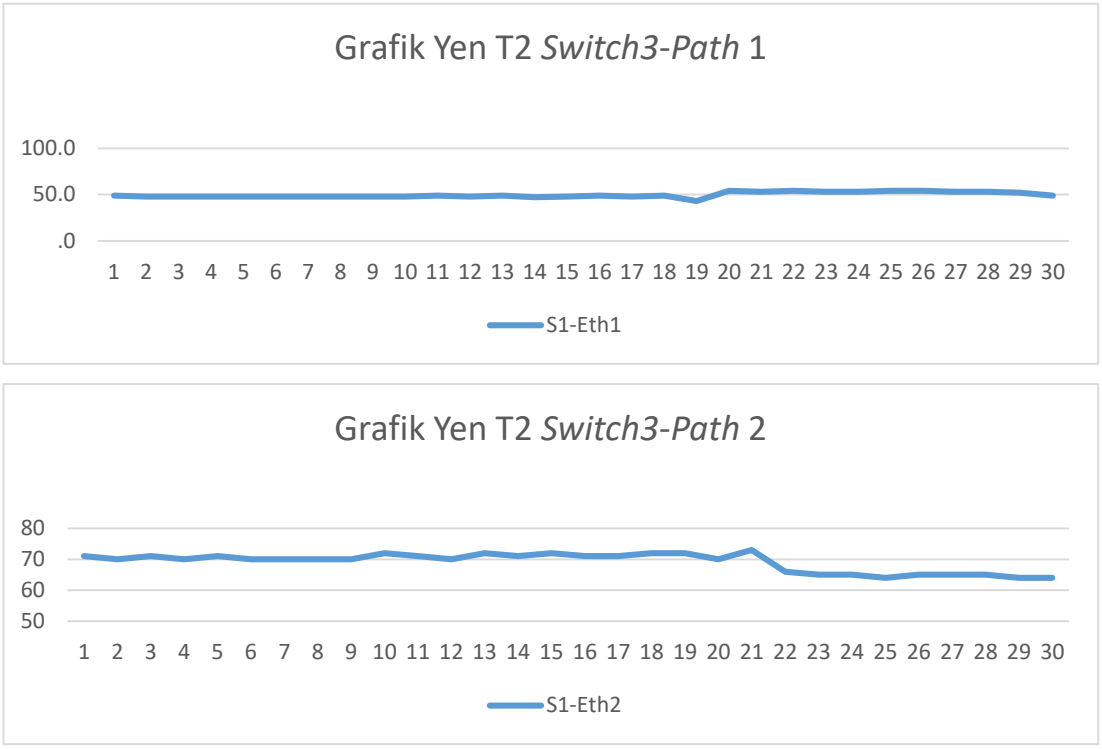

Grafik DFS *Switch4-Path* 2
120
115
110
MB
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Interval
S4-Eth2


Grafik DFS *Switch*1-*Path* 3
1
0.5
0
MB
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Interval
S9-Eth3



Gambar 6.17 Grafik pengujian *Multipath* topologi *Abilene*

Path 1-Eth1,path 2-Eth2 dan path 3-eth3 merupakan *path* yang akan dilewati melalui *port 1,2 dan 3.* Perbandingan antara *transfer* dari semua *path* yang telah ditemukan menggunakan *Yen*, dengan *transfer* data pada *path* yang sama menggunakan algoritme DFS dan Dijkstra pada topologi *Abilene* ditunjukkan pada gambar 6.17. *Yen algorithm* dan algoritme DFS untuk topologi *Abilene* berhasil menggunakan seluruh jalur yang ditemukan untuk mengirim *packet* sedangkan algoirtme Dijkstra hanya melalui lewat 1 *path* saja.



Gambar 6.18 Grafik pengujian *Multipath T2*

Digunakan topologi 2 untuk mencoba kembali serta membuktikan bahwa *Yen algorithm* sudah bisa melakukan *multipath* berbagi jalur *Path 1-Eth1, path 2-Eth2* merupakan *path* yang akan dilewati melalui *port* 1 dan 2 pada gambar *6.18.*

Tabel 6.6 Hasil pengujian *Multipath* topologi *Abilene*

Interface	**Rata-rata *transfer* (MB)**		
	Yen Algorithm	**DFS**	**Djikstra**
Path 1	57.009	103.9471384	119.827
Path 2	63.630	119.2456667	0.00359
Path 3	0.677	0.3057701123	0.33983

Rata-rata transfer pada masing-masing *path* yang telah ditemukan pada topologi *Abilene* menggunakan *Yen algorithm*, dengan *transfer data* pada *path* yang sama menggunakan algoritme DFS dan Dijkstra ditunjukkan pada tabel 6.6.

Tabel 6.7 Hasil pengujian *Multipath* topologi *2*

Interface	**Yen**
Path 1	50.265
Path 2	69.507

Rata-rata transfer pada masing-masing *path* yang telah ditemukan pada topologi *Abilene* menggunakan *Yen algorithm* ditunjukan pada tabel 6.7.

6.2 Analisis

6.2.1 Analisis Pencarian Jalur

Berdasarkan hasil pengujian pencarian jalur yang telah dilakukan, sistem berhasil menemukan sebanyak K-jalur pada topologi *Abilene.* Topologi *Abilene* menghasilkan 3 jalur yang berbeda untuk tiap *host*. Pengujian pencarian jalur dilakukan pada *host* 1 yang bertidak sebagai *server* dan *host* 6 sebagai *client.* Dari nilai *convergence time Yen algorithm* yang ditemukan, didapatkan bahwa *convergence time* akan lebih besar pada topologi dengan *switch* dan jalur yang lebih banyak. Rata-rata *convergence time* 0.0070 detik.

6.2.2 Analisis Throughput

Berdasarkan hasil pengujian throughput, dapat terlihat bahwa throughput *Yen algorithm* di koneksi 5 sebesar 387 Mbps tidak berbeda jauh dengan throughput algoritme DFS sebesar 385.4 Mbps sedangkan algoritme Djikstra sebesar 199.4 Mbps pada topologi *Abilene*. Jalur yang digunakan untuk pengiriman data pada *multipath* lebih banyak dari pada yang digunakan untuk routing DFS dan Djikstra. Ketika jumlah jalur yang digunakan dalam *multipath routing* semakin banyak, maka peningkatan throughput semakin mengecil dan tidak berbeda signifikan dengan *single-path routing*, seperti yang ditunjukkan pada hasil pengujian throughput.

6.2.3 Analisis *Packet Loss*

Berdasarkan hasil pengujian *packet loss*, ditemukan bahwa terdapat perbedaan antara *Yen algoritm* dengan algoritme DFS. Pada topologi *Abilene* rata-rata paket loss dengan menggunakan *Yen algorithm* adalah 0.6524 % jika menggunakan algortime DFS rata-rata *packet loss* adalah 0,7232 % sedangkan jika menggunakan algoritme Djikstra rata-rata *packet loss* mencapai 0.8782%. Perbedaannya packet loss Yen algorithm dengan algoritme DFS mencapai 0,0708 % dan perbedaan *packet loss Yen algorithm* dengan algoritme Djikstra mencapai 0.2258%. Terlihat banyak nya *switch* tidak berpengaruh signifikan terhadap perubahan *packet loss* namun perubahan jumlah koneksi sangat berpengaruh terhadap hasil *packet loss*. Berdasarkan hasil pengujian diatas, persentase *packet loss* untuk *Yen algorithm* dan DFS tergolong cukup kecil, sedangkan persentase *packet loss* untuk algoritme Djikstra tetap lebih besar dibandingkan dengan *Yen algorithm* dan algoritme DFS. Semakin banyak jalur yang dapat digunakan (*multipath*) untuk mengirimkan data maka *packet loss* akan semakin berkurang. Hal ini disebabkan karena semakin banyak jalur maka kemacetan jaringan akan semakin berkurang, dan paket yang akan dikirimkan mampu menghindari jalur yang telah padat.

6.2.4 Analisis *Multipath*

Untuk melakukan analisis terhadap pengujian *Multipath*, dilakukan perbandingan beban transmisi paket pada masing-masing *port* dalam *switch* yang sudah didefenisikan terlebih dahulu menjadi *path*. Pada gambar 6.17 dapat dilihat transmisi paket pada masing-masing jalur pada topologi *Abilene*, paket ditransmisikan melalui 2 jalur. *Path* tersebut dibedakan berdasarkan *port* keluaran dari *switch*. Beban *transfer* pada masing-masing jalur tersebut disesuaikan dengan *port-port* yang digunakan sebagai penghubung oleh jalur yang telah ditemukan. Masih ada kelemahan dari sistem ini karena beban yang di transmisikan oleh masing-masing *port* belum seimbang 100% dan semakin banyak *switch* yang digunakan, *transfer* pada masing-masing jalur menjadi semakin kecil.

Pada toplogi 2 (T2) dilakukan pengujian Multipath untuk *Yen algorithm* kembali untuk membuktikan bahwa sudah berhasil melakukan berbagi jalur berdasarkan K yang sudah di tentukan. S3-Eth1 dan S3-Eth2 sebagai Path untuk berbagi jalur dengan port yang digunakan sebagai penghubung oleh jalur yang telah ditemukan.