Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer e-ISSN: 2548-964X
Vol. 5, No. 2, Februari 2021, hlm. 572-581 http://j-ptiik.ub.ac.id

Implementasi *Multipath Routing* menggunakan Algoritme *Iterative Deepening Depth First Search* pada OpenFlow *Software-Defined Networking*

Ignatius Suryo Wicaksono[1], Primantara Hari Trisnawan[2]

Program Studi Teknik Informatika, Fakultas Ilmu Komputer, Universitas Brawijaya
Email: [1]ignatiussuryo15@gmail.com, [2]prima@ub.ac.id

Abstrak

Salah satu persoalan pada penerapan *multipath routing* adalah bagaimana menemukan jalur. Penelitian ini memanfaatkan algoritme *Iterative Deepening Depth First Search* (IDDFS) sebagai algoritme pencarian jalur. Kinerja algoritme IDDFS diukur dengan cara membandingkannya terhadap algoritme BFS dan DFS memakai parameter-parameter berikut: pencarian jalur, *multipath*, *path execution time*, *response time* dan *throughput* menggunakan topologi sederhana dan topologi kompleks, *fat tree*. Hasil pengujian menunjukkan bahwa ketiga algoritme, yaitu IDDFS, DFS, dan BFS, mampu membangun jalur serta menerapkan skema *multipath routing*. Kinerja IDDFS pada topologi sederhana masih di bawah kinerja dua algoritme lainnya. Hasil pengukuran dari *path execution time*: 0,0011355 ms (DFS), 0,001461 ms (BFS), 0,0018207 ms (IDDFS); *response time:* 147,8 ms (DFS), 145,9 ms (BFS), 146,9 ms (IDDFS), dan *throughput:* 16,63 Gbps (DFS); 16,79 Gbps (BFS); 16,71 Gbps (IDDFS). Namun, pada topologi kompleks, *fat tree*, diperoleh hasil sebaliknya. Hasil pengukuran untuk *path execution time*: 0,1277544 ms (DFS), 0,1760637 ms (BFS), 0,0006142 ms (IDDFS); *response time:* 2043,7 ms (DFS), 2396,1 ms (BFS), 298,3 ms (IDDFS), dan *throughput:* 12,964 Gbps (DFS), 12,533 Gbps (BFS), 14,78 Gbps (IDDFS).

Kata kunci: *multipath routing, path execution time, response time, throughput.*

Abstract

One of the main problems of the implementation of multipath routing is how to find the paths. In this research, Iterative Deepening Depth First Search (IDDFS) algorithm was used. The performance of IDDFS algorithm was measured by comparing IDDFS with other algorithms: Breadth First-Search (BFS) and Depth First-Search (DFS). Some parameters used to measure the performance of algorithms were path search, multipath, path execution time, response time, and throughput. The measurement was done by using simple topology and complex topology (fat tree). The measurement results showed that IDDFS, DFS, and BFS had ability to find paths and to implement multipath routing scheme. The IDDFS performance on simple topology was under the performance of other two algorithms. The measurement results of path execution time: 0.0011355 ms (DFS), 0.001461 ms (BFS), 0.0018207 ms (IDDFS); response time: 147.8 ms (DFS), 145.9 ms (BFS), 146.9 ms (IDDFS); and throughput: 16.63 Gbps (DFS), 16.79 Gbps (BFS), 16.71 Gbps (IDDFS). The IDDFS performance on complex topology was better. The measurement results of path execution time: 0.1277544 ms (DFS), 0.1760637 ms (BFS), 0.0006142 ms (IDDFS); response time: 2043.7 ms (DFS), 2396.1 ms (BFS), 298.3 ms (IDDFS); and throughput: 12.964 Gbps (DFS), 12.533 Gbps (BFS), 14.78 Gbps (IDDFS).

Keywords: *multipath routing, path execution time, response time, throughput.*

1. PENDAHULUAN

Metode pengiriman data tradisional, *singlepath routing*, menggunakan hanya satu jalur terbaik dan membiarkan jalur-jalur lain yang juga tersedia di dalam jaringan menjadi menganggur, *idle*, tidak termanfaatkan (Jiawei, Xiuquan, & Guoshun, 2018). Hal ini mengakibatkan distribusi trafik menjadi tidak seimbang. Oleh karena itu, sebagai alternatif lain dalam pengiriman data dengan tujuan mendistribusikan beban dan mengurangi *congestion* maka digunakan skema baru pengiriman data yaitu *multipath routing*. Hasil penelitian Chiang et. al. (2017) dalam *"A*

Multipath Transmission Sheme for the Improvement of Throughput over SDN" membuktikan bahwa *multipath routing* lebih baik daripada *singlepath routing*.

Penerapan *multipath routing* di sisi lain memunculkan tiga isu utama yaitu: pertama, bagaimana caranya membuat atau menyediakan jalur-jalur ini. Kedua, adalah berapa banyak jalur yang dibutuhkan. Dan ketiga, bagaimana mencegah terjadinya *bottleneck* (Lei, Wang, & Hsu, 2015). Satu dari tiga persoalan tersebut diambil untuk digunakan dalam penelitian ini yaitu bagaimana cara menyediakan jalur-jalur bagi skema *multipath routing*.

Beberapa algoritme telah digunakan/ dimodifikasi bagi keperluan ini. Dalam penelitian ini akan digunakan algoritme IDDFS sebagai algoritme bagi pencarian jalur-jalur dengan algoritme BFS serta DFS sebagai algoritme pembanding. Pengujian kemudian dilakukan dengan menggunakan parameter-parameter: *multipath*, *response time*, *path execution time* dan *throughput*. Dan topologi yang dipakai sebagai media pengujian adalah topologi sederhana serta topologi kompleks, *fat tree*.

Penerapan *multipath routing* sendiri telah dipermudah dengan hadirnya sebuah konsep baru dalam jaringan komputer yaitu *software defined network* (SDN), yaitu sebuah konsep dalam jaringan yang memisahkan *control plane* dan *data plane*. (Kreutz, et. al., 2015).

2. KAJIAN KEPUSTAKAAN

Penelitian yang dilakukan Chiang et. al. (2017) dalam *"A Multipath Transmission Scheme for the Improvement of Throughput over SDN"* membuktikan keunggulan *multipath routing* dari *singlepath routing*. Hal ini terlihat dari besar *throughput* yang dihasilkan oleh *multipath routing* lebih tinggi daripada singlepath routing.

Penelitian lain yang dilakukan Jiang et al. (2014) dalam "*Extending Dijkstra's Shortest Path Algorithm for Software Defined Networking*" memperlihatkan pengaruh faktor bobot terhadap besar *end-to-end latency*. Ketika bobot diterapkan pada *edge* juga *node* maka dihasilkan *end-to-end latency* terbaik.

Kemudian penelitian Wirawan et al. (2018) dalam "*Implementasi Multipath Routing Berbasis Algoritma DFS yang Dimodifikasi*" menunjukkan perbedaan kinerja algoritme-algoritme. Dari tiga algoritme yang digunakan: DFS, DFS yang dimodifikasi, dan Djikstra, secara umum, kinerja Djikstra lebih baik.

Terakhir penelitian Richard E. Korf (1985) dalam *"Depth-First Iterative-Deepening: An Optimal Admissible Tree Search"* menunjukkan kemampuan algoritme IDDFS dalam menyelesaikan persoalan yang berkaitan dengan kecerdasan buatan.

3. DASAR TEORI

3.1 *Multipath Routing*

Multipath routing terdiri atas dua kata yaitu *multipath* dan *routing*. *Multipath* bisa diartikan sebagai jalur-jalur. Dan *routing* adalah proses pemilihan jalur bagi sebuah trafik yang bergerak dari sumber (*source*) menuju tujuan (*destination*) di dalam sebuah jaringan atau diantara kumpulan jaringan (Wang et al., 2009). Sehingga *multipath routing* bisa diartikan sebagai protokol untuk menentukan jalur-jalur alternatif yang berbeda yang terdapat di dalam jaringan dalam pengiriman paket data dari *source* ke *destination*.

3.2 *Iterative Deepening Depth First Search*

IDDFS merupakan gabungan dari algoritma *Breadth-First Search* (BFS) dan *Depth-First Search* (DFS). IDDFS melakukan penelusuran node secara bertahap, mengeksplorasi kedalaman demi kedalaman (*depth by depth*) ke bawah secara iteratif sampai ditemukannya tujuan (*goal*). IDDFS bekerja berdasarkan prinsip LIFO (*Last In First Out*). Oleh karena itu, IDDFS juga menggunakan struktur data *stack*. Secara umum algoritme IDDFS dijabarkan pada Tabel 1.

Tabel 1. Pseudocode Algoritme IDDFS

```
1  function DLS(G, src, dst, lvl):
2   if src equals dst:
3    return [[src]]
4   if lvl less than or equal to 0:
5    return empty stack
6   for i in range(lvl):
7    path array to store paths
8    stack initialized as [(src, [src])]
9    while stack is not empty do:
10    pop stack, save to node, path
11    for next in (G[node]- path):
12     if next is dst:
13      append path + [next] to path
14     else if length of path less than i+1:
15      push (next, path + [next]) to stack
16    end for
17   end while
18   return path
19  end
20  function IDDFS (G, src, dst):
21   initialize valiabel lvl with 0
22   while lvl less than max_depth:
23    paths = DLS(src, dst, lvl)
24    intialize variable paths cost with length of path if
25  len(path) less then max_path else intialize with
26  max_path
27    if length of path less more or equal with
28  max_path
29     return break (stop looping)
30    if path cost not equal 2 and level is equal with
31  (max_depth – 1)
32     return break (stop looping)
33    lvl += 1
34   end while
35   return paths
36  end
```

3.3 *Depth First Search*

DFS melakukan penelusuran satu simpul bersebelahan pada kedalaman yang lebih besar pada cabang yang sama, diulang terus menerus sampai ditemukan simpul ujung, lalu dilakukan *backtracking*, dilanjutkan ekspansi simpul pada cabang lain yang belum dikunjungi hingga ditemukan seluruh jalur. Prinsip kerja DFS adalah LIFO dan diimplementasikan menggunakan struktur data *stack*. Secara umum algoritme DFS dijabarkan pada Tabel 2.

Tabel 2. Pseudocode Algoritme DFS

```
1  DFS (G,V) (V is vertex where the search start )
2   Stack S : = ();
3   for each vertex u, set set visited(u) := false ;
4   push S, v;
5   while (S is not empty) do
6    u : = pop S;
7    if (not visited[u]) then
8     visited[u] : = true;
```

Tabel 3. Pseudocode Algoritme IDDFS (Lanjutan)

```
9     for each unvisited neighbor w of u
10    push S, w;
11    end if
12   end while
13  END DFS ()
```

3.4 *Breadth First Search*

BFS bekerja dengan cara melakukan traversal terhadap semua simpul yang bersebelahan dengan simpul awal atau simpul saat ini secara melebar dan dilakukan secara bertahap kedalaman demi kedalaman (*depth by depth*). Algoritma BFS diimplementasikan menggunakan *queue data structure* menggunakan prinsip *Fisrt In First Out* (FIFO). Secara umum algoritme BFS dijabarkan pada Tabel 4.

Tabel 4. Pseudocode Algoritme BFS

```
1  Input: G(V, E)
2  Procedure: BFS(G, root)
3   Mark root as visited
4   Queue.push(root)
5   While queue is not empty:
6    v = queue.pop()
7    foreach child w of node v:
8     if (w is not visited):
9      mark w as visited
10     queue.push(w)
11    end if
12   end for
13  end while
14 END
```

3.5 *Software-Defined Networking*

Software-Defined Networking (SDN) adalah sebuah konsep atau paradigma baru dalam jaringan komputer yang memisahkan *control plane dan data plane*, sehingga jaringan yang semula cenderung bersifat terdistribusi menjadi lebih tersentralisasi, menghasilkan efisiensi pengaturan serta otomasi layanan jaringan (Astuto, Mendon,ca, Nguyen, Obraczka, & Turletti, 2014).

3.6 Openflow

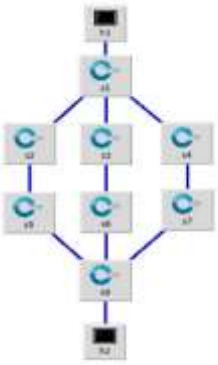
OpenFlow adalah protokol atau antarmuka komunikasi standar yang didefinisikan di antara lapisan kontrol (*control plane*) dan *forwading* (*data plane*) pada arsitektur SDN (Open Networking Foundation, 2016). OpenFlow menyediakan protokol terbuka guna memprogram *flow table* dari berbagai *router* dan *switch* yang berbeda. OpenFlow menyediakan cara baku serta terbuka bagi sebuah *controller* SDN berkomunikasi dengan OpenFlow *switch* (McKeown, et al., 2008).

Secara umum, cara kerja OpenFlow adalah sebagai berikut: ketika sebuah OpenFlow *switch* menerima paket yang belum pernah ditemui sebelumnya, dan tidak memiliki entri *flow* yang cocok, maka *switch* tersebut akan mengirimkan kembali paket ini ke *controller*. Selanjutnya *controller* akan memutuskan cara menangani paket tersebut. *Controller* dapat melakukan drop terhadap paket, atau dapat menambahkan entri *flow* sehingga *switch* mengerti bagaimana harus meneruskan paket yang serupa nantinya (Open Networking Foundation, 2016).

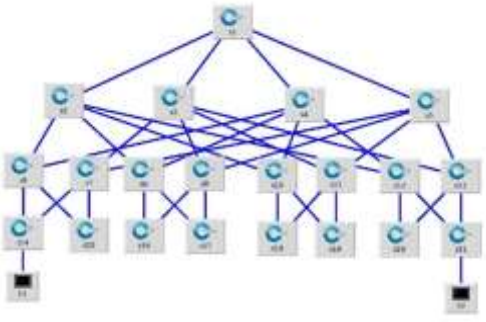
4. PERANCANGAN DAN IMPLEMENTASI

4.1 Perancangan Topologi

Dua tipe topologi yang dirancang di sini yaitu topologi sederhana dan topologi kompleks, fat tree. Kedua topologi mampu mendukung skema *multipath routing*. Gambar 1 menunjukan topologi sederhana dan Gambar 2 menunjukan topologi kompleks.



Gambar 1. Topologi Sederhana



Gambar 2. Topologi Fat Tree

4.2 Perancangan Algoritme Pencarian Jalur

Proses pencarian jalur pada penelitian ini menggunakan algoritme *Iterative Deepening Depth First Search* (IDDFS) yang akan dibandingkan dengan algoritme DFS dan algoritme BFS. Algoritme IDDFS secara teori memiliki keunggulan dibandingkan algoritme DFS dan BFS karena algoritme IDDFS menggabungkan kelebihan dari kedua algoritme tersebut. Penelusuran node pada algoritme IDDFS dilakukan bertahap, mengeksplorasi kedalaman demi kedalaman (*depth by depth*) ke bawah secara iteratif hingga batas kedalaman tertentu. Hal ini membuat algoritme IDDFS bekerja lebih optimal dibandingkan algoritme DFS dan BFS apabila node tujuan berada tepat dalam jangkauan kedalaman yang sudah ditentukan. Algoritme IDDFS menggunakan prinsip LIFO (*Last In First Out*) sama seperti algoritme DFS.

4.3 Perancangan Metrik Penilaian Jalur

Jalur yang dihasilkan di sini adalah jalur tanpa bobot. Oleh karena itu diperlukan bobot untuk memberikan *cost* dari setiap traversal. Pada penelitian ini pembobotan *link* dari jalur dihitung menggunakan rumus Cisco OSPF yang ditunjukkan oleh Persamaan (1) di bawah ini:

$$ew = \frac{B_R}{B_L} \quad (1)$$

dimana *ew* adalah *edge weight* (*link cost*), B_R adalah *reference bandwidth* yang besarnya 100 Mbps sesuai pada protokol OSPF dari Cisco (Cisco, 2005). Dan B_L adalah *link bandwidth*.

4.4 Perancangan Algoritme Pemilihan Jalur

Tidak semua jalur yang ditemukan akan digunakan dalam skema *multipath routing*. Jalur-jalur yang dipilih adalah jalur-jalur yang memiliki *cost* (pw(p)) kecil. Nilai pw(p) ditentukan melalui Persamaan (2) di bawah ini:

$$pw(p) = \sum_{e \in E} ew[e] \quad (2)$$

Banyaknya jalur yang hendak digunakanpun dibatasi tergantung dari nilai batasan yang diberikan dalam skema *multipath*, bisa 2 jalur, 3 jalur dan seterusnya. Bobot kualitatif jalur (w(p)) dihitung dengan menentukan persentase *pw(p)* dari masing-masing jalur melalui Persamaan (3) di bawah ini:

$$w(p) = \left(1 - \frac{pw(p)}{\sum_{i<n}^{i=0} pw(i)}\right) \times 10 \quad (3)$$

4.5 Perancangan Skenario Uji

Untuk mengetahui kinerja dari masing-masing algoritme dalam skema *multipath routing*, pada penelitian ini digunakan dua buah tipe topologi jaringan. Dua buah tipe topologi jaringan tersebut adalah topologi sederhana dan topologi kompleks (topologi *Fat tree*). Pada masing-masing topologi akan diujikan parameter-parameter uji, yaitu pencarian jalur, *multipath*, *path execution time*, *response time*, dan *throughput*.

Pada topologi sederhana dan topologi *Fat Tree*, parameter pencarian jalur, *multipath*, dan *throughput* akan diuji dengan menggunakan *iperf*. Parameter pencarian jalur dan *multipath* akan diuji secara bersamaan dengan melakukan *iperf* dengan host h1 sebagai *client* dan host h2 sebagai *server*. *Iperf* dilakukan selama 15 detik dengan jumlah koneksi sebanyak 5 buah. Pada parameter *throughput iperf* dilakukan dengan 2 kondisi yaitu pertama *iperf* dilakukan dengan mengubah jumlah koneksi secara tetap dalam waktu 60 detik; dan kedua menggunakan jumlah koneksi yang tetap dalam waktu 60 detik yang dilakukan sebanyak 10 kali. Parameter *path execution time*, dan *response time* akan diuji dengan, melakukan *ping* dari host h2 menuju host h1.

4.6 Implementasi

Implementasi dilakukan menggunakan lingkungan emulasi jaringan mininet dan memanfaatkan miniedit. Kontroler yang digunakan adalah Ryu dengan bahasa pemrograman Python.

5. PENGUJIAN DAN ANALISA

5.1 Topologi Sederhana

5.1.1 Pencarian jalur

Pengujian pencarian jalur dilakukan untuk mengetahui kemampuan sistem dalam menemukan jalur-jalur dari *host* sumber menuju *host* tujuan. Pengujian dilakukan dengan menggunakan *iperf* pada h1 (*client*) dan h2 (*server*) selama 15 detik dengan jumlah koneksi *client* sebanyak 5 buah.

Tabel 5. Hasil Pencarian Jalur dari Masing-Masing Algoritme Topologi Sederhana

	BFS	DFS	IDDFS
Total jalur yang ditemukan	[1, 2, 5, 9], [1, 3, 6, 9,], [1, 4, 7, 9]	[1, 4, 7, 9,], [1, 3, 6, 9], [1, 2, 5, 9]	[1, 4, 7, 9], [1, 3, 6, 9], [1, 2, 5, 9]
Jalur-jalur yang dipilih	[1, 2, 5, 9], [1, 3, 6, 9,]	[1, 4, 7, 9,], [1, 3, 6, 9]	[1, 4, 7, 9], [1, 3, 6, 9]

Hasil pengujian pencarian jalur seperti diperlihatkan oleh Tabel 5 di atas menunjukkan bahwa ketiga algoritme mampu membangun jalur-jalur antara *host client* dan *host server*. Dari jalur-jalur yang telah ditemukan tersebut kemudian dipilih beberapa jalur yang digunakan dalam skema *multipath* berdasarkan Persamaan (3). Berdasarkan persamaan tersebut jalur-jalur dengan bobot kumulatif terkecil yang dipilih. Sehingga dihasilkan 2 jalur pada masing-masing algoritme sesuai dengan nilai *max path* yang telah diatur.

5.1.2 *Multipath*

Pengujian *multipath* dilakukan untuk mengetahui kemampuan sistem dalam memanfaatkan jalur-jalur yang sudah dipilih untuk digunakan pada skema *multipath* seperti telah ditunjukkan pada Tabel 5 sebelumnya. Pengujian dilakukan dengan melihat besar paket yang melalui jalur-jalur yang telah dipilih.

Pada algoritme BFS, besar paket yang dikirimkan dilihat pada *switch* s1 yang dikirimkan keluar melalui *port* 2 dan *port* 3. Sedangkan untuk algoritme DFS dan IDDFS besar paket yang dikirimkan dilihat juga pada *switch* s1 dengan *port* 3 dan *port* 4 sebagai port pengiriman paket keluar.

Tabel 6. Hasil Pengujian Multipath pada Topologi Sederhana

Algoritme	Switch s1		
	port 2(tx)	port 3(tx)	port 4(tx)
BFS	413730	276096	-
DFS	-	123823	499096
IDDFS	-	488299	120225

Dari Tabel 6 di atas terlihat bahwa paket yang dikirimkan bisa didistribusikan pada masing-masing jalur yang telah dipilih bagi skema *multipath*. Hal ini menunjukkan bahwa skema *multipath* pada masing-masing algoritme bisa berfungsi.

5.1.3 *Response time*

Pada pengujian *respone time*, pengamatan dilakukan pada *round trip time* dari pengiriman *ping* paket pertama saat program pertama kali dijalankan. Pengujian *response time* dilakukan sebanyak 10 kali pengujian.

Tabel 7. Hasil Pengujian *Response Time* pada Topologi Sederhana

Pengujian	DFS (ms)	BFS (ms)	IDDFS (ms)
1	142	153	142
2	148	147	144
3	153	141	146
4	146	143	146
5	155	156	150
6	144	150	145
7	148	141	144
8	154	138	143
9	147	141	162
10	141	149	147
Rata-Rata	147,8	145,9	146,9

Berdasarkan Tabel 7 di atas terlihat bahwa hasil pengujian *response time* algoritme BFS memberikan rata-rata *response time* yang lebih baik dibandingkan kedua algoritme lainnya, ditunjukkan oleh nilai *response time* BFS yang lebih rendah yaitu 145,9 ms dibandingkan IDDFS (146,9 ms) dan DFS (147,8 ms). Hasil ini menunjukkan bahwa proses *filtering* pada algoritme BFS berlangsung lebih cepat atau lebih baik.

5.1.4 *Path Execution Time*

Tabel 8. Hasil Pengujian *Path Execution Time* pada Topologi Sederhana

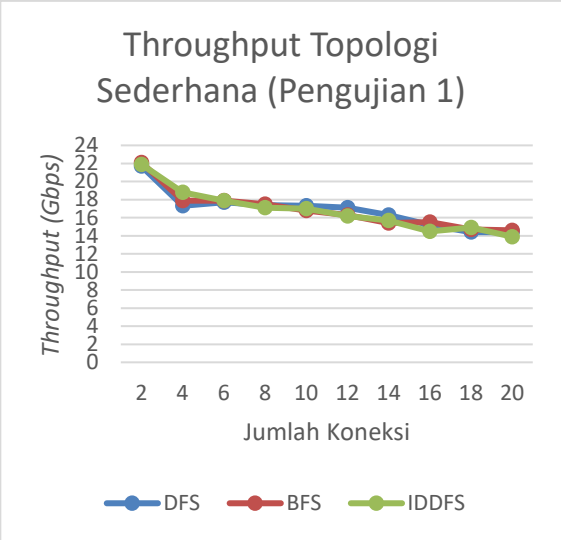
Pengujian	DFS (ms)	BFS (ms)	IDDFS (ms)
1	0,000185	0,000266	0,000172
2	0,000138	0,000128	0,000171
3	0,003307	0,005096	0,016587
4	0,000136	0,006221	0,000208
5	0,002487	0,000166	0,000173
6	0,000191	0,000138	0,000185
7	0,003355	0,000149	0,000181
8	0,000891	0,000136	0,000174
9	0,000436	0,000135	0,000183
10	0,000229	0,002175	0,000173
Rata-Rata	0,0011355	0,001461	0,0018207

Dari Tabel 8 di atas didapatkan hasil bahwa algoritme DFS lebih cepat dalam membangun jalur. Rata-rata untuk DFS adalah 0,001135 ms, lebih rendah jika dibandingkan dengan algoritme BFS yang rata-ratanya 0,001461 ms dan algoritme IDDFS yang rata-ratanya 0,001820 ms. Hal ini membuktikan optimalitas waktu pencarian jalur yang lebih baik dari algoritme DFS pada topologi sederhana.

5.1.5 *Throughput*

Pengujian *throughput* dilakukan dua kali. Pada pengujian pertama jumlah koneksi dinaikkan secara linier dan dilakukan selama 60 detik. Hasil pengujian pertama dapat dilihat pada Gambar 6.

Sedangkan pada pengujian kedua jumlah koneksi diatur pada nilai 10 dan dilakukan selama 60 detik. Hasil pengujian kedua dapat dilihat pada Tabel 6 merupakan contoh pengujian *throughput* dari sisi *client* (h1).



Gambar 3. Hasil *Throughput* untuk Jumlah Koneksi yang Berubah (t = 60s) Topologi Sederhana

Tabel 9. Hasil *Throughput* untuk Jumlah Koneksi 10 (t = 60s) Topologi Sederhana

Pengujian	DFS	BFS	IDDFS
1	16,3	16,4	16,6
2	16,1	16,7	16,5
3	16,3	16,1	17,1
4	16,4	16,7	16,3
5	16	14,5	16,4
6	19,3	15,6	17
7	16,3	16,8	17,2
8	16,8	19,1	16,2
9	16,6	18	16,9
10	16,2	18	16,9
Rata-Rata	16,63	16,79	16,71
Standar Deviasi	0,9661	1,3152	0,3541

Dari Gambar 3 hasil pengujian *throughput* terlihat bahwa ketiga algoritme menunjukkan penurunan *throughput* ketika koneksi berubah secara linier dari 2 hingga 20. Sedang pada Tabel 9 terlihat bahwa rata-rata *throughput* yang dihasilkan algoritme BFS (16,79 Gbps) lebih besar daripada rata-rata *throughput* IDDFS (16,71 Gbps) dan DFS (16,63 GBps).

5.2 Topologi Fat Tree

5.2.1 Pencarian jalur

Pengujian ini dilakukan berdasarkan rancangan topologi *fat tree* seperti terlihat pada Gambar 5. Topologi menggunakan besar *link bandwidth* yang sama yaitu 100 *Mbps*.

Tabel 10. Hasil Pencarian Jalur Masing-Masing Algoritme Topologi *Fat Tree* (Max Path = 2)

	BFS	DFS	IDDFS
Total jalur yang ditemukan	Banyak	Banyak	[14, 7, 5, 13, 21], [14, 7, 3, 13, 21], [14, 6, 4, 12, 21], [14, 6, 2, 12, 21]
Jalur-jalur yang dipilih	[14, 6, 2, 12, 21], [14, 6, 4, 12, 21]	[14, 7, 5, 13, 21], [14, 7, 3, 13, 21]	[14, 7, 5, 13, 21], [14, 7, 3, 13, 21]

Tabel 11. Hasil Pencarian Jalur dari Masing-Masing Algoritme Topologi *Fat Tree* (Max Path = 3)

	BFS	DFS	IDDFS
Total jalur yang ditemukan	Banyak	Banyak	[14, 7, 5, 13, 21], [14, 7, 3, 13, 21], [14, 6, 4, 12, 21], [14, 6, 2, 12, 21]
Jalur-jalur yang dipilih	[14, 6, 2, 12, 21], [14, 6, 4, 12, 21], [14, 7, 3, 13, 21]	[14, 7, 5, 13, 21], [14, 7, 3, 13, 21], [14, 6, 4, 12, 21]	[14, 7, 5, 13, 21], [14, 7, 3, 13, 21], [14, 6, 4, 12, 21]

Hasil pengujian, seperti diperlihatkan oleh Tabel 10 dan Tabel 11 di atas, menunjukkan bahwa ketiga algoritme mampu membangun jalur-jalur antara host *client* dan *host server*. Sama seperti pada topologi sederhana, dari jalur-jalur yang telah ditemukan tersebut kemudian dipilih beberapa jalur yang digunakan dalam skema *multipath* berdasarkan Persamaan 4.3. Berdasarkan persamaan tersebut jalur-jalur dengan bobot kumulatif terkecil yang dipilih. Sehingga masing-masing algoritme menghasilkan 2 jalur pada Tabel 10 dan 3 jalur pada Tabel 11 sesuai dengan nilai *max path* yang telah diatur.

5.2.2 *Multipath*

Pengujian *multipath* dilakukan sama seperti pengujian yang dilakukan pada topologi sederhana. Pengujian ini digunakan untuk mengetahui kemampuan sistem dalam memanfaatkan jalur-jalur yang sudah dipilih untuk digunakan pada skema *multipath* seperti telah ditunjukkan pada Tabel 10 sebelumnya. Pengujian dilakukan dengan melihat besar paket yang melalui jalur-jalur yang telah dipilih.

Pada algoritme BFS, besar paket yang dikirimkan dilihat pada *switch* s6 yang dikirimkan keluar melalui *port* 1 dan *port* 2. Sedangkan untuk algoritme DFS dan IDDFS besar paket yang dikirimkan dilihat juga pada *switch* s7 dengan *port* 1 dan *port* 2 sebagai port pengiriman paket keluar.

Tabel 12. Hasil Pengujian Multipath pada Topologi *Fat Tree*

Algoritme	Switch s6		Switch s7	
	port 1(tx)	port 2(tx)	port 1(tx)	port 2(tx)
BFS	154239	235726	-	-
DFS	-	-	222674	149270
IDDFS	-	-	98077	144217

Dari Tabel 12 di atas terlihat bahwa paket yang dikirimkan bisa didistribusikan pada masing-masing jalur yang telah dipilih bagi skema *multipath*. Hal ini menunjukkan bahwa skema *multipath* pada masing-masing algoritme bisa berfungsi

5.2.3 *Response time*

Pada pengujian *respone time*, dilakukan pengamatan pada *round trip time* dari pengiriman *ping* paket pertama dari h2 ke h1 saat program pertama kali dijalankan. Pengujian *response time* dilakukan sebanyak 10 kali menggunakan topologi *fat tree*. Tabel 10 menunjukkan hasil pengujian *response time* dari masing-masing algoritme.

Tabel 13. Hasil Pengujian *Response Time* pada Topologi *Fat Tree*

Pengujian	DFS (ms)	BFS (ms)	IDDFS (ms)
1	2895	1003	355
2	1148	2659	220
3	1928	2440	191
4	2186	2932	245
5	1374	2489	392
6	2073	1614	350
7	2734	2513	200
8	2141	2937	378
9	2366	2465	278
10	2092	2909	374
Rata-Rata	2093,7	2396,1	298,3

Tabel 14. Hasil Pengujian *Response Time* pada Topologi *Fat Tree* (Lanjutan)

Tabel 13 dan Tabel 14 hasil pengujian *response time* di atas menunjukkan bahwa algoritme IDDFS memberikan rata-rata *response time* yang lebih baik dibandingkan kedua algoritme lainnya, seperti ditunjukkan oleh nilai *response time* IDDFS, 298,3 ms, adalah yang terendah dibandingkan dua algoritme lainnya. Hasil ini menunjukkan bahwa proses *filtering* pada algoritme IDDFS berlangsung lebih cepat atau lebih baik di sini, di topologi *fat tree*.

5.2.4 *Path Execution Time*

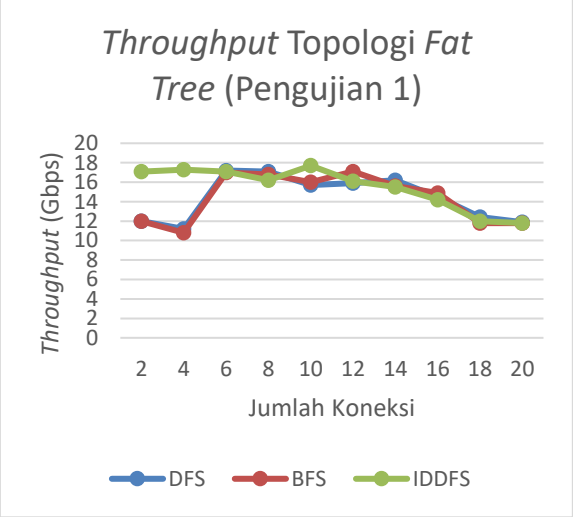
Tabel 15. Hasil Pengujian *Path Execution Time* pada Topologi *Fat Tree*

Pengujian	DFS (ms)	BFS (ms)	IDDFS (ms)
1	0,123735	0,166643	0,000642
2	0,122642	0,173979	0,000603
3	0,124211	0,169041	0,000578
4	0,172266	0,16474	0,000571
5	0,123038	0,166191	0,000646
6	0,124416	0,171734	0,000545
7	0,117571	0,216803	0,000535
8	0,127539	0,185507	0,000705
9	0,122591	0,175499	0,000628
10	0,119535	0,1705	0,000689
Rata-Rata	0,1277544	0,1760637	0,0006142

Tabel 15 di atas memperlihatkan bahwa algoritme IDDFS jauh lebih cepat dalam membangun jalur, dengan rata-rata 0,0006142 ms, jika dibandingkan dengan dua algoritme lain yaitu: DFS dengan rata-rata 0,1277544 ms dan BFS dengan rata-rata 0,1760637 ms. Hal ini membuktikan bahwa optimalitas waktu pencarian yang lebih baik dari algoritme IDDFS pada topologi *fat tree*.

5.2.5 *Throughput*

Pengujian *throughput* dilakukan dua kali. Pada pengujian pertama jumlah koneksi dinaikkan secara linier dan dilakukan selama 60 detik. Hasilnya dapat dilihat pada Gambar 4. Sedangkan pada pengujian kedua jumlah koneksi diatur pada nilai 10 dan dilakukan selama 60 detik. Hasilnya terlihat pada Tabel 16.



Gambar 4. Hasil *Throughput* untuk Jumlah Koneksi yang Berubah (t =60s) Topologi *Fat Tree*

Tabel 16. Hasil *Throughput* untuk Jumlah Koneksi 10 (t =60s) Topologi *Fat Tree*

Pengujian	DFS	BFS	IDDFS
1	7,54	8,28	12,5
2	7,46	8,1	15,3
3	14,6	7,05	15
4	15,7	13,7	14,8
5	15,8	14,9	14,5
6	15,2	14,2	15,6
7	13,5	14,5	14,7
8	9,74	15,5	14,9
9	15,3	14,4	15,3
10	14,8	14,7	15,2
Rata-Rata	12,964	12,533	14,78
Standar Deviasi	3,3736	3,3066	0,8651

Gambar 4 hasil pengujian *throughput* di atas menunjukkan bahwa ketiga algoritme mulai menunjukkan penurunan *throughput*. Sedang pada Tabel 16 terlihat bahwa rata-rata *throughput* yang dihasilkan algoritme IDDFS (14,78 Gbps) lebih besar daripada rata-rata *throughput* DFS (12,964 Gbps) dan BFS (12,533 Gbps).

6. KESIMPULAN

Berdasarkan hasil pengujian serta analisa bisa didapatkan kesimpulan sebagai berikut:

1. Ketiga algoritme, DFS-BFS-IDDFS mampu membangun jalur dan mampu menerapkan skema *multipath routing*.
2. Pada topologi sederhana kinerja algoritme IDDFS sedikit tertinggal bila dibandingkan dengan kinerja algoritme BFS, dalam hal *response time* (BFS = 145,9 ms, IDDFS = 146,9 ms, DFS = 147,8 ms). Dan untuk *throughput*, BFS juga memberikan hasil lebih baik yaitu 16,79 Gbps dibandingkan IDDFS = 16,71 Gbps dan DFS = 16,63 Gbps. Hasil ini menunjukkan bahwa pada topologi yang sederhana algoritme BFS bisa menjadi pilihan yang lebih baik dibandingkan algoritme IDDFS dan DFS.
3. Pada topologi yang kompleks, fat tree, hasil yang diperoleh berubah. Algoritme IDDFS menunjukkan kinerja yang lebih baik, dibandingkan dua algoritme lainnya yaitu BFS dan DFS. *Response time* IDDFS = 298,3 ms lebih baik daripada DFS = 2093,7 ms dan BFS = 2396,1 ms. Demikian juga throughput IDDFS = 14,78 Gbps lebih baik daripada DFS = 12,964 Gbps dan BFS = 12,533 Gpbs. Hasil ini menunjukkan bahwa dalam jaringan yang kompleks, *fat tree*, algoritme IDDFS lebih tepat diterapkan daripada algoritme BFS dan DFS.

7. DAFTAR PUSTAKA

Austuto, B. N., Mendon,ca, M., Nguyen, X. N., Obraczka, K., & Turletti, T. (2014). *A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks, Communications Surveys and Tutorials. IEEE Communications Surveys & Tutorials.* 16.10.1109/SURV.2014.012214.00186

Chiang, Y. R., Ke, C. H., Yu, Y. S., Chen, Y. S., Pan, C. J. (2017). *A multipath Transmission Scheme for the Improvement of Throughput over SDN. Proceedings of the 2017 IEEE International Conference on Applied System innovation*, 1247-1250.

Heap, D., (2002). *Depth First Search (DFS)*. [online] Tersedia di: <http://www.cs.toronto.edu/~heap/270F0

2/node36.html> [Diakses 6 April 2019]

Jiang, J. R., Huang, H. W., Liao, J. H., and Chen, S. Y. (2014). *Extending Dijkstra's Shortest Path Algorithm for Software Defined Networking. The 16th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, 1-4.

Jiang Zhenghong. (2019) *Algorithm*. [online] Tersedia di https://jiang-zhenghong.github.io/blogs/algorithm.html

Korf, Richard E. (1985). *Depth-First Iterative-Deepening: An Optimal Admissible Tree Search. Artificial Intelligence, 27*, 97-109.

Kreutz, D., Verissimo, P.E., Azodolmolky, S., Ramos, F. M., Rothenberg, C.E., & Uhlig, S. (2015). *Software-Defined Networking: A Comphrehensive Survey. Proceedings of the IEEE*. IEEE.

Lei, Y.-C., Wang, K., & Hsu, Y.-H. (2015). *Multipath Routing in SDN-Based Data Center Networks. European Conference on Networks and Communications (EuCNC)*, 365-369.

Maulana, W.,Yahya,W & Akbar, S (2017). *Multipath Routing dengan Load-Balancing pada OpenFlow Software-Defined Network. Jurnal Pengembangan Teknologi Informasi Dan Ilmu Komputer*.

Mininet Team. (2016). *Mininet Overview*. Diambil kembali dari Mininet: http://mininet.org/overview/

Open Networking Foundation. (2016). *OpenFlow*. Diambil kembali dari Open Networking Foundation: https://www.opennetworking.org/sdn-resource/openflow

OpenvSwitch. (2016). *How to Build Debian Packages for OpenvSwitch*. Diambil kembali dari OpenvSwitch: http://openvswitch.org/support/dist-docs-2.5/INSTALLDebian.md.html

Poole, David & Alan Mackworth. (2018). *Artificial Intelligence: Foundations of Computational Agents, 2nd Edition*. Cambridge University Press

R, Gayatri. (2019). Comparative Analysis of Various Uninformed Searching Algorithms in AI. *International Journal of Computer Science and Mobile Computing, 8*, 57 – 56.

Rouse, M. (2012, November). SDN Controller (Software-Defined Netwroking Controller). Diambil kembali dari TechTarget: SearchSDN: http://searchsdn.techtarget.com/definition/SDN-controller-software-defined-networking-controller

Rouse, M. (2013, March). OpenFlow Controler. Diambil kembali dari TechTarget: SearchSDN: http://searchsdn.techtarget.com/definition/OpenFlow-controller

Tri Wirawan, U., Yahya, W., & Basuki, A. Implementasi Multipath Routing Berbasis Algoritme DFS yang Dimodifikasi. Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer, vol. 2, no. 11, p. 4741-4749, mei 2018. ISSN 2548-964X. Tersedia pada: <http://j-ptiik.ub.ac.id/index.php/j-ptiik/article/view/3062>. Tanggal Akses: 25 mar. 2019.