JURNAL RESISTOR | ISSN 2598-7542 | E-ISSN 2598-9650
Vol. 7 No 1 – Agustus 2024 | https://s.id/jurnalresistor
DOI: -
Publishing: Prahasta Publisher

Perbandingan Algoritma Dijkstra dan Floyd-Warshall Menggunakan Software Defined Network untuk Rute Terpendek

I Made Adi Bhaskara[1], I Made Surya Kumara[2], I Gede Wira Darma[3], I Kadek Agus Wahyu Raharja[4]

[1234]Program Studi Teknik Komputer, Fakultas Teknik dan Perencanaan, Universitas Warmadewa
Jl. Terompong No.24, Sumerta Kelod, Kec. Denpasar Tim., Kota Denpasar, Bali, Indonesia

e-mail: madeadi.bhaskara@warmadewa.ac.id[1], madesurya.kumara@warmadewa.ac.id[2], gede.wiradarma@warmadewa.ac.id[3], raharja.wahyu.agus.kadek@gmail.com[4]

Received : June, 2024	Accepted : August, 2024	Published : August, 2024

Abstract

The development of network technology has led to the emergence of Software Defined Network (SDN), a network architecture that is managed and controlled through centralized software. With SDN, network administrators can accelerate connections and manage network traffic from a central location without having to access the hardware directly. OpenFlow is a communication protocol used to provide access between the forwarding plane on switches and the controller, enabling full network monitoring and control. Dijkstra's Algorithm is commonly used in SDN to determine the shortest path, but the Floyd-Warshall Algorithm can also be utilized with a dynamic programming approach. This study was conducted to compare these two algorithms in obtaining the shortest path in SDN. First, the Floyd-Warshall Algorithm was modified to resemble Dijkstra's Algorithm on the controller. Then, three network topology schemes were constructed using Mininet, each consisting of two hosts (source and destination), one controller, and several switches. The testing was conducted on the controller with different algorithms using POX tools across the three network topology schemes. This research found that, for obtaining the shortest path in SDN, Dijkstra's Algorithm proved to be superior to the Floyd-Warshall Algorithm.

Keywords: *Software Defined Network (SDN), OpenFlow, mininet, POX, Controller, Floyd-Warshall algorithm, Dijkstra algorithm*

Abstrak

Perkembangan teknologi jaringan yang pesat telah mendorong lahirnya arsitektur Software Defined Network (SDN) yang memungkinkan mengatur dan mengontrol jaringan melalui perangkat lunak terpusat. Dengan SDN, administrator jaringan dapat mempercepat koneksi dan mengelola lalu lintas jaringan dari satu lokasi pusat tanpa harus mengakses perangkat keras secara langsung. Openflow adalah protokol komunikasi yang memungkinkan pengawasan dan kontrol penuh terhadap jaringan. Urgensi dari penelitian ini terletak pada kebutuhan untuk menentukan jalur terpendek secara efisien dalam SDN agar mendapatkan performa jaringan yang terbaik. Algoritma Dijkstra sering digunakan dalam SDN untuk menentukan jalur terpendek, namun Algoritma Floyd-Warshall juga dapat digunakan dengan pendekatan pemrograman dinamis. Penelitian dilakukan untuk membandingkan kedua algoritma ini dalam memperoleh jalur terpendek pada SDN. Metode yang digunakan yakni pertama, Algoritma Floyd-Warshall diubah menjadi Algoritma Dijkstra pada controller. Kemudian, dibangun tiga skema topologi jaringan menggunakan mininet, masing-masing terdiri dari dua host (host awal dan tujuan), satu controller, dan beberapa switch. Pengujian dilakukan pada controller dengan algoritma berbeda menggunakan tools POX

pada tiga skema topologi jaringan. Pada penelitian ini diperoleh untuk mendapatkan jalur terpendek pada SDN, Algoritma Dijkstra terbukti lebih unggul dibandingkan Algoritma Floyd-Warshall. Dengan ini menunjukkan bahwa pemilihan algoritma yang tepat pada controller SDN dapat meningkatkan efisiensi jaringan.

Kata Kunci: *Software Defined Network (SDN), OpenFlow, mininet, POX, Controller, Algoritma Floyd-Warshall, Algoritma Dijkstra*

1. PENDAHULUAN

Dalam dunia informatika, perkembangan teknologi yang cepat memungkinkan manusia untuk mendapatkan perangkat yang lebih canggih dan *user-friendly*. Selain itu, perangkat yang dikembangkan untuk memenuhi kebutuhan manusia menjadi semakin sederhana, menggantikan perangkat lama yang biasanya lebih rumit. Salah satu inovasi tersebut adalah Software Defined Network (SDN). Dalam jaringan tradisional, penggabungan antara *data plane* dan *control plane* membuat penerusan data, pemrosesan, dan pengelolaan jaringan menjadi sulit dan kompleks. Setiap switch mengambil keputusannya sendiri, yang membuat jaringan menjadi terdesentralisasi secara logis. Untuk mengatasi keterbatasan dalam jaringan tradisional, para insinyur mengembangkan model jaringan baru yang dikenal sebagai Software Defined Network (SDN) [1]. Berdasarkan penelitian Nugroho dan Rakhmatsyah [2], Algoritma Floyd-Warshall unggul dibandingkan Algoritma Johnson untuk menemukan jalur optimal dalam pencarian backup Ketika terjadi kegagalan tautan dalam topologi jellyfish. Oleh sebab itu, Algoritma Floyd-Warshall perlu untuk dibandingkan dengan Algoritma Dijkstra yang sering digunakan di Software Defined Network untuk mendapatkan algoritma yang lebih unggul dalam pencarian jalur terpendek. Pada penelitian ini menggunakan SDN yang memiliki keunggulan yakni memiliki arsitektur yang terpusat sehingga menyediakan visibilitas jaringan untuk membantu dalam pemanfaatan sumber daya yang efisien dan kinerja tinggi [3]. Selain itu, SDN dalam penyelesaian suatu masalah lebih adaptif, dinamis, manageable, dan programmable [4].

Software Defined Network merupakan teknologi jaringan komputer yang sangat fleksibel karena pengaturannya terpusat melalui perangkat lunak. Dari pengembangan di Universitas Stanford, SDN memungkinkan administrator untuk meningkatkan kecepatan proses pengaturan koneksi jaringan dan mengendalikan lalu lintas secara terpusat tanpa perlu mengakses perangkat keras secara langsung. SDN juga memungkinkan bebas dari ketergantungan pada vendor atau produk khusus dalam penerapan jaringan, menghasilkan solusi jaringan yang lebih serbaguna.

Dalam melakukan implementasi, *Software Defined Network* digunakan untuk berbagai pekerjaan seperti *forwarding* pada layer 2 dan 3, routing pada layer 3, firewall, *Spanning Tree Protocol*, pemantauan jaringan, penemuan jaringan, *load balancing IP*, dan lainnya. Pada implementasi ini, fokusnya adalah pada proses routing di layer 3, yang dikendalikan oleh sebuah controller dalam SDN. Di dalam *script controller* yang bernama 12_multi.py terdapat Algoritma Floyd-Warshall yang dipakai untuk mencari jalur terpendek . Selain itu, diketahui algoritma lain untuk menentukan rute terpendek yaitu Algoritma Dijkstra. Sehingga perlunya membandingkan Algoritma Floyd-Warshall dengan Algoritma Dijkstra pada penerapan proses routing di SDN untuk menentukan algoritma mana yang paling baik digunakan untuk menentukan rute terpendek.

3. MODEL ANALISIS, DESAIN, DAN IMPLEMENTASI

Pada penelitian penentuan jalur terpendek [5] mengatakan penelitian lebih lanjut diperlukan untuk menggunakan algoritma selain Algoritma A* dalam penentuan rute terpendek. Pada penelitian yang lain juga, Algoritma Dijkstra dan Algoritma Floyd-Warshall dilakukan perbandingan dengan menggunakan NS2 (Network Simulator 2) [6]. Berdasarkan penelitian sebelumnya tersebut penulis memutuskan untuk mengimplementasikan Algoritma Floyd-Warshall dan Algoritma Dijkstra menggunakan SDN.

3.1 Software Defined Network (SDN)

Software Defined Network (SDN) merupakan suatu konsep yang tergolong baru dalam

memecahkan masalah jaringan dengan memisahkan *control plane* dan *data plane* pada perangkat yang berbeda [7]. SDN merupakan arsitektur tiga lapis yang masih berkembang yang mendefinisikan lapisan data, kontrol, dan aplikasi. *data planes* dan *control planes* masing-masing mengimplementasikan fungsi *forwarding* dan *routing*. *application planes* berisi proses komunikasi [8]. Software pengatur jaringan bernama NOX diciptakan sebagai prototipe awal. NOX, Salah satu pionir SDN dikembangkan menggunakan bahasa pemrograman C. Upaya awal untuk mengembangkan SDN dimulai di Stanford University pada tahun 2008, dengan penerapannya dalam jaringan kampus internal. Sudah banyak teknologi SDN yang sudah diajukan dan diterapkan untuk memudahkan dalam mendefinisikan dan memanajemen sumber jaringan yang dinamis [9]. Publikasi jurnal dari hasil implementasi ini memperkenalkan konsep openflow sebagai salah satu bentuk topologi SDN.

3.2 OpenFlow

Openflow adalah protokol manajemen dan pemantauan perangkat jaringan, serta spesifikasi switch, yang memungkinkan penelitian, eksperimen, dan penerapan jaringan yang ditentukan oleh perangkat lunak pada umumnya, dan jaringan pusat data pada khususnya [10]. Openflow adalah protokol komunikasi yang digunakan dalam SDN, menjadi standar industri yang memfasilitasi akses antara *forwarding plane* dari switch atau router dalam jaringan dengan *network controller*. Brocade menyediakan produk yang mendukung atau kompatibel dengan openflow di seluruh lini switch dan router mereka, seperti router seri brocade MLX, switch brocade ICX, dan switch brocade VDX. Dukungan penuh untuk OpenFlow memastikan bahwa organisasi dapat memaksimalkan manfaat SDN di pusat data, WAN, dan jaringan kampus.

3.3 Mininet

Mininet adalah platform yang memungkinkan pembuatan jaringan besar menggunakan sumber daya dari satu sistem kecil atau mesin virtual. Dengan Mininet, pengguna dapat dengan cepat membuat jaringan virtual yang realistis, menjalankan kernel, switch, dan aplikasi perangkat lunak pada komputer pribadi. Platform ini memudahkan pembuatan, interaksi, penyesuaian, dan berbagi prototipe Software Defined Network (SDN) untuk mensimulasikan topologi jaringan yang menggunakan switch terbuka.

3.4 POX

POX adalah pengembangan dari NOX. POX adalah openflow SDN *controller open source* berbasis python yang dirancang untuk pengembangan dan pembuatan prototipe aplikasi jaringan baru secara cepat dan POX sudah terpasang sebelumnya pada mesin virtual mininet [11]. Pada dasarnya, ini adalah salah satu dari banyak kerangka kerja (seperti NOX, floodlight, trema, dan lainnya) yang membantu dalam penulisan *controller* openflow. POX tidak hanya berfungsi sebagai kerangka kerja untuk berinteraksi dengan switch OpenFlow, tetapi juga dapat digunakan sebagai basis untuk berbagai pengembangan disiplin baru dari Software Defined Network. POX dapat dimanfaatkan untuk mengeksplorasi dan mendistribusikan prototipe, debugging SDN, virtualisasi jaringan, desain controller, pemrograman model, dan lainnya. Menurut penelitian Noman dan Jasim, disarankan untuk menggunakan kontroler POX untuk pengembangan dan pembuatan prototipe sistem kontrol jaringan dengan cepat serta sebagai *platform* untuk berinteraksi dengan switch openflow [11].

3.5 Controller

Controller atau program kendali adalah fitur yang memungkinkan pengguna menentukan algoritma sistem jaringan. Program kendali ini dapat dirancang menggunakan bahasa pemrograman Java atau bisa menggunakan Bahasa C atau bahasa pemrograman lainnya baik bahasa pemrograman tingkat rendah atau bahasa tingkat tinggi. Setelah itu, lapisan virtualisasi mengubah informasi dari program kendali menjadi bahasa yang lebih universal, seperti assembly. Program yang telah divirtualisasikan tersebut kemudian dijalankan oleh Network Operating System (NOS), yang berperan sebagai perantara antara perangkat lunak dan perangkat keras (seperti switch).

3.6 Algoritma Floyd-Warshall

Algoritma Floyd-Warshall secara sistematis mencoba semua probabilitas jalur yang mungkin dilewati di dalam graf untuk setiap pasangan simpul. Keistimewaan algoritma ini adalah

kemampuannya untuk melakukan perbandingan sebanyak V^3 kali, berbeda dengan V^2 kali (kuadrat dari jumlah simpul) seperti pada graf, di mana setiap kombinasi jalur diuji. Ini dimungkinkan karena algoritma ini secara iteratif memutuskan jalur terpendek antara dua simpul sampai nilai tersebut optimal yang dijelaskan alur pseudocode dari Algoritma Floyd-Warshall pada Gambar 1.

```
FloydWarshall(matriksBobot):
    n = jumlah simpul dalam matriksBobot

    dist = matriksBobot

    for k from 0 to n-1:
        for i from 0 to n-1:
            for j from 0 to n-1:
                if dist[i][j] > dist[i][k] + dist[k][j]:
                    dist[i][j] = dist[i][k] + dist[k][j]

    return dist
```

Gambar 1: Alur Pseudocode dari Algoritma Floyd-Warshall

Script SDN yang mengelola proses *routing* pada lapisan 3 dikenal sebagai l2_multi.py.

```
def _calc_paths():
  switches = get_openflow_switches()
  clear_path_map()

  untuk setiap switch dalam switches:
    untuk setiap tetangga, port dalam adjacency[switch]:
      jika port bukan None:
        path_map[switch][tetangga] = (1, None)
    path_map[switch][switch] = (0, None)
```

Gambar 2: Script l2_multi.py untuk Inisialisasi

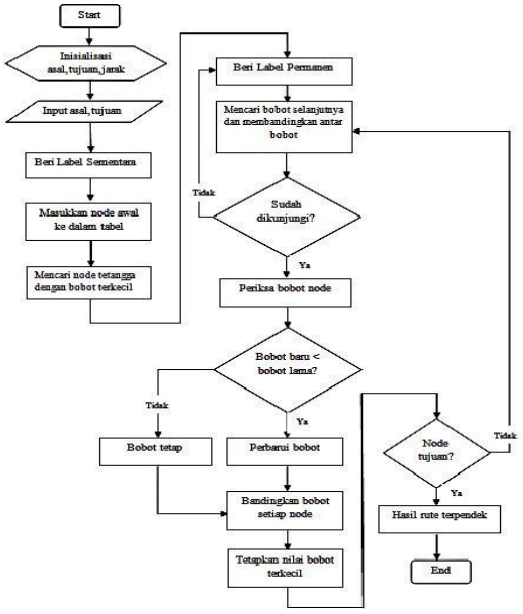
Script l2_multi.py pada gambar 2 untuk inisialisasi switch dan rute jalan serta inisialisasi jarak berdasarkan matriks ketetanggaan. Sedangkan pada proses menemukan jalur terpendek variable k yang ada pada switch ditampilkan pada pseudocode Gambar 3.

```
untuk setiap i dalam switches:
  untuk setiap j dalam switches:
    jika path_map[i][k][0] bukan None
    dan path_map[k][j][0] bukan None:
      jarak_baru = path_map[i][k][0] + path_map[k][j][0]
       jika path_map[i][j][0] adalah None atau jarak_baru
       lebih kecil dari path_map[i][j][0]:
        path_map[i][j] = (jarak_baru, k)
```

Gambar 3: Algoritma Floyd-Warshall dalam Menemukan Jalur Terpendek untuk k yang Ada di Switch

3.2 Algoritma Dijkstra

Algoritma ini digunakan dalam berbagai skenario pencarian rute terpendek, termasuk mencari jalur terpendek antara dua simpul tertentu (*pair shortest path*), menemukan jalur terpendek antara semua pasangan simpul (*all pairs shortest path*), menemukan jalur terpendek dari satu simpul ke semua simpul lainnya (*single-source shortest path*), serta mencari jalur terpendek antara dua simpul dengan melalui simpul-simpul tertentu (*intermediate shortest path*). Pada Gambar 4 merupakan diagram alir dari Algoritma Dijkstra.



Gambar 4: Flowchart Algoritma Dijkstra

Dalam Algoritma Dijkstra, strategi *greedy* diterapkan dengan memilih sisi yang memiliki bobot paling rendah pada setiap langkahnya, menghubungkan *node* yang belum dipilih dengan *node* yang sudah dipilih. Rute dari *node* awal ke *node* baru harus merupakan rute terpendek di antara semua rute menuju *node* yang belum dipilih. Penjelasan berupa *pseudocode* dari Algoritma Dijkstra dijelaskan pada gambar 5.

```
untuk i dari 1 hingga numNodes lakukan
  selected[i] ← 0
  distance[i] ← ∞
akhir untuk
distance[startNode] ← 0
```

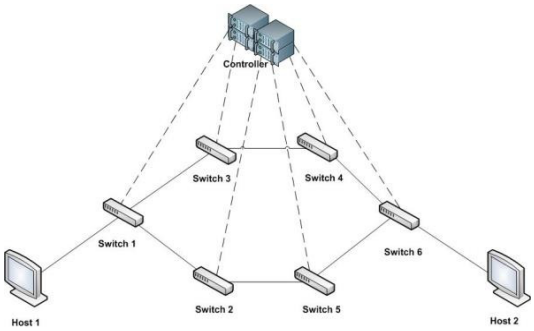
```
untuk iterasi dari 1 hingga numNodes-1 lakukan
  minValue ← ∞
  untuk i dari 1 hingga numNodes lakukan
    jika selected[i] = 0 dan distance[i] < minValue maka
      minValue ← distance[i]
      j ← i
    akhir jika
  akhir untuk

  selected[j] ← 1 { simpul j sudah terpilih }

  untuk i dari 1 hingga numNodes lakukan
    jika selected[i] = 0 dan adjacencyMatrix[j][i] ≠ 0 maka
      distance[i] ← min{distance[i],
        distance[j] + adjacencyMatrix[j][i]}
    akhir jika
  akhir untuk
akhir untuk
Keterangan :
  selected: array [1..numNodes] of integer
  distance: array [1..numNodes] of integer
  i, j: integer
  numNodes: integer {jumlah simpul}
```

Gambar 5: Pseudocode Algoritma Dijkstra

Kompleksitas waktu Algoritma Dijkstra adalah $O(n^2)$, di mana n menunjukkan jumlah simpul dalam graf. Penggunaan struktur data seperti *adjacency list* atau *priority queue* bisa mengurangi kompleksitasnya menjadi O((m+n) log n), yang merupakan pendekatan yang lebih efisien dalam konteks komputasi.

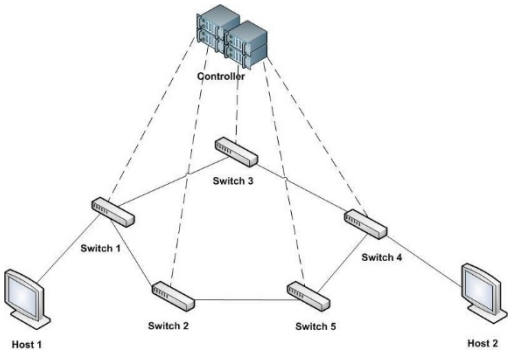
3.3 Desain

Didesain seperti graf, jalur dari *host* awal menuju ke *host* tujuan melibatkan beberapa titik yang disebut vertex, yang menghubungkan kedua titik tersebut. Vertex ini merepresentasikan openflow switch dalam desain ini. Semua openflow switch dikendalikan oleh satu controller openflow yang bertanggung jawab atas proses routing dalam SDN. Algoritma pada layer 3 SDN berfungsi menentukan jalur terpendek melalui switch OpenFlow dengan jumlah switch minimal untuk mencapai host 2 (tujuan). Tiga desain jaringan dibuat untuk digunakan menguji efektivitas algoritma dalam menemukan jalur terpendek menuju host 2 dari host 1.



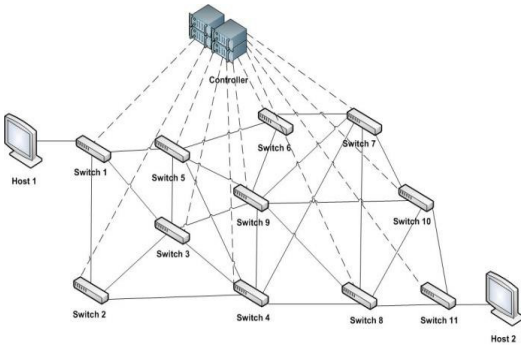
Gambar 6: Desain Topologi Jaringan 1

Pada desain topologi pertama pada Gambar 6 terdiri dari 1 buah *controller* serta 2 host (host awal dan host tujuan) dan 6 switch yang di mana switch diberikan label penomeran secara acak. 6 switch tersebut terhubung dengan controller agar dapat menentukan jalur mana yang akan diambil.



Gambar 7: Desain Topologi Jaringan 2

Pada topologi jaringan Gambar 7 terdiri dari 2 host sebagai host awal dan host tujuan, 1 buah controller dan 5 buah switch yang terhubung dengan controller. Berdasarkan topologi dari host 1 menuju host 2 terdapat 2 jalur yang digunakan yaitu melalui switch 1 menuju switch 3 lalu menuju switch 4 dan switch 1 dan menuju switch 2 menuju switch 5 dan terakhir menuju switch 4.



Gambar 8: Desain Topologi Jaringan 3

Pada desain topologi jaringan Gambar 8 terdiri dari 1 buah controller, 11 switch yang mana terhubung dengan controller, dan 2 buah host yang terdiri dari host awal dan host tujuan.
Untuk mengimplementasikan proses *routing* SDN pada desain topologi jaringan, digunakan alat POX. POX adalah sebuah platform pengembangan kontrol perangkat lunak jaringan dan pembuatan prototipe cepat yang menggunakan python. POX tidak hanya digunakan untuk berkomunikasi dengan switch openflow, tetapi juga sebagai dasar untuk berbagai proyek pengembangan dalam teknologi jaringan SDN.

4. SKENARIO UJI COBA

Dalam uji coba ini, langkah pertama yang dilakukan adalah memodifikasi skrip *Software Defined Network* (SDN) pada lapisan 3 dengan mengimplementasikan Algoritma Dijkstra sebagai pengganti Algoritma Floyd-Warshall untuk *routing*. Namun, *backup script* asli dilakukan agar kedua algoritma tersebut dapat dibandingkan secara efektif. Langkah selanjutnya adalah merancang tiga topologi jaringan yang terdiri dari satu *controller*, dua host (host awal dan host tujuan), serta beberapa switch OpenFlow dengan jumlah yang berbeda-beda di setiap topologi. Topologi ini dibangun menggunakan Mininet untuk menguji efektivitas routing Algoritma Dijkstra, dengan fokus pada penemuan rute yang meminimalkan jumlah switch sesuai struktur jaringan yang direncanakan.
Setelah topologi dirancang, *script* POX dijalankan, di mana satu *script* menggunakan Algoritma Floyd-Warshall (misalnya, l2_multi.py), sementara versi lainnya telah dimodifikasi untuk menggunakan Algoritma Dijkstra. Selanjutnya, ketiga topologi tersebut diaktifkan melalui terminal dengan menggunakan MiniEdit. Koneksi jaringan dari host 1 ke host 2 diuji untuk memeriksa kestabilan serta keberhasilan *routing*.
Pada tahap akhir, dilakukan analisis terhadap hasil uji tiga skema jaringan di Mininet untuk menentukan apakah rute terpendek menuju host 2 telah ditemukan. Perbandingan antara Algoritma Floyd-Warshall dan Algoritma Dijkstra dilakukan untuk mengevaluasi keunggulan serta kelemahan masing-masing algoritma dalam menentukan rute terpendek pada graf, khususnya dalam konteks jaringan SDN yang diujikan. Hasil evaluasi ini diharapkan memberikan wawasan lebih lanjut terkait efektivitas kedua algoritma dalam konteks *routing* pada jaringan berbasis SDN.

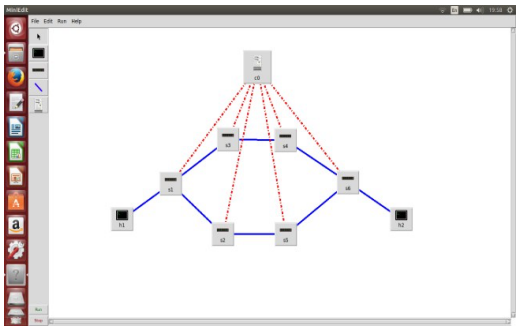
5. HASIL DAN PEMBAHASAN

script l2_multi.py yang berisi Algoritma Dijkstra yang sudah disesuaikan pada lapisan 3 SDN dijabarkan pada gambar 9.

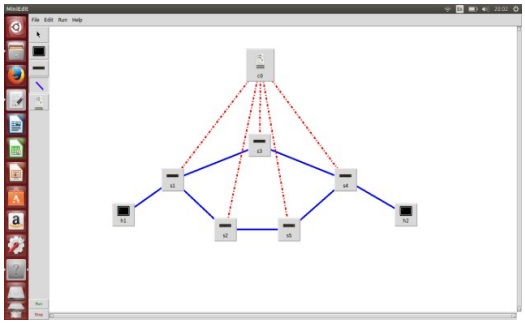
```
fungsi _get_raw_path(source, dest):
inf = tak hingga
switches = himpunan dari semua switch openflow
dist = kamus dengan nilai default inf untuk setiap switch
previous = kamus dengan nilai default none untuk setiap switch
dist[source] = 0

selama switches tidak kosong:
   u = switch dengan jarak terkecil di dist
   hapus u dari switches

   jika dist[u] == inf atau u adalah dest:
      hentikan loop

   untuk setiap tetangga v dan biaya di adjacency[u]:
      alt = dist[u] + 1  # Mengasumsikan setiap edge memiliki biaya 1
      jika alt < dist[v]:  # Relaksasi
         dist[v] = alt
         previous[v] = u

path = daftar kosong
u = dest
selama previous[u] bukan None:
   masukkan u di awal daftar path
   u = previous[u]

masukkan u (source) di awal daftar path
return path
```

Gambar 9: Script Algoritma Dijkstra pada l2_multi.py

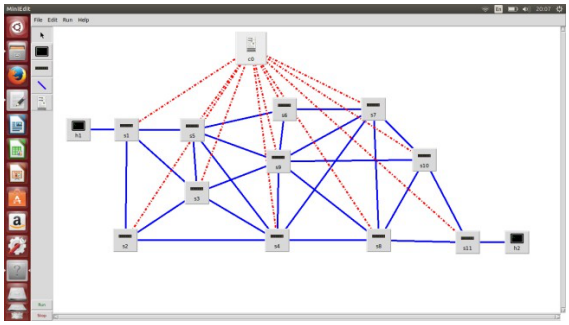
Mininet secara berurutan dibuatkan skema jaringan yakni Gambar 10, Gambar 11, dan Gambar 12.



Gambar 10: Topologi Jaringan 1 pada Mininet

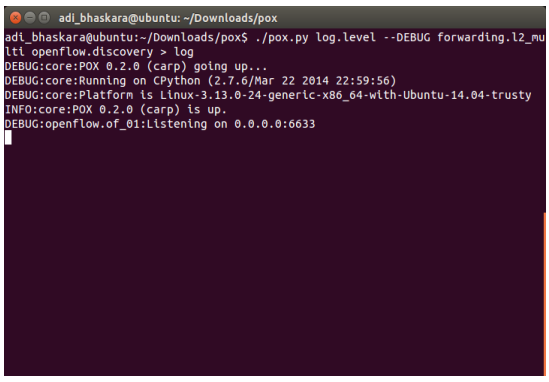


Gambar 11: Topologi Jaringan 2 pada Mininet



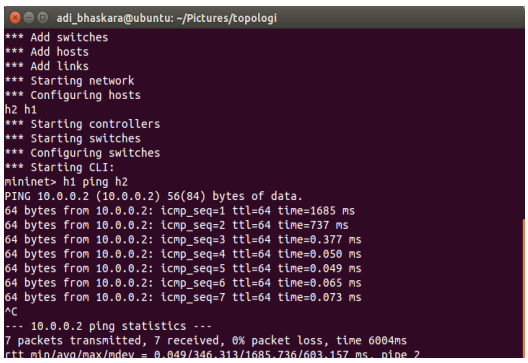
Gambar 12: Topologi Jaringan 3 pada Mininet

Untuk menjalankan program POX menggunakan Python, jalankan source code POX dengan perintah ./pox.py log.level --DEBUG forwarding l2_multiopenflow. Discovery > log. Log hasil operasi akan disimpan dalam file bernama log, yang bisa dilihat dengan menggunakan perintah cat log. Tunggu hingga topologi jaringan siap untuk dieksekusi.



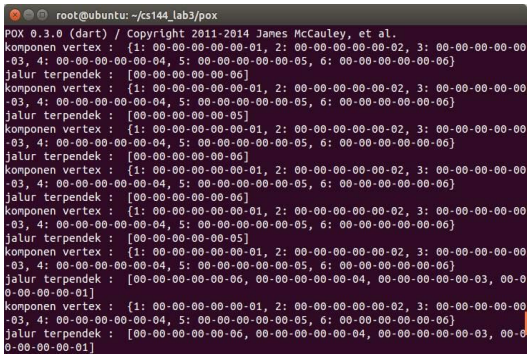
Gambar 13: Menjalankan 12_multi dan File Log

Jalankan topologi di miniedit dengan membuka terminal baru dan menjalankan perintah sudo python topologi.py untuk membuka mininet. Program POX yang telah dijalankan pada terminal sebelumnya akan mulai berproses secara otomatis. Untuk mengecek koneksi, ketik perintah h1 ping h2 pada terminal mininet. Setelah beberapa saat menunggu, hentikan ping dan program POX .



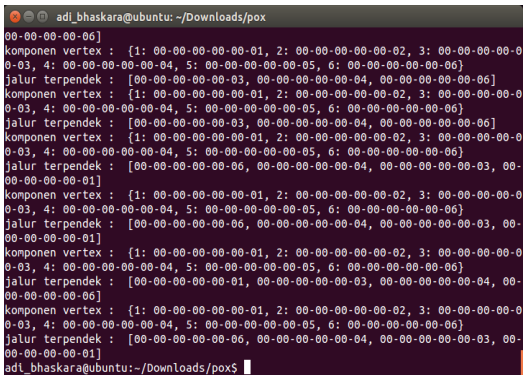
Gambar 14: Penggunaan pada Topologi dan Periksa Konektivitas

Perbandingan visual dari hasil jalur terpendek dengan menerapkan Algoritma Dijkstra dan floyd-warshall pada tiga graf yang telah dibuat diperlihatkan pada gambar berikut :



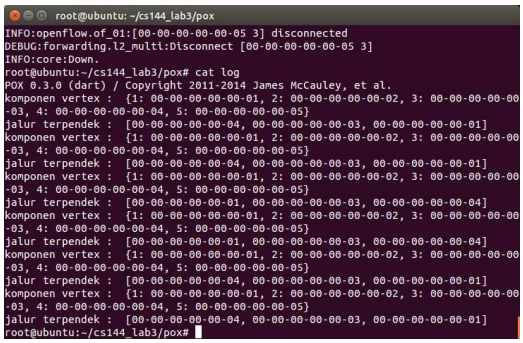
Gambar 15: Hasil Penentuan Rute Terpendek Topologi 1 dengan Algoritma Floyd- Warshall

Hasil yang didapatkan untuk menentukan rute terpendek pada percobaan topologi 1 menggunakan Algoritma Floyd-Warshall terdapat 3 yaitu s5, s6, dan s1 menuju s3 lanjut menuju s4 dan berakhir di s6.



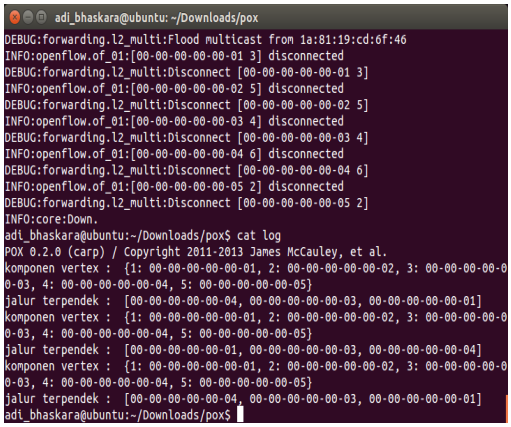
Gambar 16: Hasil Penentuan Jalur Terpendek Topologi 1 dengan Algoritma Dijkstra

Hasil yang diperoleh dari penentuan jalur terpendek pada topologi 1 menggunakan Algoritma Dijkstra yaitu melalui jalur s1 → s3 → s4→ s6.



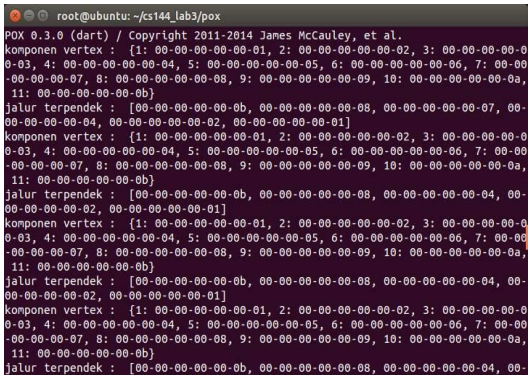
Gambar 17: Hasil Penentuan Rute Terpendek Topologi 2 Menggunakan Algoritma Floyd-Warshall

Penentuan jalur terpendek dimulai dari host satu ke host lainnya menggunakan Algoritma Floyd-Warshall pada topologi 2 yakni s1 → s3→ s4..



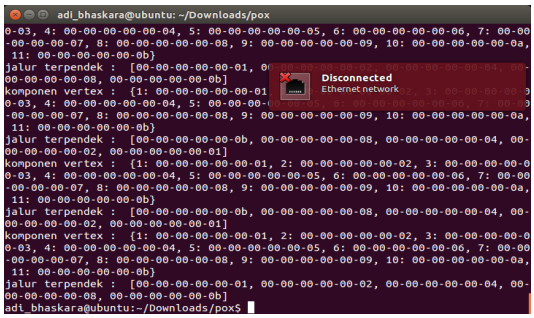
Gambar 18: Hasil Penentuan Rute Terpendek Topologi 2 dengan Algoritma Dijkstra

Gambar 13 menampilkan hasil pengujian jalur terpendek menggunakan Algoritma Dijkstra dari host 1 menuju host 2 pada topologi 2 melalui jalur dari s1 menuju s3 dan berakhir di s4.



Gambar 19: Hasil pengujian jalur terpendek topologi 3 menggunakan Algoritma Floyd-Warshall

Hasil pengujian jalur terpendek pada percobaan topologi 3 menggunakan Algoritma Floyd-Warshall adalah s1 → s2→ s4→ s8→ s11 dan s1 → s2→ s4→ s7→ s8 → s11.



Gambar 20: Hasil penentuan rute terpendek topologi 3 dengan Algoritma Dijkstra

Pengeksekusian dengan cara yang sama pada topologi 3 menggunakan algoritma Dijkstra diperoleh hasil jalur terpendek yakni s1 →s2→ s4→ s8→ s11.

6. KESIMPULAN

Dengan berbagai kelebihan yang dimiliki SDN (Software Defined Network) yang digunakan sebagai tools untuk membandingkan Algoritma Floyd-Warshall dengan Algoritma Dijkstra, menghasilkan hasil yang serupa dalam pencarian jalur terpendek . Namun, setelah diamati lebih seksama pada hasil log Algoritma Floyd-Warshall terdapat beberapa kesalahan untuk pencarian jalur terpendek pada awal eksekusi program sehingga berdasarkan hal tersebut membuktikan pencarian terpendek Algoritma Dijkstra lebih cepat untuk mendapat hasil jalur terpendek lebih baik dibanding Algoritma Floyd-Warshall. Dengan demikian,

dari segi pencarian rute terpendek, Algoritma Dijkstra lebih cepat daripada Algoritma Floyd-Warshall.

DAFTAR PUSTAKA

[1] "Performance Evaluation of Pox Controller for Software Defined Networks," *International Journal of Innovative Technology and Exploring Engineering*, 2019, [Online]. Available: https://api.semanticscholar.org/CorpusID:241473923

[2] M. A. Nugroho and A. Rakhmatsyah, "Simulation Of Jellyfish Topology Link Failure Handling Using Floyd Warshall and Johnson Algorithm in Software Defined Network Architecture," *2021 9th International Conference on Information and Communication Technology (ICoICT)*, pp. 144–148, 2021, doi: 10.1109/ICoICT52021.2021.9527523.

[3] D. B. Rawat and S. R. Reddy, "Software Defined Networking Architecture, Security and Energy Efficiency: A Survey," *IEEE Communications Surveys & Tutorials*, vol. 19, pp. 325–346, 2017, [Online]. Available: https://api.semanticscholar.org/CorpusID:34643772

[4] Naimullah, S. I. Ullah, A. W. Ullah, A. Salam, M. Imad, and F. Ullah, "Performance Analysis of POX and RYU Based on Dijkstra's Algorithm for Software Defined Networking," in *Lecture Notes in Networks and Systems*, Springer Science and Business Media Deutschland GmbH, 2021, pp. 24–35. doi: 10.1007/978-3-030-77246-8_3.

[5] I. Bagus, G. Wahyu, and A. Dalem, "PENERAPAN ALGORITMA A* (STAR) MENGGUNAKAN GRAPH UNTUK MENGHITUNG JARAK TERPENDEK," Online, 2018. [Online]. Available: http://jurnal.stiki-indonesia.ac.id/index.php/jurnalresistor

[6] A. P. Patil and S. P. Gaikwad, "Network Throughput Improvement using Dijkstra's and Floyd Warshall Algorithm," *Networking and Communication Engineering*, vol. 7, pp. 39–44, 2015, [Online]. Available: https://api.semanticscholar.org/CorpusID:60179436

[7] E. Fatur Rohman and Y. Afrianto, "Fail Path Analysis on Openflow Network Using Floyd-Warshall Algorithm," *Jurnal Mantik*, vol. 4, no. 3, 2020, [Online]. Available: https://iocscience.org/ejournal/index.php/mantik

[8] T. Javid, T. Riaz, and A. Rasheed, "A layer2 firewall for software defined network," *2014 Conference on Information Assurance and Cyber Security (CIACS)*, pp. 39–42, 2014, [Online]. Available: https://api.semanticscholar.org/CorpusID:16530746

[9] T. Galinac Grbac and N. Domazet, "On the applications of Dijkstra's shortest path algorithm in software defined networks," *Studies in Computational Intelligence*, vol. 737, pp. 39–45, 2017, doi: 10.1007/978-3-319-66379-1_4.

[10] J. Miguel-Alonso, "A Research Review of OpenFlow for Datacenter Networking," *IEEE Access*, vol. 11, pp. 770–786, 2023, [Online]. Available: https://api.semanticscholar.org/CorpusID:255336598

[11] H. M. Noman and M. N. Jasim, "POX Controller and Open Flow Performance Evaluation in Software Defined Networks (SDN) Using Mininet Emulator," *IOP Conf Ser Mater Sci Eng*, vol. 881, 2020, [Online]. Available: https://api.semanticscholar.org/CorpusID:225426647