

Analisis Komparatif Algoritma Shortest Path dan Bottleneck Path (Max-Min) dalam Optimasi Throughput Jaringan Komputer

Kelompok 7 Kelas C

Ariwata Alfajri (L0124088)

Azzaral Aswad Asshiddiqy (L0124090)

Bintang A'raaf Stevan Putra (L0124091)

Daffa Dewanda Putra (L0124094)

Dzaky Ghiffary Susilo (L0124097)

Program Studi Informatika

Fakultas Teknologi Informasi dan Sains Data

Universitas Sebelas Maret

2025

Abstrak

Algoritma *routing Shortest Path First* (SPF) tradisional, yang mendominasi operasi jaringan dengan meminimalkan metrik aditif (cost/tunda), kini menghadapi tantangan serius. Dengan meningkatnya lalu lintas data bervolume tinggi, jaringan rentan terhadap kemacetan (*bottleneck*) karena SPF mengabaikan kapasitas *bandwidth* (*concave metric*). Masalah *routing* yang melibatkan konflik metrik ini sering kali NP-complete, menyebabkan SPF memilih jalur terpendek yang justru sempit, membatasi kinerja jaringan. Penelitian ini melakukan analisis kuantitatif komparatif antara SPF dan algoritma *Highest Bottleneck Bandwidth* (HBB) atau *Max-Min Path*, yang dirancang untuk memaksimalkan kapasitas minimum jalur.

Implementasi dan evaluasi dilakukan dalam konteks *Software Defined Network* (SDN), yang memfasilitasi deteksi bottleneck dan implementasi routing berbasis kapasitas secara dinamis. Hasil simulasi menunjukkan disparitas kinerja signifikan. Meskipun algoritma SPF memilih jalur 2 hop terpendek, *throughput*-nya dibatasi drastis menjadi 10 Mbps. Sebaliknya, HBB memilih jalur 4 hop yang lebih panjang, tetapi berhasil menghindari *bottleneck* dan memberikan throughput maksimal 500 Mbps. Peningkatan *throughput* yang mencapai 50 kali lipat ini membuktikan kelemahan mendasar metrik routing tunggal berbasis jarak. Dalam analisis ini, algoritma HBB terbukti unggul dalam optimasi *throughput* jaringan, terutama untuk transfer data bervolume besar, dengan mengubah paradigma operasi matriks dari $(\min, +)$ menjadi (\max, \min) . Temuan ini mendukung penerapan model *routing* adaptif di lingkungan SDN: SPF untuk lalu lintas sensitif latensi dan HBB untuk maksimasi utilisasi jaringan dan *throughput* data bervolume besar.

Kata Kunci: *Software Defined Network* (SDN), Algoritma *Shortest Path*, *Bottleneck Bandwidth*, Jalur *Max-Min*, Optimasi *Throughput*, *Quality of Service* (QoS).

1. Pendahuluan

Dalam era di mana penggunaan dan eksploitasi internet berkembang pesat, arsitektur jaringan tradisional seringkali tidak lagi memadai untuk memenuhi kebutuhan perusahaan maupun pengguna akhir. Peningkatan lalu lintas internet, beban data, dan permintaan *bandwidth* yang terus melonjak menuntut manajemen jaringan yang lebih cerdas. Salah satu komponen vital dalam menjaga kualitas layanan (*Quality of Service* atau QoS) dalam jaringan adalah algoritma *routing*.

Secara tradisional, protokol jaringan yang umum digunakan, seperti OSPF (*Open Shortest Path First*), sangat bergantung pada algoritma pencarian jalur terpendek (*Shortest Path First* atau SPF), seperti Algoritma Dijkstra. Logika dasar dari algoritma ini adalah meminimalkan biaya jalur berdasarkan jarak atau jumlah lompatan (*hop count*). Meskipun algoritma ini mampu mengirimkan paket ke tujuan dengan penundaan waktu (*delay*) yang rendah, algoritma ini memiliki kelemahan fundamental: ia seringkali mengabaikan kapasitas *bandwidth* dari jalur yang dipilih.

Kelemahan ini sering kali berujung pada masalah *bottleneck* (leher botol). Sebuah jaringan mungkin mencapai titik di mana aliran data dibatasi oleh sumber daya jaringan yang sempit pada satu titik tertentu, meskipun jalur tersebut adalah jalur terpendek secara geografis atau topologi. Ketika *bandwidth* pada jalur terpendek tersebut tidak mampu menampung besarnya data yang dikirim, kemacetan terjadi, dan perangkat akhir tidak dapat dimanfaatkan kapasitas penuhnya. Dengan kata lain, memilih "jalan pintas" yang sempit seringkali lebih merugikan daripada memilih jalan yang sedikit lebih jauh namun memiliki kapasitas lebar.

Untuk mengatasi permasalahan ini, diperlukan pendekatan yang tidak lagi hanya berfokus pada jarak, melainkan pada kapasitas *bandwidth*. Solusi yang diusulkan adalah penerapan algoritma *Highest Bottleneck Bandwidth* (HBB) dalam lingkungan *Software Defined Network* (SDN). Berbeda dengan Dijkstra konvensional yang menghitung jarak minimum, algoritma HBB memodifikasi pendekatan tersebut untuk menghitung "lebar minimum" dari setiap tepi jalur, dan kemudian memilih jalur yang memiliki *bottleneck bandwidth* tertinggi (Max-Min) dari sumber ke tujuan.

Dalam pendekatan ini, *controller* pada SDN yang memiliki visibilitas penuh terhadap topologi jaringan dapat mendeteksi titik-titik *choke* (kemacetan) dan mengarahkan paket

melalui jalur yang mungkin memiliki lebih banyak *hop*, namun menawarkan kapasitas *bandwidth* yang jauh lebih besar. Artikel ini bertujuan untuk mengevaluasi dan membuktikan bahwa dalam konteks pengiriman data bervolume besar, pendekatan "Jalur Terlebar" (HBB) mampu memberikan *throughput* yang lebih baik dan mengurangi risiko *packet loss* dibandingkan dengan pendekatan "Jalur Terpendek" (SPF) yang masih banyak digunakan saat ini.

2. Tinjauan Pustaka

Dalam upaya mengoptimalkan kinerja jaringan komputer, pemilihan algoritma routing memegang peranan krusial dalam menentukan efisiensi transmisi data. Bagian ini meninjau literatur terkait metrik routing, karakteristik algoritma *Shortest Path* dan *Bottleneck Path*, serta analisis komparatif keduanya dalam konteks *throughput* dan *Quality of Service* (QoS).

2.1 Klasifikasi Routing Metric: Additive vs. Concave

Pondasi utama dalam algoritma routing adalah bagaimana biaya (*cost*) sebuah jalur dihitung. Literatur membedakan metrik jaringan ke dalam dua kategori utama yang mempengaruhi *throughput*: *additive metric* dan *concave metric*. Riedl dan Schupke (2007) menjelaskan bahwa sebagian besar protokol routing konvensional berbasis pada metrik aditif. Jika sebuah jalur P terdiri dari sekumpulan tautan e_1, e_2, \dots, e_k , maka total biaya jalur $w(P)$ didefinisikan sebagai penjumlahan bobot tiap tautan:

$$w(P) = \sum_{i=1}^k w(e_i)$$

Sebaliknya, untuk optimasi throughput, digunakan metrik concave, di mana kapasitas jalur ditentukan oleh tautan dengan kapasitas terkecil:

$$w(P) = \min_{i=1}^k w(e_i)$$

Pemahaman terhadap perbedaan operasi 'min-sum' (*additive*) dan 'max-min' (*concave*) ini adalah dasar untuk membedakan tujuan antara minimasi latensi dan maksimasi kapasitas (Van Mieghem et al., n.d.)."

2.2 Algoritma Shortest Path dan Keterbatasannya

Algoritma Shortest Path, seperti Dijkstra dan Bellman-Ford, merupakan standar de facto dalam penentuan rute jaringan selama beberapa dekade. Fokus utama algoritma ini adalah meminimalkan total biaya akumulatif dari sumber ke tujuan. Sajid et al. (2018) dalam evaluasi komprehensifnya pada jaringan Software Defined Network (SDN) mencatat bahwa meskipun algoritma Dijkstra sangat efisien dalam menemukan jalur dengan latensi terendah (latency-based), algoritma ini sering kali mengabaikan kapasitas bandwidth yang tersedia.

Kelemahan pendekatan ini menjadi nyata dalam skenario transfer data bervolume besar. Jika jalur "terpendek" melalui tautan dengan kapasitas kecil, maka throughput keseluruhan akan terhambat, terlepas dari seberapa rendah latensnya. Hal ini sejalan dengan temuan bahwa routing metrik tunggal (hanya jarak) sering kali tidak cukup untuk memenuhi tuntutan lalu lintas jaringan modern yang kompleks (Riedl & Schupke, 2007).

2.3 Algoritma Bottleneck Path (Max-Min)

Berbeda dengan pendekatan jarak terpendek, algoritma Bottleneck Path (sering disebut sebagai Widest Path atau Maximum Capacity Path) bertujuan untuk memecahkan masalah Max-Min. Masalah ini didefinisikan sebagai pencarian jalur di mana kapasitas tautan minimum (bottleneck) dimaksimalkan (Deaconu & Tayyеби, 2020). Masalah ini didefinisikan sebagai pencarian jalur di mana kapasitas tautan minimum (*bottleneck*) dimaksimalkan (Deaconu & Tayyеби, 2020).

Secara formal, jika $P_{s,t}$ adalah himpunan semua jalur yang mungkin dari sumber s ke tujuan t , maka algoritma *Bottleneck Path* bertujuan mencari jalur optimal P^* yang memenuhi persamaan:

$$Width(P^*) = \max_{P \in P_{s,t}} (\min_{e \in P} Capacity(e))$$

Secara matematis, konsep ini berfokus pada identifikasi "pipa terbesar" yang tersedia antar node. Duan (2010) menguraikan bahwa masalah All-Pairs Bottleneck Paths (APBP) dapat diselesaikan menggunakan operasi matriks (max, min), yang secara fundamental berbeda dari operasi (min, +) pada algoritma jalur terpendek. Dalam konteks aplikasi, Sajid et al. (2018) menunjukkan bahwa algoritma Highest Bottleneck Bandwidth secara signifikan lebih unggul dalam menjaga ketersediaan throughput tinggi, karena algoritma ini secara proaktif menghindari tautan yang padat atau sempit yang biasanya dipilih oleh algoritma Shortest Path hanya karena jaraknya lebih dekat.

2.4 Analisis Komparatif dan Relevansi pada Jaringan Terprogram (SDN)

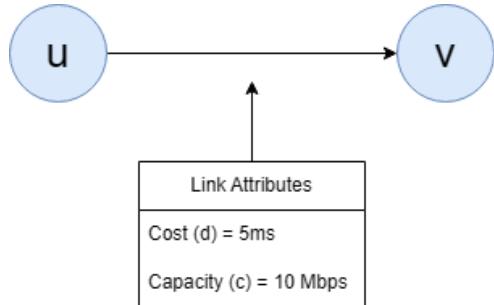
Pergeseran paradigma menuju Software Defined Networking (SDN) telah membuka peluang baru untuk menerapkan algoritma Bottleneck Path secara lebih dinamis. Jain et al. (2013) dalam studi kasus implementasi jaringan WAN Google (B4), mendemonstrasikan bahwa dengan memisahkan control plane, operator dapat melakukan Traffic Engineering (TE) terpusat yang mendorong utilisasi tautan hingga mendekati 100%. Dalam lingkungan seperti ini, optimasi throughput melalui pemilihan jalur berkapasitas maksimal (bottleneck-aware) menjadi lebih kritikal dibandingkan sekadar meminimalkan jarak hop.

Studi komparatif oleh Sajid et al. (2018) menyimpulkan bahwa ada trade-off yang jelas: algoritma Shortest Path unggul dalam kecepatan konvergensi dan latensi untuk paket kecil, sedangkan algoritma berbasis Bottleneck Path (Max-Min) memberikan stabilitas dan throughput maksimal untuk aliran data besar. Oleh karena itu, penelitian modern mulai melirik pendekatan hibrida atau pemilihan algoritma adaptif berdasarkan jenis lalu lintas untuk menutupi celah kinerja di antara kedua pendekatan ekstrem ini.

3. Metode Penelitian

Penelitian ini menggunakan pendekatan analisis komparatif berbasis simulasi untuk mengevaluasi perbedaan kinerja antara algoritma *routing* berbasis *Shortest Path* dan algoritma *routing* berbasis *Bottleneck Path (Max-Min)* dalam konteks optimasi *throughput* jaringan komputer. Fokus utama penelitian adalah mengkaji bagaimana perbedaan karakteristik metrik *routing*, yaitu *additive metric* dan *concave metric*, memengaruhi pemilihan jalur serta performa jaringan secara keseluruhan.

3.1 Pemodelan Jaringan Secara Matematis



Gambar 3.1.1. Representasi atribut sisi (*edge*) pada graf jaringan komputer

Jaringan komputer dimodelkan sebagai sebuah graf berarah $G = (V, E)$, di mana V merepresentasikan himpunan *node* (*router* atau *switch*) dan E merepresentasikan himpunan link antar node. Setiap sisi berarah $(u, v) \in E$ memiliki dua atribut bobot utama yang mencerminkan karakteristik jaringan nyata, yaitu:

1. *Cost / Delay* ($d_{u,v}$)

Mewakili *additive metric* seperti *latency*, jarak logis, atau *hop count*. Nilai ini digunakan oleh algoritma *Shortest Path*.

2. *Capacity / Bandwidth* ($c_{u,v}$)

Mewakili *concave metric* berupa kapasitas maksimum *link* dalam satuan Mbps. Nilai ini digunakan oleh algoritma *Bottleneck Path*.

Pemodelan ini memungkinkan analisis perbedaan perilaku algoritma *routing* ketika dihadapkan pada konflik antara jalur terpendek dan jalur berkapasitas terbesar.

3.1.1 Formulasi Masalah Shortest Path (SP)

Sesuai dengan tinjauan Riedl & Schupke (2007), algoritma *Shortest Path* bertujuan meminimalkan total biaya jalur. Jika P adalah jalur dari sumber s ke tujuan t , maka fungsi objektifnya adalah:

$$\min_{P \in P_{s,t}} \left(\sum_{(u,v) \in P} d_{u,v} \right)$$

Pendekatan ini menggunakan operasi aljabar (\min , $+$), sehingga sangat efektif dalam meminimalkan *end-to-end latency*, namun tidak mempertimbangkan kapasitas *bandwidth* sebagai faktor utama dalam pemilihan jalur.

3.1.2 Formulasi Masalah Bottleneck Path (BP)

Masalah ini, yang juga dikenal sebagai *Maximum Capacity Path*, bertujuan untuk memaksimalkan kapasitas *bottleneck* dalam jalur. Berdasarkan definisi masalah *Inverse Maximum Capacity* oleh Deaconu & Tayyebi (2020), kita mencari jalur P^* yang memenuhi:

$$Width(P^*) = \max_{P \in P_{s,t}} (\min_{(u,v) \in P} c_{u,v})$$

di mana $\min_{(u,v) \in P} c_{u,v}$ adalah kapasitas efektif dari jalur tersebut. Pendekatan ini menggunakan operasi aljabar (\max , \min) yang berbeda secara fundamental dari operasi (\min , $+$) pada algoritma standar (Duan, 2010).

3.2 Implementasi Algoritma

Untuk menyelesaikan permasalahan di atas, penelitian ini membandingkan dua algoritma:

1. Dijkstra Standar

Digunakan untuk menyelesaikan masalah *Shortest Path* dengan memanfaatkan *min-priority queue*. Algoritma ini memilih *node* dengan jarak kumulatif terkecil pada setiap iterasi.

2. Modified Dijkstra (*Max-Min*):

Digunakan untuk menyelesaikan masalah *Bottleneck Path* dengan memanfaatkan *max-priority queue*. Alih-alih menjumlahkan bobot, algoritma ini menyebarkan nilai kapasitas minimum maksimum dari sumber ke seluruh *node*.

Berikut adalah desain *pseudocode* untuk algoritma *Bottleneck Path* yang digunakan dalam simulasi, diadaptasi dari pendekatan *Widest Path* yang dievaluasi oleh Sajid et al. (2018) yang di mana nilai *width* suatu *node* merepresentasikan kapasitas maksimum yang dapat dicapai dari sumber ke *node* tersebut:

```

// Input: Graph G(V, E), Source Node s, Destination Node t
// Output: Path P with maximum bottleneck capacity

FOR each vertex v in V:
    width[v] ← 0           // inisialisasi kapasitas path ke 0
    parent[v] ← NULL        // untuk melacak jalur
    width[s] ← ∞            // kapasitas di sumber tak terhingga
    Q ← PriorityQueue()      // max-priority queue
    Q.add(s, width[s])

WHILE Q is not empty:
    u ← Q.extract_max()    // ambil node dengan kapasitas width terbesar saat ini
    IF u == t THEN BREAK   // tujuan tercapai

    FOR each neighbor v of u:
        // hitung kapasitas potensial lewat jalur u -> v
        // kapasitas dibatasi oleh link terkecil (bottleneck)
        new_width ← min(width[u], capacity(u, v))

        // jika jalur baru lebih lebar dari jalur lama
        IF new_width > width[v]:
            width[v] ← new_width
            parent[v] ← u
            // update prioritas di antrian
            IF v in Q:
                Q.update_key(v, new_width)
            ELSE:
                Q.add(v, new_width)

RETURN ReconstructPath(parent, t), width[t]

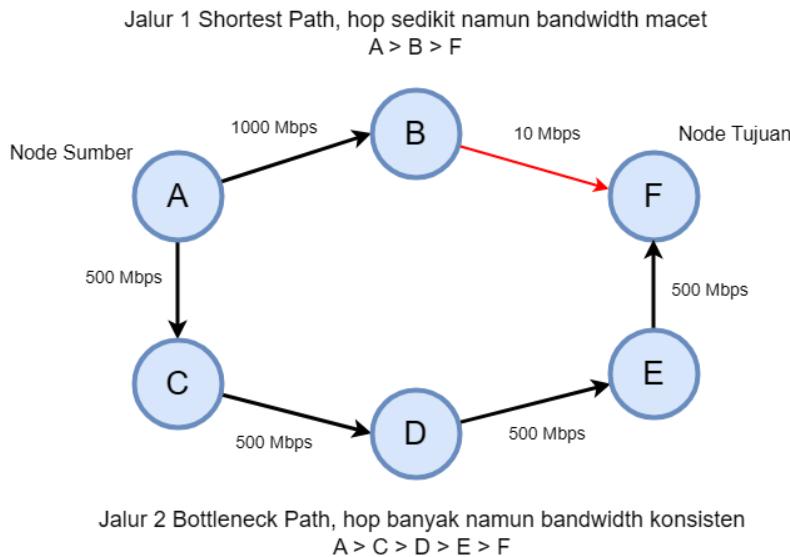
```

Pseudocode 3.2.1. Modified Dijkstra untuk algoritma *Bottleneck Path*

3.3 Desain Skenario Pengujian

Untuk memvalidasi kinerja algoritma, dirancang sebuah topologi jaringan representatif yang mengandung konflik antara *hop count* dan kapasitas *bandwidth*. Skenario ini mensimulasikan kondisi *Traffic Engineering* pada jaringan WAN seperti yang dibahas pada studi kasus Google B4 (Jain et al., 2013).

3.3.1 Topologi Jaringan



Gambar 3.3. Topologi pengujian simulasi dengan skenario *bottleneck link* pada jalur terpendek.

Graf terdiri dari 6 *node* (A, B, C, D, E, F) dengan *node* A sebagai sumber dan *node* F sebagai tujuan. Terdapat dua jalur utama yang terbentuk:

1. Jalur atas (*shortest path*): Jalur dengan jumlah *hop* sedikit namun memiliki *link* dengan kapasitas rendah (*bottleneck*).
2. Jalur bawah (*widest path*): Jalur dengan jumlah *hop* lebih banyak namun memiliki kapasitas konsisten tinggi.

3.3.2 Dataset Parameter Link

Berikut adalah tabel parameter bobot untuk setiap sisi (*edge*) dalam simulasi:

Link (Edge)	Cost (Hop/Metric)	Capacity (Mbps)	Keterangan
A → B	1	1000	<i>Gigabit link</i>
B → F	1	10	<i>Bottleneck link</i>
A → C	1	500	<i>Link alternatif awal</i>
C → D	1	500	<i>Link alternatif tengah</i>
D → E	1	500	<i>Link alternatif tengah</i>
E → F	1	500	<i>Link alternatif akhir</i>

Tabel 3.3.1.1. Parameter topologi simulasi

3.3.3 Skenario Kasus Uji

Sistem akan diuji dengan mengirimkan aliran data dari *node A* ke *node F*.

1. Hipotesis 1: Algoritma *Shortest Path* akan memilih jalur $A \rightarrow B \rightarrow F$ (*total cost*: 2) namun *throughput* dibatasi oleh *link B → F* sebesar 10 Mbps.
2. Hipotesis 2: Algoritma *Bottleneck Path* akan memilih jalur $A \rightarrow C \rightarrow D \rightarrow E \rightarrow F$ (*total cost*: 4) namun memberikan *throughput* sebesar 500 Mbps.

3.4 Parameter Evaluasi

Analisis komparatif akan dilakukan berdasarkan parameter berikut, merujuk pada metrik QoS standar (Van Mieghem et al., n.d.):

Parameter	Definisi	Satuan	Tujuan Optimasi
<i>End-to-end throughput</i>	Kapasitas minimum yang tersedia sepanjang jalur terpilih.	Mbps	<i>Maximize</i>
<i>Hop count</i>	Jumlah <i>router/node</i> perantara yang dilalui.	Hops	<i>Minimize</i>
<i>Path efficiency</i>	Rasio antara <i>throughput</i> jalur terpilih dibanding <i>throughput</i> maksimum teoritis graf.	%	<i>Maximize</i>

Tabel 3.4.1. Parameter Evaluasi Kinerja

4. Hasil & Pembahasan

4.1 Hasil Eksekusi Algoritma

Berdasarkan topologi pengujian dengan node sumber A dan node tujuan F, berikut adalah hasil penelusuran jalur yang dihasilkan oleh kedua algoritma:

4.1.1 Penelusuran Algoritma Shortest Path (Dijkstra)

Algoritma ini bekerja dengan meminimalkan metrik aditif (biaya/jarak). Berdasarkan parameter bobot pada Tabel 1, perhitungan biaya jalur adalah sebagai berikut:

1. Jalur 1 ($A \rightarrow B \rightarrow F$):

$$TotalCost = Cost(A, B) + Cost(B, F) = 1 + 1 = 2$$

2. Jalur 2 ($A \rightarrow C \rightarrow D \rightarrow E \rightarrow F$):

$$TotalCost = Cost(A, C) + Cost(C, D) + Cost(D, E) + Cost(E, F) = 1 + 1 + 1 + 1 = 4$$

Karena $2 < 4$, algoritma Shortest Path memilih jalur $A \rightarrow B \rightarrow F$. Namun, kapasitas efektif (*end-to-end throughput*) jalur ini dibatasi oleh tautan dengan kapasitas terkecil (*concave metric*):

$$Throughput_{SPF} = \min(Capacity(A, B), Capacity(B, F))$$

$$Throughput_{SPF} = \min(1000 \text{ Mbps}, 10 \text{ Mbps}) = 10 \text{ Mbps}$$

Hasil ini mengkonfirmasi Hipotesis 1, di mana algoritma memilih jalur dengan biaya terendah namun terjebak pada *link bottleneck*.

4.1.2 Penelusuran Algoritma Bottleneck Path (Max–Min)

Algoritma ini bertujuan memaksimalkan kapasitas *bottleneck*. Algoritma menghitung “lebar” jalur (*Width*) berdasarkan kapasitas minimum di sepanjang jalur tersebut:

1. Jalur 1 ($A \rightarrow B \rightarrow F$):

$$Width = \min(1000, 10) = 10 \text{ Mbps}$$

2. Jalur 2 ($A \rightarrow C \rightarrow D \rightarrow E \rightarrow F$):

$$Width = \min(500, 500, 500, 500) = 500 \text{ Mbps}$$

Karena $500 > 10$, algoritma *Bottleneck Path* memilih jalur $A \rightarrow C \rightarrow D \rightarrow E \rightarrow F$. Meskipun memiliki jumlah *hop* yang lebih banyak (4 *hops*), karena jalur ini menawarkan kapasitas yang jauh lebih besar.

Hasil ini mengkonfirmasi Hipotesis 2, di mana algoritma berhasil menghindari jalur sempit dan memberikan *throughput* maksimal sebesar 500 Mbps

```

[1] ALGORITMA SHORTEST PATH (DIJKSTRA)
    - Jalur Terpilih : A -> B -> F
    - Total Cost      : 2 (Hops)
    - Throughput      : 10 Mbps

[2] ALGORITMA BOTTLENECK PATH (MAX-MIN)
    - Jalur Terpilih : A -> C -> D -> E -> F
    - Total Cost      : 4 (Hops)
    - Throughput      : 500 Mbps

```

Gambar 4.1.2.1. *Screenshot* hasil simulasi membandingkan jalur dan *throughput* algoritma Dijkstra (SPF) dan Modified Dijkstra (HBB) dengan *Python*.

4.2. Analisis Komparatif Kinerja

Perbandingan kinerja kedua algoritma berdasarkan parameter evaluasi (Tabel 2) disajikan dalam tabel berikut:

Parameter Evaluasi	Shortest Path (SPF)	Bottleneck Path (HBB)	Selisih/Peningkatan
Jalur Terpilih	$A \rightarrow B \rightarrow F$	$A \rightarrow C \rightarrow D \rightarrow E \rightarrow F$	–
Hop Count (Minimize)	2	4	SPF lebih efisien 2 hop
End-to-End Throughput (Maximize)	10 Mbps	500 Mbps	HBB unggul 490 Mbps
Path Efficiency	Rendah (Dibatasi Bottleneck)	Tinggi (Optimal)	–

Tabel 4.2.1. Perbandingan Hasil Simulasi

Data di atas menunjukkan disparitas kinerja yang signifikan. Penggunaan algoritma Shortest Path mengakibatkan penurunan potensi *throughput* jaringan secara drastis karena terjebak pada *link B → F* yang hanya memiliki kapasitas 10 Mbps. Sebaliknya, pendekatan Bottleneck Path mampu memanfaatkan jalur alternatif yang memiliki kapasitas konsisten 500 Mbps.

Dalam konteks pengiriman data bervolume besar, peningkatan *throughput* dari 10 Mbps menjadi 500 Mbps (peningkatan 50 kali lipat) oleh algoritma HBB jauh lebih baik dibandingkan penghematan latensi dari selisih 2 *hop* yang diberikan oleh SPF.

4.3 Pembahasan

Hasil simulasi ini menyoroti kelemahan fundamental dari metrik routing tunggal berbasis jarak (*additive metric*) dalam menangani kebutuhan jaringan modern. Algoritma Dijkstra standar mengabaikan kapasitas *bandwidth*, sehingga rentan menyebabkan kongesti pada satu titik *bottleneck* meskipun jalur tersebut adalah yang terpendek secara topologi.

Penerapan algoritma Bottleneck Path (Max–Min) terbukti efektif memecahkan masalah ini dengan mengubah paradigma operasi matriks dari $(\min, +)$ menjadi (\max, \min) . Algoritma ini secara proaktif menghindari tautan padat atau sempit yang biasanya dipilih oleh Shortest Path.

Ini relevan dalam implementasi *Software Defined Network* (SDN). Dengan visibilitas penuh terhadap topologi global, *controller* SDN dapat mendeteksi adanya *link bottleneck* (seperti link $B \rightarrow F$) dan mengarahkan lalu lintas data besar melalui jalur yang lebih lebar ($A \rightarrow C \rightarrow D \rightarrow E \rightarrow F$), memaksimalkan utilisasi jaringan secara keseluruhan.

Meskipun demikian, perlu dicatat adanya *trade-off*. Algoritma Bottleneck Path menggunakan lebih banyak sumber daya jaringan (4 *link* aktif dibanding 2 *link* pada SPF). Oleh karena itu, pendekatan ini paling optimal diterapkan secara adaptif, dengan menggunakan SPF untuk paket kecil yang sensitif terhadap latensi, dan menggunakan HBB untuk transfer data bervolume besar.

5. Kesimpulan

Berdasarkan rangkaian simulasi dan analisis komparatif yang telah dilakukan dalam penelitian ini, kami dapat menarik benang merah mengenai efektivitas algoritma *routing* dalam menangani tantangan jaringan modern. Berikut adalah poin-poin kesimpulan utama:

1. *"Jalan Pintas" Tidak Selalu Lebih Cepat* Penelitian ini membuktikan bahwa algoritma *Shortest Path* (seperti Dijkstra), yang selama ini menjadi standar industri, memiliki keterbatasan fatal dalam skenario transfer data besar. Meskipun algoritma ini sangat efisien dalam mencari rute terpendek secara topologi (hanya 2 *hops*), ia "buta" terhadap kondisi kapasitas jalan. Akibatnya, lalu lintas data terjebak pada jalur sempit (*bottleneck*), yang membuat *throughput* jatuh drastis ke angka 10 Mbps .
2. *Efektivitas Pendekatan "Jalan Lebar" (HBB)* Sebaliknya, algoritma *Highest Bottleneck Bandwidth* (HBB) menunjukkan keunggulannya dengan mengubah cara pandang dalam memilih rute. Dengan tidak terpaku pada jarak terdekat, algoritma ini mampu menghindari jalur yang macet dan secara cerdas memilih rute alternatif yang lebih panjang (4 *hops*), namun memiliki kapasitas "pipa" yang jauh lebih besar. Hasilnya, *throughput* yang didapatkan stabil di angka 500 Mbps .
3. *Lonjakan Kinerja yang Signifikan* Data simulasi menunjukkan perbedaan performa yang sangat mencolok. Penerapan algoritma HBB mampu meningkatkan *throughput* hingga 50 kali lipat dibandingkan algoritma *Shortest Path* . Temuan ini menegaskan bahwa untuk kebutuhan transfer data bervolume besar, mengorbankan sedikit latensi (akibat rute yang memutar) adalah harga yang pantas dibayar demi mendapatkan kecepatan transfer yang maksimal.
4. *Masa Depan pada Jaringan SDN* Implementasi algoritma HBB ini menjadi sangat relevan dan memungkinkan berkat arsitektur *Software Defined Network* (SDN). Keberadaan *controller* pusat yang memiliki "pandangan mata burung" (*global view*) terhadap seluruh jaringan memungkinkan deteksi dini terhadap tautan yang mengalami *bottleneck*. Hal ini memungkinkan jaringan untuk secara dinamis mengarahkan lalu lintas ke jalur terbaik sebelum kemacetan terjadi .

6. Daftar Pustaka

- Deaconu, A. M., & Tayyebi, J. (2020). Inverse maximum capacity path problems under sum-type and max-type distances and their practical application to transportation networks. *IEEE Access*.
- Duan, R. (2010). Fast algorithms for (max, min)-matrix multiplication and bottleneck shortest paths. *Proceedings of the SODA*.
- Jain, S., Kumar, A., Mandal, S., Ong, J., Poutievski, L., Singh, A. & Vahdat, A. (2013). B4: Experience with a globally-deployed software defined WAN. *ACM SIGCOMM Computer Communication Review*, 43(4), 3-14.
- Riedl, A., & Schupke, D. A. (2007). Routing optimization in IP networks utilizing additive and concave link metrics. *IEEE/ACM Transactions on Networking*, 15(5), 1136-1149.
- Sajid, A. S., Niloy, S. F. N., Hossain, K. A., & Rahman, T. (2018). Comprehensive Evaluation of Shortest Path Algorithms and Highest Bottleneck Bandwidth Algorithm in Software Defined Networks (Bachelor's thesis, BRAC University).
- Van Mieghem, P., Kuipers, F. A., Korkmaz, T., Krunz, M., Curado, M., Monteiro, E.& Sanchez-Lopez, S. (n.d.). Quality of Service Routing.