Jurnal Teknologi Informasi dan Ilmu Komputer (JTIIK) DOI: 10.25126/jtiik.202072402
Vol. 7, No. 5, Oktober 2020, hlm. 961-970 p-ISSN: 2355-7699
Akreditasi KEMENRISTEKDIKTI, No. 36/E/KPT/2019 e-ISSN: 2528-6579

ANALISIS KINERJA ALGORITME TCP *CONGESTION CONTROL* BERDASARKAN *SINGLE* DAN *MULTIPLE FLOW* PADA *MULTI-PATH ROUTING*

Bayu Sutawijaya*[1], Achmad Basuki[2], Fitra Abdurrachman Bachtiar[3]

[1, 2, 3] Fakultas Ilmu Komputer, Universitas Brawijaya
Email: [1]bayu.sutawijaya@student.ub.ac.id, [2]abazh@ub.ac.id, [3]fitra.bachtiar@ub.ac.id
*Penulis Korespondensi

(Naskah masuk: 19 Agustus 2019, diterima untuk diterbitkan: 05 Oktober 2020)

Abstrak

Teknik *multi-path routing* merupakan solusi efektif untuk menambah kapasitas *bandwidth* jaringan. Namun, TCP menggunakan *multiple paths* sama dengan di *single path*. Penelitian ini melakukan analisis kinerja algoritme TCP *congestion control* Reno, BIC, CUBIC, dan BBR pada *multi-path routing* dengan setiap *multiple paths* menggunakan *cost* yang sama. Analisis yang dilakukan meliputi perbandingan antara *single path routing* dengan *multi-path routing*, *single flow*, dan *multiple flow*. Analisis *single flow* meliputi *link delay* dan *loss rate*. Sedangkan analisis *multiple flow* meliputi *inter* TCP *protocol fairness* dan *fairness* antara TCP dengan UDP. Semua evaluasi dilakukan berdasarkan emulasi pada VirtualBox. Berdasarkan hasil emulasi, *multi-path routing* dapat berdampak pada *packet reordering*, tetapi tidak mengakibatkan penurunan rata-rata *throughput* yang signifikan. Pada *single flow*, BBR merupakan algoritme TCP *congestion control* terbaik pada *multi-path routing*. Namun, pada *multiple flow*, CUBIC merupakan algoritme TCP *congestion control* terbaik pada *multi-path routing*. Pada evaluasi *link delay*, rata-rata RTT BBR lebih rendah hingga 58 ms dibandingkan dengan Reno, BIC, dan CUBIC. Sedangkan pada evaluasi *loss rate*, rata-rata *throughput* BBR lebih tinggi hingga 12 Mbps dibandingkan dengan Reno, BIC, dan CUBIC. Pada evaluasi *inter* TCP *protocol fairness* dan *fairness* antara TCP dengan UDP, *fairness* CUBIC paling mendekati nilai 1 dibandingkan dengan Reno, BIC, dan BBR.

Kata kunci: *kinerja TCP*, *TCP fairness, multi-path routing*

PERFORMANCE ANALYSIS OF TCP CONGESTION CONTROL ALGORITHMS BASED ON SINGLE AND MULTIPLE FLOW UNDER MULTI-PATH ROUTING

Abstract

The multi-path routing technique is an effective solution to increase network bandwidth capacity. However, TCP uses multiple paths similar to a single path. This study analyzes the performance of TCP congestion control algorithms Reno, BIC, CUBIC, and BBR on multi-path routing with each multiple paths using the same cost. The analysis includes a comparison between single path routing and multi-path routing, single flow, and multiple flows. In a single flow, the analysis includes link delay and loss rate. Whereas in multiple flows, the analysis includes inter TCP protocol fairness and fairness between TCP and UDP. All evaluations are based on emulation in VirtualBox. Based on the results from emulation, multi-path routing can have an impact on packet reordering but does not result in a significant degrade in average throughput. In a single flow, BBR is the best TCP congestion control algorithm on multi-path routing. However, in multiple flows, CUBIC is the best TCP congestion control algorithm on multi-path routing. In the link delay evaluations, the average RTT on BBR up to 58 ms lower than Reno, BIC, and CUBIC. Whereas in the loss rate evaluations, the average throughput on BBR up to 12 Mbps higher than Reno, BIC, and CUBIC. In the evaluation of inter TCP protocol fairness and fairness between TCP and UDP, fairness on CUBIC is closest to 1 than Reno, BIC, and BBR.

Keywords: *TCP performance, TCP fairness, multi-path routing*

1. PENDAHULUAN

Transmission Control Protocol (TCP) merupakan protokol *transport* yang paling banyak digunakan di jaringan Internet. TCP banyak digunakan karena kemampuan pengiriman TCP yang reliabel (Ros & Welzl, 2013). Salah satu mekanisme dalam implementasi TCP yang paling penting dan kompleks adalah algoritme TCP *congestion control* (Carpa, dkk., 2017). Fungsi algoritme TCP *congestion control* adalah mengatur

kecepatan pengiriman *packet* untuk mencegah *congestion* pada jaringan (Kurose & Ross, 2017).

Namun, desain awal algoritme TCP *congestion control* adalah untuk *single path routing* (Carpa, dkk., 2017). Pada *single path routing*, pengiriman *packet* hanya dilakukan melalui satu jalur saja, walaupun *Internet Service Provider* (ISP) menyediakan dua atau lebih jalur menuju *destination host*. Untuk menambah kapasitas *bandwidth* jaringan, jalur-jalur yang disediakan oleh ISP dapat digunakan untuk *load balancing* trafik jaringan melalui teknik *multi-path routing* (Chaitanya & Varadarajan, 2016).

Multi-path routing adalah teknik *routing* yang dilakukan dengan cara membagi trafik jaringan melalui *multiple paths*. Distribusi *flow* pada *multi-path routing* dapat berbasis *packet,* yaitu *sub-flow* dikirimkan secara merata ke dalam *multiple paths*. Penggunaan *multi-pah routing* telah terbukti dapat meningkatkan *throughput* dan menurunkan *Round-Trip Time* (RTT) dibandingkan dengan *single path routing* (Liu, dkk., 2014; Kanagevlu & Aung, 2015).

Namun, hanya sedikit yang dapat diketahui tentang dampak kinerja algoritme TCP *congestion control* pada *multi-path routing*. *Multi-path routing* dapat mengakibatkan *packet reordering* jika *cost* pada *multiple paths* berbeda. *Packet reordering* dapat mengakibatkan penurunan *throughput* pada algoritme TCP *congestion control* (Carpa, dkk*.,* 2017; Karlsson, dkk., 2012). Oleh karena itu, perlu dilakukan analisis lebih lanjut kinerja algoritme TCP *congestion control* pada *multi-path routing*.

Kinerja algoritme TCP *congestion control* pada *multi-path routing* dipengaruhi oleh *single flow* dan *multiple flow*. Pada *single flow*, *packet* dikirimkan dari satu *source host* menuju satu *destination host*. Sedangkan pada *multiple flow*, *packet* dikirimkan dari dua atau lebih *source host* menuju dua atau lebih *destination host.* Kinerja algoritme TCP *congestion control* pada *single flow* dipengaruhi oleh *link delay* dan *loss rate* (Lukaseder, dkk., 2016). Sedangkan kinerja algoritme TCP *congestion control* pada *multiple flow* dipengaruhi oleh *fairness* antar algoritme TCP *congestion control* (*inter* TCP *protocol fairness*) dan *fairness* antara TCP dengan UDP (Yue, dkk*.,* 2012; Kurose & Ross, 2017).

Analisis *single flow* dan *multiple flow* telah dilakukan sebelumnya. Yue, dkk. (2012) melakukan evaluasi *link delay*, *loss rate*, dan *inter* TCP *protocol fairnes.* Kemudian, Arokkiam, dkk. (2014) melakukan evaluasi *link delay* dan *fairness* antara TCP dengan UDP. Sedangkan Lukaseder, dkk. (2016) melakukan evaluasi *link delay*, *loss rate*, dan *inter* TCP *protocol fairness.* Terakhir, Hock, Bless, & Zitterbart. (2017) melakukan evaluasi *link delay* dan *inter* TCP *protocol fairness*. Namun, semua penelitian tersebut hanya dilakukan pada *single path routing*.

Oleh karena itu, penelitian ini melakukan analisis perbandingan kinerja beberapa algoritme TCP *congestion control* pada *multi-path routing.* Setiap jalur pada *multiple paths* menggunakan *cost* yang sama karena kondisi tersebut merupakan kondisi terbaik pada *multiple paths* (Dixit, dkk., 2013). Analisis yang dilakukan meliputi perbandingan antara *single path routing* dengan *multi-path routing*, *link delay*, *loss rate*, *inter* TCP *protocol fairness,* dan *fairness* antara TCP dengan UDP. Algoritme TCP *congestion control* yang dilakukan perbandingan adalah Reno (Jacobson, 1990), *Binary Increase Congestion Control* (BIC) (Xu, Harfoush, & Rhee, 2004), CUBIC (Ha, Rhee, & Xu, 2008), dan *Bottleneck Bandwidth and Round-trip propagation time* (BBR) (Cardwell dkk., 2016a). Reno, BIC, dan CUBIC merupakan algoritme TCP *congestion control* yang paling banyak digunakan pada sistem operasi sampai saat ini. Sedangkan BBR merupakan algoritme TCP *congestion control* yang paling baru untuk saat ini.

2. METODE PENELITIAN

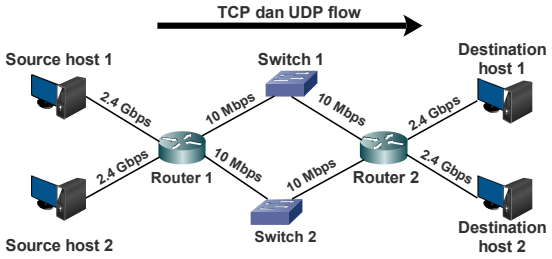
2.1. Perancangan *Testbed*

Untuk melakukan evaluasi kinerja algoritme TCP *congestion control* pada *multi-path routing,* maka dilakukan pembuatan *testbed*. *Testbed* dibuat dengan menggunakan *virtual machine* (VM) pada VirtualBox (VirtualBox, n.d.). VirtualBox berjalan pada komputer dengan spesifikasi CPU Intel Core i5-7200U dan RAM 8 GB. *Testbed* pada penelitian ini menggunakan 8 VM yang terdiri dari 4 *host*, 2 router, dan 2 switch. Semua VM disusun sehingga membentuk topologi seperti pada Gambar 2. Setiap VM dihubungkan dengan *virtual ethernet*. Sistem operasi yang digunakan semua VM adalah Ubuntu Server 16.04 dengan Linux Kernel 4.15. Setiap VM menggunakan CPU 1 *core* dan RAM 768 MB. Khusus pada router 1 menggunakan RAM 2 GB karena router 1 digunakan untuk *capture* TCP *packet*. Kemudian, setiap *interface* pada semua VM menggunakan Intel PRO/1000 MT Desktop.

Switch diimplementasikan pada setiap *multiple paths* untuk melakukan emulasi *cost*. Masing-masing switch menggunakan *cost* yang sama. *Cost* yang digunakan adalah *bottleneck bandwidth*, *link delay*, *loss rate*, dan ukuran *buffer*. Emulasi *bottleneck bandwidth* dilakukan dengan menggunakan *Token Bucket Filter* (TBF) (Kuznetsov, n.d.) sebesar 10 Mbps. Konfigurasi TBF dilakukan pada kedua *interface* setiap switch. *Bandwidth* antara *host* dengan router adalah *bandwidth default* sistem VirtualBox, yang setelah dilakukan pengukuran adalah 2.4 Gbps. Oleh karena itu, *bandwidth* maksimum pada saat pengiriman *packet* dari *source host* menuju *destination host* dibatasi oleh konfigurasi TBF.

Selanjutnya, untuk melakukan emulasi *link delay*, *loss rate*, dan ukuran *buffer* pada switch, penelitian ini menggunakan NetEM (NetEM, n.d.). Konfigurasi *link delay* pada semua evaluasi adalah

sebesar 10 ms, kecuali pada evaluasi variasi *link delay* yang mengikuti skenario evaluasi. *Link delay* pada penelitian ini merupakan representasi dari *two-way propagation delay* antara router 1 dan router 2. Oleh karena itu, untuk melakukan emulasi *link delay* 10 ms, setiap *interface* switch dilakukan konfigurasi sebesar 5 ms. Sedangkan *link delay* antara *host* dan router merupakan *link delay default* dari sistem VirtualBox, yang setelah dilakukan pengukuran adalah 0.1 ms. Oleh karena itu, *delay* antara *source*



Gambar 1. Topologi *testbed*.

host dengan *destination host* lebih dipengaruhi oleh emulasi *link delay* pada NetEM. Selanjutnya, untuk konfigurasi *loss rate* akan disesuaikan dengan skenario evaluasi. Sedangkan konfigurasi ukuran *buffer* setiap switch adalah 100 *packet*.

Setiap switch terhubung dengan dua router yang berfungsi untuk melakukan *multi-path routing*. Jumlah jalur yang digunakan untuk *multi-path routing* adalah dua jalur. Untuk mendistribusikan *packet* ke dalam *multiple paths*, maka penelitian ini menggunakan *True (or Trivial) Link Equalizer* (TEQL) (Hubert, dkk., 2002). TEQL mendistribusikan *packet* secara merata ke dalam *multiple paths* berdasarkan *round-robin. Round-robin* cocok digunakan pada *multiple paths* yang menggunakan cost sama (Singh, Das, & Jukan, 2015).

Setiap router terhubung dengan *host* yang digunakan untuk evaluasi algoritme TCP *congestion control*. Penelitian ini menggunakan 4 *host*, yaitu 2 *source host* dan 2 *destination host*. Pada setiap *host,* dilakukan pengaktifan *Selective Acknowledgment* (SACK), *Duplicate* SACK (DSACK), *timestamp*, dan *window scaling option.*

Untuk melakukan observasi kinerja algoritme TCP *congestion control* pada *multi-path routing*, maka untuk mengaktifkan Reno, BIC, CUBIC, dan BBR cukup dilakukan pada *source host* (Cardwell, dkk., 2016). CUBIC merupakan algoritme TCP *congestion control default* di Linux Kernel 4.15. Untuk mengaktifkan Reno, BIC, dan BBR, dilakukan dengan mengaktifkan modul Reno, BIC, dan BBR pada Linux. Parameter yang digunakan Reno, BIC, CUBIC, dan BBR adalah parameter *default* dari Linux Kernel 4.15. Khusus pada BBR, dilakukan tambahan konfigurasi, yaitu mengaktifkan *fair queuing* pada *source host* (Cheng & Cardwell, 2016).

2.2. Pengukuran Kinerja

Untuk mengukur kinerja Reno, BIC, CUBIC, dan BBR pada jaringan *multi-path routing*, maka evaluasi dilakukan berdasarkan parameter pada Tabel 1. Untuk semua evaluasi, pengiriman TCP/UDP *flow* dilakukan dengan menggunakan iPerf3 (iPerf, n.d.) selama 100 detik. Selama evaluasi berlangsung, dilakukan *capture* TCP *packet* dengan menggunakan Tcpdump (Tcpdump, n.d.) sebagai bahan analisis. Evaluasi dilakukan berdasarkan tiga skenario, yaitu perbandingan

Tabel 1. Parameter evaluasi

Parameter	Deskripsi
Waktu evaluasi	100 detik
TCP dan UDP *packet generator*	iPerf3
Packet monitoritng	Tcpdump
Skenario evaluasi	Perbandingan antara *single path routing* dengan *multi-path routing, link delay, loss rate, inter* TCP *protocol fairness,* dan *fairness* antara TCP dengan UDP
Perulangan evaluasi	10 kali
Parameter evaluasi	*Throughput,* RTT, SACK blocks, *Jain's fairness index*

antara *single path routing* dengan *multi-path routing*, *single flow*, dan *multiple flow*. Untuk menjaga tingkat akurasi, maka setiap skenario evaluasi dilakukan perulangan sebanyak 10 kali.

Evaluasi pertama adalah perbandingan antara *single path routing* dengan *multi-path routing.* Evaluasi dilakukan dengan mengirimkan satu TCP *flow* dari *source host* 1 menuju *destination host* 1. TCP *flow* pertama dikirimkan melalui satu jalur (*single path routing*), sedangkan TCP *flow* kedua dikirimkan melalui dua jalur (*multi-path routing*). Algoritme TCP *congestion control* Reno, BIC, CUBIC, dan BBR dilakukan evaluasi secara bergantian. Berdasarkan hasil *evaluasi*, kemudian dilakukan perbandingan jumlah SACK *blocks* dan rata-rata *throughput* Reno, BIC, CUBIC, dan BBR antara *single path routing* dengan *multi-path routing*.

Evaluasi kedua adalah *single flow* yang dilakukan dengan cara mengirimkan satu TCP *flow* ke dalam *multiple paths*. TCP *flow* dikirimkan dari *source host* 1 menuju *destination host* 1. Pada evaluasi *single flow*, kinerja yang diukur adalah rata-rata *throughput* dan rata-rata RTT. Kinerja *single flow* diukur berdasarkan 2 skenario evaluasi yaitu:

1) Variasi *link delay*: Evaluasi dilakukan berdasarkan variasi *link delay* 10 ms – 100 ms dengan kelipatan 10 ms untuk setiap pengukuran kinerja. Algoritme TCP *congestion control* Reno, BIC, CUBIC, dan BBR dilakukan evaluasi secara bergantian untuk setiap variasi *link delay*.

2) Variasi *loss rate*: Evaluasi dilakukan berdasarkan variasi *loss rate* 1% – 10% dengan

kelipatan 1% untuk setiap pengukuran kinerja. *Loss rate* hanya dilakukan pada saat pengiriman *packet* dari *source host* menuju *destinaton host*. Algoritme TCP *congestion control* Reno, BIC, CUBIC, dan BBR dilakukan evaluasi secara bergantian untuk setiap variasi *loss rate*.

Evaluasi ketiga adalah *multiple flow* yang dilakukan dengan cara mengirimkan dua *flow* ke dalam *multiple paths*. *Flow* pertama dikirimkan dari *source host* 1 menuju *destination host* 1, dan *flow* kedua dikirimkan dari *source host* 2 menuju *destination host* 2. Kinerja *multiple flow* diukur berdasarkan 2 skenario evaluasi, yaitu:

Tabel 2. Konfigurasi *host inter* TCP *protocol fairness*

Evaluasi	*Source host* 1	*Source Host* 2
1	CUBIC	Reno
2	BIC	CUBIC
3	BIC	Reno
4	BBR	CUBIC
5	BBR	Reno
6	BBR	BIC

1) *Inter* TCP *protocol fairness*: Evaluasi dilakukan dengan mengirimkan dua TCP *flow* dengan setiap *flow* menggunakan algoritme TCP *congestion control* yang berbeda. *Source host* 1 mengirimkan TCP *flow* dari detik ke 1 – 100 dan *source host* 2 mengirimkan TCP *flow* dari detik ke 6 – 100. Konfigurasi algoritme TCP *congestion control* pada *source host* 1 dan *source host* 2 untuk setiap evaluasi dilakukan berdasarkan Tabel 2.

2) *Fairness* antara TCP dengan UDP: Evaluasi dilakukan dengan *source host* 1 mengirimkan TCP *flow* dari detik ke 1 – 100 dan *source host* 2 mengirimkan UDP *flow* dari detik ke 6 – 100. Pengiriman UDP *flow* dilakukan berdasarkan variasi *constant bit rate* (CBR) 2 Mbps – 20 Mbps dengan kelipatan 2 Mbps untuk setiap pengukuran kinerja. Algoritme TCP *congestion control* Reno, BIC, CUBIC, dan BBR dilakukan evaluasi secara bergantian untuk setiap variasi CBR.

Pada *multiple flow*, kinerja yang diukur adalah *fairness*. *Fairness* dihitung berdasarkan *Jain's fairness index* (Jain, Chiu, & Hawe, 1984) pada persamaan (1). Pada persamaan (1), x_i merupakan *throughput* dari *flow* i dengan total *flow* n. *Jain's fairness index* memiliki rentang nilai 0 – 1. Algoritme TCP *congestion control* dikatakan mendapatkan *fairness* paling tinggi jika *Jain's fairness index* bernilai 1.

$$J(x_1, x_2, \dots, x_n) = \frac{(\sum_{i=1}^{n} x_i)^2}{n * \sum_{i=1}^{n} x_i^2}, \quad J \in [0,1] \qquad (1)$$

3. PERILAKU TCP *CONGESTION CONTROL* PADA *MULTI-PATH ROUTING*

3.1. Algoritme TCP *Congestion Control*

Algoritme TCP *congestion control* menggunakan mekanisme *congestion window* (cwnd) untuk menentukan jumlah pengiriman *packet*. Apabila *bandwidth* jaringan masih tersedia, algoritme TCP *congestion control* menaikkan cwnd. Namun, jika jaringan mengalami *congestion*, maka algoritme TCP *congestion control* menurunkan cwnd untuk mencegah *congestion* terjadi terus-menerus (Kurose & Ross, 2017). Algoritme TCP *congestion control* yang digunakan pada penelitian ini adalah sebagai berikut.

1) Reno

Reno menggunakan teknik *Additive Increase Multiplicative Decrease* (AIMD) untuk menentukan ukuran cwnd. Pada *additive increase*, Reno menaikkan cwnd sebesar 1 *packet* untuk setiap RTT. Apabila terjadi *packet loss*, Reno menjalankan *multiplicative decrease*, yaitu menurunkan cwnd sebesar setengah (Jacobson, 1990).

2) BIC

BIC menggunakan tiga fase untuk menentukan ukuran cwnd, yaitu *additive increase*, *binary search increase*, dan *slow start max probing*. Apabila jarak antara cwnd dengan *titik packet loss* masih jauh, BIC melakukan fase *additive incrase*, yaitu menaikkan cwnd secara agresif. Kemudian, jika cwnd mendekati titik *packet loss*, maka BIC melakukan fase *binary search increase*, yaitu menurunkan keagreasifan dalam menaikkan cwnd. Apabila cwnd lebih tinggi dari titik *packet loss* yang sebelumnya, maka BIC mengindikasikan terjadi perubahan pada *bandwidth* jaringan. Oleh karena itu, BIC melakukan fase *slow start max probing,* yaitu menaikkan cwnd secara agresif untuk mencari titik *packet loss* yang baru. (Xu, Harfoush, & Rhee, 2004).

3) CUBIC

CUBIC memulai pengiriman *packet* melalui fase *hybrid slow start* (Ha & Rhee, 2008), yaitu menaikkan cwnd secara agresif dan berhenti jika memenuhi salah satu dari dua kondisi. Kondisi pertama adalah pada saat cwnd mencapai estimasi BDP. Kondisi kedua adalah pada saat *delay* mengalami kenaikan yang signifikan dibandingkan dengan *delay* yang sebelumnya. Kemudian, CUBIC melakukan fase *congestion avoidance*. Pada fase *congestion avoidance*, CUBIC menaikkan cwnd berdasarkan fungsi *cubic* (Ha, Rhee, & Xu, 2008).

4) BBR

BBR menggunakan estimasi dari *Bandwidth Delay Product* (BDP) untuk menentukan ukuran cwnd. Estimasi BDP pada BBR merupakan estimasi dari *bottleneck bandwidth* maksimum dan RTT minimum jaringan. BBR menaikkan cwnd berdasarkan empat fase. Fase pertama adalah *startup*, yaitu menaikkan cwnd secara agresif sampai titik BDP jaringan. Fase kedua adalah *drain*, yaitu menurunkan cwnd karena fase *startup* mengakibatkan *packet* BBR mengisi *buffer*. Fase ketiga adalah *probe bandwidth* untuk melakukan estimasi *bottleneck bandwidth* maksimum. Fase keempat adalah *probe* RTT untuk melakukan estimasi RTT minimum (Cardwell, dkk., 2017).

Gambar 2 menunjukkan perbandingan perilaku perubahan cwnd terhadap BDP untuk TCP Reno, BIC, CUBIC, dan BBR pada lingkungan pengujian dengan *link delay* 100 ms, *loss rate* 0%, dua *link paths* dengan *bandwidth* masing-masing 10 Mbps (kumulatif 20 Mbps). Reno, BIC, dan CUBIC menaikkan cwnd secara bertahap untuk mencapai kapasitas maksimum *bandwidth* jaringan. Apabila cwnd Reno, BIC, dan CUBIC telah mencapai kapasitas *bandwidth* maksimum jaringan, *packet* Reno, BIC, dan CUBIC mulai mengisi *buffer,* seperti yang ditunjukkan pada saat ukuran cwnd lebih tinggi dibandingkan dengan BDP. Reno, BIC, dan CUBIC terus mengisi *buffer* sampai mendeteksi *packet loss* karena *buffer* terisi penuh. Pada saat jumlah *packet loss* melebihi batas toleransi, Reno, BIC, dan CUBIC menurunkan cwnd karena Reno, BIC, dan CUBIC mendeteksi *packet loss* sebagai tanda *congestion* pada jaringan.

Namun, pada saat pengujian selama 100 detik, BIC dan CUBIC lebih sering mengalami *packet loss* dibandingkan dengan Reno. Penyebabnya adalah BIC dan CUBIC tidak menurunkan ukuran *window* sedrastis Reno pada saat terjadi *packet loss*. Selain itu, BIC dan CUBIC menaikkan cwnd lebih agresif dibandingkan dengan Reno. Oleh karena itu, BIC dan CUBIC membutuhkan waktu yang lebih pendek dibandingkan dengan Reno untuk menaikkan cwnd sampai terjadi *packet loss.*

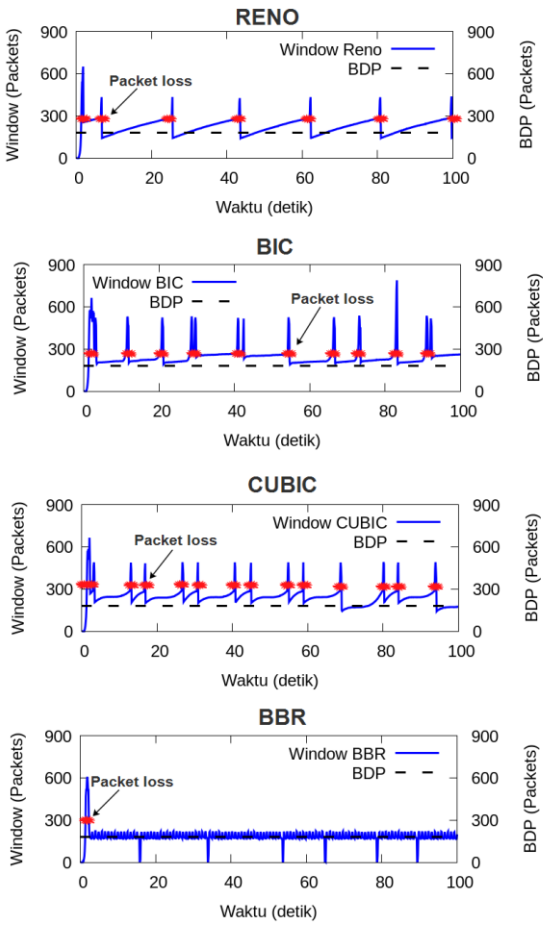
Sebaliknya, BBR mendefinisikan *congestion* pada saat *packet* mulai mengisi *buffer*. Pada saat awal pengiriman *packet*, BBR menaikkan cwnd secara agresif untuk melakukan pencarian BDP jaringan. Kenaikan cwnd secara agresif tersebut dapat mengakibatkan BBR mengalami *packet loss.*

Setelah BBR melakukan estimasi BDP, BBR menjaga ukuran cwnd di sekitar BDP. Kemudian, BBR secara berkala melakukan estimasi BDP kembali. Untuk melakukan estimasi *bottleneck bandwidth*, BBR menaikkan cwnd. Apabila *bottleneck bandwidth* tidak berubah, cwnd BBR akan sedikit lebih tinggi dibandingkan dengan BDP. Kemudian, untuk melakukan estimasi RTT, BBR menurunkan cwnd secara drastis.

3.2. *Multi-path Routing*

Penggunaan algoritme TCP *congestion control* pada *multi-path routing* dapat mengakibatkan *packet reordering*. *Packet reordering* dapat disebabkan oleh perbedaan *link delay*, perbedaan ukuran *packet*, dan perbedaan waktu transmisi (Dixit dkk., 2013). *Packet reordering* dapat berdampak pada perilaku dan kinerja algoritme TCP *congestion control*, yaitu:

1) *Packet rerdering* dapat menyebabkan algoritme TCP *congestion control* mendeteksi *packet reordering* sebagai *packet loss.* Oleh karena itu, *packet reordering* dapat mengakibatkan algoritme TCP *congestion control* berbasis *packet loss* menurunkan ukuran cwnd sebelum *bandwidth* jaringan terisi penuh (Bennett, Partridge, & Shectman, 1999).



Gambar 2. Perilaku Reno, BIC, CUBIC, dan BBR

2) *Packet reordering* dapat memicu algoritme TCP *congestion control* melakukan *retransmission,* walaupun tidak terjadi *packet loss.* Oleh karena itu, *packet* yang sama akan dikirimkan dua kali (Bennett, Partridge, & Shectman, 1999).

Linux memiliki mekanisme untuk mengatasi permasalahan *packet reordering*. Namun, mekanisme pada Linux dapat bekerja dengan baik jika *link delay* pada *multiple paths* hanya berbeda sedikit dan *source host* menerima ACK secara urut (Karlsson, dkk., 2012; Carpa, dkk., 2017).

4. HASIL EVALUASI DAN PEMBAHASAN

4.1. Perbandingan Antara *Single Path Routing* dengan *Multi-path Routing*

Evaluasi ini dilakukan untuk mengetahui perbedaan perilaku dan kinerja Reno, BIC, CUBIC, dan BBR antara *single path routing* dengan *multi-path routing*. Tabel 3 menunjukkan perbandingan jumlah SACK *blocks* dan rata-rata *throughput* Reno, BIC, CUBIC, dan BBR antara *single path routing* dengan *multi-path routing*. Pada *multi-path routing,* jumlah SACK *blocks* Reno, BIC, CUBIC, dan BBR mengalami peningkatan yang signifikan

dibandingkan dengan *single path routing*, yaitu lebih dari 11 kali lipat. Pada *single path routing*, *destination host* hanya mengirimkan SACK *blocks* pada saat terjadi *packet loss*. Namun, pada *multi-path routing*, *destination host* terus-menerus mengirimkan SACK *blocks* selama 100 detik, walaupun tidak pada saat terjadi *packet loss.*

Tabel 3. Perbandingan *single path routing* dan *multi-path routing*

TCP	*Single path routing*		*Multi-path routing*	
	SACK *blocks*	*Throughput (Mbps)*	SACK *blocks*	*Throughput (Mbps)*
Reno	1780	9.57	73263	19.1
BIC	12089	9.57	129205	19.14
CUBIC	4541	9.57	177040	19.08
BBR	0	9.45	70380	18.69

Fenomena SACK *blocks* pada *multi-path routing* menunjukkan bahwa *multi-path routing* dapat mengakibatkan *packet reordering*, walaupun *multiple paths* menggunakan *cost* yang sama dan TEQL mendistribusikan *packet* secara merata. Packet *reordering* yang terjadi pada penelitian ini disebabkan oleh sedikit perbedaan waktu pengiriman *packet* antara switch 1 dengan switch 2

Namun, rata-rata *throughput* Reno, BIC, CUBIC, dan BBR pada *multi-path routing* mengalami peningkatan hampir 2 kali lipat dibandingkan dengan *single path routing*. Fenomena rata-rata *throughput* pada *multi-path routing* menunjukkan bahwa Reno, BIC, CUBIC, dan BBR tetap dapat memaksimalkan *bandwidth* yang tersedia pada *multiple paths*, walaupun terjadi *packet reordering*. Berdasarkan fenomena *packet reordering* dan rata-rata *throughput* pada *multi-path routing,* maka dilakukan observasi lebih lanjut kinerja algoritme TCP *congestion control* pada *multi-path routing* untuk evaluasi *single flow* dan *multiple flow.*

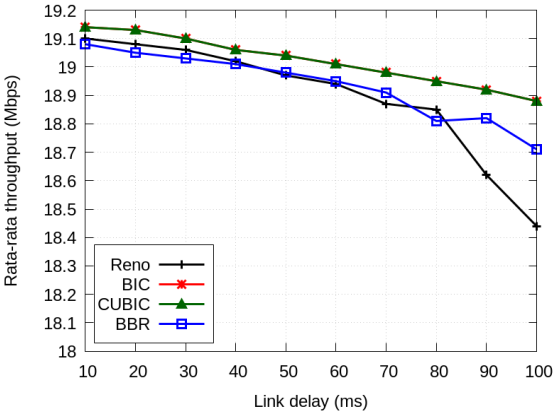
4.2. Evaluasi *Single Flow*

A. Variasi *Link delay*

Evaluasi *single flow* yang pertama adalah melakukan variasi *link delay* pada *multiple paths*. Gambar 3 menunjukkan perbandingan rata-rata *throughput* antara Reno, BIC, CUBIC, dan BBR pada variasi *link delay* 10 ms – 100 ms. Pada variasi *link delay* 10 ms – 100 ms, rata-rata *throughput* Reno, BIC, CUBIC, dan BBR terus mengalami penurunan, tetapi tidak signifikan. Rata-rata *throughput* Reno, BIC, CUBIC, dan BBR tetap tinggi, yaitu di atas 18 Mbps. Fenomena rata-rata *throughput* pada variasi *link delay* 10 ms – 100 ms menunjukkan bahwa Reno, BIC, CUBIC, dan BBR dapat memaksimalkan lebih dari 90% dari *bandwidth* yang tersedia pada *multiple paths* walaupun terjadi *packet reordering*.

Rata-rata *throughput* Reno, BIC, dan CUBIC, terus mengalami sedikit penurunan karena Reno, BIC, dan CUBIC menaikkan cwnd jika menerima ACK. Semakin tinggi *link delay*, semakin lama Reno, BIC, dan CUBIC menerima ACK. Namun, karena BIC dan CUBIC menaikkan cwnd lebih agresif dibandingkan dengan Reno, maka penurunan rata-rata *throughput* BIC dan CUBIC lebih rendah dibandingkan dengan Reno.

Sementara itu, rata-rata *throughput* BBR tinggi karena BBR mengirimkan *packet* sebesar estimasi *bottleneck bandwidth* dan *link delay* pada *multiple*



Gambar 3. Perbandingan *throughput* terhadap variasi *link delay*

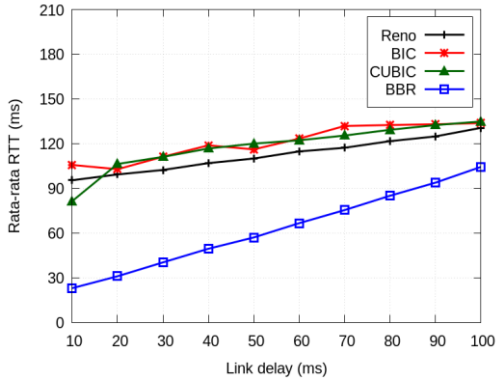
paths. Namun, semakin tinggi *link delay*, semakin rendah estimasi *bottleneck bandwidth* karena BBR melakukan estimasi *bottleneck bandwidth* berdasarkan kecepatan penerimaan ACK. Oleh karena itu, rata-rata *throughput* BBR terus mengalami sedikit penurunan seiring bertambah tinggi *link delay*.

Namun, walaupun rata-rata *throughput* Reno, BIC, CUBIC, dan BBR sama-sama dapat mencapai di atas 18 Mbps, tetapi terdapat perbedaan pada rata-rata RTT. Gambar 4 menunjukkan perbandingan rata-rata RTT antara Reno, BIC, CUBIC pada *variasi link delay* 10 ms – 100 ms. Pada semua variasi *link delay*, rata-rata RTT Reno, BIC, CUBIC, dan BBR cenderung lebih tinggi dari *link delay*. Namun, rata-rata RTT BBR sekitar 26 ms – 58 ms lebih rendah dibandingkan dengan Reno, BIC, dan CUBIC. Fenomena rata-rata RTT yang lebih tinggi dari *link delay* menunjukkan bahwa Reno, BIC, CUBIC, dan BBR cenderung mengirimkan *packet* sampai mengisi *buffer* switch, sehingga dapat mengakibatkan *queuing delay*. Namun, karena BBR lebih sedikit mengisi *packet* ke dalam *buffer*, *destination host* BBR menerima *packet* lebih cepat hingga 58 ms dibandingkan dengan Reno, BIC, dan CUBIC.

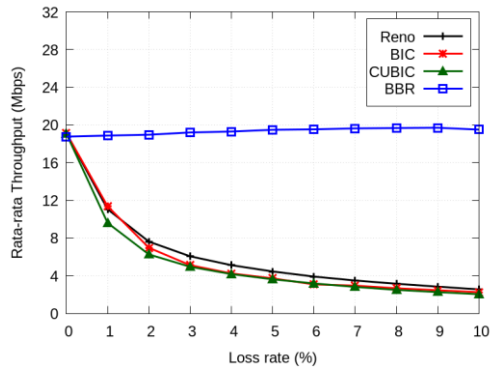
Rata-rata RTT BBR lebih rendah dibandingkan dengan Reno, BIC, dan CUBIC disebabkan BBR menjaga ukuran cwnd sebesar estimasi BDP untuk mencegah mengisi *packet* terlalu banyak ke dalam *buffer*. Sebaliknya, Reno, BIC, dan CUBIC terus mengisi *packet* ke dalam *buffer* sampai penuh dan terjadi *packet loss*. Oleh karena itu, *queuing delay* Reno, BIC, dan CUBIC lebih tinggi dibandingkan dengan BBR.

B. Variasi *Loss Rate*

Evaluasi *single flow* yang kedua adalah melakukan variasi *loss rate* pada *multiple paths*. Gambar 5 menunjukkan perbandingan rata-rata *throughput* Reno, BIC, CUBIC, dan BBR pada variasi *loss rate* 1% – 10%. Pada *loss rate* 1%, rata-rata *throughput* Reno, BIC, dan CUBIC mengalami penurunan sekitar 7 Mbps. Semakin tinggi *loss rate*, rata-rata *throughput* Reno, BIC, dan CUBIC terus



Gambar 4. Perbandingan RTT terhadap variasi *link delay*



Gambar 5. Perbandingan *throughput* terhadap variasi *loss rate*

mengalami penurunan. Pada loss rate 10%, rata-rata *throughput* Reno, BIC, dan CUBIC mengalami penurunan sekitar 17 Mbps. Sebaliknya, rata-rata *throughput* BBR tetap dapat mencapai 19 Mbps pada semua variasi *loss rate*. Fenomena rata-rata *throughput* pada variasi *loss rate* 1% – 10% menunjukkan bahwa BBR tetap dapat memaksimalkan 95% *bandwidth* yang tersedia pada *multiple paths* walaupun pada *loss rate* 10%. Sedangkan Reno, BIC, dan CUBIC hanya dapat memaksimalkan kurang dari 55% *bandwidth* yang tersedia pada *multiple paths* untuk semua variasi *loss rate*. Hasil ini mengonfirmasi penelitian Cardwell, dkk. (2016b) yang dilakukan pada *single path routing*. Oleh karena itu, *packet reordering* yang terjadi pada *multi-path routing* tidak berdampak pada rata-rata *throughput* Reno, BIC, CUBIC, dan BBR untuk evaluasi *loss rate*.

Rata-rata *throughput* Reno, BIC, dan CUBIC mengalami penurunan disebabkan Reno, BIC dan CUBIC menurunkan cwnd secara drastis pada saat terjadi *packet loss*. Reno, BIC, dan CUBIC menurunkan cwnd sebelum mencapai kapasitas *bandwidth* maksimum jaringan. Semakin tinggi *loss rate*, semakin tinggi frekuensi Reno, BIC, dan CUBIC menurunkan cwnd. Oleh karena itu, rata-rata *throughput* Reno, BIC, dan CUBIC terus mengalami penurunan seiring bertambah tinggi *loss rate*.

Sebaliknya, BBR pada saat terjadi *packet loss* tidak menurunkan cwnd sedrastis Reno, BIC, dan CUBIC. Pada saat terjadi *packet loss*, BBR menjaga ukuran cwnd sebesar *packet* yang berhasil dikirimkan. Selain itu, pada saat *retransmission* selesai dilakukan, BBR langsung mengembalikan cwnd ke ukuran sebelum terjadi *packet loss* (Cardwell, dkk., 2017).

Pengukuran kinerja selanjutnya adalah rata-rata RTT Reno, BIC, CUBIC, dan BBR. Gambar 6 menunjukkan perbandingan rata-rata RTT Reno, BIC, CUBIC, dan BBR pada variasi *loss rate* 1% – 10%. Pada *loss rate* 1%, rata-rata RTT Reno, BIC, dan CUBIC mengalami penurunan drastis menjadi sekitar 12 ms. Sebaliknya, rata-rata RTT BBR pada *loss rate* 1% tetap stabil pada sekitar 24 ms. Kemudian, pada loss rate 2% – 10%, rata-rata RTT Reno, BIC, CUBIC, dan BBR cenderung stabil. Fenomena rata-rata RTT pada variasi *loss rate* 1% – 10% menunjukkan bahwa *loss rate* dapat mengakibatkan *destination host* Reno, BIC, dan CUBIC menerima *packet* lebih cepat dibandingkan dengan tanpa *loss rate*. Selain itu, *loss rate* juga dapat mengakibatkan *destination* host Reno, BIC, dan CUBIC menerima *packet* sekitar 12 ms lebih cepat dibandingkan dengan BBR.

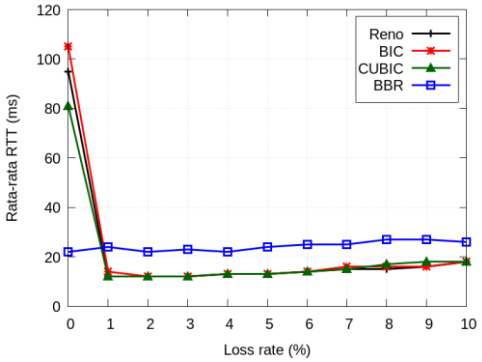
Rata-rata RTT BBR lebih tinggi dibandingkan dengan Reno, BIC, dan CUBIC disebabkan BBR tetap dapat mengirimkan *packet* sampai kapasitas maksimum *bandwidth* jaringan walaupun pada *loss rate* 10%. Oleh karena itu, rata-rata RTT BBR tetap stabil pada semua variasi *loss rate*. Sebaliknya, Reno, BIC, dan CUBIC mengalami *packet loss* sebelum *bandwidth* jaringan terisi penuh, sehingga rata-rata RTT Reno, BIC, dan CUBIC mengalami penurunan drastis. Oleh karena itu, walaupun selama 100 detik *destination host* Reno, BIC, dan CUBIC menerima *packet* lebih cepat dibandingkan dengan BBR, tetapi *destination host* BBR menerima *packet* lebih banyak dibandingkan dengan Reno, BIC, dan CUBIC.

4.3. Evaluasi *Multiple Flow*

A. *Inter* TCP *Protocol Fairness*

Evaluasi *multiple flow* yang pertama adalah mengirimkan dua TCP *flow* dengan setiap TCP *flow* menggunakan algoritme TCP *congestion control* yang berbeda. Tabel 4 menunjukkan rata-rata *throughput* dan perhitungan *Jain's fairness index* dari kedua TCP *flow*. Pada semua evaluasi *inter* TCP *protocol fairness*, total rata-rata *throughput* kedua TCP *flow* mencapai 19 Mbps. Fenomena rata-rata *throughput* pada evaluasi *inter* TCP *protocol fairness* menunjukkan bahwa kedua TCP *flow* tetap

dapat memaksimalkan 95% *bandwidth* yang tersedia pada *multiple paths* walaupun terjadi *packet reordering*.



Gambar 6. Perbandingan RTT terhadap variasi *loss rate*

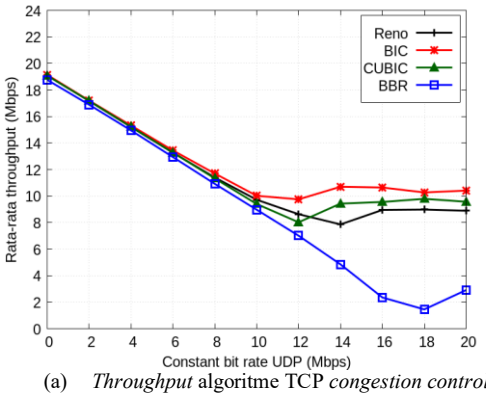
Tabel 4. Perbandingan *inter* TCP *protocol fairness*.

***Source host* 1**		***Source host* 2**		***Jain's fairness index***
TCP	***Through put (Mbps)***	**TCP**	***Through put (Mbps)***	
CUBIC	10.61	Reno	8.27	0.985
BIC	10.89	CUBIC	8.05	0.978
BIC	12.22	Reno	6.93	0.929
BBR	7.73	CUBIC	11.04	0.969
BBR	7.11	Reno	11.42	0.949
BBR	7.27	BIC	11.29	0.955

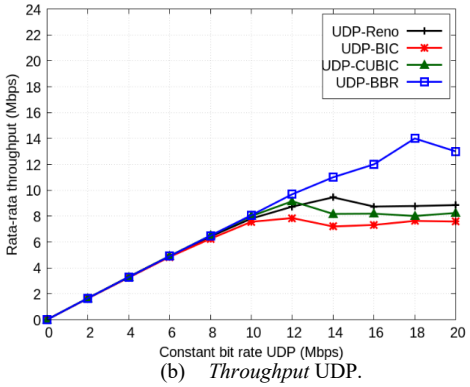
Kemudian, jika melihat *Jain's fairness index* pada Tabel 4, *Jain's fairness index* pada semua pengujian *inter* TCP *protocol fairness* mencapai di atas 0.9. Namun, *fairness* CUBIC lebih tinggi dibandingkan dengan Reno, BIC, dan BBR. *Jain's fairness index* CUBIC dengan Reno, CUBIC dengan BIC, dan CUBIC dengan BBR paling mendekati nilai 1 dibandingkan dengan yang lainnya. Oleh karena itu, jika dua algoritme TCP *congestion control* mengirimkan *packet* secara bersamaan ke dalam *multiple paths*, CUBIC dapat berbagi *throughput* lebih adil dibandingkan dengan Reno, BIC, dan BBR.

Fairness CUBIC lebih tinggi dibandingkan dengan Reno, BIC, dan BBR karena CUBIC menggunakan mode TCP. Pada mode TCP, CUBIC mengirimkan *packet* dengan kecepatan yang sama dengan Reno. Oleh karena itu, *fairness* tertinggi terjadi pada saat CUBIC dengan Reno. Walaupun CUBIC mengirimkan *packet* sedikit lebih agresif dibandingkan dengan Reno, tetapi kondisi tersebut mengakibatkan *fairness* antara CUBIC dengan BIC lebih tinggi dibandingkan *fairness* antara Reno dengan BIC. Sementara itu, jika melihat rata-rata *throughput* BIC pada Tabel 4, BIC mengirimkan *packet* lebih agresif dibandingkan dengan Reno, CUBIC, dan BBR. BIC lebih agresif karena BIC mengirimkan *packet* dengan menggunakan mode *scalability*.

Sebaliknya, *fairness* BBR lebih ditentukan oleh ukuran *buffer* pada switch. Apabila ukuran *buffer* lebih besar dari 1.5 BDP, rata-rata throughput BBR lebih rendah dibandingkan dengan algoritme TCP *congestion control* berbasis *packet loss* (Cardwell, dkk.,2016b). Ukuran *buffer* yang digunakan pada



(a) *Throughput* algoritme TCP *congestion control*



(b) *Throughput* UDP.

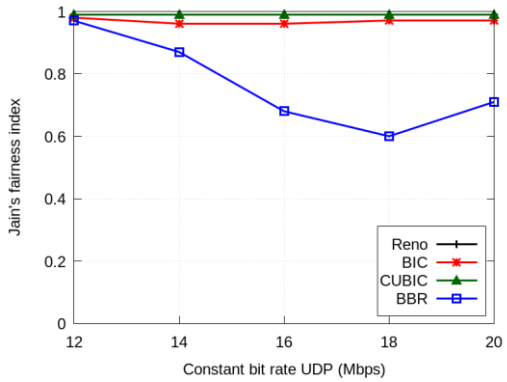
Gambar 7. Perbandingan *throughput* TCP dan UDP

penelitian ini adalah 8 BDP. Oleh karena itu, rata-rata *throughput* BBR lebih rendah dibandingkan dengan Reno, BIC, dan CUBIC untuk semua pengujian *inter* TCP *protocol fairness* BBR pada Tabel 4.

B. *Fairness* Antara TCP dengan UDP

Evaluasi *multiple flow* yang kedua adalah mengirimkan dua *flow* dengan setiap *flow* menggunakan protokol *transport* yang berbeda. *Flow* pertama adalah TCP *flow* dan *flow* kedua adalah UDP *flow*. Gambar 7 menunjukkan perbandingan rata-rata *throughput* Reno, BIC, CUBIC, BBR, dan UDP pada variasi CBR 2 Mbps – 20 Mbps. Berdasarkan rata-rata *throughput* TCP pada Gambar 7 (a), pada CBR 2 Mbps – 10 Mbps, rata-rata *throughput* Reno, BIC, dan CUBIC mengalami penurunan sebesar variasi CBR. Namun, pada CBR 12 Mbps – 20 Mbps, rata-rata *throughput* Reno, BIC, dan CUBIC stabil pada sekitar 8 Mbps – 10 Mbps. Sebaliknya, rata-rata *throughput* BBR pada CBR 2 Mbps – 20 Mbps terus mengalami penurunan hampir sebesar variasi CBR.

Fenomena rata-rata *throughput* TCP juga sama dengan rata-rata *throughput* UDP pada Gambar 7 (b). Pada CBR 2 Mbps – 10 Mbps, rata-rata *throughput* UDP pada Reno, BIC, dan CUBIC terus mengalami peningkatan. Namun, pada CBR 12



Gambar 8. Perbandingan *fairness* antara TCP dengan UDP.

Mbps – 20 Mbps, rata-rata *throughput* Reno, BIC, dan CUBIC stabil pada sekitar 7 Mbps – 9 Mbps. Sebaliknya, rata-rata *throughput* UDP pada BBR terus mengalami peningkatan pada semua variasi CBR. Fenomena rata-rata *throughput* TCP dan UDP menunjukkan bahwa pada CBR 2 Mbps – 10 Mbps, Reno, BIC, CUBIC, dan BBR sama-sama dapat berbagi *throughput* secara adil dengan UDP. Namun, pada CBR 12 Mbps – 20 Mbps, UDP lebih agresif dibandingkan dengan BBR. Sedangkan Reno, BIC, dan CUBIC pada CBR 12 Mbps – 20 Mbps tetap dapat berbagi *throughput* secara adil dengan UDP, walaupun rata-rata *throughput* Reno, BIC, dan CUBIC berbeda-beda. Oleh karena itu, untuk mengetahui algoritme TCP *congestion control* yang mendapatkan *fairness* paling tinggi, maka *Jain's fairness index* dihitung pada CBR 12 Mbps – 20 Mbps.

Gambar 8 menunjukkan perbandingan *Jain's fairness index* Reno, BIC, CUBIC, dan BBR pada CBR 12 Mbps – 20 Mbps. Pada CBR 12 Mbps – 20 Mbps, Reno, BIC dan CUBIC mendapatkan *fairness* yang tinggi, yaitu di atas 0.9. Namun, *Jain's fairness index* Reno dan CUBIC paling mendekati 1, yaitu 0.99. Sedangkan pada BBR, *Jain's fairness* BBR pada CBR 14 Mbps – 20 Mbps di bawah 0.9. Oleh karena itu, Reno dan CUBIC lebih adil dalam berbagi *throughput* dengan UDP dibandingkan dengan BIC dan BBR.

Reno dan CUBIC mendapatkan *fairness* yang sama disebabkan CUBIC menggunakan mode TCP. Sedangkan BIC beroperasi dengan menggunakan mode *scalability.* Oleh karena itu, BIC mengirimkan *packet* lebih agresif dibandingkan dengan Reno dan CUBIC. Sementara itu, BBR mengirimkan *packet* berdasarkan estimasi dari *bandwidth* yang tidak digunakan oleh UDP. Oleh karena itu, rata-rata *throughput* BBR terus mengalami penurunan hampir sebesar CBR.

5. KESIMPULAN

Multi-path routing dapat mengakibatkan TCP *Multi-path routing* dapat mengakibatkan TCP *packet reordering* walaupun *multiple paths* menggunakan *cost* yang sama dan *packet* didistribusikan secara merata. Namun, *packet reordering* tidak berdampak pada penurunan rata-rata *throughput* yang signifikan. Pada *single flow,* BBR adalah algoritme TCP *congestion control* yang terbaik pada *multi-path routing.* Namun, pada *multiple flow,* CUBIC adalah *algoritme* TCP *congestion control* yang terbaik pada *multi-path routing.*

Pada *single flow,* BBR lebih baik dalam memaksimalkan *bandwidth* yang tersedia pada *multiple paths* dibandingkan dengan Reno, BIC, dan CUBIC. Pada evaluasi *link delay,* Reno, BIC, CUBIC, dan BBR dapat memaksimalkan lebih dari 90% dari *bandwidth* yang tersedia pada *multiple paths.* Namun, rata-rata RTT BBR lebih rendah hingga 58 ms dibandingkan dengan Reno, BIC, dan CUBIC. Pada evaluasi *loss rate,* BBR tetap dapat memaksimalkan 95% dari *bandwidth* yang tersedia pada *multiple paths,* walaupun pada *loss rate* 10%. Sedangkan Reno, BIC, dan CUBIC hanya dapat memaksimalkan kurang dari 55% *bandwidth* yang tersedia pada *multiple paths* untuk semua variasi *loss rate.*

Pada *mutliple flow,* fairness CUBIC lebih tinggi dibandingkan dengan Reno, BIC, dan BBR. Pada evaluasi *inter TCP protocol fairness, Jain's fairness index* CUBIC paling mendekati nilai 1 dibandingkan dengan Reno, BIC, dan BBR. Sedangkan pada evaluasi *fairness* antara TCP dengan UDP, *Jain's fairness index* Reno dan CUBIC paling mendekati nilai 1 dibandingkan dengan BIC dan BBR.

DAFTAR PUSTAKA

ARROKKIAM, J. A., WU, X., BROWN, K. N. dan SREENAN, C. J., 2014. Experimental Evaluation of TCP Performance over 10Gb/s Passive Optical Networks (XG-PON). In: IEEE Global Communications Conference, pp. 2223–2228.

BENNETT, J. C. R., PARTRIDGE, C. dan SHECTMAN, N., 1999. Packet Reordering is Not Pathological Network Behavior. IEEE/ACM Transactions on Networking, 7(6), pp. 789–798.

CARDWELL, N., CHENG, Y., GUNN, C. S., YEGANEH, S. H. dan JACOBSON, V., 2016a. BBR Congestion-Based Congestion Control. ACM Queue, 14(5), pp. 20–53.

CARDWELL, N., CHENG, Y., GUNN, C. S., YEGANEH, S. H. dan JACOBSON, V., 2016b. BBR Congestion Control, IETF 97. [online] Tersedia di: <https://www.ietf.org/proceedings/97/slides/slides-97-iccrg-bbr-congestion-control-02.pdf> [Diakses 23 Juli 2019].

CARDWELL, N., CHENG, Y., YEGANEH, S. H. dan JACOBSON, V., 2017. BBR Congestion Control. Internet Congestion Control Research Group - Internet Draft.

CARPA, R., DIAS DE ASSUNCAO, M., GLUCK, O., LEFEVRE, L. dan MIGNOT, J. -C., 2017. Evaluating the Impact of SDN-Induced Frequent Route Changes on TCP Flows. In: 13th International Conference on Network and Service Management.

CHAITANYA, N. K. dan VARADARAJAN, S., 2016. Load distribution using multipath-routing in wired packet networks: A comparative study. Elsevier Perspectives in Science, 8, pp. 234–236.

CHENG, Y. dan CARDWELL, N., 2016. Making Linux TCP Fast, In: Netdev 1.2.

DIXIT, A., PRAKASH, P., HU, Y. C. dan KOMPELLA, R. R., 2013. On the Impact of Packet Spraying in Data Center Networks. In: IEEE INFOCOM, pp. 2130–2138..

HA, S. dan RHEE, I., 2008. Hybrid Slow Start for High-Bandwidth and Long-Distance Networks. In: PFLDNet.

HA, S., RHEE, I. dan XU, L., 2008. CUBIC : A New TCP-Friendly High-Speed TCP Variant. ACM SIGOPS Operating Systems Review - Research and developments in the Linux kernel, 42(5), pp. 64–74.

HOCK, M., BLESS, R. and ZITTERBART, M., 2017. Experimental Evaluation of BBR Congestion Control. In: IEEE 25th International Conference on Network Protocols (ICNP)

HUBERT, B., MAXWELL, G., MOOK, R., OOSTERHOUT, M., SCHROEDER, P. B. dan SPAANS, J., 2002. Linux Advanced Routing & Traffic Control HOWTO. [online] Tersedia di: <https://www.tldp.org/HOWTO/Adv-Routing-HOWTO/lartc.loadshare.html> [Diakses 23 Juli 2019].

iPerf - The ultimate speed test tool for TCP, UDP and SCTP, n.d. [online] Tersedia di: <https://iperf.fr/> [Diakses 23 Juli 2019.

JACOBSON, V., 1990. modified TCP congestion avoidance algorithm. End2end-interest mailing list.

JAIN, R. K., CHIU, D.-M. dan HAWE, W. R., 1984. A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Computer System. Technical Report TR-301, Digital Equipment Corporation.

KANAGEVLU, R. dan AUNG, K. M. M., 2015. SDN controlled Local re-routing to reduce congestion in cloud Data Centers. In: International Conference on Cloud Computing Research and Innovation, pp. 80–88.

KARLSSON, J., HURTIG, P., BRUNSTROM, A., KASSLER, A. dan STASI, G. D., 2012. Impact of Multi-path Routing on TCP Performance. In: IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks.

KUROSE, J. F. dan ROSS, K. W., 2017. Computer Networking: A Top-Down Approach. 7th ed. New Jersey: Pearson.

KUZNETSOV, A. N., n.d. TBF - Token Bucket Filter. [online] Tersedia di: <https://linux.die.net/man/8/tc-tbf> [Diakses 23 Juli 2019].

LIU, J., LI, J., SHOU, G., HU, Y., GUO, Z. dan DAI, W., 2014. SDN Based Load Balancing Mechanism for Elephant Flow in Data Center Networks. In: International Symposium on Wireless Personal Multimedia Communications, pp. 486–490.

LUKASEDER, T., BRADATSCH, L., ERB, B., HEIJDEN, R. W. V. D. dan KARGL, F., 2016. A Comparison of TCP Congestion Control Algorithms in 10G Networks. In: IEEE 41st Conference on Local Computer Networks, pp. 706–714.

NetEM, n.d. [online] Tersedia di: <https://wiki.linuxfoundation.org/networking/netem> [Diakses 23 Juli 2019].

ROS, D. dan WELZL, M., 2013. Less-than-Best-Effort Service: A Survey of End-to-End Approaches. IEEE Communications Surveys and Tutorials, 15(2), pp. 898–908.

SINGH, S. K., DAS, T. dan JUKAN, A., 2015. A Survey on Internet Multipath Routing and Provisioning. IEEE Communications Surveys and Tutorials, 17(4), pp. 2157–2175.

Tcpdump, n.d. [online] Tersedia di: <https://www.tcpdump.org> [Diakses 23 Juli 2019].

VirtualBox, n.d. [online] Tersedia di: <https://www.virtualbox.org/> [Diakses 23 Juli 2019].

XU, L., HARFOUSH, K. dan RHEE, I., 2004. Binary Increase Congestion Control (BIC) for Fast, Long-Distance Networks. In: INFOCOM, pp. 2514–2524.

YUE, Z., ZHANG, X., REN, Y., LI, J. dan ZHONG, Q., 2012. The Performance Evaluation and Comparison of TCP-based High-speed Transport Protocols. In: IEEE 3rd International Conference on Software Engineering and Service Science, pp. 509–512.