ANALISISPENGGUNAAN *MULTI-PATH ROUTING* TERHADAP KINERJA BEBERAPA ALGORITME TCP *CONGESTION CONTROL*

TESIS

Untuk memenuhi sebagian persyaratan
memperoleh gelar Magister Komputer

Disusun oleh:
Bayu Sutawijaya
NIM:156150100011009

PROGRAM STUDI MAGISTER ILMU KOMPUTER
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2019

PENGESAHAN

ANALISIS PENGGUNAAN *MULTI-PATH ROUTING* TERHADAP KINERJA BEBERAPA ALGORITME TCP *CONGESTION CONTROL*

TESIS

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Magister Komputer

Disusun Oleh:
Bayu Sutawijaya
NIM: 156150100011009

Tesis ini telah diuji dan dinyatakan lulus pada
17 Desember 2019
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Achmad Basuki, S.T., M.MG., Ph.D	Dr. Eng. Fitra Abdurrachman Bachtiar, ST., M.Eng.
NIP: 197411182003121002	NIP: 198406282019031006

Mengetahui
Ketua Jurusan Teknik Informatika

Tri Astoto Kurniawan, S.T, M.T, Ph.D

NIP: 197105182003121001

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah Tesis ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar pustaka.

Apabila ternyata di dalam naskah Tesis ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia Tesis ini digugurkan dan gelar akademik yang telah saya peroleh (Magister) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 30 Januari 2019

Bayu Sutawijaya

NIM: 156150100011009

KATA PENGANTAR

Puji syukur kehadirat Allah SWT yang telah melimpahkan rahmat, taufik, dan hidayah-Nya sehingga laporan tesis yang berjudul "Analisis Penggunaan Multi-path Routing terhadap Kinerja Beberapa Algoritme TCP Congestion Control" ini dapat terselesaikan. Penulis menyadari bahwa tesis ini tidak akan berhasil tanpa bantuan dari beberapa pihak. Oleh karena itu, penulis ingin menyampaikan rasa hormat dan terima kasih kepada:

1. Bapak Achmad Basuki, S.T, M.MG, Ph.D dan Bapak Dr.Eng. Fitra Abdurrachman Bachtiar, ST., M.Eng. selaku pembimbing tesis yang telah dengan sabar membimbing dan mengarahkan penulis sehingga dapat menyelesaikan tesis ini,
2. Keluarga penulis Bapak Prof. Dr. Ir. Abdul Latief Abadi, MS, Ibu Rr. Dewi Anggrahini (Almh), dan seluruh keluarga besar atas segala nasihat, semangat, dan doa demi terselesaikannya tesis ini,
3. Dian Faqih Puspitasari yang terus memberikan semangat kepada penulis,
4. Rhadin Angger Pribadi yang telah menjadi teman diskusi penulis selama penyusunan tesis ini.
5. Seluruh angkatan 3 Magister Ilmu Komputer dan laboratorium ICN yang telah banyak memberikan bantuan dan dukungan selama penulis menempuh studi di Magister Ilmu Komputer Universitas Brawijaya dan selama penyelesaian tesis ini.

Penulis menyadari bahwa dalam penyusunan tesis ini masih banyak kekurangan, sehingga saran dan kritik yang membangun sangat penulis harapkan. Akhir kata penulis berharap tesis ini dapat membawa manfaat bagi semua pihak yang menggunakannya.

Malang, 30 Januari 2019

Penulis

Email: bayu.sutawijaya@student.ub.ac.id

ABSTRAK

Bayu Sutawijaya, Analisis Penggunaan Multi-path routing terhadap Kinerja Beberapa Algoritme TCP Congestion Control

Pembimbing: Achmad Basuki, S.T, M.MG, Ph.D dan Dr.Eng. Fitra Abdurrachman Bachtiar, ST., M.Eng.

Penggunaan*multi-path routing*melalui*multiple paths*dengan*cost*yang sama merupakan solusi efektif untuk menambah kapasitas *bandwidth* jaringan. Namun, algoritme TCP *congestion control*menggunakan*multiple paths*sama dengan *single path.* Oleh karena itu, salah satu tantangan dalam penggunaan *multi-path routing* adalah memilih algoritme TCP *congestion control*yang terbaik, sehingga *bandwidth* jaringan pada *multiple paths* dapat digunakan secara maksimal.Penelitian ini melakukan analisis kinerja Reno, BIC, CUBIC, dan BBRpada *multi-path routing*dengan setiap *multiple paths* menggunakan *cost* yang sama. Analisis yang digunakan meliputi perbandingan antara *single path* dan *multi-path routing*, variasi *link delay*, variasi *loss rate*, *inter* TCP *protocol fairness*, dan *fairness* antara TCP dengan UDP. Berdasarkan hasil emulasi, penggunaan *multi-path routing* dapat mengakibatkan paket *reordering* pada Reno, BIC, CUBIC, dan BBR, tetapi tidak mengakibatkan penurunan rata-rata *throughput*. Pada saat hanya satu TCP *flow*yang mengirimkan paket data melalui*multiple paths*, BBR merupakan algoritme TCP *congestion control* terbaik pada *multi-path routing*. Namun, jika dua *flow*mengirimkan paket data melalui*multiple paths*, CUBIC merupakan algoritme TCP *congestion control* terbaik pada *multi-path routing*. Pada evaluasi variasi *link delay*, rata-rata RTT BBR lebih rendah hingga 58 ms dibandingkan dengan Reno, BIC, dan CUBIC. Sedangkan pada evaluasi variasi *loss rate*, rata-rata *throughput* BBR lebih dari 45% lebih tinggi dibandingkan dengan Reno, BIC, dan CUBIC. Kemudian, pada evaluasi *inter* TCP *protocol fairness* dan *fairness* antara TCP dengan UDP, *fairness* CUBIC lebih tinggi dibandingkan dengan Reno, BIC, dan BBR. *Jain's fairness index* CUBIC paling mendekati nilai 1 dibandingkan dengan Reno, BIC, dan BBR.

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

ABSTRACT

Bayu Sutawijaya, The Analysis of Multi-path Routing Usage on the Performance of TCP Congestion Control Algorithms

Supervisor: Achmad Basuki, S.T, M.MG, Ph.D and Dr.Eng. Fitra Abdurrachman Bachtiar, ST., M.Eng.

The use of multi-path routing through multiple paths with equal-cost is an effective solution to increase network bandwidth capacity. However, the TCP congestion control algorithm uses multiple paths similar to a single path. Therefore, one of the challenges in multi-path routing is to choose the best TCP congestion control algorithm, so it can maximize the network bandwidth on multiple paths. This research analyzes the performance of Reno, BIC, UCBIC, and BBR on multi-path routing with each multiple paths using the equal-cost. The analysis includes the comparison between single path routing and multi-path routing, link delay variations, loss rate variations, inter TCP protocol fairness, and fairness between TCP and UDP. Based on the results of the emulation, the use of multi-path routing triggers packet reordering on Reno, BIC, CUBIC, and BBR, but does not degrade in average throughput. When a single TCP flow that sends data packets through multiple paths, BBR is the best TCP congestion control algorithm in multi-path routing. However, when two flows send data packets through multiple paths, CUBIC is the best TCP congestion control algorithm in multi-path routing. In the link delay variations, the average throughput on BBR is more than 45% higher than Reno, BIC, and CUBIC. In the evaluation of inter TCP protocol fairness and fairness between TCP and UDP, fairness on CUBIC is higher than Reno, BIC, and BBR. Jain's fairness index on CUBIC is closest to 1 than Reno, BIC, and BBR.

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

DAFTAR ISI

PENGESAHAN i
PERNYATAAN ORISINALITAS ii
KATA PENGANTAR iii
ABSTRAK iv
ABSTRACT v
DAFTAR ISI vi
DAFTAR GAMBAR viii
DAFTAR TABEL x
BAB 1 PENDAHULUAN 1
1.1 Latar Belakang 1
1.2 Rumusan Masalah 3
1.3 Tujuan 3
1.4 Manfaat 3
1.5 Batasan Masalah 3
1.6 Sistematika Pembahasan 4
BAB 2 LANDASAN KEPUSTAKAAN 5
2.1 Penelitian Terdahulu 5
2.2 Algoritme TCP *Congestion Control* 7
2.2.1 Reno 9
2.2.2 BIC 10
2.2.3 CUBIC 12
2.2.4 BBR 14
2.3 Multi-path Routing 17
2.4 Parameter Pengukuran Kinerja 20
BAB 3 METODOLOGI 26
3.1 Studi Literatur 26
3.2 Kriteria Evaluasi 27
3.3 Perancangan *Testbed* 29
3.4 Skenario Pengujian 33
3.5 Analisis 35

3.6 Penarikan Kesimpulan Penelitian 35
BAB 4 KONFIGURASI EKSPERIMENTAL 36
4.1 Konfigurasi *Testbed* 36
4.1.1 Konfigurasi Switch 36
4.1.2 Konfigurasi Router 38
4.1.3 Konfigurasi *Host* 41
4.2 Konfigurasi Perangkat Lunak untuk Evaluasi 41
BAB 5 HASIL DAN PEMBAHASAN 43
5.1 Perbandingan antara *Single Path Routing* dengan *Multi-path Routing* 43
5.2 Variasi *Link Delay* 50
5.3 Variasi *Loss Rate* 53
5.4 Inter TCP *Protocol Fairness* 58
5.5 *Fairness* Antara TCP dengan UDP 64
BAB 6 PENUTUP 70
6.1 Kesimpulan 70
6.2 Saran 70
DAFTAR PUSTAKA 71

DAFTAR GAMBAR

Gambar 2.1 Perubahan *window* pada Reno 9
Gambar 2.2 Perubahan *window* pada BIC 11
Gambar 2.3 Perubahan *window* pada CUBIC..... 12
Gambar 2.4*State machine* pada BBR 16
Gambar 2.5 Paket *reordering* pada *multi-path routing* 18
Gambar 2.6 SACK untuk mendeteksi paket *loss* 21
Gambar 2.7 SACK untuk mendeteksi paket *reordering* 23
Gambar 2.8*Throughput* kedua TCP *flow*..... 25
Gambar 3.1 Diagram alir metodologi penelitian 26
Gambar 3.2 Topologi *multi-path routing* 30
Gambar 4.1 Perintah TBF pada Linux Ubuntu..... 37
Gambar 4.2 Perintah NetEM pada Linux Ubuntu 37
Gambar 4.3 Perintah penggabungan TBF dengan NetEm 38
Gambar 4.4 Konfigurasi IP *forwarding* pada router..... 38
Gambar 4.5 Konfigurasi *multi-path routing* pada router 1..... 39
Gambar 4.6 Konfigurasi *multi-path routing* pada router 2..... 39
Gambar 4.7 Konfigurasi *single path routing* pada router 1 40
Gambar 4.8 Konfigurasi *single path routing* pada router 2 40
Gambar 4.9 Hasil uji ping jalur pertama 40
Gambar 4.10 Hasil uji ping jalur kedua 41
Gambar 5.1 Aliran paket data BBR pada *multi-path routing*..... 44
Gambar 5.2 Perilaku Reno, BIC, CUBIC, dan BBR pada *multi-path routing* 48
Gambar 5.3 Perbandingan rata-rata *throughput* terhadap variasi *link delay* 50
Gambar 5.4 Perilaku Reno, BIC, CUBIC, dan BBR terhadap *link delay* 100 ms 51
Gambar 5.5 Perbandingan rata-rata RTT terhadap variasi *link delay*..... 53
Gambar 5.6 Perbandingan rata-rata *throughput* terhadap variasi *loss rate* 54
Gambar 5.7 Perilaku Reno, BIC, CUBIC, dan BBR terhadap *loss rate* 1%..... 56
Gambar 5.8 Perilaku Reno, BIC, CUBIC, dan BBR terhadap *loss rate* 10%..... 56
Gambar 5.9 Perbandingan rata-rata RTT terhadap variasi *loss rate* 57
Gambar 5.10 Perilaku *fairness* antaraCUBIC dengan Reno 60

Gambar 5.11 Perilaku *fairness* antaraBIC dengan Reno 61
Gambar 5.12 Perilaku *fairness* antaraCUBIC dengan BIC 61
Gambar 5.13 Perilaku *fairness* antara BBR dengan Reno, BIC, dan CUBIC..... 62
Gambar 5.14 Perbandingan rata-rata *throughput* antara TCP dengan UDP 65
Gambar 5.15 Perbandingan *fairness* antara TCP dengan UDP 65
Gambar 5.16 Perilaku Reno, BIC, dan CUBIC terhadap trafik UDP pada CBR 2 Mbps..... 66
Gambar 5.17 Perilaku Reno, BIC, dan CUBIC terhadap trafik UDP pada CBR 20 Mbps..... 68
Gambar 5.18 Perilaku BBR terhadap trafik UDP 69

DAFTAR TABEL

Tabel 2.1 Perbandingan antara penelitian terdahulu dengan penelitian yang dilakukan 5

Tabel 3.1 Konfigurasi VM 30

Tabel 3.2 Konfigurasi IP pada setiap *interface* VM 31

Tabel 3.3 Konfigurasi *source host* pada evaluasi *inter* TCP *protocol fairness* 34

Tabel 5.1 Perbandingan antara *single path routing* dengan *multi-path routing*. 43

Tabel 5.2 Perbandingan *inter* TCP *protocol fairness* 59

BAB 1 PENDAHULUAN

1.1 Latar Belakang

*Routing*merupakanteknik pemilihan jalur terbaik untuk pengiriman paket dataantara *source host* dengan *destination host*. Teknik*routing* yang umum digunakan adalah *single path routing*. Pada *single path routing*, pengiriman paket datadilakukan melalui satu jalur. Namun, semakin berkembangnya penggunaan jaringan Internet, kapasitas *bandwidth* jaringan pada *single path routing* tidak mencukupi untuk aplikasi yang membutuhkan *throughput tinggi* dan*Round-Trip Time*(RTT) yang rendah (Singh, S., Das, T.dan Jukan, A., 2015). Oleh karena itu, *Internet Service Provider* (ISP) menyediakan dua atau lebih jalur antara *source host* dengan *destination host* yang digunakanuntuk menambah kapasitas *bandwidth* jaringan melalui teknik *multi-path routing*(Bennett, J. C. R., Partridge, C. dan Shectman, N., 1999; He, J.dan Rexford, J., 2008).

Multi-path routing merupakan teknik *routing* yang dilakukan dengan caramendistribusikantrafik jaringan melalui dua jalur atau lebih (*multiple paths*).Distribusitrafikjaringan *pada multi-path routing* dapat berbasis paket data, yaitu semua paket datadari *source host* menuju *destination host*dikirimkan secara merata melalui *multiple paths*(Leung, K. -C., Li, V. O. dan Yang, D., 2007). Namun, jika *source host* mengirimkanpaket datalebih cepat dibandingkan dengan kapasitas *bandwidth* jaringan, maka dapat mengakibatkan kemacetan pada jaringan*multiple paths* atau disebut dengan *congestion*.

Algoritme *Transmission Control Protocol* (TCP) *congestion control* merupakan mekanisme yang berjalan pada *source host*untuk mencegah *congestion* pada jaringan. Algoritme TCP *congestion control* mengatur kecepatan pengiriman paket data untuk memaksimalkan penggunaan *bandwidth* jaringan sekaligus mencegah *congestion* terjadi secara terus-menerus (Hock, M., Bless, R. dan Zitterbart, M., 2017).Algoritme TCP *congestion control* yang paling banyak digunakan pada sistem operasi saat ini adalah algoritme TCP *congestion control* berbasis paket *loss*, seperti Reno (Jacobson, V., 1990), *Binary Increase Congestion control* (BIC) (Xu, L., Harfoush, K. dan Rhee, I., 2004), dan CUBIC (Ha, S., Rhee., L. dan Xu, L., 2008). Algoritme TCP *congestion control* berbasis paket *loss*beroperasi dengan menaikkan kecepatan pengiriman paket data sampai buffer router terisi penuh, sehingga terjadi paket loss. Namun,pengisianpaket data ke dalam *buffer*routerdapat mengakibatkan *queuing delay* yang tinggi (Hock, M., Bless, R. dan Zitterbart, M., 2017).

Bottleneck Bandwidth and Round-trip propagation time (BBR) (Cardwell, N., et al., 2016a) merupakan algoritme TCP *congestion control*berbasis *congestion*untuk mengatasi permasalahan *queuing delay.* BBR mendeteksi *congestion*pada saat paket data mulai mengisi *buffer* router.BBR menjaga pengirimanpaket datasebesarestimasi kapasitas maksimum *bandwidth* jaringanuntuk mencegah paket datamengisi ke dalam *buffer* routerterlalu

banyak, sehingga BBR dapat mencegah *congestion* sekaligus meminimalisir *queuing delay*.

Namun, algoritme TCP *congestion control*akanmenggunakan *multiple paths* sama dengan di *single path*.Penyebabnya adalah algoritme TCP *congestion control*merupakan proses komunikasi antar *host*. Berdasarkan sudut pandang algoritme TCP *congestion control*, *host* akan terhubung dengan *host* yang lainnyasecara langsung melalui *single path*(Kurose, J.F.dan Ross, K. W., 2017). Akibatnya adalah algoritme TCP *congestion control*akan tetap melakukan proses pengiriman paket data berdasarkan*single path,* walaupun router mengirimkan paket data melalui *multiple paths.*Oleh karena itu, perlu dilakukan analisis untuk mengetahui perilaku dan kinerja algoritme TCP *congestion control* pada *multi-path routing*.

Akan tetapi, hanya sedikit yang dapat diketahui tentang perilaku dan kinerja algoritme TCP *congestion control* pada *multi-path routing*. Berdasarkan penelitian yang dilakukan oleh Karlsson, J., et al (2012)dan Carpa, R., et al(2017), *multi-path routing* dapat mengakibatkan paket *reordering* jika *link delay* pada *multiple paths* berbeda. Paket *reordering* merupakan fenomena paket datayang diterima tidak sesuai dengan urutan yang telah dikirimkan. Apabila algoritme TCP *congestion control* mengalami paket *reordering* yang tinggi, maka algoritme TCP *congestion control*akan menginterpretasikan paket *reordering* sebagai paket *loss*, sehinggadapat menurunkan rata-rata *throughput*.

Oleh karena itu, penelitian ini melakukan analisis untuk mengetahui lebih lanjut dampak penggunaan *multi-path routing* terhadap kinerja algoritme TCP *congestion control* Reno, BIC, CUBIC, dan BBR.Perbedaan *multiple paths* yang digunakan pada penelitian ini dengan penelitian yang telah dilakukan sebelumnya oleh Karlsson, J., et al (2012)dan Carpa, R., et al(2017) adalah penelitian ini menggunakan nilai bobot *link* (*cost*) yang sama pada *multiple paths*. *Cost* yang digunakan meliputi *link delay*, *loss rate*, dan *bottleneck bandwidth*. Penggunaan *cost* yang sama pada *multiple paths* merupakan kondisi terbaik untuk melakukan *multi-path routing*(Dixit, A., et al., 2013).

Analisisyang dilakukan pada penelitian ini merupakan analisis empiris berdasarkan perilaku dankinerja algoritme TCP *congestion control* Reno, BIC, CUBIC, dan BBR pada *multi-path routing*.Analisis yangpertama adalah melakukan perbandingan jumlah*Selective Acknowledgment*(SACK)*block* dan rata-rata *throughput*antara *single path routing* dengan*multi-path routing* pada Reno, BIC, CUBIC, dan BBR.Tujuannya adalahuntuk mengetahui apakah paket *reordering*terjadi pada *multi-path routing* dan bagaimana dampaknya terhadap rata-rata *throughput* Reno, BIC, CUBIC, dan BBR. Hasil kajian pada analisis perbandingan *single path routing* dengan *multi-path routing* menjadi dasar untuk melakukan analisis selanjutnya, yaitu perbandingankinerja algoritme TCP *congestion control* Reno, BIC, CUBIC, dan BBR pada *multi-path routing*.

Tujuan analisis perbandingan kinerja algoritme TCP *congestion control* Reno, BIC, CUBIC, dan BBR adalah untuk mengetahui algoritme TCP *congestion* terbaik

pada *multi-path routing*. Parameter yang mempengaruhi kinerja algoritme TCP *congestion control* Reno, BIC, CUBIC, dan BBR pada *multi-path routing*adalahrata-rata *throughput*, rata-rata RTT, dan *fairness*. Rata-rata *throughput*dan rata-rata RTT dipengaruhi oleh kondisi *link* pada *multiple paths*, yaitu *link delay* dan *loss rate*. Sedangkan *fairness* dipengaruhi pada saat dua algoritme TCP *congestion control* yang berbeda mengirimkan paket data melalui *multiple paths* yang sama atau disebut dengan *inter* TCP *protocol fairness*. Selain itu, *fairness* juga dipengaruhi pada saat algoritme TCP *congestion control* dan *User Datagram Protocol* (UDP) mengirimkan paket datamelalui *multiple paths* yang sama. Oleh karena itu, penelitian ini menggunakananalisis variasi *link delay*, variasi *loss rate*, *inter* TCP *protocol fairness*, dan *fairness* antara TCP dengan UDP untuk melakukananalisis perbandingan kinerja algoritme TCP *congestion control* Reno, BIC, CUBIC, dan BBR pada *multi-path routing*.

1.2 Rumusan Masalah

Berdasarkan latar belakang yang telah dijelaskan, maka rumusan masalah pada penelitian ini adalah:

1. Bagaimana analisis kinerja algoritme TCP *congestion control*Reno, BIC, CUBIC, dan BBRpada*multi-path routing* jika dibandingkan dengan *single path routing.*
2. Bagaimana perbandingan kinerja algoritme TCP *congestion control* Reno, BIC, CUBIC, dan BBRpada *multi-path routing.*

1.3 Tujuan

Berdasarkan rumusan masalah yang telah dijelaskan, maka tujuan penelitian ini adalahuntuk mengetahui dampak penggunaan *multi-path routing* terhadapkinerja algoritme TCP *congestion control* Reno, BIC, CUBIC, dan BBR.

1.4 Manfaat

Penelitian ini diharapkan dapat memberikan kontribusi pada ilmu pengetahuan khususnya di bidang jaringan, sehingga penelitian ini dapatdijadikan sebagai pertimbangan dalam pemilihan algoritme TCP *congestion control*untuk digunakan pada jaringan*multi-path routing*dengan kondisi setiapjalur menggunakan *cost* yang sama.

1.5 Batasan Masalah

Untuk menjaga supaya lingkup penelitian ini fokus pada permasalahan yang akan diteliti, maka batasan masalah pada penelitian ini adalah:

1. *Link delay*antara *host*dengan router merupakan link *delay default*dari sistem yang setelah dilakukan pengukuran adalah sebesar 0.1 ms.
2. *Bandwidth* antara *host*dengan routermerupakan *bandwidth default*dari sistem yang setelah dilakukan pengukuran adalah sebesar 2.4 Gbps.

1.6 Sistematika Pembahasan

Pembahasan pada penelitian ini dibagi menjadi 6bab dengan sistematika sebagai berikut.

BAB 1 PENDAHULUAN

Bab pendahuluan berisi latar belakang, rumusan masalah, tujuan, manfaat, batasan masalah, dan sistematika pembahasan.

BAB 2 LANDASAN KEPUSTAKAAN

Bab landasan kepustakaanberisi penelitian terdahulu yang terkait dengan penelitian ini dan teori-teori pendukung yang meliputialgoritmeTCP *congestion control* Reno, BIC, CUBIC, dan BBR;*multi-path routing;*dan parameter kinerja.

BAB 3 METODOLOGI

Bab metodologi berisiperancangan untuk melakukananalisis dampak penggunaan *multi-path routing* terhadap kinerja Reno, BIC, CUBIC, dan BBR. Perancangan yang dilakukan meliputi studi literatur, kriteria evaluasi, perancangan *testbed*, skenario pengujian untuk setiap evaluasi, cara melakukan analisis, dan penarikan kesimpulan penelitian.

BAB 4 KONFIGURASIEKSPERIMENTAL

Bab konfigurasi eksperimental berisi konfigurasi yang digunakan untuk melakukananalisis dampak penggunaan *multi-path routing* terhadap kinerja Reno, BIC, CUBIC, dan BBR.Konfigurasi yang dilakukan meliputi konfigurasi switch, router, host, dan perangkat lunak yang digunakan untuk evaluasi.

BAB 5 HASIL DAN PEMBAHASAN

Bab hasil dan pembahasan berisi hasil pengukuran kinerja dan analisis untuk setiap evaluasi. Evaluasi yang dilakukan meliputi perbandingan antara *single path routing* dengan *multi-path routing,* variasi *link delay*, variasi *loss rate*, *inter* TCP *protocol fairness*, dan *fairness* antara TCP dengan UDP.

BAB 6 KESIMPULAN

Bab kesimpulan berisi kesimpulan dari hasil analisis kinerja algoritme TCP *congestion control*Reno, BIC, CUBIC, dan BBR pada*multi-path routing* untuk setiap evaluasi.

BAB 2 LANDASAN KEPUSTAKAAN

2.1 Penelitian Terdahulu

Penelitian analisis variasi*link delay*, variasi*loss rate*, *inter* TCP *protocol fairness*, dan *fairness* antara TCP dengan UDP pada algoritme TCP *congestion control* Reno, BIC, CUBIC, dan BBR telah dilakukan sebelumnya. Tabel 2.1 menunjukkan perbedaan penelitian yang telah dilakukan sebelumnya dengan penelitian ini.Arokkiam, J., et al.(2014) melakukan analisis *link delay* dan *fairness*antara TCP dengan UDP pada Reno, CUBIC, dan H-TCP di lingkungan jaringan XG-PON. Hasil penelitian Arokkiam, J., et al. (2014) adalah pada *link delay* 5 ms dan 20 ms dengan *downstream* 10 Gbps, rata-rata *throughput* CUBIC lebih tinggi dibandingkan dengan Reno dan H-TCP. Namun, pada *link delay* 8 ms, 38 ms, dan 62 ms dengan *downstream* 10 Gbps, rata-rata *throughput* H-TCP lebih tinggi dibandingkan dengan Reno dan CUBIC. Selanjutnya, pada *link delay* 5 ms – 20 ms dengan *upstream* 2.5 Gbps, rata-rata *throughput* CUBIC lebih tinggi dibandingkan dengan Reno dan H-TCP. Namun, pada *link delay* 38 ms dan 62 ms dengan *upstream* 2.5 Gbps, rata-rata *throughput* CUBIC dan H-TCP lebih tinggi dibandingkan dengan Reno. Kemudian, jika TCP dan UDP mengirimkan paket datasecara bersamaan, TCP dan UDP mendapatkan *fairness* yang tinggi. Namun, penelitian Arokkiam, J., et al.(2014) tidak menyebutkan algoritme TCP *congestion control* yang digunakan untuk melakukan evaluasi *fairness* antara TCP dengan UDP.

Tabel 2.1Perbandingan antara penelitian terdahulu dengan penelitian yang dilakukan

Penelitian	**Algoritme TCP *congestion control***	***Routing***	**Parameter analisis**			
			Link delay	***Loss rate***	***Inter* TCP *protocol fairness***	***Fairness*antara TCP dengan UDP**
Arokkiam, J., et al.(2014)	Reno, CUBIC, dan H-TCP	*Single path*	✓	–	–	✓
Lukaseder, T., et al. (2016)	Reno, Scalable TCP, HS-TCP, BIC, CUBIC, dan H-TCP	*Single path*	✓	✓	✓	–
Hock, M., Bless, R.	BBR dan CUBIC	*Single path*	✓	–	✓	–

(lanjutan)

Tabel 2.1Perbandingan antara penelitian terdahulu dengan penelitian yang dilakukan

Penelitian	Algoritme TCP *congestion control*	*Routing*	Parameter analisis			
			Link delay	***Loss rate***	***Inter* TCP *protocol fairness***	***Fairness*antara TCP dengan UDP**
dan Zitterbart, M.(2017)						
Karlsson, J., et al.(2012)	New Reno	*Multi-path*	✓	–	–	–
Carpa, R., et al(2017)	CUBIC	*Multi-path*	✓	–	–	–
Penelitian yang dilakukan	Reno, BIC, CUBIC, BBR	*Multi-path*	✓	✓	✓	✓

Kemudian, Lukaseder, T., et al. (2016) melakukan analisis *link delay*, *loss rate*, dan *inter* TCP *protocol fairness* pada Reno, Scalable TCP, HS-TCP, BIC, CUBIC, dan H-TCP. Hasil penelitian Lukaseder, T., et al. (2016) adalah pada semua variasi *link delay* dan *loss rate*, rata-rata *throughput* BIC lebih tinggi dibandingkan dengan Reno, Scalable TCP, HS-TCP, CUBIC, dan H-TCP. Sedangkan pada inter TCP *protocolfairness*antara Reno dengan Scalable TCP, HS-TCP, BIC, CUBIC, dan H-TCP, *fairness* CUBIC lebih tinggi dibandingkan dengan Scalable TCP, HS-TCP, BIC, dan H-TCP.

Selanjutnya, Hock, M., Bless, R. dan Zitterbart, M.(2017) melakukan analisis*link delay* pada *multiple* BBR *flow* dan *inter* TCP protocol *fairness*antara BBR dengan CUBIC. Berdasarkan penelitian Hock, M., Bless, R. dan Zitterbart, M.(2017), jika *multiple* BBR *flow*mengirimkan paket datadengan kondisi*buffer* router berukuran besar, maka rata-rata RTT BBR mencapai dua kali lipat dari *link delay* yang diujikan. Kemudian, pada evaluasi*inter* TCP *protocol fairness* antara BBR dengan CUBIC, jika *buffer* router berukuran besar, rata-rata *throughput* CUBIC lebih tinggi dibandingkan dengan BBR. Sebaliknya, jika *buffer* router berukuran kecil, rata-rata *throughput* BBR lebih tinggi dibandingkan dengan CUBIC. Namun, analisis algoritme TCP *congestion control*yang dilakukan oleh Arokkiam, J., et al.(2014), Lukaseder, T., et al. (2016), dan Hock, M., Bless, R. dan Zitterbart, M.(2017) hanya dilakukan pada lingkungan*single path routing*.

Sementara itu, analisis algoritme TCP *congestioncontrol* pada *multi-path routing* telah dilakukan sebelumnya oleh Karlsson, J., et al.(2012) dan Carpa, R.,et al(2017). Karlsson, J., et al.(2012) melakukan perbandingan kinerja antara New Reno tanpa mekanisme mitigasi paket *reordering*, New Reno dengan mekanisme mitigasi paket *reordering* pada Linux, dan New Reno dengan mekanisme mitigasi paket *reorderingNon-Congestion Robustness* (NCR). Karlsson, J., et al.(2012) juga melakukan perbandingan antara *multi-path routing* berbasis *flow* pada saat pengiriman paket datadan ACK, *multi-path routing* berbasis *flow* hanya pada saat pengiriman paket data, *multi-path routing* berbasis *flow* hanya pada saat pengiriman ACK, dan *multi-path routing* berbasis paket data. Semua evaluasi dilakukan pada *multi-path routing* dengan kondisi setiap jalur menggunakan*link delay* yang berbeda. Berdasarkan penelitian Karlsson, J., et al.(2012), rata-rata *throughput* pada *multi-path routing* berbasis paket datalebih rendah dibandingkan dengan *multi-path routing* berbasis *flow.* Selain itu, rata-rata *throughput* pada *multi-path routing* berbasis *flow* pada saat pengiriman paket data lebih rendah dibandingkan dengan *multi-path routing* berbasis *flow* pada saat pengiriman ACK.

Kemudian,Carpa, R., et al(2017)melakukan penelitian dampak frekuensi *reroruting* pada CUBIC di *Software Defined Network* (SDN). Berdasarkan penelitian Carpa, R., et al(2017), jika perbedaan *link delay* pada *multiple paths* kecil dan frekuensi *rerouting* juga kecil, paket *reordering* yang terjadi pada *multi-path routing* tidak berpengaruh terhadap rata-rata *throughput* CUBIC. Sebaliknya, jika *rerouting* dilakukan dengan frekuensi yang tinggi, rata-rata *throughput* CUBIC mengalami penurunan sebesar 35%.

Namun, penelitian yang dilakukan oleh Karlsson, J., et al.(2012) dan Carpa, R., et al(2017) hanya fokus pada*multipath routing* dengan masing-masing jalur menggunakan *cost* yang berbeda. Sedangkan yang dilakukan pada penelitian ini adalah *multi-path routing* dengan masing-masing jalur menggunakan *cost* yang sama. Selain itu, penelitian yang dilakukan oleh Karlsson, J., et al.(2012) dan Carpa, R., et al(2017) hanya melakukan evaluasi pada *link delay*. Untuk analisis yang lebih komprehensif, maka penelitian ini tidak hanya melakukan evaluasi variasi *link delay*, tetapi juga variasi *loss rate*, *inter* TCP *protocol fairness*, dan*fairness* antara TCP dengan UDP. Kemudian, penelitian yang dilakukan oleh Karlsson, J., et al.(2012) dan Carpa, R., et al(2017) hanya melakukan analisis berdasarkan satu algoritme TCP *congestion control*. Untuk mengetahui algoritme TCP *congestion control* yang mendapatkan kinerja paling baik pada jaringan *multi-path routing*, maka penelitian ini melakukan perbandingan 4 algoritme TCP *congestion control*, yaitu Reno, BIC, CUBIC, dan BBR.

2.2 Algoritme TCP *Congestion Control*

*Congestion*terjadi jika router menerima paket datadengan kecepatan yang lebih besar dibandingkan dengan yang dapat dikirimkan. Apabila router tidak dapat mengirimkan semua paket data secara langsung, maka sebagian paket dataakan disimpan terlebih dahulu pada *buffer* router. Apabila *buffer* router

terisi penuh, maka paket dataselanjutnya yang diterima akan dibuang oleh router, sehingga mengakibatkan paket *loss*(Lar, S. dan Liao, X., 2013). Paket *loss* dan pengisian paket data ke dalam *buffer* router dapat mengakibatkan penurunan kinerja pada jaringan, yaitu *throughput* dan RTT.

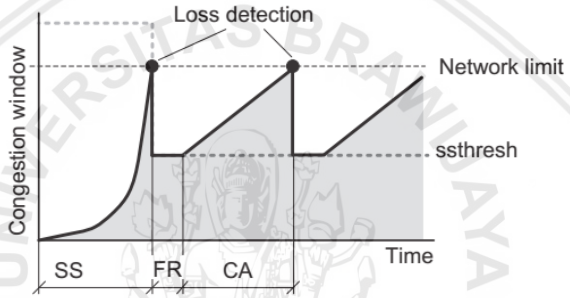
Algoritme TCP *congestion control*mencegah *congestion* dengan caramengatur kecepatan pengiriman paket data melalui mekanisme *window*. *Window* merupakan representasi dari jumlah maksimum paket data yang dapatdikirimkan oleh algoritme TCP *congestion control* sebelum menerima *acknowledgment* (ACK) dari *destination host* untuk paket data tersebut.ACK berisi informasi bahwa paket data dengan *sequence number* yang dikirimkan oleh algoritme TCP *congestion control* telah diterima oleh *destination host*. Sebagai contoh, pada saat ukuran *window* algoritme TCP *congestion control*adalah 5 paket data, maka algoritme TCP *congestion control* hanya dapat mengirimkan 5 paket data sebelum menerima ACK dari *destination host* untuk salah satu atau kelima paket data tersebut. Apabilaalgoritme TCP *congestion control* menerima ACK dari *destination host*, maka algoritme TCP *congestion control*mendeteksi jaringan belum mengalami *congestion*, sehingga algoritme TCP *congestion control*akanmenaikkan ukuran *window*.

Algoritme TCP *congestion control* menaikkan ukuran *window* untuk berusaha mencapai*Bandwidth Delay Product*(BDP) secepat mungkin. BDP merupakan representasi dari jumlah paket datayang harus dikirimkan oleh algoritme TCP *congestion control* untuk dapat mencapai kapasitas maksimum *link* jaringan. BDP dihitung berdasarkan perkalian antara *bottleneck bandwidth* dengan RTT. *Bottleneck bandwidth* merupakan *bandwidth* minimum pada jalur antara *source host* dengan *destination host*. Sedangkan RTT merupakan waktu yang dibutuhkan oleh *source host* untuk mengirimkan paket data menuju *destination host* sampai menerima ACK dari *destination host*. Apabila BDP dianalogikan sebagai pipa pada *link* jaringan, *bottleneck bandwidth* merupakan besar diameter pipa, sedangkanRTT merupakan panjang pipa. Namun, jika algoritme TCP *congestion control* mendeteksi *congestion* pada jaringan, maka algoritme TCP *congestion control* menurunkan ukuran *window* untuk mencegah *congestion* terjadi secara terus menerus (Kurose, J.F.dan Ross, K. W., 2017).

Algoritme TCP *congestion control* yang paling banyak digunakan adalah algoritme TCP *congestion control* berbasis paket *loss*, seperti Reno, BIC, dan CUBIC. Algoritme TCP *congestion control* berbasis paket *loss* mendeteksi *congestion* jika paket datayang dikirimkan oleh *source host* mengalami paket *loss*. Namun, terdapat jenis algoritme TCP *congestion control* selain berbasis paket *loss*, yaitu algoritme TCP *congestion control* berbasis *congestion*, seperti BBR. Algoritme TCP *congestion control* berbasis *congestion* mendeteksi *congestion* pada saat paket datayang dikirimkan oleh *source host* lebih besar dibandingkan dengan BDP, sehingga paket datamulai mengisi *buffer* router.

2.2.1 Reno

Renomerupakanalgoritme TCP *congestion control* berbasis paket *loss* yang menggunakan teknik *Additive Increase Multiplicative Decrease*(AIMD) untukmenentukan ukuran *window*. Apabila Renobelum mendeteksipaket *loss*, maka Reno menaikkan ukuran *window* sebesar satu paket datauntuk setiap *round* (*additive increase*). Satu *round* dihitung pada saat*source host* menerima ACK untuk semua paket datapada ukuran *window* saat ini. Sebagai contoh, pada saat ukuran *window* Reno adalah 5 paket data, Reno akan menaikkan ukuran *window* sebesar 1 paket datajika Reno sudah menerima ACK untuk kelima paket datatersebut. Kemudian, jika Reno mendeksipaket *loss*, maka Reno menurunkan ukuran *window* sebesar setengah (*multiplicative decrease*). Mekanisme Reno dibagi menjaditiga fase, yaitu fase *slow start,* fase *congestion avoidance,* dan fase *fast recovery*(Jacobson, V., 1990).Perubahan *window* Reno pada fase *slow start*, fase *congestion avoidance*, dan fase *fast recovery* ditunjukkan pada Gambar 2.1.



Gambar 2.1 Perubahan *window* pada Reno

Sumber: Afanasyev, A., et al.(2010)

Reno memulai pengiriman paket datamelalui fase *slow start*. Pada fase *slow start*, Reno menaikkan ukuran *window* sebesar satu paket datauntuk setiappenerimaan ACK, sehingga Reno menaikkan ukuran *window* 2 kali lipat setiap *round*. Oleh karena itu, ukuran *window*Reno pada fase *slow start*mengalami kenaikan secara eksponensial. Fase *slow start*Reno berhenti pada saat Reno mendeteksipaket *loss*.

Reno menggunakan *fast retransmit*untuk mendeteksi paket *loss*(Stevens, W., 1997). Apabila Reno menerima empat nomor ACK yang sama atau disebut dengan tiga duplikat ACK, maka Reno mengindikasikan bahwa paket datadengan nomor ACK tersebut mengalami paket *loss*. Oleh karena itu, Reno menurunkan ukuran *window* sebesar setengah untuk mencegah *congestion* terjadi secara terus menerus. Pada saat Reno menurunkan ukuran *window,* ukuran *window* tersebut menjadi nilai *slow start threshold*.

Kemudian, Reno melakukan fase *fast recovery*. Pada fase *fast recovery*, Reno melakukan *retransmisson*, yaitu mengirimkan kembali paket datayang

mengalami paket *loss*. Pada saat Reno melakukan *retransmission*, Reno jugamengirimkan paket datadengan *sequence number* yang baru untuk setiap duplikat ACK yang diterima. Penyebabnya adalah duplikat ACK juga mengindikasikan bahwa *destination host* sudah menerima paket datadengan *sequence number* yang lebih besar dari paket datayang mengalami paket *loss*. Oleh karena itu, pengiriman paket datadengan *sequence number* yang baru berfungsi untuk menjaga ukuran *window* Reno tetap setengah dari ukuran *window* pada saat terjadi paket *loss*. Apabila Reno sudah menerima ACK untuk paket datayang mengalami paket *loss*, maka fase *fast recovery* selesai.

Setelah melakukan fase *fast recovery*, maka fase Reno yang selanjutnya tergantung pada nilai *slow start threshold*. Apabila ukuran *window*Reno lebih rendah dibandingkan dengan nilai*slow start threshold*, maka Reno kembali melakukan fase *slow start*. Namun, jika ukuran *window*Renosama dengan ataulebih tinggi dibandingkan dengan nilai*slow start threshold*, maka Reno melakukan fase *congestion avoidance*. Pada fase *congestion avoidance*, Reno menaikkan ukuran*window*sebesar satu paket datauntuk setiap *round* sampai terjadi paket *loss*. Oleh karena itu, ukuran *window* Reno pada fase *congestion avoidance*mengalami kenaikan secara linier.

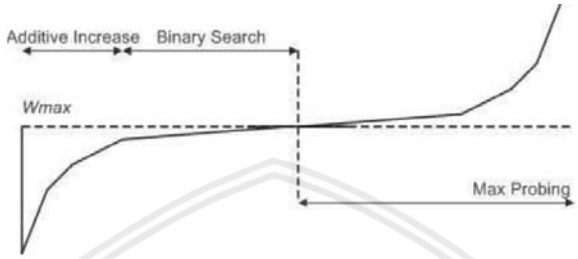
2.2.2 BIC

Binary Increase Congestion control (BIC) merupakan algoritme TCP*congestion control*berbasis paket *loss*yangdibuat dengan mempertimbangkan tiga faktor, yaitu *scalability*, RTT *fairness*, dan TCP *friendliness*. *Scalability* merupakan kemampuan BICuntukmemaksimalkan penggunaan*bandwidth*yang tersedia pada jaringan dengan BDPyangtinggi. RTT *fairness* merupakan kemampuan BIC untuk dapat berbagi *throughput* secara adil pada saat dua atau lebih BIC *flow* mengirimkan paket datasecara bersamaan,jika *link delay* masing-masing BIC *flow* berbeda. *Flow* merupakan semua paket datayang dikirimkan oleh*source host*menuju*destination host*. Sedangkan TCP *friendliness* merupakan kemampuan BIC untuk dapat berbagi *throughput* secara adil pada saat BIC dan Reno mengirimkan paket datasecara bersamaan(Xu, L., Harfoush., K. dan Rhee, I., 2004).

BIC memulai pengiriman paket datamelalui fase *slow start*. Fase *slow start* BIC menggunakan mekanisme yang sama dengan fase *slow start* Reno, yaitu menaikkan ukuran *window* sebesar 1 paket datauntuk setiap penerimaan ACK. Fase *slow* start BIC dilakukan sampai terjadi paket *loss*. Kemudian, BIC melakukan fase *congestion avoidance*. Fase *congestion avoidance* BIC terdiri dari tiga fase, yaitu *additive increase*, *binary search increase,* dan *slow start max probing.* Gambar 2.2 menunjukkan perubahan *window* BIC pada fase *additive increase*, *binary search*, dan *slow start max probing*.

BIC melakukan fase *additive increase* dan *binary search increase* tergantung pada jarak antara W_{max}dan W_{min}. W_{max}merupakan ukuran *window* pada saat BIC mengalami paket *loss* yang sebelumnya. Sedangkan W_{min} merupakan ukuran *window* pada saat BIC menurunkan ukuran *window* yang disebakan oleh paket

loss. Apabila titik tengah antara W_{max} dan W_{min} lebih besar dari S_{max} yang bernilai 32, maka BIC melakukan fase *additive increase*. Pada fase *additive increase*, BIC menaikkan ukuran *window*sebesar S_{max}. Kemudian, jika titik tengah antara W_{max} dan W_{min} lebih rendah dibandingkan dengan S_{max}, maka BIC melakukan fase *binary search increase*(Xu, L., Harfoush., K. dan Rhee, I., 2004).



Gambar 2.2 Perubahan *window* pada BIC

Sumber: Ha, S., Rhee, I. dan Xu, L.(2008)

Pada fase *binary search increase,* BIC menaikkan ukuran window sebesar setengah dari W_{max}dan W_{min}. Apabila pada saat fase *additive increase* atau *binary search increase*tidak terjadi paket *loss*, maka ukuran *window* saat ini menjadi nilai W_{min} yang baru. Kenaikan ukuran *window*sebesar setengah dari W_{max} dan W_{min} dilakukan sampai jarak antara W_{max} dan W_{min} kurang dari S_{min} yang bernilai 0.01. Apabila jarak antara W_{max} dan W_{min} kurang dari S_{min}, maka BIC mengindikasikan ukuran *window* saat ini mendekati titik paket *loss* yang sebelumnya. Oleh karena itu, BIC menaikkan ukuran *window* sebesar S_{min}. BIC menaikkan ukuran *window* sebesarS_{min} sampai ukuran *window*BIC lebih tinggidibandingkan dengan W_{max}. Pada saat ukuran *window* BIClebih tinggi dibandingkan dengan W_{max}, maka BIC mengindikasikan *bandwidth*jaringan saat ini lebih besar dibandingkan dengan *bandwidth* yang sebelumnya. Oleh karena itu, BIC melakukan fase *slow start max probing*(Xu, L., Harfoush., K. dan Rhee, I., 2004).

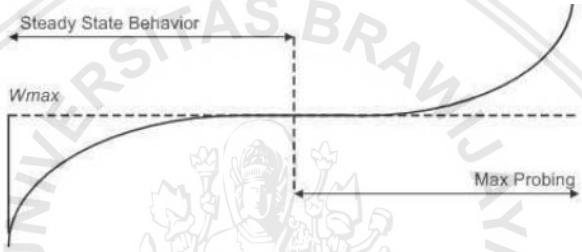
Tujuan fase*slow startmax probing* adalah mencari nilaiW_{max} yang baru. BIC melakukan fase *slow start max probing* dengan cara menaikkan ukuran *window* sebesar S_{min} sampai ukuran *window* BIC lebih tinggi dibandingkan dengan$W_{max} + \mathrm{S_{max}}$. Apabila ukuran *window*BIC lebih tinggi dibandingkan dengan$W_{max} + \mathrm{S_{max}}$, maka BIC menaikkan ukuran *window* sebesar $\mathrm{S_{max}}$ sampai terjadi paket *loss*(Xu, L., Harfoush., K. dan Rhee, I., 2004).

Kemudian, BIC menggunakan*fast convergence* untuk mempercepat *fairness* antar *flow*. Pada saat terjadipaket *loss*, jika nilai W_{max} saat ini lebih rendah dibandingkan dengan nilaiW_{max} yang sebelumnya, maka BIC mengindikasikan terdapat*flow*lain yangmengirimkan paket datake dalam*link* jaringan yang sama.

Oleh karena itu, BIC melakukan pengaturan ulang nilaiW_{max}, yaitu sebesar setengah antara W_{max} dan W_{min}, sehingga BIC menurunkan ukuran window lebih drastis. Tujuannya adalah memberikan ruang kepada*flow* lain untuk menaikkan ukuran *window*, sehingga dapat tercapai *fairness*(Xu, L., Harfoush., K. dan Rhee, I., 2004).

2.2.3 CUBIC

CUBIC merupakan pengembangan dari BIC untuk TCP *friendliness*karena BIC menaikkan ukuran *window* lebih agresif dibandingkan dengan Reno. Namun, CUBICtetap mempertahankan *scalability* dan RTT *fairness* pada BIC. Gambar 2.3 menunjukkan grafik perubahan*window* pada CUBIC. CUBIC menggunakan fungsi *cubic*untukmenaikkanukuran *window*. Oleh karena itu, grafik perubahan*window* pada CUBIC memiliki pola yang sama dengan BIC pada Gambar 2.2. Mekanisme CUBIC dibagi menjadi 2, yaitu fase *slow start* dan fase *congestion avoidance*(Ha, S., Rhee, I. dan Xu, L., 2008).



Gambar 2.3 Perubahan *window* pada CUBIC

Sumber :Ha, S., Rhee, I. dan Xu, L.(2008)

2.2.3.1 *Slow Start*

CUBIC memulai pengiriman paket datamelalui fase *slow start.* Fase *slow start* CUBIC menggunakan metode *hybrid slow start*(Ha, S. dan Rhee, I., 2008). Pada *hybrid slow start,* CUBIC menaikkan ukuran *window* dengan mekanisme yang samaseperti fase *slow start* Reno.Namun,*hybrid slow start*CUBIC tidak berhenti pada saat terjadi paket *loss* yang disebabkan oleh *buffer* router terisi penuh, melainkanberhenti jika memenuhi salah satu dari dua kondisi. Kondisi pertama adalah pada saat ukuran *window* CUBIC sama dengan estimasi BDP. Sedangkan kondisi kedua adalah pada saat *delay*untukACK yang diterima oleh CUBIC mengalami kenaikan signifikan dibandingkan dengan *delay* untuk ACK yang diterima sebelumnya. Oleh karena itu, CUBIC menggunakan dua mekanisme sebagai titik berhenti *slow start*, yaitu ACK *train* dan kenaikan *delay.*

1. ACK *train*

CUBIC menggunakan mekanisme ACK *train* untuk melakukan estimasi BDP. ACK *train* merupakan *delay*antar ACK dalam satu *round*. Apabila *delay*ACK

*train*saat ini lebih besar dibandingkan dengan*delay*ACK *train*yang sebelumnya, maka CUBIC mengindikasikan ukuran *window* sama dengan BDP. Oleh karena itu,CUBIC menghentikan fase *slow start*(Ha, S. dan Rhee, I., 2008).

2. Kenaikan *delay*

Pada saat dua *flow*atau lebih mengirimkan paket datasecara bersamaan, estimasiBDP yang didapatkan dari ACK *train* tidak akurat. Oleh karena itu, *hybridslow start* menggunakan mekanisme kenaikan *delay*sebagai salah satu kondisi titik berhenti *slow start*. Mekanisme kenaikan *delay* didapatkan dari sampel *delay*untuk setiap *round*. Apabila sampel*delay* melebihi ambang batas dari *delay*yang telah ditentukan, maka CUBIC mengindikasikan ukuran *window* sudah mencapai kapasitas maksimum *bandwidth*jaringan, sehinggapaket datamulai mengisi ke dalam*buffer* router. Oleh karena itu, CUBIC menghentikan fase *slow start*. Kenaikan*delay* pada CUBIC dihitung berdasarkan Persamaan 2.1, yaitu:

$$RTT_k \geq RTT_{k-1} + \eta \quad (2.1)$$

dimana RTT_k merupakan nilai RTT minimum pada k *round,*sedangkanη merupakan ambang batas kenaikan *delay* yang dihitung berdasarkan Persamaan 2.2(Ha, S. dan Rhee, I., 2008).

$$\eta = \min(8, \max(2, \lceil \mathrm{RTT}_{k-1}/16 \rceil)) \quad (2.2)$$

2.2.3.2 *Congestion Avoidance*

Setelah fase *slow start* selesai, CUBIC melakukan fase *congestion avoidance*. Pada fase *congestion avoidance*, CUBIC menggunakan fungsi *cubic*berdasarkanPersamaan 2.3, yaitu:

$$W_{cubic}(t) = C\,(t-K)^3 + W_{max} \quad (2.3)$$

dimana C merupakan nilai tetap parameter CUBIC, yaitu 0.4, t merupakan jarak antara waktu saat ini dengan waktu dimulainya fase *congestion avoidance*, W_{max} merupakan ukuran *window* pada saat terjadi paket *loss*, dan K dihitung berdasarkan Persamaan 2.4. Pada Persamaan 2.4, β merupakan nilai penurunan ukuran *window* pada saat terjadi paket *loss*, yaitu 0.2. Apabila paket *loss*disebabkanoleh tiga duplikat ACK, maka nilai W_{max} CUBIC ditentukan sebesar ukuran *window* pada saat terjadi paket *loss*. Namun, jika paket *loss*disebabkan oleh*timeout*, nilai W_{max}ditentukan sebesar ukuran *window* pada saat dimulainya fase *congestion avoidance*(Rhee, I., et al., 2018).

$$K = \sqrt[3]{\frac{W_{max}\beta}{C}} \quad (2.4)$$

Berdasarkan ukuran *window*pada fase *congestion avoidance*, CUBIC menjalankan tiga mode, yaitu:

1. Apabila ukuran *window* CUBIC lebih rendah dibandingkan dengan ukuran *window*yang akan diperoleh Reno dalam waktu t, maka CUBIC menjalankan mode *TCP friendliness*(Rhee, I., et al., 2018).
2. Apabila ukuran *window* CUBIC lebih rendah dibandingkan dengan W_{max} dan tidak dalam mode TCP *friendliness*, maka CUBIC menjalankan mode *concave*. Pada mode *concave*, CUBICmenaikkan ukuran *window* sebesar$\frac{(W(t+RTT)-cwnd)}{cwnd}$ untuk setiap penerimaan ACK (Rhee, I., et al., 2018).
3. Apabila ukuran *window*CUBIC lebih tinggi dibandingkan dengan W_{max} dan tidak dalam mode TCP *friendliness*, maka CUBIC menjalankan mode *convex* untuk mencari nilai W_{max} yang baru. Pada mode *convex,* CUBIC menaikkan ukuran *window* sama seperti mode *concave* (Rhee, I., et al., 2018).

2.2.3.3 *TCP Friendliness*

Pada jaringan dengan BDP yangrendah, Reno menaikkan ukuran *window* lebih agresif dibandingkan dengan CUBIC. Oleh karena itu, CUBIC menjalankan mode TCP *friendliness* untuk mencapai ukuran *window* yang sama seperti Reno. Pada mode TCP *friendliness*, CUBIC menaikkan ukuran *window*berdasarkan Persamaan 2.5. Apabila nilai W_{tcp}pada Persamaan 2.5 lebih tinggi dibandingkan dengan W_{cubic} pada Persamaan 2.3, makaCUBIC menaikkan ukuran *window*berdasarkanW_{tcp}. Sebaliknya, jikaW_{cubic}pada Persamaan 2.3 lebih tinggi dibandingkan dengan W_{tcp} pada Persamaan 2.5, maka CUBIC menaikkan ukuran *window* berdasarkanW_{cubic}(Rhee, I., et al., 2018).

$$W_{tcp} = W_{max}\,\beta + 3\,\frac{1-\beta}{1+\beta}\frac{t}{RTT} \tag{2.5}$$

2.2.4 BBR

BBR merupakan algoritme TCP *congestion control* berbasis *congestion*yang mencegah *congestion* dengan caramenjaga pengirimanpaket databerdasarkan estimasi BDP. BBRmelakukandua estimasi untuk mendapatkan estimasi BDP, yaitu:

1. *Bottleneck Bandwidth* (BtlBw) maksimum. BtlBw merupakan *bandwidth* minimum pada jalur antara *source host* dengan *destination host*. BtlBw berpengaruh terhadap kecepatan maksimum pada saat pengiriman paket datadari *source host* menuju *destination host*.
2. *Round-trip propagation delay* (RTprop) minimum. RTprop merupakan waktu yang dibutuhkan untuk mengirimkan paket datadari *source host* menuju *destination host*sampai*source host* menerima ACK dari *destination host* untuk paket datatersebut.

Estimasi BtlBw diperoleh dari*delivery rate*dalam waktu T. *Delivery rate*merupakan kecepatan maksimum pada saat pengirimanpaket datasampai penerimaanACK. Estimasi BtlBw dihitungberdasarkan Persamaan 2.6. Pada Persamaan 2.6, W_B merupakan perhitungan estimasi *delivery rate* yang dilakukan setiap 10 *round*.

$$BtlBw = \max(deliveryRate_t)\ \forall t\ \in [T - W_B, T] \quad (2.6)$$

Kemudian, BBRmendapatkan estimasi RTprop berdasarkan sampel rentang waktu minimum antara pengiriman paket datasampai penerimaan ACK dalamwaktu T. Estimasi RTpropdihitung berdasarkan Persamaan 2.7. Pada Persamaan 2.7, W_R merupakan perhitungan estimasi RTprop yang dilakukan setiap 10 detik, dan η merupakan *noise* yang disebabkan oleh ACK *aggregation, queuing delay*, dan lain-lain (Cardwell *et al.*, 2016a).

$$RTprop = RTprop + \min(\eta_t) = \min(RTT_t)\ \ \forall t\ \in [T - W_R, T] \quad (2.7)$$

2.2.4.1 Parameter Kontrol

BBR menggunakan tiga parameter kontrol untuk menentukan kecepatan pengiriman paket data, yaitu *pacing rate*, *quantum*, dan*window*(Cardwel, N., et al., 2017).

1. *Pacing rate*

Pacing rate merupakan parameter kontrol utama BBR untuk menentukan waktu pengiriman paket datayang dilakukanberdasarkan Persamaan 2.8. Pada Persamaan 2.8, *pacing_gain*merupakan faktor skalakecepatan pengiriman paket data. Penentuan nilai*pacing_gain*dilakukan berdasarkan *state machine* BBR yang akan dijelaskan pada sub bab 2.2.4.2(Cardwel, N., et al., 2017).

$$next\_send\_time = now() + ukuran\_\text{paket} / pacing\_rate * pacing\_gain \quad (2.8)$$

2. *Quantum*

*Quantum*merupakan batasjumlahpaket datayang dapat dikirimkan untuk mengurangi CPU *overhead*. Apabila*pacing rate*lebih rendah dibandingkan dengan 1.2 Mbps, maka nilai *quantum*ditentukan sebesar 1 *Maximum Segment Size* (MSS). Apabila *pacing rate*lebih tinggi dibandingkan dengan 1.2 Mbps dan lebih rendah dibandingkan dengan 24 Mbps, maka nilai *quantum*ditentukan sebesar 2 MSS. Terakhir, jika*pacing rate* lebih tinggi dibandingkan dengan 24 Mbps, maka nilai *quantum*ditentukan dari nilai minimum antara64KB dengan*pacing rate* selama 1 ms(Cardwel, N., et al., 2017).

3. *Window*

*Window*BBR berfungsi untuk membatasi jumlah maksimum paket datayang dikirimkan sebelum BBR menerima ACK untuk paket datatersebut. Ukuran *window*BBRditentukan berdasarkan Persamaan 2.9.Pada Persamaan 2.9,*window_gain merupakan* faktor skala kenaikan ukuran *window.* Nilai *window_gain*ditentukan berdasarkan *state machine* BBR yang akan dijelaskan pada sub bab 2.2.4.2.

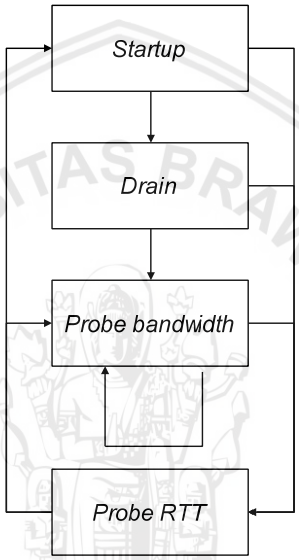
$$window = window\_gain * BtlBw\ * RTprop + quantum \quad (2.9)$$

Apabila terjadi paket *loss* yang disebabkan oleh *timeout*, maka BBR menurunkan ukuran *window* menjadi 1 paket data. Namun, jika paket *loss*

disebabkan oleh tiga duplikat ACK, maka pada saat *retransmission* untuk *round* pertama, ukuran *window* BBR diturunkan sebesar paket datayang mengalami paket *loss*. Untuk *round* selanjutnya, paket datayang dikirimkan tidak boleh melebihi dua kali lipat dari ACK yang diterima. Pada saat *retransmission* selesai dilakukan, ukuran *window*BBR dikembalikan ke ukuran tepat pada saat sebelum terjadi paket *loss*(Cardwel, N., et al., 2017).

2.2.4.2 *State Machine* BBR

State machine BBR merupakan urutan proses mekanisme BBR.Gambar 2.4 menunjukkan *state machine* BBR yang terdiri dari*startup*, *drain*, *probe bandwidth*, dan *probe* RTT (Cardwel, N., et al., 2017).



Gambar 2.4*State machine* pada BBR

Sumber : Cardwel, N., et al.(2017)

1. *Startup*

State startup merupakan mekanisme awal pengiriman paket datapada BBR. Pada *state startup*, *pacing_gain* dan *window_gain* ditentukan sebesar 2/ln(2)~=2.899, sehingga ukuran *window* pada BBR mengalami kenaikan secara eksponensial. *State startup*BBR berakhir jika *delivery rate*dalam tiga *round*kurang dari BtlBw * 1.25 (Cardwel, N., et al., 2017).

2. *Drain*

Statedrain berfungsi untuk mengosongkan *buffer* pada router karena pengiriman paket datapada *state startup* mengakibatkan paket dataBBR mengisi ke dalam*buffer*router. Pada *state drain*, *pacing_gain* ditentukan sebesar 1/2.89.

Apabila jumlah paket datayang dikirimkansama dengan estimasi BDP, maka BBR mengindikasikan *buffer* router telah kosong (Cardwel, N., et al., 2017).

3. *Probe bandwidth*

Sebagian besar operasi BBR adalah menjalankan*state probe bandwidth*. *State probe bandwidth* berfungsi untuk melihat jika terjadi perubahan pada *bottleneck bandwidth*. Pada *state probe bandwidth*, ukuran *window_gain* ditentukan sebesar 2 dan ukuran *pacing_gain* ditentukan berdasarkan 8 siklus *pacing_gain*, yaitu: 5/4, 3/4, 1, 1, 1, 1, 1, 1. Pada siklus pertama, *pacing rate* dinaikkan sebesar 5/4 untuk melihat jika *bottleneckbandwidth*mengalami kenaikan. Apabila *bottleneck bandwidth* tidak berubah, maka ukuran *pacing_gain* sebesar 3/4 pada siklus kedua berfungsi untuk mengosongkan *buffer*router. Pada siklus ketiga sampai delapan, BBR mengirimkan *packe*t sebesar estimasi *bottleneckbandwidth*.BBR secara berkala mengacak ukuran *pacing_gain*pada siklus pertamaberdasarkan ukuran *pacing_gain* dari 8 siklus, kecuali *pacing_gain* 3/4 (Cardwel, N., et al., 2017).

4. *Probe* RTT

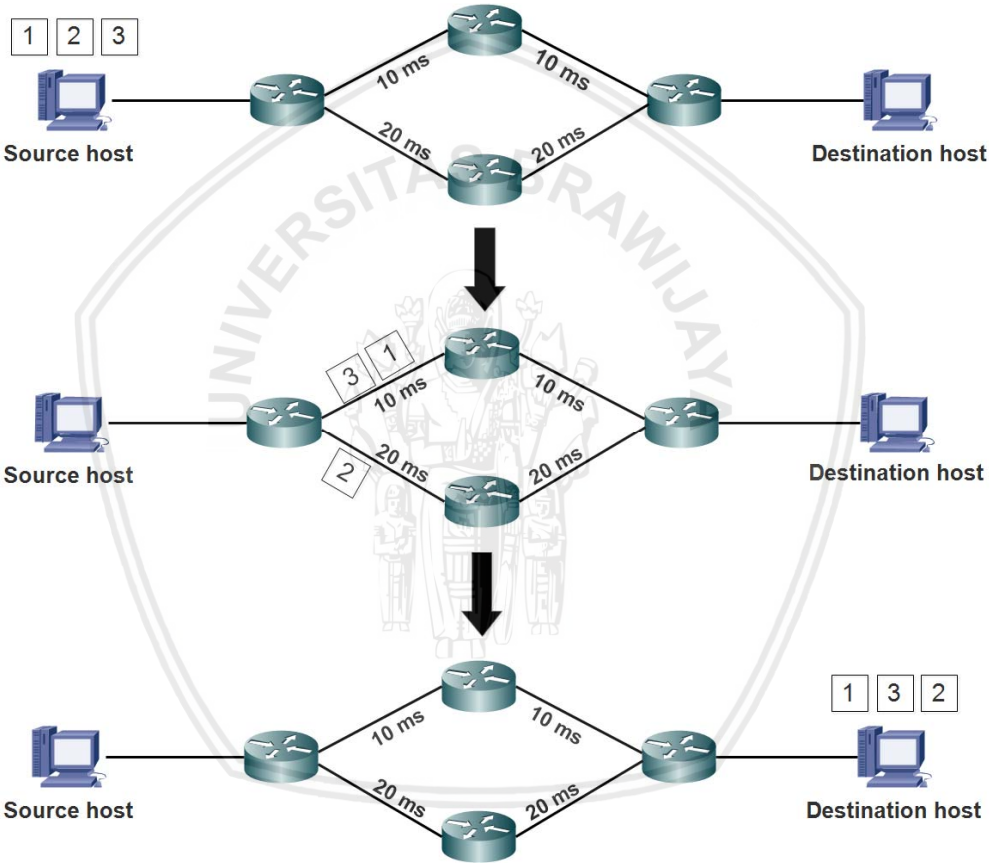
State probe RTT berfungsi untuk melakukan estimasi RTT minimum, jika estimasiRTT minimum tidak mengalami perubahan lebih dari 10 detik. *Probe* RTT dilakukan dengan menurunkan ukuran *window* sebesar 4 paket data. RTT minimum yang diperoleh dari *probe* RTT akan menjadi estimasi RTT minimum yang baru (Cardwel, N., et al., 2017).

2.3 Multi-path Routing

Multi-path routing merupakan teknik *routing* yang dilakukan dengan caramendistribusikan paket datasecara merata melalui *multiple paths*.*Round-robin* merupakan salah satu algoritme untuk mendistribusikan paket datamelalui*multiple paths*. *Round-robin*cocok digunakan pada *multiple paths* yang menggunakan *cost* sama. Selain itu, *round-robin* hanya dapat mengakibatkan sedikit *overhead* pada router (He, J. dan Rexford, J., 2008., Singh, S., Das, T.dan Jukan, A., 2015).

Namun, penggunaan algoritme TCP *congestion control* pada *multi-path routing* dapat mengakibatkan paket *reordering*. Paket *reordering* merupakan fenomena paket datayang diterima oleh*destination host* tidak sesuai dengan urutan yang dikirimkan oleh *source host*. Paket *reordering*dapat disebabkan oleh perbedaan *link delay* atau perbedaan ukuran *buffer*, sehingga terdapat perbedaan waktu *pengiriman* paket datapada *multiple paths*(Leung, K. -C., Li, V. O. dan Yang, D., 2007). Sebagai contoh, *source host* mengirimkan 3 paket datamenuju *destination host*. Pengiriman paket datadilakukan melalui 2 jalur berdasarkan *round-robin*. *Link delay* antara jalur pertama dan kedua tidak sama. Total *link delay* pada jalur pertama adalah 20 ms,sedangkan total *link delay* pada

jalur kedua adalah 40 ms. Gambar 2.5 menunjukkan aliran pengiriman paket datayang dilakukan melalui *multiple paths*. *Source host* mengirimkan 3 paket datamenuju *destination host*dengan*sequence number* 1 – 3. Apabila pengiriman paket datadilakukan berdasarkan *round-robin*, maka paket datadengan *sequence number* 1 dan 3 dikirimkan melalui jalur pertama, sedangkan paket datadengan *sequence number* 2 dikirimkan melalui jalur kedua. *Destination host* mengharapkan paket datayang diterima sesuai dengan urutan *sequence number* yang dikirimkan oleh *source host*. Namun, *link delay* jalurkedua lebih tinggi dibandingkan dengan *link delay* jalur pertama, sehingga *destination host* menerima paket datadari jalur kedua lebih lama dibandingkan dengan jalur pertama.



Gambar 2.5Paket *reordering* pada *multi-path routing*

Sumber :Visualisasi penulis

Pada saat *destination host* menerima paket data dengan *sequencenumber* 1, *destination host* mengirimkan ACK 2 kepada *source host*. ACK 2 menunjukkan bahwa *destination host* sudah menerima paket datadengan *sequence number* 1, dan paket dataselanjutnya yang diharapkan diterima oleh *destination host* adalah paket datadengan *sequence number* 2. Namun, *destination host*

menerima paket datadengan *sequence number* 3 karena proses pengiriman paket datadengan *sequence number* 2 lebih lama dibandingkan dengan paket data dengan *sequence number* 3. Oleh karena itu, *source host* mengirimkan 1 duplikat ACK, yaitu ACK 2 karena *source host*mengharapkanpaket datayang diterima adalah dengan *sequence number* 2. Kemudian, *destination host* menerima paket datadengan *sequence number* 2, sehingga *destination host mengirimkan* ACK baru, yaitu 3.

Paket *reordering* dapat terjadi pada saat pengiriman paket datadari *source host*atau pada saat pengiriman ACK dari *destination host*. Dampak paket *reordering* adalah:

1. Algoritme TCP *congestion control* menggunakan tiga duplikat ACK untuk mendeteksi paket *loss*. Namun,jika algoritme TCP *congestion control* berbasis paket *loss* menerima tiga duplikat ACK yangdisebabkan olehpaket *reordering*, maka algoritme TCP *congestion control* berbasis paket *loss* mendeteksi jaringan mengalami *congestion*. Oleh karena itu, algoritme TCP *congestion control* berbasis paket *loss* menurunkan ukuran *window*. Apabila paket *reordering* terjadi secara terus menerus, algoritme TCP *congestioncontrol*berbasis paket *loss*sulit untuk menaikkan ukuran *window*sampai kapasitas maksimum*bandwidth* jaringan. Oleh karena itu, packet *reordering*dapat mengakibatkan rata-rata *throughput*algoritme TCP *congestion control* berbasis paket *loss* mengalami penurunan(Bennett, J.C.R., Partridge, C. dan Shectman, N., 1999).
2. Apabilapaket *reordering*memicu algoritme TCP *congestion control*untuk melakukan *retransmission*, maka paket datayang sama akan dikirimkan dua kali, sehingga algoritme TCP *congestion control* menggunakan *bandwidth* jaringan tidak efisien(Bennett, J.C.R., Partridge, C. dan Shectman, N., 1999).
3. Berdasarkan penelitian Karlsson, J., et al (2012), paket *reordering* yang terjadi pada saat pengiriman ACK dari *destination host* dapat menurunkan rata-rata *throughput* yang lebih drastis dibandingkan dengan paket *reordering* yang terjadi pada saat pengiriman paket datadari *source host*.
4. Paket *rerordering* juga dapat berdampak pada kinerja *destination host*. Apabila terjadi paket *reordering*, maka *destination host* harus menyimpan paket datadengan *sequence number* yang tidak urut sampai *destination host* menerima paket datadengan *sequence number* yang urut. Proses pengurutanpaket datapada saat terjadi paket *reordering*dapat mengakibatkan penggunaan sumber daya komputer pada *destination host*yang tidak efisien (Bennett, J.C.R., Partridge, C. dan Shectman, N., 1999).

Linux memiliki mekanisme untuk mengatasi permasalahan paket *reordering* melalui *timestamp* dengan algoritme *Eiffel*(Ludwig, R. dan Katz, R. H., 2000), *DuplicateSelective Acknowledment* (DSACK)(Floyd, S., et al., 2000), dan *reordering heuristic*.*Timestamp* dan DSACK berfungsi untuk mencegah algoritme TCP *congestion control* menurunkan ukuran *window* pada saat algoritme TCP *congestion control* salah mendeteksi paket *reordering*sebagai paket *loss*.

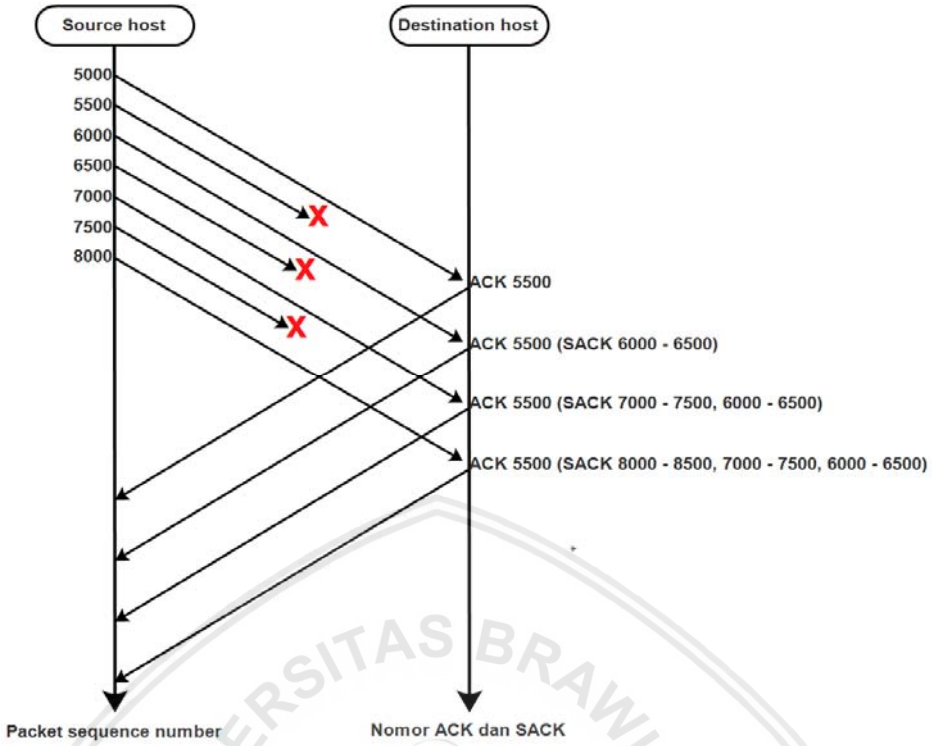
Sedangkan pada *reordering heuristic*, Linux melakukan penambahan jumlah toleransi penerimaan duplikat ACK sebelum algoritme TCP *congestion control* melakukan *retransmission*. Jumlah toleransi penerimaan duplikat ACK bertambah secara bertahap dari 3 duplikat ACK ke nilai maksimum yang ditentukan oleh Linux. Namun, semua mekanisme mitigasi paket *reordering* pada Linux hanya dapat bekerja dengan baik jika perbedaan *cost*pada masing-masing *multiple paths* kecil dan ACK diterima secara urut(Karlsson, J., et al*,* 2012; Carpa, R., et al., 2017).

2.4 Parameter Pengukuran Kinerja

Penelitian ini menggunakan 4 parameter kinerja untuk mengukur kinerja algoritme TCP *congestion control* pada *multi-path routing,* yaituSACK*block*,rata-rata *throughput*, rata-rata RTT, dan *fairness.*

1. *SACK block*

SACK(Mathis, M., et al., 1996) merupakan salah satu mekanisme ACKpada TCP. Apabila terjadi paket *loss*, *destination host* mengirimkan ACK dengan SACK *block* kepada *source host*. Setiap *block* pada SACK berisi informasi bahwa terdapat *gap* antara *sequence number*paket data yang diterima oleh *destination host*, sehingga *source host* mengetahui secara pasti paket datayang mengalami paket *loss*. Sebagai contoh, *source host* mengirimkan 7 paket datasecara bersamaan dengan *sequence number* 5000 – 8000 dengan kelipatan 500 untuk setiap *sequence number*. Namun, paket datadengan *sequence number* 5500, 6500, dan 7500 mengalami paket *loss*. Gambar 2.6 menunjukkan aliran paket dataantara *source host* dengan *destination host* jika terjadi paket *loss* dengan menggunakan SACK.



Gambar 2.6SACK untuk mendeteksi paket *loss*

Sumber :Visualisasi penulis

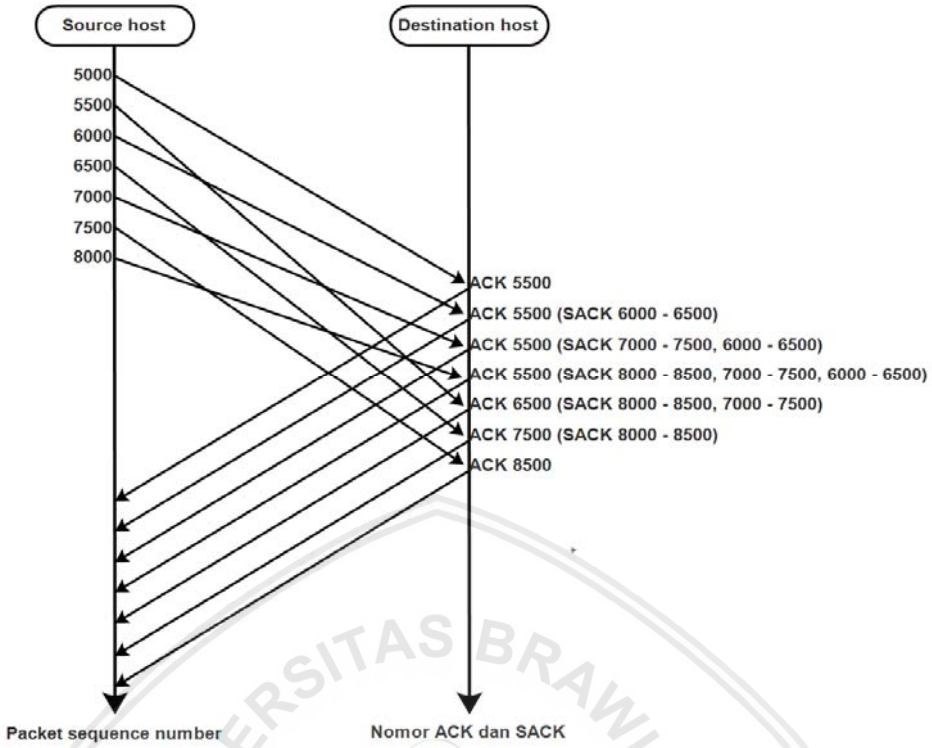
Pada saat *destination host* menerima paket datadengan *sequence number* 5000, *destination host* mengirimkan ACK 5500. ACK 5500menunjukkan bahwa *destination host* sudah menerima paket datadengan *sequence number* 5000, dan paket dataselanjutnya yang diharapkan diterima oleh *destination host* adalah paket datadengan *sequence number* 5500. Namun, paket dataselanjutnya yang diterima oleh *destination host* adalah dengan *sequence number* 6000 karena paket datadengan *sequence number* 5500 mengalami paket *loss*. Oleh karena itu, *destination host* mengirimkan ACK 5500 dengan 1 SACK *block*, yaitu 6000 – 6500. ACK dan SACK *block* yang dikirimkan oleh *destination host* menunjukkan bahwa *destination host* mengharapkan paket datayang diterima adalah dengan *sequence number* 5500, tetapi *destination host* menerima paket datadengan *sequence number* 6000. Paket dataselanjutnya yang diharapkan diterima oleh *destination host* setelah *sequence number* 6000 adalah 6500.

Kemudian, *destination host* kembali menerima paket datayang tidak sesuai dengan yang diharapkan oleh *destination host*, yaitu paket datadengan *sequence number* 7000. *Destination host* kembali mengirimkan ACK 5500, tetapi saat ini *destination host* mengirimkan 2 SACK *block*. SACK *block* pertama adalah 6000 – 6500 dan SACK *block* kedua adalah 7000 – 7500. *Destination host* mengirimkan 2 SACK *block* karena *destination host* mendeteksi 2 *gap* pada paket datayang diterima. *Gap* yang pertama adalah *destination host* mengharapkan paket

datayang diterima adalah dengan *sequence number* 5500, tetapi *destination host* menerima paket datadengan *sequence number* 6000. *Gap* yang kedua adalah setelah *destination host* menerima paket datadengan *sequence number* 6000, paket dataselanjutnya yang diharapkan diterima oleh *destination host* adalah paket datadengan *sequence number* 6500. Namun, *destination host* menerima paket datadengan *sequence number* 7000.

Selanjutnya, *destination host* menerima paket datadengan *sequence number* 8000. Oleh karena itu, *destination host* kembali mengirimkan ACK 5500, tetapi dengan 3 SACK *block*karena *destination host* mendeteksi terdapat 3 gappadapaket datayang diterima. Berdasarkan informasi ACK dan SACK *block* tersebut, *source host* mengetahui bahwa paket datadengan *sequence number*5500, 6500, dan 7500 mengalami paket *loss*.*Destination host*akan menghapus SACK *block* jika *destination host* sudah tidak mendeteksi *gap* pada paket datayang diterima, yaitu pada saat *source host* melakukan *retransmission* untuk paket datadengan *sequence number* 5500, 6500, dan 7500.

Pada *multi-path routing*, informasi SACK *block* juga dapat digunakan untuk mengetahui jika terjadi paket *reordering*(Bennett, J.C.R., Partridge, C.dan Shectman, N., 1999). *Gap* antara *sequence number*pada paket data yang diterima oleh *destination host* tidak hanya disebabkan oleh paket *loss*, tetapi juga dapat disebabkan oleh paket *reordering*. Sebagai contoh adalah kasus yang sama dengan Gambar 2.6, tetapi paket datadengan *sequence number* 5500, 6500, dan 7500 tidak mengalami paket *loss*, melainkan datang terlambat pada *destination host*. Gambar 2.7 menunjukkan paket *reordering* yang terjadi pada *multi-path routing*berdasarkan informasi dari SACK *block*. Sama seperti Gambar 2.6, pada saat *destination host* mengharapkan paket datayang diterima adalah dengan *sequence number* 5500, *destination host* menerima paket datadengan *sequence number* 6000. Oleh karena itu, *destination host* mengirimkan ACK dengan 1 SACK *block*, yaitu 6000 – 6500. Kemudian, *destination host* menerima paket datadengan *sequence number* 7000, sehingga *destination host* kembali mengirimkan ACK 5500, tetapi dengan 2 SACK *block*, yaitu 6000 – 6500 dan 7000 – 7500. Selanjutnya, *destination host* menerima paket datadengan *sequence number* 8000, sehingga *destination host* mengirimkan ACK 5500, tetapi dengan 3 SACK *block*, yaitu 6000 – 6500, 7000 – 7500, dan 8000 – 8500.



Gambar 2.7SACK untuk mendeteksi paket *reordering*

Sumber :Visualisasi penulis

Namun, pada aliran paket datayang ke 5, *destination host* menerima paket datadengan *sequence number* 5500. Oleh karena itu, *destination host* mengirimkan ACK 6500 yang artinya adalah *destination host* saat ini sudah menerima paket datadengan *sequence number* 5500, dan paket dataselanjutnya yang diharapkan diterima oleh *destination host* adalah dengan *sequence number* 6500. *Destination host* juga mengirimkan SACK *block*, tetapi *destination host* hanya mengirimkan 2 SACK *block*. *Destination host* menghapus SACK *block* 6000 – 6500 karena*destination host* mendeteksi sudah tidak ada gap pada paket data dengan *sequencenumber* 5000 – 6000.

Selanjutnya, *destination host* menerima paket datadengan *sequence number* 6500. Oleh karena itu, *destination host* mengirimkan ACK 7500 yang artinya adalah *destination host* sudah menerima paketdata dengan *sequence number* 6500, dan paket dataselanjutnya yang diharapkan diterima oleh *destination host* adalah dengan *sequence number* 7500. SACK *block* yang dikirimkan *destination host* saat ini hanya 1 SACK *block*, yaitu 8000 – 8500 karena*destination host* mendeteksi sudah tidak ada *gap*padapaket data dengan *sequence number*5000 – 7000.

Kemudian, *destination host* menerima paket datadengan *sequence number* 7500. Oleh karena itu, *destination host* mengirimkan ACK 8500, yang artinya adalah *destination host* sudah menerima paket datadengan *sequence number*

7500, dan *destination host* mengharapkan paket dataselanjutnya yang diterima adalah dengan *sequence number* 8500. Namun, *destination host* saat ini tidak mengirimkan SACK *block* karena *destination host*tidak mendeteksi *gap* pada semuapaket datayang diterima. Oleh karena itu, *destination host* tidak hanya mengirimkan SACK *block* yang disebabkan oleh paket *loss*, tetapi juga disebabkan oleh paket *reordering*. Penelitian ini menggunakan parameter SACK *block* untuk mengetahui apakah paket *reordering* terjadi pada *multi-path routing.*

2. *Rata-rata throughput*

Throughput merupakan kecepatan penerimaan paket datapada*destination host*. Apabila total semuapaket datayang dikirimkan oleh algoritme TCP *congestion control*adalah F Mbit dan*destination host* membutuhkan waktu T detik untuk menerima semua paket data, maka rata-rata *throughput* adalah F/T atau Mbit/detik atau Megabit *per second* (Mbps)(Kurose, J.F. dan Ross, K. W., 2017). Tujuan algoritme TCP *congestion control* adalah memaksimalkan penggunaan *bandwidth* jaringan pada *multiple paths*, sehingga didapatkanrata-rata *throughput* yang tinggi.

3. *Rata-rata RTT*

*Delay*terdiri dari lima jenis, yaitu *propagation delay*, *queuing delay*, *transmission delay*, *processing delay,* dan *end-to-end delay*. *Propagation delay* merupakan jarak fisik medium pengiriman paket dataantara *source host* dengan*destination host* dibagi dengan kecepatan medium pengirimanpaket data. Kemudian, *queuing delay* merupakan waktu yang dibutuhkan daripaket datamasuk ke dalam *buffer*router sampai paket datatersebut dikeluarkan dari *buffer* router. Sedangkan, *transmission delay* merupakan waktu yang dibutuhkan untuk mengirimkan semua bit dalam satu paket datake dalam *link* jaringan. Selanjutnya, *processing delay* merupakan waktu yang dibutuhkan untuk memeriksapaket *header* dan menentukan *interface* pengiriman paket data. Terakhir,*end-to-end delay* merupakan gabungan antara *propagation delay, queuing delay, transmission delay, dan processing delay*(Kurose, J.F. dan Ross, K. W., 2017). Rata-rata RTT merupakan rata-rata *end-to-end delay* pada saat *source host* mengirimkan semua paket datasampai menerima ACK untuk semua paket datatersebut. Semakin rendah rata-rata RTT, maka semakin tinggi kinerja algoritme TCP *congestion control*.

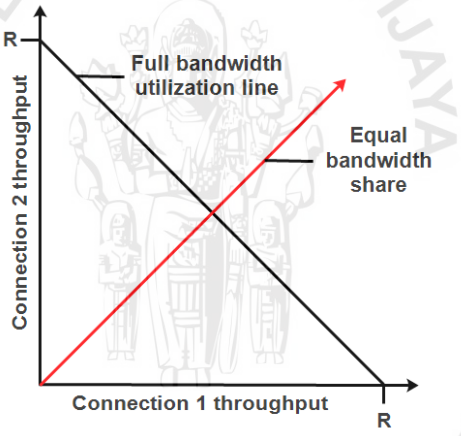
4. *Fairness*

Tujuan algoritme TCP *congestion control* selain mencegah *congestion* pada jaringan adalahuntuk mencapai *fairness*(Hasegawa, G. dan Murata, M., 2001). *Fairness* merupakan parameter kinerja algoritme TCP *congestion control* jika minimal *dua flow*mengirimkan paket datasecara bersamaan. Tujuan *fairness* adalah semua *destination host* memperoleh rata-rata*throughput* yang sama (Jain, R., Chiu, D. dan Hawe, W. R., 1984; Hasegawa, G. dan Murata, M., 2001). Sebagai contoh, 2 algoritme TCP *congestion control* mengirimkan paket

datamenuju 2 *destination host* melalui jalur yang sama, dimana masing-masing *source host* mengirimkan paket datamenuju *destination host* yang berbeda. Link delay antara *source host* 1 dan *destination host* 1 sama denganlink delay antara*source host* 2 dan *destination host* 2. Gambar 2.8menunjukkan *throughput* kedua algoritme TCP *congestion control*. Algoritme TCP *congestion control* dikatakan mencapai *fairness*tertinggi jika *throughput*kedua algoritme TCP *congestion control*sama, sehingga mencapai sudut 45 derajat. Apabila tidak tercapai *fairness* (*unfairness*), maka *throughput* kedua algoritme TCP *congestion control* tidak sama.

Fairness algoritme TCP *congestion control*dapat dihitung berdasarkanrata-rata *throughput* kedua algoritme TCP *congestion control* dengan menggunakan*Jain's fairness index*(Jain, R., Chiu, D. dan Hawe, W. R., 1984)padaPersamaan 2.10. Pada Persamaan 2.10, x_i merupakan rata-rata*throughput flow* i dengan total *flow* n. *Jain's fairness index* memiliki rentang nilai antara 0 – 1. Algoritme TCP *congestion control* dikatakan mendapatkan*fairness* paling tinggi jika *Jain's fairness index* bernilai 1.

$$J(x_1, x_2, \dots, x_n) = \frac{(\sum_{i=1}^{n} x_i)^2}{n * \sum_{i=1}^{n} x_i^2}, \; J \in [0,1] \qquad (2.10)$$



Gambar 2.8*Throughput* kedua TCP *flow*

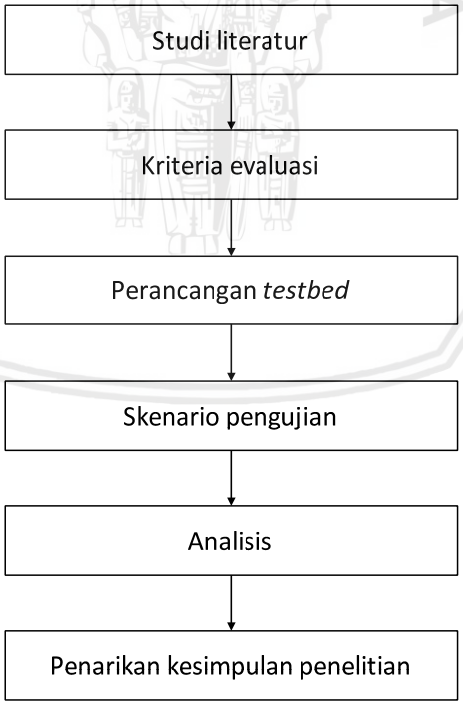
Sumber : Kurose, J.F. dan Ross, K. W.(2017)

BAB 3 METODOLOGI

Pada bab metodologi penelitian dilakukan perancangan analisis kinerja kinerja algoritme TCP *congestion control* Reno, BIC, CUBIC, dan BBR pada *multi-path routing*. Analisis yang dilakukan pada penelitian ini merupakan analisis empiris berdasarkan perilaku dan kinerja algoritme TCP *congestion control* Reno, BIC, CUBIC, dan BBR pada *multi-path routing.*Setiap*multiple paths*akanmenggunakan *cost* yang sama. *Cost*yang digunakan meliputi *link delay*, *loss rate*, ukuran *buffer*, dan *bottleneck bandwidth*. Gambar 3.1 menunjukkan diagram alir urutan penelitian yang akan dilakukan. Urutan penelitian yang dilakukan meliputi studi literatur, kriteriaanalisis algoritme TCP *congestion control* pada jaringan *multi-path routing*, perancangan *testbed*untuk melakukan evaluasi, skenario pengujian untuk masing-masing evaluasi, cara melakukan analisis untuk masing-masing evaluasi, dan penarikan kesimpulan berdasarkan hasil evaluasi dan analisis.

3.1 Studi Literatur

Tahapan studi literatur merupakan proses pencarian literatur dan pemahaman konsep algoritme TCP *congestion control* Reno, BIC, CUBIC, BBR, *multi-pathrouting*, dan parameter kinerja algoritme TCP *congestion control*. Literatur yang digunakan pada penelitian ini meliputi buku, jurnal, dan prosiding.



Gambar3.1 Diagram alir metodologi penelitian

3.2 Kriteria Evaluasi

Penelitian ini melakukan 5 evaluasi untuk analisis kinerja algoritme TCP *congestion control* Reno, BIC, CUBIC, dan BBR pada *multi-path routing*. Evaluasi yang pertama adalah melakukan perbandingan Reno, BIC, CUBIC, dan BBR antara *single path routing* dengan *multi-path routing*. Evaluasi kedua dan ketiga adalah memberikan perlakuan pada *link multiple paths*, yaitu variasi *link delay* dan variasi *loss rate*. Sedangkan evaluasi keempat dan kelima adalah memberikan perlakuan pada trafik yang melewati *multiple paths*, yaitu *inter* TCP *protocolfairness* dan *fairness* antara TCP dengan UDP.

1. Perbandingan antara *single path routing* dengan *multi-path routing*

Tujuan analisis perbandingan antara *single path routing* dengan *multi-path routing* adalah untuk mengetahui apakah paket *reordering* tetap dapat terjadi pada *multi-path routing*, walaupun *costmultiple paths* yang digunakan pada penelitian ini sama. Seperti yang telah dijelaskan pada sub bab 2.3, paket *reordering* yang terjadi pada *multi-path routing* dapat berpengaruh terhadap rata-rata *throughput*. Oleh karena itu, pada analisis perbandingan antara *single pathrouting* dengan *multi-path routing*, parameter yang digunakan untuk mengukur kinerja algoritme TCP *congestion control* Reno, BIC, CUBIC, dan BBR adalah SACK *block* dan rata-rata *throughput*. SACK *block* digunakan untuk mengetahui apakah terjadi paket *reordering* pada *multi-path routing*. Seperti yang telah dijelaskan pada sub bab 2.4, SACK *block* pada *single path routing* hanya terjadi pada saat terjadi paket *loss*. Namun, jika terjadi paket *reordering* pada *multi-path routing*, maka jumlah SACK *block* pada Reno, BIC, dan CUBIC akan mengalami kenaikan dibandingkan dengan *single path routing* karena SACK *block* juga terjadi pada saat terjadi paket *rerodering*. Hasil kinerja yang didapatkan Reno, BIC, CUBIC, dan BBR pada *multi-path routing*akan menjadi dasar untuk melakukan observasi kinerja Reno, BIC, CUBIC, dan BBR pada evaluasi variasi *link delay*, variasi *loss rate*, *inter* TCP *protocol fairness*, dan *fairness* antara TCP dengan UDP.

2. Variasi *link delay*

Variasi link delay merupakan representasi dari perbedaan letak geografis antara *source host* dengan *destination host*. Semakin tinggi *link delay*, maka semakin tinggi BDP, sehingga semakin lama waktu yang dibutuhkan Reno, BIC, dan CUBIC untuk menaikkan ukuran *window*sampai BDP. Waktu yang dibutuhkan Reno, BIC, dan CUBIC untuk menaikkan ukuran *window* sampai BDP tergantung pada keagresifan mekanisme Reno, BIC, dan CUBIC dalam menaikkan ukuran *window*.

Sementara itu, BBR menggunakan estimasi *bottleneck bandwidth* dan estimasi RTT untuk menentukan ukuran *window*. Semakin tinggi *link delay*, maka semakin tinggi RTT, sehingga untuk tetap dapat mencapaiBDP, maka BBR harus menyesuaikan estimasi RTT dengan *link delay* pada *multiple paths.* Oleh karena itu, evaluasi variasi *link delay* dilakukan untuk mengetahui perilaku dan kinerja algoritme TCP *congestion control* Reno, BIC, CUBIC, dan BBR untuk dapat

mencapai BDP jika *link delay* pada *multiple paths* semakin tinggi. Parameter kinerja yang diukur adalah rata-rata *throughput* dan rata-rata RTT.

3. Variasi *loss rate*

Padasaat terdapat banyak *flow* yang mengirimkan paket datamelalui*link* jaringan yang sama, paket *loss*yang dialami oleh satu *flow*sering dimodelkan sebagai proses yang terjadi secara acak (Li, Y. T., Leith, D. dan Shorten, R. N., 2007). Reno, BIC, dan CUBIC merupakan algoritme TCP *congestion control* berbasis paket *loss* yang menggunakan paket *loss* sebagai tanda *congestion* pada jaringan. Pada saat terjadi paket *loss*, Reno, BIC, dan CUBIC menurunkan ukuran *window* untuk mencegah *congestion* terjadi secara terus-menerus. Pada saat Reno, BIC, dan CUBIC menurunkan ukuran *window*, Reno, BIC, dan CUBIC membutuhkan waktu untuk kembali menaikkan ukuran *window* sampai BDP. Panjang atau pendeknya waktu yang dibutuhkan untuk mencapai BDP tergantung pada keagresifan Reno, BIC, dan CUBIC dalam menaikkan ukuran *window* dan seberapa drastis Reno, BIC, dan CUBIC menurunkan ukuran *window.* Sebaliknya, BBR merupakan algoritme TCP *congestion control* yang tidak menggunakan paket *loss* sebagai tanda *congestion* pada jaringan. Oleh karena itu, evaluasi *loss rate* dilakukan untuk mengetahui perilaku dan kinerja algoritme TCP *congestion control* Reno, BIC, CUBIC, dan BBR, jika dilakukan variasijumlah paket *loss*berdasarkan*loss rate* pada *multiple paths*. Kinerja yang diukur adalah rata-rata *throughput* dan rata-rata RTT

4. *Inter*TCP *protocol fairness*

Pada jaringan Internet,terdapat kemungkinan dua TCP *flow*mengirimkan paket datamelalui *link* jaringan yang sama, dimana masing-masing TCP *flow* menggunakan algoritme TCP *congestion control* yang berbeda. Pada saat dua TCP *flow* mengirimkan paket datamelalui*link* jaringan yang sama, maka kinerja algoritme TCP *congestioncontrol* dipengaruhi oleh *fairne*ss(Hasegawa, G. dan Murata, M., 2001). Reno, BIC, CUBIC, dan BBR dikatakan mencapai *fairness* jika masing-masing *destination host* mendapatkan rata-rata *throughput* yang sama. Sebaliknya, jika terjadi *unfairness*, maka terdapat *destination host* yang mendapatkan rata-rata *throughput* lebih tinggi dan lebih rendah. *Fairness* dipengaruhi oleh keagresifan mekanisme Reno, BIC, CUBIC, dan BBR dalam menaikkan ukuran *window*. Oleh karena itu, evaluasi *inter* TCP protocol *fairness* dilakukan untuk mengetahui perilaku dan *fairness* algoritme TCP *congestion control* Reno, BIC, CUBIC, dan BBR jika dua algoritme TCP *congestion control* yang berbeda mengirimkan paket datasecara bersamaan melalui*multiple paths*. *Jain's fairness index* digunakan untuk mengukur *fairness* kedua algoritme TCP *congestion control* yang mengirimkan paket datamelalui *multiple paths* yang sama.

5. *Fairness* antara TCP dengan UDP

Pada jaringan Internet juga memungkinkan TCP dan UDP *flow* mengirimkan paket datamelalui*link* jaringan yang sama. Aplikasi seperti *Voice over IP* (VoIP) dan *video streaming*menggunakan UDP sebagai protokol *transport* karena UDP mengirimkan paket datalebih cepat dibandingkan dengan TCP. Penyebabnya adalah UDP tidak menggunakan mekanisme ACK, tidak memiliki mekanisme pendeteksian paket *loss*, dan tidak memiliki mekanisme *congestion control*, sehingga UDP mengirimkan paket datadengan kecepatan yang konstan berdasarkan *constant bit rate* (CBR)(Kurose, J.F.dan Ross, K. W., 2017). Oleh karena itu, aplikasi yang membutuhkan pengiriman paket datasecepat mungkin dan memiliki toleransi terhadap paket *loss*, biasanya menggunakan UDP.

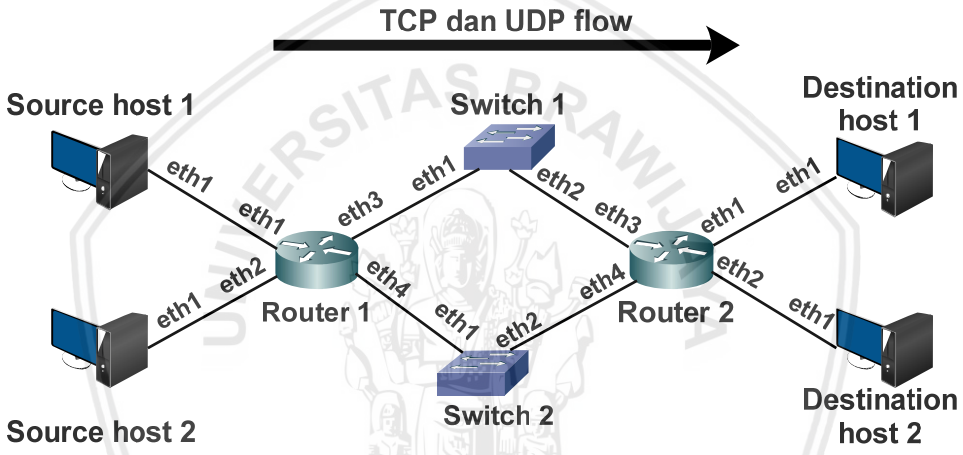
Namun, UDP *flow* dapat mengakibatkan *unfairness* pada algoritme TCP *congestioncontrol*(Kurose, J.F.dan Ross, K. W., 2017). Apabila *buffer*router terisi penuh dan terjadi paket *loss*, algoritme TCP *congestion control* menurunkan ukuran *window.* Namun, UDP terus mengirimkan paket datasebesar CBR karena UDP tidak memiliki mekanisme pendeteksian *congestion,* sehingga *bandwidth* jaringan akan didominasi oleh UDP(Chang, H. -Y., Lee, W. -T. dan Wei, H. -W., 2014). Oleh karena itu, evaluasi *fairness* antara TCP dengan UDP dilakukan untuk mengetahui perilaku dan *fairness* algoritme TCP *congestion* control Reno, BIC, CUBIC, dan BBR jika CBR UDP semakin tinggi. *Jain's fairness index* digunakan untuk mengukur *fairness* antara TCP dengan UDP.

Berdasarkan penjelasan 5 kriteria evaluasi yang digunakan untuk analisis algoritme TCP *congestion control* Reno, BIC, CUBIC, dan BBR pada *multi-path routing*, maka variabel bebas yang digunakan pada penelitian ini adalah perbandingan antara *single path routing* dengan *multi-path routing*,variasi *link delay*, variasi *loss rate*, *inter* TCP *protocol fairness*, dan *fairness* antara TCP dengan UDP. Evaluasi perbandingan antara *single path routing* dengan *multi-path routing* berpengaruh terhadap parameter kinerja SACK *block* dan rata-rata *throughput*. Evaluasi variasi *link delay* dan variasi *loss rate* berpengaruh terhadap parameter kinerja rata-rata *throughput* dan rata-rata RTT. Sedangkan evaluasi *inter* TCP *protocol fairness* dan *fairness* antara TCP dengan UDP berpengaruh terhadap parameter kinerja *fairness* yang dihitung berdasarkan *Jain's fairness index.* Oleh karena itu, variabel terikat yang digunakan pada penelitian ini adalah SACK *block*, rata-rata *throughput*, rata-rata RTT, dan *fairness*.

3.3 Perancangan *Testbed*

Penelitian ini melakukan pembuatan *testbed* yang digunakan untuk melakukan analisis algoritme TCP *congestion control*Reno, BIC, CUBIC, dan BBRpada *multi-path routing*berdasarkan 5 kriteria evaluasi. *Testbed* dibuat dengan menggunakan 8 *virtualmachine* (VM) pada VirtualBox (VirtualBox, n.d.). VirtualBox berjalan pada komputer dengan spesifikasi CPU Intel Core i5-7200U dan *Random Access Memory* (RAM) sebesar 8 GB. Semua VM dihubungkan dengan*virtual ethernet,* sehingga membentuk topologi seperti yang ditunjukkan

pada Gambar 3.2. Konfigurasi setiap VM dilakukan berdasarkan Tabel 3.1.Setiap VM menggunakan sistem operasi Linux Ubuntu Server 16.04 dengan Linux Kernel 4.15 yang sudah mendukung algoritme TCP *congestion control* Reno, BIC, CUBIC, dan BBR. Setiap VM menggunakan CPU sebesar 1 *core*. Kemudian, semua VM selain router 1 menggunakan RAM sebesar 768 MB karena keterbatasan sumberdaya komputer yang digunakan. Router 1 menggunakan RAM sebesar 2 GB karena router 1 juga berfungsi untuk melakukan *capture* TCP paket datayang akan digunakan sebagai bahan analisis, sehingga membutuhkan RAM yang lebih besar. Kemudian, setiap *interface* VM menggunakan *network adapter* Intel PRO/1000 MT Desktop yang merupakan *network adapter default* pada VirtualBox. Konfigurasi IP pada setiap *interface* VM selain switch 1 dan switch 2 ditunjukkan pada Tabel 3.2. Switch tidak perlu dilakukan konfigurasi IP karena switch hanya mengenal alamat *Media Access Control* (MAC).



Gambar3.2Topologi *multi-path routing*

Tabel 3.1 Konfigurasi VM

Konfigurasi	Deskripsi
Sistem operasi	Linux Ubuntu Server 16.04 dengan Linux Kernel 4.15
CPU	1 *core*
RAM	768 MB, kecuali router 1 sebesar 2 GB
Network adapter	Intel PRO/1000 MT Desktop

Penelitian ini mengimplementasikan 2 switch, dimana setiap jalur pada *multiple paths* diimplementasikan satu switch. Implementasi switch berfungsi untuk melakukan emulasi *cost* pada *multiple paths*. Emulasi *cost* yang dilakukan adalah *bottleneck bandwidth*, *link delay*, *loss rate*, dan ukuran *buffer*. Emulasi

bottleneck bandwidth pada penelitian ini menggunakan *Token Bucket Filter* (TBF) (Kuznetsov, A. N., n.d.). *Bottleneck bandwidth* pada setiap jalur dibatasi sebesar 10 Mbps. Penggunaan *bottleneck bandwidth*sebesar 10 Mbps juga dilakukan oleh Wang, J., et al.(2013). Oleh karena itu, penggunaan *bottleneck bandwidth* sebesar 10 Mbps sudah sesuai dengan kaidah penelitian terdahulu untuk melakukan analisis kinerja algoritme TCP *congestion control* pada *multi-path routing*. Emulasi *bottleneck bandwidth* 10 Mbps dilakukan pada setiap *interface* switch, yaitu eth1 dan eth2. Sedangkan *bandwidth* antara *host* dengan router merupakan *bandwidth default* sistem VirtualBox,yang setelah dilakukan pengukuran adalah sebesar 2.4 Gbps. Oleh karena itu, *bandwidth* maksimum yang tersedia pada saat pengiriman paket datadari *source host* menuju *destination host* dibatasi oleh emulasi *bottleneck bandwidth*. Apabila dilakukanpengiriman paket datamelalui mekanisme *multi-path routing* antara *source host* dengan *destination host*, makatotal maksimum *bandwidth* yang tersedia adalah 20 Mbps.

Tabel 3.2 Konfigurasi IP pada setiap *interface* VM

Virtual machine	**Konfigurasi IP**
Source host 1	eth1 = 172.16.1.2
Source host 2	eth1 = 172.16.2.2
Destination host 1	eth1 = 172.16.5.2
Destination host 4	eth1 = 172.16.6.2
Router 1	- eth1 = 172.16.1.1 - eth2 = 172.16.2.1 - eth3 = 172.16.4.1 - eth4 = 172.16.5.1
Router 2	- eth1 = 172.16.5.1 - eth2 = 172.16.6.1 - eth3 = 172.16.3.2 - eth4 = 172.16.4.2

Selanjutnya, emulasi *link delay*, *loss rate*, dan ukuran *buffer* pada setiap switch dilakukan dengan menggunakan NetEM (NetEM, n.d.). Pada semua evaluasi selain evaluasi variasi *link delay*,*link delay* setiap *multiple paths*ditetapkan sebesar 10 ms. *Link delay* 10 ms termasuk *link delay* yang rendah, sehingga pada saat melakukan evaluasi perbandingan antara *single path routing* dengan *multi-path routing*, variasi *loss rate*, *inter* TCP *protocol fairness*, dan *fairness* antara TCP dengan UDP, kinerja Reno, BIC, CUBIC, dan BBR tidak dipengaruhi oleh *link delay*. *Link delay* pada penelitian ini merupakan representasi dari RTT antara router 1 dengan router 2. Oleh karena itu, untuk

melakukan konfigurasi *link delay* 10 ms, setiap *interface* switch dilakukan konfigurasi *link delay* sebesar 5 ms.

Kemudian,emulasi*loss rate* pada penelitian ini hanya dilakukan pada saat pengiriman paketdata dari *source host* menuju *destination host,* tanpa dilakukan pada saat pengiriman ACK dari *source host* menuju *destination host*. Penyebabnya adalah perubahan*window* algoritme TCP *congestion control*Reno, BIC, CUBIC, dan BBR dipengaruhi pada saat paketdata yang dikirimkan oleh *source host* mengalami paket *loss*. Oleh karena itu, emulasi *loss rate* cukup dilakukan pada *interface* eth2 pada switch 1 dan switch 2. Ukuran*loss rate*pada setiap switch disesuaikan dengan skenario evaluasi variasi *loss rate* yang akan dijelaskan pada sub bab 3.4.

Terakhir, emulasi ukuran *buffer*pada setiap switch ditetapkan sebesar 100 paket data. Apabila dilakukan pengiriman paket datamelaluiteknik *multi-path routing*, total ukuran *buffer* yang tersedia adalah 200 paket data. Pada evaluasi variasi*link delay* dengan variasi yang tertinggi, ukuran *buffer* 200 paket databernilai sekitar 1 BDP. Ukuran *buffer* sebesar 1 BDP sudah sesuai dengan ukuran *buffer* yang disarankan oleh Villamizar, C. dan Song, C.(1994). Kemudian, pada evaluasi perbandingan antara *single path* dan *multi-path routing*, variasi *loss rate*, *inter* TCP *protocol fairness*, dan *fairness* antara TCP dengan UDP, ukuran *buffer* sebesar 200 paket databernilai sekitar 11 BDP. Ukuran *buffer* yang lebih besar dari 1 BDP masih sesuai dengan realita jaringan Internet saat ini (Gettys, J. dan Nichols, K., 2011).

Masing-masing switch terhubung dengan 2 router yang digunakan untuk melakukan *multi-path routing*. Jumlah jalur yang digunakan adalah 2 jalur karena merupakan jumlah jalur minimum untuk melakukan *multi-path routing*(He, J. dan Rexford, J., 2008). Paket datayang melewati routerakandikirimkan secara merata melalui dua jalur tersebut berdasarkan *round-robin*. Penelitian ini menggunakan *round-robin* karena *round-robin* cocok digunakan pada *multiple paths* yang menggunakan *cost* sama (Singh, S. K., Das, T. dan Jukan, A., 2015). Selain itu, *round-robin* hanya mengakibatkan sedikit *overhead* pada router (He, J. dan Rexford, J., 2008).Penelitian ini menggunakan *True* (*or Trivial*) *Link Equalizer* (TEQL) (Hubert, B., et al., 2002) untuk melakukan pengiriman paket databerdasarkan *round-robin.*

Setiap router terhubung dengan *host* untuk melakukan evaluasi algoritme TCP *congestion control* Reno, BIC, CUBIC, dan BBR pada *multi-path routing*. Jumlah *host* yang digunakan pada penelitian ini adalah 4 *host*, dimana 2 *host* merupakan *source host* dan 2 *host* merupakan *destination host*. Penggunaan 2 *host* pada *source host* dan *destination host* dilakukan untuk mengakomodir evaluasi *inter* TCP *protocol fairness* dan *fairness* antara TCP dengan UDP. Kemudian, untuk memperbesar ukuran TCP *receive window*, maka fitur *window scalling option*(Jacobson, V., Braden, R. dan Borman, D., 1992) diaktifkan pada setiap *host*. Fitur SACKjuga diaktifkan pada setiap *host,* sehingga *source host* mengetahui secara pasti paket datayang mengalami paket *loss*.Selain itu, SACK juga digunakan untuk mengetahui apakah terjadi paket *reordering* pada *multi-*

path routing.Kemudian, fitur DSACK, *timestamp*, dan *reordering heuristic* juga diaktifkan pada setiap *host* untuk mengatasi permasalahan paket *reordering*seperti yang telah dijelaskan pada sub bab 2.3. Penelitian ini menggunakan konfigurasi *reordering heuristic* sesuai dengan *default* dari Linux Kernel 4.15, yaitu 300 paket data. Selanjutnya, konfigurasi Reno, BIC, CUBIC, dan BBR hanya perlu dilakukan pada *source host*(Cardwell, N., et al., 2016a). Parameter yang digunakan Reno, BIC, CUBIC, dan BBR merupakan parameter *default* pada Linux Kernel 4.15. Khusus pada BBR, dilakukan pengaktifan fitur *fair queuing* sebagai mekanisme *queuing* pada *source host*(Cheng, Y. dan Cardwell, N., 2016).

3.4 Skenario Pengujian

Skenario pengujian dilakukan untuk mengukur kinerja algoritme TCP *congestion control* Reno, BIC, CUBIC, dan BBR pada *multi-path routing* untuk setiap evaluasi. Semua evaluasi dilakukan dengan cara mengirimkan TCP/UDP paket datadari *source host* menuju *destination host*selama 100 detik dengan menggunakan iPerf3 (iPerf, n.d.). Pengiriman paket datadengan menggunakan iPerf berdasarkan durasi detik juga dilakukan sebelumnya olehLukaseder, T., et al. (2016), Hock, M., Bless, R. dan Zitterbart, M.(2017), dan Yue, Z., et al.(2012). Oleh karena itu, pengiriman TCP/UDP paket datadengan menggunakan iperf3 selama 100 detik sudah sesuai dengan kaidah penelitian sebelumnya untuk mengetahui perilaku dan kinerja algoritme TCP *congestion control* Reno, BIC, CUBIC, dan BBR pada *multi-path routing*. Kemudian, semua evaluasi dilakukan perulangan sebanyak 10 kali untuk menjaga tingkat akurasi hasil pengukuran kinerja.Hasil pengukuran kinerja yang akan dilakukan analisis merupakan hasil pengukuran kinerja berdasarkan rata-rata dari perulangan sebanyak 10 kali tersebut.

1. Perbandingan antara *single path routing* dengan multi-path routing

Evaluasi perbandingan antara *single path routing* dengan *multi-path routing* dilakukan dengan cara mengirimkan satu TCP *flow* dari *source host* 1 menuju *destination host* 1. Evaluasi *single path routing* dilakukan dengan cara mengirimkan semuapaket datamelalui satu jalur. Sedangkan evaluasi *multi-path routing* dilakukan dengan cara mengirimkan semuapaket datamelalui dua jalur. Algoritme TCP *congestion control* Reno, BIC, CUBIC, dan BBR dilakukan evaluasi secara bergantian untuk *single path routing* dan *multi-path routing*.

2. Variasi *link delay*

Evaluasi variasi *link delay* dilakukan dengancara mengirimkan satu TCP *flow*melalui*multiple paths* dari *source host* 1 menuju *destination host* 1. Setiap evaluasi dilakukan variasi pada *link delay.* Variasi *link delay* yang digunakan pada setiap *multiple paths* adalah 10 ms – 100 ms dengan kelipatan 10 ms untuk setiap pengukuran kinerja. Penentuan variasi *link delay* maksimum 100 ms juga dilakukan sebelumnya oleh Wang, J., et al.(2013). Oleh karena itu, penentuan variasi *link delay* 10 ms – 100 ms sudah sesuai dengan kaidah penelitian

sebelumnya untuk melakukan analisis algoritme TCP *congestion control* pada *multi-path routing*.Algoritme TCP *congestion control* Reno, BIC, CUBIC, dan BBR dilakukan evaluasi secara bergantian untuk setiap variasi *link delay*.

3. Variasi *loss rate*

Evaluasi variasi *loss rate* dilakukan dengan cara mengirimkan satu TCP *flow*melalui*multiple paths* dari *source host* 1 menuju *destination host* 1. Setiap evaluasi dilakukan variasi pada *loss rate*. Variasi *loss rate* yang digunakan pada setiap *multiple paths* adalah 1% – 10% dengan kelipatan 1% untuk setiap pengukuran kinerja. Penentuan *variasi loss rate*1% – 10% dilakukan untuk mensimulasikan *loss rate* rendah sampai *loss rate* tinggi, sehingga dapat diketahui perbedaan kinerja algoritme TCP *congestion control*Reno, BIC, CUBIC, dan BBR untuk setiap variasi *loss rate*.Algoritme TCP *congestion control* Reno, BIC, CUBIC, dan BBR dilakukan evaluasi secara bergantian untuk setiap variasi *loss rate*.

4. Inter TCP *protocol fairness*

Evaluasi *inter* TCP *protocol fairness* dilakukan dengan cara mengirimkan dua TCP *flow*melalui*multiple paths* dengan masing-masing TCP*flow* menggunakan algoritme TCP *congestion control* yang berbeda. TCP *flow* pertama dikirimkan dari*source host* 1 menuju *destination host* 1, sedangkan TCP *flow* kedua dikirimkan dari *source host* 2 menuju *destination host* 2. Pengiriman TCP *flow* pertama dilakukan dari detik ke 1 – 100, sedangkan TCP *flow* kedua dikirimkan dari detik ke 6 – 100. Pengiriman kedua TCP *flow* tidak dilakukan secara bersamaan untuk mengetahui perubahan perilaku algoritme TCP *congestion control*pada saat pengiriman satu TCP *flow*dandua TCP *flow*. Konfigurasi algoritme TCP *congestion control* pada *source host* 1 dan *source host* 2 untuk setiap evaluasi dilakukan berdasarkan Tabel 3.3.

Tabel 3.3 Konfigurasi *source host* pada evaluasi *inter* TCP *protocol fairness*

Pengujian	*Source host*1	*Source host*2
1	CUBIC	Reno
2	BIC	CUBIC
3	BIC	Reno
4	BBR	CUBIC
5	BBR	Reno

6	BBR	BIC

5. *Fairness* antara TCP dengan UDP

Evaluasi *fairness* antara TCP dengan UDP dilakukan dengan caramengirimkan dua*flow*melalui*multiple paths*. *Flow* pertama merupakan TCP *flow* dan *flow* kedua merupakan UDP *flow*. TCP *flow* dikirimkan dari *source host* 1 menuju *destination host* 1. Sedangkan UDP*flow*dikirimkan dari *source host* 2 menuju *destination host* 2. Pengiriman kedua *flow* tidak dilakukan secara bersamaan untuk mengetahui perubahan perilaku dan kinerja algoritme TCP *congestion control* Reno, BIC, CUBIC, dan BBR tanpa dan dengan UDP *flow.*TCP *flow* dikirimkan dari detik ke 1 – 100, sedangkan UDP *flow* dikirimkan dari detik ke 6 – 100. Pengiriman UDP *flow* dilakukan berdasarkan variasi CBR 2 Mbps – 20 Mbps dengan kelipatan 2 Mbps untuk setiap pengukuran kinerja. Penentuan variasi CBR 2 Mbps – 20 Mbps dilakukan untuk mengetahui perubahan perilaku dan kinerja algoritme TCP *congestion control*Reno, BIC, CUBIC, dan BBR pada saat UDP mengirimkan paket datadari CBR yang rendah sampai CBR yang bernilai sama dengan total *bandwidth* yang tersedia pada *multiple paths*. Algoritme TCP *congestion control* Reno, BIC, CUBIC, dan BBR dilakukan evaluasi secara bergantian untuk setiap variasi CBR.

3.5 Analisis

Analisis yang digunakan pada penelitian ini merupakan analisis empiris berdasarkan perilaku dan kinerja algoritme TCP *congestion control* Reno, BIC, CUBIC, dan BBR pada*multi-path routing*. Hasil eksperimen didapatkandari*capture* paket data dengan menggunakan Tcpdump (Tcpdump & Libcap, n.d.).*Capture* paket data dilakukan pada semua paket data yang keluar dan masuk pada router 1. Kemudian, dilakukan pengukuran kinerja dan observasi perilaku Reno, BIC, CUBIC, dan BBRdari hasil *capture* paket data dengan menggunakan Wireshark (Wireshark, n.d.). Observasi perilaku Reno, BIC, CUBIC, dan BBR yang dilakukan meliputi perubahan *window* Reno, BIC, CUBIC, dan BBR selama 100 detik evaluasi.

3.6 Penarikan Kesimpulan Penelitian

Berdasarkan hasil evaluasi dan analisis kinerja algoritme TCP *congestion control* Reno, BIC, CUBIC, dan BBR pada *multi-path routing*untuk setiap pengukuran kinerja, maka dilakukan penarikan kesimpulan. Kesimpulan yang dilakukan adalah bagaimana dampak penggunaan*multi-path routing* terhadap kinerja algoritme TCP *congestion control*Reno, BIC, CUBIC, dan BBR. Kemudian, berdasarkan kinerja masing-masing algoritme TCP *congestion control*, algoritme TCP *congestion control* manakah yang mendapatkan kinerja paling baik pada *multi-path routing*.

BAB 4 KONFIGURASIEKSPERIMENTAL

Pada babkonfigurasi eksperimentalakan dijelaskan proses konfigurasiyang digunakan untuk melakukan analisis kinerja algoritme TCP *congestion control* Reno, BIC, CUBIC, dan BBR pada *multi-path routing*. Konfigurasi yang dilakukan meliputi konfigurasi *testbed* dan perangkat lunak yang digunakan untuk evaluasi.

4.1 Konfigurasi *Testbed*

Pada konfigurasi *testbed*akan dijelaskan proses konfigurasi yang digunakan untuk membuat *testbed*. Konfigurasi *testbed* terdiri dari konfigurasi switch, konfigurasi router, dan konfigurasi *host*.

4.1.1 Konfigurasi Switch

Penelitian ini menggunakan bridge-utils (bridge-utils, 2019) untuk mengimplementasikan switch pada Linux Ubuntu Server 16.04. Konfigurasi yang dilakukan pada bridge-utils adalah menambahkan kedua *interface* switch ke dalam parameter *bridge,* sehingga semuapaket datayang melewati switch dapat dilakukan pengiriman berdasarkan alamat MAC. Kemudian, penelitian ini juga melakukan konfigurasi pada VirtualBox untuk mengaktifkan mode switch pada VirtualBox dengan cara mengaktifkan mode *promiscuous*. Mode *promiscuous* diaktifkan pada kedua *interface* switch melalui perintah `VboxManage modifyvm s1 --nicpromisc3 allow-all` dan `VboxManage s1 modifyvm --nicpromisc4 allow all`.Perintah `s1` merupakan nama VM switch 1 pada VirtualBox. Sedangkan perintah `--nicpromisc3` dan `--nicpromisc4` merupakan nama*interface* switch pada *network adapter* VirtualBox. Mode *promiscuous* juga diaktifkan pada switch 2 melalui perintah yang sama dengan switch 1.

Setelah implementasi switch, maka proses konfigurasi selanjutnya adalah melakukan emulasi *cost*. Konfigurasi yang pertama adalah konfigurasi TBF untuk melakukan emulasi *bottleneck bandwidth*. Konfigurasi TBF dilakukan pada kedua *interface* setiap switch. Gambar 4.1 menunjukkan perintah TBF pada Linux Ubuntu Server 16.04 untuk melakukan emulasi *bottleneck bandwidth* 10 Mbps. Perintah `enp0s9` dan `enp0s10` merupakan nama*interface* switch yang diberikan oleh VirtualBox. Kemudian, konfigurasi TBF dilakukan berdasarkan tiga parameter, yaitu *rate*, *burst*, dan *limit*. *Rate* merupakan kecepatan rata-rata pengiriman paket data. *Burst* merupakan jumlah *token* yang disimpan pada *bucket*. Satu *token*sama dengan satu byte. Sedangkan *limit* merupakan hasil penjumlahan dari *burst* dengan ukuran *buffer*(Wagner, K., 2001). Konfigurasi *rate* pada TBF dilakukanmelalui perintah `rate 10mbit` karena *bottleneck bandwidth* yang digunakan pada penelitian ini adalah 10 Mbps. Kemudian, konfigurasi *burst size* dilakukan berdasarkan *rate* dibagi dengan *clock tick*(Kuznetsov, A., n.d.). *Clock tick default* pada Linux Kernel 4.15 adalah 250 Hz. Oleh karena itu, konfigurasi *burst* pada TBF adalah 5 KB melalui perintah `burst 5kb`. Selanjutnya, pada konfigurasi *limit,* jika nilai *limit* sama dengan *burst*, maka ukuran *buffer*

adalah 0 (Wagner, K., 2001). Seperti yang telah dijelaskan pada sub bab 3.3, penelitian ini melakukan emulasi ukuran *buffer* hanyapada NetEM. Oleh karena itu, konfigurasi *limit* pada TBF sama dengan konfigurasi *burst*, yaitu 5 KBmelalui perintah `limit 5kb`.

```
Tc qdisc add dev enp0s9 tbf rate 10mbit burst 5kb limit 5kb
Tc qdisc add dev enp0s10 tbf rate 10mbit burst 5kb limit 5kb
```

Gambar4.1Perintah TBF pada Linux Ubuntu

Konfigurasi selanjutnya adalah konfigurasi pada NetEM untuk melakukan emulasi *link delay*, *loss rate*, dan ukuran *buffer*. Gambar 4.2 menunjukkan perintah NetEM pada Linux Ubuntu Server 16.04 untuk emulasi *link delay* 10 ms, *loss rate* 1%, dan ukuran *buffer* 100 paket data. Seperti yang telah dijelaskan pada sub bab 3.3, *link delay*pada penelitian ini merupakan representasi dari RTT. Oleh karena itu, emulasi *link delay* 10 ms dilakukan dengan carakonfigurasi*linkdelay* 5 ms pada setiap *interface switch* melalui perintah `delay 5ms` pada NetEM. Kemudian, emulasi *loss rate* 1% dilakukan melalui perintah `loss1%`pada NetEM. Seperti yang telah dijelaskan pada sub 3.3, *loss rate*hanya dilakukan pada saat pengiriman paket datadari *source host* menuju *destination host*. Oleh karena itu, konfigurasi *loss rate* hanya dilakukan pada *interface*`enp0s9` setiap switch yang merupakan *outgoing interface* switch dari *source host* menuju *destination host*. Terakhir, emulasi ukuran *buffer* sebesar 100 paket datadilakukan melalui perintah `limit 100` pada NetEM untuk setiap *interface* switch.

```
Tc qdisc add dev enp0s9 netem delay 5ms limit 100 loss 1%
Tc qdisc add dev enp0s10 netem delay 5ms limit 100
```

Gambar4.2Perintah NetEM pada Linux Ubuntu

```
Tc qdisc add dev enp0s9 root handle 1:0  tbf rate 50mbit burst 25kb limit 25kb

Tc qdisc add dev enp0s10 root handle 1:0  tbf rate 50mbit burst 25kb limit 25kb

Tc qdisc add dev enp0s9 parent 1:1 handle 10:  netem delay 5ms limit 100 loss 1%

Tc qdisc add dev enp0s10 parent 1:1 handle 10:  netem delay 5ms limit 100
```

Gambar4.3Perintah penggabungan TBF dengan NetEm

Kemudian, dilakukan konfigurasi penggabungan antara TBF dengan NetEM, sehingga TBF dan NetEM dapat berjalan pada switch. Gambar 4.3 menunjukkan perintah penggabungan antara TBF dengan NetEM. Pada Gambar 4.3, konfigurasiTBF menjadi *root* pada *traffic control* Linux dan konfigurasi NetEM menjadi *parent*pada *traffic control* Linux. Oleh karena itu, setiappaket datayang melewati switch, maka TBF dilakukan eksekusi pertama dan NetEM dilakukan eksekusi setelah TBF.

4.1.2 Konfigurasi Router

Router pada penelitian ini berfungsi untuk melakukan *multi-path routing*. *Routing* yang digunakan pada penelitian ini merupakan*routing* statis, sehingga jalur pengiriman paket datadari *source host* menuju *destination host*, dan jalur pengiriman ACK dari *destination host* menuju *source host*dilakukan konfigurasi secara manual. Konfigurasi router yang pertama adalah konfigurasi IP *forwarding*. Gambar 4.4 menunjukkan konfigurasi IP *forwarding* pada Linux Ubuntu Server 16.04 untuk setiap router. Konfigurasi IP *forwarding* dilakukan pada `/etc/sysctl.conf`. Pada Gambar 4.4, baris pertama merupakan perintah pada router untuk mengirimkan paket datajika *interface* router menerima paket datadengan tujuan IP jaringan yang berbeda.Sedangkan baris kedua dan ketiga merupakan perintah *loose reverse path* pada kedua *interface* router yang digunakan sebagai jalur untuk melakukan *multi-path routing*. Fungsi *loose reverse path* adalah mengizinkan router untuk tetap mengirimkan paket datapada saat paket datadari *destination host* menuju *source host* melewati jalur yang berbeda dengan paket datadari *source host* menuju *destination host.*

```
net.ipv4.ip_forward=1

net.ipv4.conf.enp0s16.rp_filter=2

net.ipv4.conf.enp0s17.rp_filter=2
```

Gambar4.4 Konfigurasi IP *forwarding* pada router

Kemudian, untuk mendistribusikanpaket datamelalui*multiple paths*, maka dilakukan konfigurasi pada TEQL. Konfigurasi yang pertama adalah mengaktifkan modul TEQL pada Linux Kernel 4.15 dengan cara menambahkan perintah `sch_teql`pada `/etc/modules`. Setelah mengaktifkan TEQL, maka konfigurasi selanjutnya adalah menjadikan TEQL sebagai mekanisme distribusipaket datapada *multi-path routing*. Gambar 4.5 menunjukkan perintah router 1 untuk mengirimkanpaket datadari *source host* menuju *destination host* melalui TEQL, sedangkan Gambar 4.6 menunjukkan perintah router 2 untuk mengirimkan ACK dari *destination* menuju *source host* melalui TEQL. Semua perintah pada router 1 dan router 2 ditambahkan pada `/etc/rc.local`, sehingga semua perintah secara otomatis dijalankan setiap router dihidupkan. Pada Gambar 4.5 dan 4.6, baris pertama dan kedua merupakan perintah untuk menggunakan TEQL sebagai mekanisme distribusipaket datapada kedua *interfacemultiple paths*. Sedangkan baris ketiga merupakan perintah untuk memberikan alamat IP *virtual* pada TEQL. Baris keempat merupakan perintah untuk menghidupkan TEQL. Terakhir, baris kelima dan keenam merupakan perintah untuk mengirimkan paket datamenuju alamat IP virtualTEQL, jika router 1 menerima paket datadengan tujuan *destination host*, dan jika router 2 menerima ACKdengan tujuan *source host*.

```
sudo tc qdisc add dev enp0s16 root teql0
sudo tc qdisc add dev enp0s17 root teql0
sudo ip address add 172.16.7.1/24 dev teql0
sudo ip link set teql0 up
sudo ip route add 172.16.5.0/24 via 172.16.7.2 dev teql0
sudo ip route add 172.16.6.0/24 via 172.16.7.2 dev teql0
```

Gambar4.5Konfigurasi *multi-path routing* pada router 1

```
sudo tc qdisc add dev enp0s16 root teql0
sudo tc qdisc add dev enp0s17 root teql0
sudo ip address add 172.16.7.2/24 dev teql0
sudo ip link set teql0 up
sudo ip route add 172.16.1.0/24 via 172.16.7.1 dev teql0
sudo ip route add 172.16.2.0/24 via 172.16.7.1 dev teql0
```

Gambar4.6Konfigurasi *multi-path routing* pada router 2

Kemudian, pada router 1 dan 2 juga dilakukan konfigurasi *single path routing* untuk melakukan evaluasi perbandingan antara *single path routing* dengan *multi-*

path routing. Gambar 4.7 menunjukkan perintah konfigurasi *single path routing* pada router 1. Perintah pada baris pertama dan kedua merupakan perintah untuk mengirimkan paket datamenuju router 2, jika router 1 menerima paket datadengan tujuan *destination host* 1 dan *destination host* 2. Selanjutnya, Gambar 4.8 menunjukkan perintah konfigurasi *single path routing* pada router 2. Perintah pada baris pertama dan kedua merupakan perintah untuk mengirimkan paket datamenuju router 1, jika router 2 menerima paket datadengan tujuan *source host* 1 dan *source host* 2.

```
up route add -net 172.16.4.0/24 gw 172.16.3.2 dev enp0s16
up route add -net 172.16.5.0/24 gw 172.16.3.2 dev enp0s16
```

Gambar4.7Konfigurasi *single path routing* pada router 1

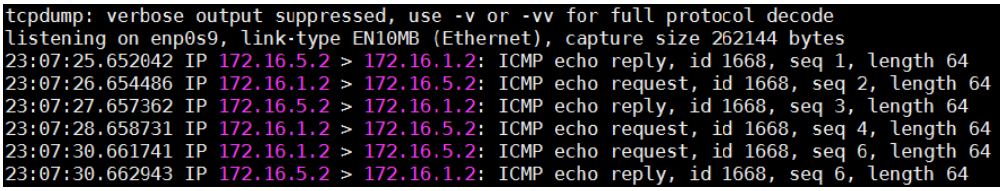
```
up route add -net 172.16.1.0/24 gw 172.16.3.1 dev enp0s16
up route add -net 172.16.2.0/24 gw 172.16.3.1 dev enp0s16
```

Gambar 4.8 Konfigurasi *single path routing* pada router 2

Setelah semua konfigurasi router selesai dilakukan, maka dilakukan pengujian ping dari *source host* 1 menuju *destination host* 1 untuk menguji mekanisme pengiriman paket datamelalui *multiple paths* berhasil dilakukan. Gambar 4.9 dan Gambar 4.10 menunjukkan hasil pengujianping pada jalur pertama dan kedua. Berdasarkan Gambar 4.9 dan Gambar 4.10, ICMP *echo request* dengan *sequence number* ganjil dikirimkan melalui jalur pertama dan ICMP *echo request* dengan *sequence number* genap dikirimkan melalui jalur kedua. Kemudian, pada ICMP *echo reply*, tidak semua ICMP *echo reply* dikirimkan berdasarkan*round-robin*. Namun, ICMP *echo reply* pada jalur pertama dan kedua memiliki jumlah yang sama.

```
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on enp0s9, link-type EN10MB (Ethernet), capture size 262144 bytes
23:07:25.580156 IP 172.16.1.2 > 172.16.5.2: ICMP echo request, id 1668, seq 1, length 64
23:07:26.584777 IP 172.16.5.2 > 172.16.1.2: ICMP echo reply, id 1668, seq 2, length 64
23:07:27.584508 IP 172.16.1.2 > 172.16.5.2: ICMP echo request, id 1668, seq 3, length 64
23:07:28.588670 IP 172.16.5.2 > 172.16.1.2: ICMP echo reply, id 1668, seq 4, length 64
23:07:29.589002 IP 172.16.1.2 > 172.16.5.2: ICMP echo request, id 1668, seq 5, length 64
23:07:29.590271 IP 172.16.5.2 > 172.16.1.2: ICMP echo reply, id 1668, seq 5, length 64
```

Gambar4.9Hasil uji ping jalur pertama



Gambar4.10Hasil uji ping jalur kedua

4.1.3 Konfigurasi *Host*

Konfigurasi *host* pertama yang dilakukan adalah mengaktifkan algoritme TCP *congestion control* Reno, BIC, CUBIC, dan BBR pada *source host*. Algoritme TCP *congestion control default* pada Linux Kernel 4.15 adalah CUBIC, sedangkan Reno, BIC, dan BBR merupakan algoritme TCP *congestion control* opsional.Algoritme TCP *congestion control* Reno dapat diaktifkan melalui perintah `sysctl -w net.ipv4.tcp_congestion_control=Reno`. Sedangkan algoritme TCP *congestion control* BIC dan BBR dapat diaktifkan melalui perintah yang samadengan Reno dan mengganti Reno dengan BIC atau BBR. Khusus pada BBR, *fair queuing* dapat diaktifkan melalui perintah `sysctl -w net.core.default_qdisc=fq`. Selanjutnya,mekanisme SACK, DSACK, *window scaling option*, *reordering heuristic*, dan *timestamp* secara *default* telah aktif pada Linux Kernel 4.15.

4.2 Konfigurasi Perangkat Lunak untuk Evaluasi

Perangkat lunak yang digunakan untuk evaluasi algoritme TCP *congestion control* Reno, BIC, CUBIC, dan BBR pada *multi-path routing* adalah iPerf3 dan Tcpdump. IPerf3 berfungsisebagai mekanisme pengirimanTCP dan UDPpaket data, sedangkanTcpdump sebagai mekanismeuntuk *capture* TCP paket data. Konfigurasi yang dilakukan pada iPerf3 dan Tcpdump adalah sebagai berikut.

1. iPerf3

Pada iPerf3, *source host*merupakaniPerf3 *client,* dan *destination host* merupakaniPerf3 server. Pengiriman TCP paket datapada*source host*dapat dilakukan melalui perintah `iperf3 -c 172.16.5.2 -t 100`. Perintah `-c` merupakan perintah iPerf3 untuk menjalankan *source host* sebagai *client*. Kemudian, perintah `172.16.5.2` merupakan perintah iPerf3 untuk mengirimkanpaket data TCP menuju alamat IP 172.16.5.2 yang merupakan alamat IP *destination host*. Terakhir, perintah `-t 100` merupakan perintah iPerf3 untuk mengirimkan TCP paket dataselama 100 detik. Selanjutnya, *destination host* yang merupakan tujuan pengiriman TCP paket datadari iPerf3*client* menjalankan perintah `iperf3 -s`. Perintah `-s` merupakan perintah iPerf3 untuk menjalankan *destination host* sebagai server.

Kemudian, pengirimanpaket data UDP pada*source host*dapat dilakukan melalui perintah `iperf3 -c 172.16.6.2 -u -b 10M -t 100`. Perintah –u merupakan perintah iPerf3 untuk mengirimkanpaket data UDP. Kemudian,

perintah `-b 10M`merupakan perintah iPerf3 untuk mengirimkanpaket data UDPdengan CBR 10 Mbps. Selanjutnya, *destination host* yang merupakan tujuan paket data UDP dari iPerf3 *client* menjalankan perintah `iperf3 -s`yang merupakan perintah iPerf3 untuk menjalankan *destination host* sebagai server.

2. Tcpdump

Pada penelitian ini, *capture*paket data TCP dilakukan pada router 1, sehingga Tcpdump diaktifkan pada router 1. *Capture* TCP paket datadengan Tcpdump dilakukan melalui perintah `tcpdump -i enp0s9 -w tesis.pcap -B 1800000`. Perintah `-i enp0s9` merupakan perintah pada Tcpdump untuk melakukan *capture* semuadata TCPyang masuk dan keluar dari *interface*router 1 yang terhubung dengan *source host*. Perintah `-w tesis.pcap` merupakan perintah Tcpdump untuk menyimpan semua hasil *capture*paket data TCP dengan nama*file* tesis dan ekstensi *file* *.pcap. Ekstensi *file* *.pcap merupakan salah satujenis ekstensi *file* yangmendukung untuk dibuka pada Wireshark. Kemudian, perintah `-B 1800000` merupakan perintah Tcpdump untuk menyediakan *buffer* sebesar 1800000 KB. *Buffer* yang digunakan pada penelitian ini berukuran besar supaya tidak ada paket data TCPyang gagal dilakukan *capture*selama evaluasi.

BAB 5 HASIL DAN PEMBAHASAN

Pada bab hasil dan pembahasan akan dijelaskan hasil evaluasi dananalisis kinerja algoritme TCP *congestion control* Reno, BIC, CUBIC, dan BBR pada *multi-path routing*. Evaluasi yang dilakukan meliputi variasi *link delay*, variasi *loss rate*, *inter* TCP *protocol fairness*, dan *fairness* antara TCP dengan UDP. Berdasarkan hasil evaluasi, kemudian dilakukan analisis untuk setiap pengukuran kinerja.

5.1 Perbandingan antara *Single Path Routing* dengan *Multi-path Routing*

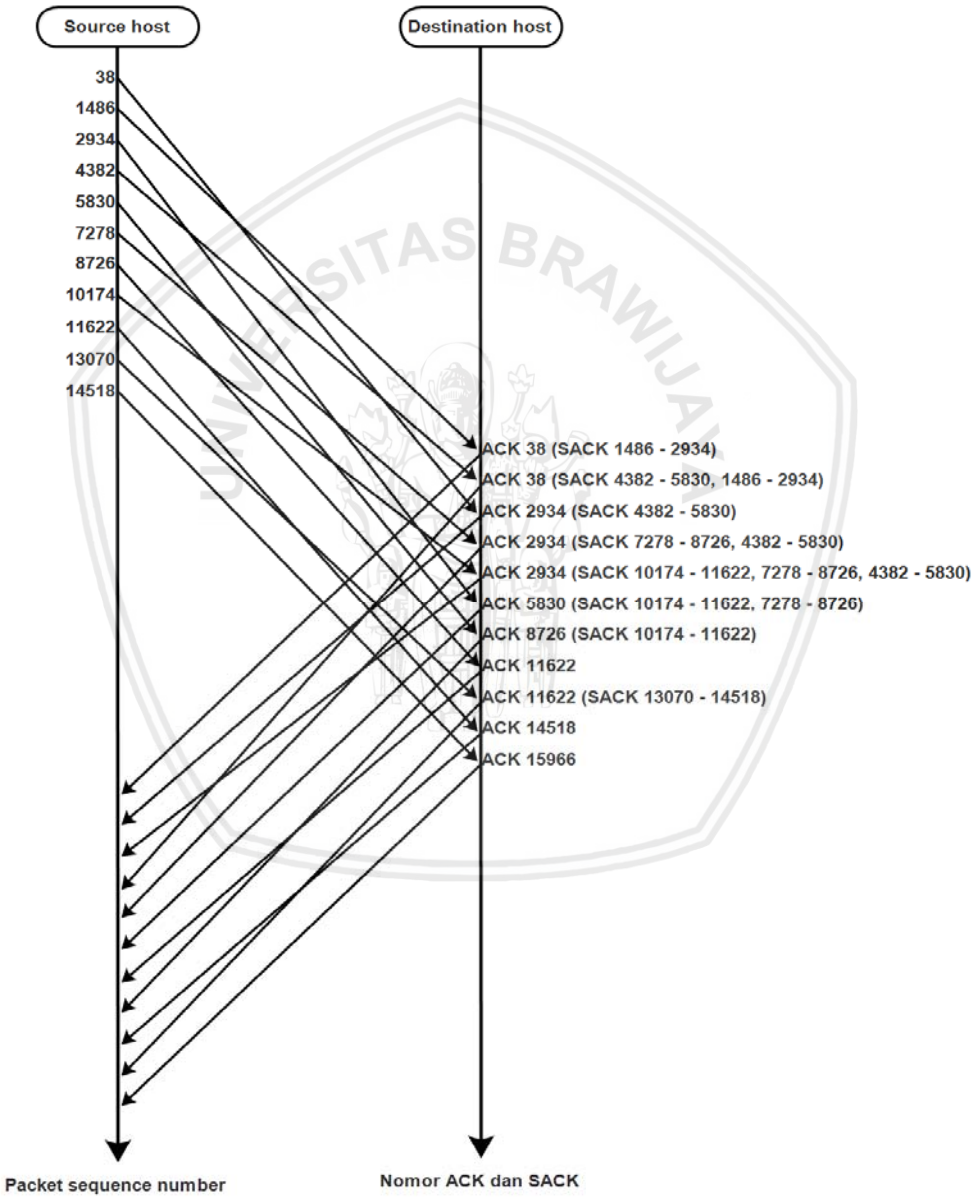
Evaluasi pertama adalah melakukanperbandingan antara *single path routing* dengan *multi-path routing* untuk mengetahui perbedaan kinerja dan perilaku Reno, BIC, CUBIC, dan BBR antara *single path routing* dengan *multi-path routing*. Tabel 5.1 menunjukkan perbandingan jumlah SACK *block* dan rata-rata *throughput* antara *single path routing* dengan *multi-path routing* pada Reno, BIC, CUBIC, dan BBR. Berdasarkan kolom jumlah SACK *block*, jumlah SACK *block* Reno, BIC, CUBIC, dan BBR pada *multi-path routing* mengalami kenaikan yang signifikan dibandingkan dengan *single path routing*. Jumlah SACK *block* Reno mengalami kenaikan 41 kali lipat, BIC mengalami kenaikan 27 kali lipat, CUBIC mengalami kenaikan 78 kali lipat, dan SACK *block* pada BBR hanya ditemukan pada *multi-path routing*, yaitu 70380.

Tabel 5.1Perbandingan antara *single path routing* dengan *multi-path routing*

TCP	*Single path routing*		*Multi-path routing*	
	Jumlah *SACK block*	**Rata-rata *throughput(Mbps)***	**Jumlah *SACK block***	**Rata-rata *throughput(Mbps)***
Reno	1157	9.57	56215	19.1
BIC	12089	9.57	83036	19.14
CUBIC	4541	9.57	92976	19.08
BBR	0	9.45	61399	18.69

Penyebab jumlah SACK *block* pada *multi-path routing* mengalami kenaikan yang signifikan ditunjukkan pada Gambar 5.1. Gambar 5.1 menunjukkan aliran 11paket datapertama antara *source host* dengan *destination host*pada BBR *multi-path routing*. Pada saat awalpengiriman paket data, BBR mengirimkan 11 paket datadengan *sequence number* 38 – 14518 dengan kelipatan 1448 untuk setiap*sequence number*. Namun, *destination host* BBR tidak menerima paket data dengan *sequence number*yang sesuai urutan pada saat dikirimkan oleh

source host. Seperti yang telah dijelaskan pada sub bab 2.4, pada saat *destination host* BBR mendeteksi *gap*antara paket data dengan*sequence number*yang sudah diterima sebelumnya dengan yang diterima saat ini, *destination host*BBR mengirimkan ACK dengan SACK *block* kepada *source host*. *Destination host*BBR mengirimkan ACK untuk paket data dengan*sequence number*yang diharapkan diterima oleh *destination host*. SedangkanSACK *block* berisi informasi tentangpaket data dengan *sequence number* yang sudah diterima oleh *destination host* BBR, tetapi tidak sesuai dengan urutan yang diharapkan oleh *destination host* BBR.



Gambar 5.1Aliran paketdata BBR pada *multi-path routing*

Pada aliran paket datayang pertama, *destination host* BBR mengharapkan paket datayang diterima adalah dengan *sequence number* 38. Namun,*destination host*BBR menerima paket datadengan *sequence number* 1486. Oleh karena itu, *destination hos*t BBR mengirimkan ACK38 dengan satu SACK *block*, yaitu 1486 – 2934. ACK dan SACK *block* yang dikirimkan oleh *destination host* BBR menunjukkan bahwa*destination host* BBR mendeteksi 1 *gap*untukpaket datayang diterima.

Kemudian, *destination host*BBR menerima paket datadengan *sequence number* 4386. Oleh karena itu, *destination host*BBR kembali mengirimkan ACK 38 dengan SACK *block*. Namun,saat ini *destination host*BBR mengirimkan 2 SACK *block*, yaitu 1486 – 2934 dan 4382 – 5830karena*destination host*BBR mendeteksi 2 *gap*padapaket datayang diterima. *Gap* yang pertama adalah *destination host*BBR mengharapkan menerima paket datadengan *sequence number* 38, tetapi *destination host*BBR menerima paket datadengan *sequence number* 1486. Sedangkan *gap* yang kedua adalah setelah *sequence number* 1486, paketdata selanjutnya yang diharapkan diterima oleh *destination host*BBR adalah dengan *sequence number* 2934. Namun, *destination host* BBR menerima paket datadengan *sequence number* 4382.

Setelah itu, *destination host*BBR menerima paket datadengan *sequence number* yang diharapkan, yaitu 38. Pada saat *destination host* BBR menerima paket data dengan *sequence number* 38, *destination host*BBR mengirimkan ACK 2934yang artinya adalah *destination host* BBR sudah menerima semua paket datadengan *sequence number* 38 – 1486. Paket dataselanjutnya yang diharapkan diterima oleh *destination host* BBR adalah paket datadengan *sequence number* 2934. Oleh karena itu, *destination host* BBR menghapus SACK *block* 1486 – 2934. Namun, *destination host* BBR tetap mengirimkan 1 SACK *block*, yaitu 4382 – 5830 karena *destination host* BBR masih mendeteksi *gap* antara paket data *sequence number* 1486 dengan 4382.

Berdasarkan informasi ACK dan SACK *block*, maka dapat diketahui bahwa paket data dengan *sequence number* 38 tidak mengalami paket *loss*, tetapi hanya datang terlambat pada *destination host* BBR, sehingga terjadi paket *reordering*. Paket *rerordering* pada*destination host* BBR juga terjadi di aliran paket dataselanjutnya sampai detik ke 100. Selain itu, paket *reordering*juga terjadi pada Reno, BIC, dan CUBIC. Oleh karena itu, SACK *block*Reno, BIC, CUBIC, dan BBR pada *multi-path routing* tidak hanya disebabkan oleh paket *loss*, melainkan juga disebabkan oleh paket *reordering.* Sedangkan SACK *block*Reno, BIC, CUBIC, dan BBR pada *single path routing* hanya disebabkan oleh paket *loss,* sehinggajumlah SACK *block* Reno, BIC, CUBIC, dan BBR pada *multi-path routing* mengalami kenaikan yang signifikan dibandingkan dengan *single path routing*.

Namun, paket *reordering*pada *multi-path routing*tidak hanya terjadi pada saat pengiriman paket datadari*source host*menuju*destination host*. Paket *reordering* juga terjadi pada saat pengiriman ACK dari*destination*

*host*menuju*source host*. Berdasarkan Gambar 5.1, *source host* tidak menerima ACK sesuai dengan urutan yang dikirimkan oleh *destination host*. Fenomena paket *reordering* pada saat pengiriman paket datadan ACK menunjukkan bahwa paket *reordering*tetap dapat terjadi pada *multi-path routing*, walaupun *cost* pada *multiple paths* sama dan TEQL mendistribusikan paket datasecara merata berdasarkan *round-robin*. Namun, penelitian ini tidak membahas lebih lanjut penyebab terjadinyapaket *reordering*.

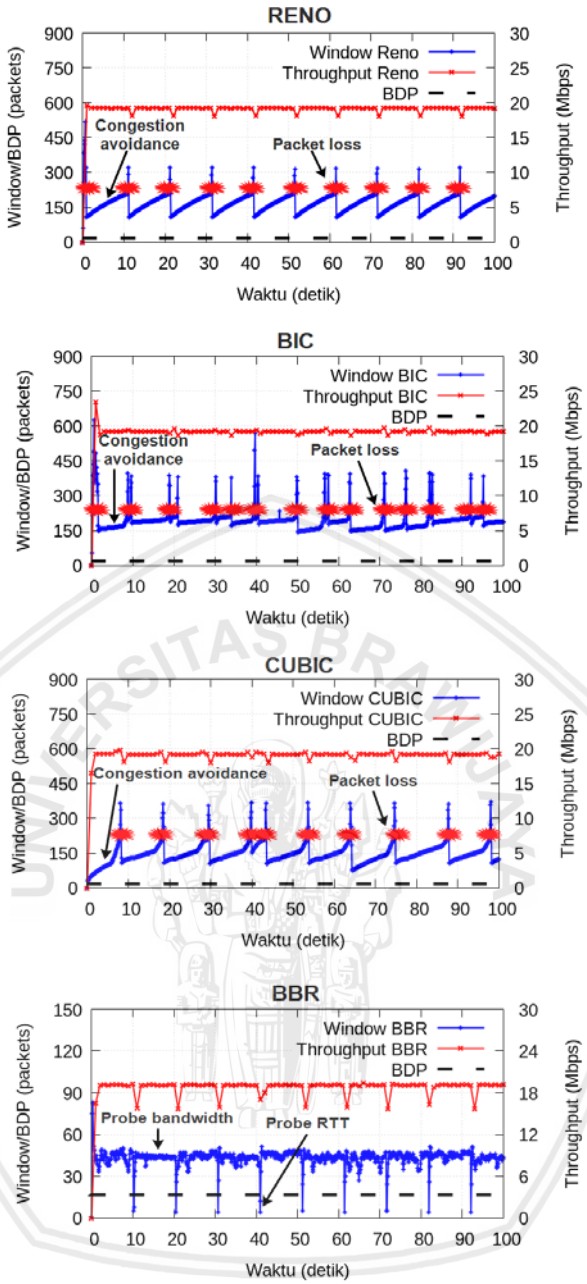
Kemudian, berdasarkan kolom rata-rata *throughput* pada Tabel 5.1, rata-rata *throughput* Reno, BIC, CUBIC, dan BBR pada *multi-path routing* mengalami kenaikan hampir 2 kali lipat dibandingkan dengan *single path routing*. Fenomena rata-rata *throughput* pada *multi-path routing* menunjukkan bahwa Reno, BIC, CUBIC, dan BBR dapat memaksimalkan 95% *bandwidth* yang tersedia pada *multiple paths*, walaupun terjadi paket *reordering*. Berdasarkanrata-rata *throughput* pada *multi-path routing*, maka dilakukan observasi perubahan *window* Reno, BIC, CUBIC, dan BBR untuk mengetahui penyebab rata-rata *throughput*yang didapatkan oleh Reno, BIC, CUBIC, dan BBR pada *multi-path routing*.

Gambar 5.2 menunjukkan perubahan *window* Reno, BIC, CUBIC, dan BBR terhadap BDP dan *throughput* selama 100 detik. Seperti yang telah dijelaskan pada sub bab 2.2, *window* merupakan representasi dari jumlah maksimumpaket datayang dapat dikirimkan oleh algoritme TCP *congestion control* sebelum menerima ACK dari *destination host* untuk paket datatersebut. Sedangkan BDP merupakan jumlah paket datayang harus dikirimkan oleh algoritme TCP *congestion control*untuk dapat mencapai kapasitas maksimum *link* jaringan pada *multiple paths*. Sebagian besar mekanisme Reno, BIC, dan CUBIC selama 100 detik adalah melakukan fase *congestion avoidance*. Walaupun mekanisme fase *congestion avoidance* Reno, BIC, dan CUBIC berbeda, tetapi Reno, BIC, dan CUBIC memiliki kesamaan dalam melakukan fase *congestion avoidance*, yaitu menaikkan ukuran *window* sampai terjadi paket *loss*. Pada fase *congestion avoidance*, Reno, BIC, dan CUBIC menaikkan ukuran *window* untuk mencapai BDP dalam waktu yang singkat. Apabilaukuran *window*sudah mencapai BDP, Reno, BIC, dan CUBIC terus menaikkan ukuran *window*, sehingga paket dataReno, BIC, dan CUBIC mulai mengisi ke dalam *buffer switch*. Reno, BIC, dan CUBIC terus menaikkan ukuran *window* sampai terjadi paket *loss* yang disebabkan oleh *buffer* switch terisi penuh.

Berdasarkan Gambar 5.2, Reno, BIC, dan CUBIC memulai fase *congestion avoidance*sekitar detik ke 2. Reno, BIC, dan CUBICterus menaikkan ukuran *window* sampai terjadi paket *loss* sekitar detik ke 10. Namun, *throughput* Reno, BIC, dan CUBIC pada detik ke 2 – 10 cenderung stabil sekitar 19 Mbps. Penyebabnya adalah pada saat Reno, BIC, dan CUBIC memulaifase *congestion avoidance*, ukuran *window* Reno, BIC, dan CUBIC sudah mencapai BDP. Oleh karena itu, pada detik ke 2 – 10, Reno, BIC, dan CUBIC sudah menggunakan seluruh kapasitas *bandwidth* pada *multiple paths*, sehingga pada saat Reno, BIC,

dan CUBIC menaikkan ukuran window, paket dataReno, BIC, dan CUBIC masuk ke dalam *buffer* switch.

Kemudian, pada saat terjadi paket *loss* sekitar detik ke 10, Reno, BIC, dan CUBIC mendeteksi *congestion* pada jaringan. Oleh karena itu, Reno, BIC, dan CUBIC menurunkan ukuran *window* untuk mencegah *congestion* terjadi secara terus-menerus.Pada saat Reno, BIC, dan CUBIC menurunkan ukuran *window*, *throughput* Reno, BIC, dan CUBIC sedikit mengalami penurunan, yaitu sekitar 1Mbps. Kemudian, Reno, BIC, dan CUBIC melakukan *retransmission*, yaitu mengirimkan kembali paket datayang mengalami paket *loss*.



Gambar5.2Perilaku Reno, BIC, CUBIC, dan BBR pada *multi-path routing*

SetelahReno, BIC, dan CUBIC selesai melakukan *retransmission*, Reno, BIC, dan CUBIC kembali melakukan fase *congestion avoidance*. Namun,ukuran *window*Reno, BIC, dan CUBICtetap lebih tinggi dibandingkan dengan BDP.Oleh karena itu, Reno, BIC, dan CUBIC hanya perlu menaikkanukuran *window* sedikit untuk kembali mencapai *throughput* 19 Mbps. Siklus perubahan *window* Reno, BIC, dan CUBIC terhadap *throughput*pada fase *congestion avoidance*terjadi secara terus-menerus sampai detik ke 100. Oleh karena itu, rata-rata *throughput*

Reno, BIC, dan CUBIC pada *multi-path routing*dapat mencapai 19 Mbps, walaupun terjadi paket *reordering*. Reno, BIC, dan CUBIC tidak sampai mendeteksi paket *reordering* sebagai paket *loss*, sehingga Reno, BIC, dan CUBIC tidak menurunkan ukuran *window* pada saat terjadi paket *reordering*.

Kebalikan dari Reno, BIC, dan CUBIC yang menaikkan ukuran *window* sampai *buffer* switch terisi penuh, BBR menjaga ukuran *window*berdasarkan estimasi BDP. BBR melakukan estimasi BDP berdasarkan estimasi *bottleneck bandwidth*maksimum dan estimasi RTT minimum pada *multiple paths*. BBR melakukan *probe bandwidth* untuk melakukan estimasi *bottleneck bandwidth* maksimum,dan *probe* RTT untuk melakukan estimasi RTT minimum. BBR secara berkala melakukan *probe bandwidth* dan *probe* RTT untuk tetap menjaga ukuran *window* sebesar estimasi BDP.

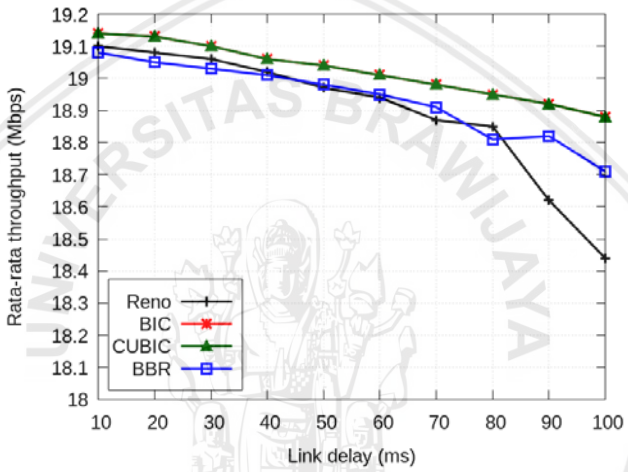
Berdasarkan Gambar 5.2, sebagian besar mekanisme BBR adalah melakukan *probe bandwidth* untuk mencari perubahan pada *bottleneck bandwidth*. BBR melakukan *probe bandwidth* sekitar detik ke 2 – 100. Pada sekitar detik ke 2, BBR sudah mendapatkan estimasi BDP dari *state startup* yang dilakukan pada detik ke 1. BBR melakukan *probe bandwidth* dengan cara menaikkan ukuran *window* dari estimasi BDP yang didapatkan sebelumnya. Namun, *bottleneck bandwidth*yang digunakan pada penelitian ini bersifat tetap selama 100 detik. Oleh karena itu, pada saat BBR menaikkan ukuran *window*, paket dataBBR mulai mengisi ke dalam *buffer* switch karena BBR sudah menggunakan seluruh kapasitas *bandwidth* pada *multiple paths*. Akibatnya adalah ukuran *window* BBR pada saat melakukan *probe bandwidth*selama 100 detik selalulebih tinggi dibandingkan dengan BDP, sehingga *throughput* BBR tetap stabil sekitar 19 Mbps.

Kemudian, BBR melakukan *probe* RTT setiap 10 detik untuk melakukan estimasi RTT minimum. BBR melakukan *probe* RTT dengan cara membatasi pengiriman paket datasebesar 4 paket data. Oleh karena itu,ukuran *window* BBR setiap kelipatan 10 detik mengalami penurunan yang drastis. Akibatnya adalah*throughput* BBR pada saat melakukan probe RTT mengalami penurunan. Penurunan *throughput* BBR pada saat *probe* RTT lebih drastis dibandingkan dengan penurunan *throughput* Reno, BIC, dan CUBIC pada saat Reno, BIC, dan CUBIC menurunkan ukuran *window* akibat paket *loss*. Oleh karena itu, rata-rata *throughput* BBR padaTabel 5.1 sedikit lebih rendah dibandingkan dengan Reno, BIC, dan CUBIC.

Pengiriman paket databerdasarkan estimasi BDP mengakibatkan BBR tidak mengisi paket datake dalam *buffe*r switch sampai penuh yang dapat mengakibatkan paket *loss*. Oleh karena itu, pada Gambar 5.2 tidak ditemukan paket *loss* pada BBR, sehingga paket *reordering* yang terjadi pada *multi-path routing* tidak mengakibatkan BBR mendeteksi paket *reordering* sebagai paket *loss*. Berdasarkan fenomena rata-rata *throughput* Reno, BIC, CUBIC, dan BBR pada *multi-path routing*, maka dilakukan observasi lebih lanjut kinerja Reno, BIC, CUBIC, dan BBRpada *multi-path routing* untuk evaluasi variasi *link delay*, variasi *loss rate*, *inter* TCP *protocol fairness*, dan *fairness* antara TCP dengan UDP.

5.2 Variasi *Link Delay*

Evaluasi kedua adalah melakukan variasi *link delay* 10 ms – 100 ms pada *multiple paths*.Gambar 5.3 menunjukkan perbandingan rata-rata *throughput*antara Reno, BIC, CUBIC, dan BBR terhadap 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 sedikit penurunan. Namun,rata-rata *throughput* Reno, BIC, CUBIC, dan BBR pada variasi *link delay* 10 ms – 100 ms tetap tinggi, yaitu lebih dari 18 Mbps dari total *bandwidth* 20 Mbps. Fenomena rata-rata *throughput* Reno, BIC, CUBIC, dan BBR pada variasi *link delay* 10 ms – 100 ms menunjukkan bahwa Reno, BIC, CUBIC, dan BBR dapat memaksimalkan lebih dari 90% *bandwidth* yang tersedia pada *multiple paths,* walaupun *link delay* semakin tinggi hingga 100 ms.

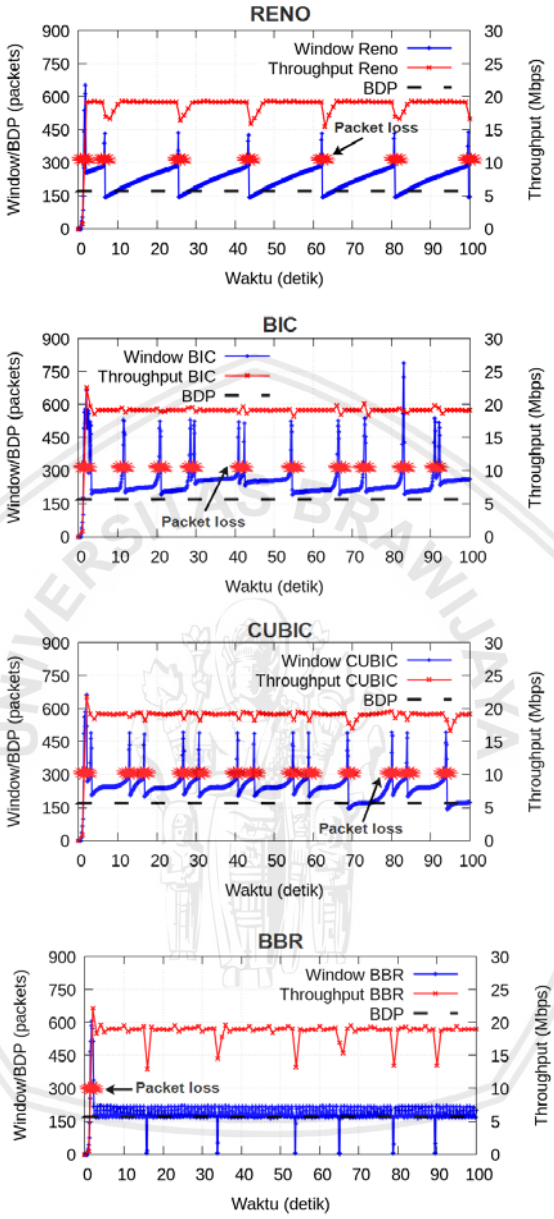


Gambar5.3Perbandingan rata-rata *throughput* terhadap variasi *link delay*

Penyebab rata-rata *throughput* Reno, BIC, dan CUBIC tetap tinggi hingga *link delay*100 ms ditunjukkan pada Gambar 5.4. Gambar 5.4 menujukkan perubahan *window* Reno, BIC, CUBIC, dan BBR terhadap *throughput* dan BDP pada variasi *link delay* 100 ms. Apabila dibandingkan dengan Gambar 5.2, BDP pada *link delay* 100 ms lebih tinggi dibandingkan dengan BDP pada *link delay* 10 ms karena BDP dihitung berdasarkan perkalian antara *bottleneck bandwidth* dengan variasi*link delay*.

Pada *link delay* 100 ms, sebagian besar ukuran *window* Reno, BIC, dan CUBIC pada fase *congestion avoidance* dari detik ke 2 – 100 tetap lebih tinggi dibandingkan dengan BDP. Walaupun ukuran *window* Reno dan CUBIC pada fase *congestion avoidance* terkadang lebih rendah dibandingkan dengan BDP, tetapiReno dan CUBIC hanya membutuhkan waktu yang singkat untuk menaikkan ukuran *window* sampai BDP. Penyebabnya adalah selisih jarak antara BDP dengan *window*Reno dan CUBIC kecil. Oleh karena itu, *throughput* Reno,

BIC, dan CUBIC pada fase *congestion avoidance* tetap dapat mencapai sekitar 19 Mbps, walaupun *link delay* semakin tinggi hingga 100 ms.



Gambar5.4Perilaku Reno, BIC, CUBIC, dan BBR terhadap *link delay* 100 ms

Kemudian, pada saat Reno, BIC, dan CUBIC mengalami paket *loss* yang disebabkan oleh *buffer* switch terisi penuh, Reno, BIC, dan CUBIC menurunkan ukuran *window*. Namun, pada saat Reno, BIC, dan CUBIC menurunkan ukuran *window*, *throughput* Reno, BIC, dan CUBIC pada *link delay* 100 ms mengalami penurunan yang lebih drastis dibandingkan dengan *link delay* 10 ms pada

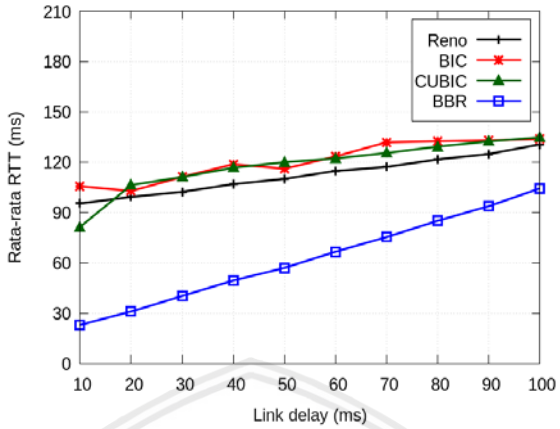
Gambar 5.2. Penyebabnya adalah ukuran BDP semakin tinggi seiring bertambah tingginya *link delay*, sehingga ukuran maksimum *window* yang dapat dicapai Reno, BIC, dan CUBIC sampai terjadipaket *loss* pada *link delay* 100 ms lebih tinggi dibandingkan dengan *link delay* 10 ms. Oleh karena itu, pada saat terjadi paket *loss*, Reno, BIC, dan CUBIC pada *link delay* 100 ms menurunkan ukuran *window* lebih drastis dibandingkan dengan *link delay* 10 ms. Akibatnya adalahrata-rata *throughput* Reno, BIC, dan CUBIC pada Gambar 5.3 terus mengalami sedikit penurunan seiring bertambah tingginya *link delay*. Namun, Reno mengalami penurunan *throughput* yang lebih drastis dibandingkan dengan BIC dan CUBIC karena Reno menurunkan ukuran *window* lebih drastis dibandingkan dengan BIC dan CUBIC pada saat terjadi paket *loss*. Oleh karena itu, rata-rata *throughput* Reno pada Gambar 5.3 lebih rendah dibandingkan dengan BIC dan CUBIC untuk semua variasi *link delay*.

Sementara itu, berdasarkan Gambar 5.4, ukuran *window* BBR pada *link delay* 100 ms tetap dapat mencapai BDP pada saat BBR melakukan*probe bandwidth*dari detik ke 2 – 100. Penyebabnya adalah BBR menyesuaikan estimasi RTT minimum berdasarkan*link delay*pada *multiple paths* untuk menentukan ukuran *window*. Oleh karena itu, *throughput* BBR pada saat melakukan *probe bandwidth* selama 100 detik tetap mencapai 19 Mbps, walaupun *link delay* semakin tinggi hingga 100 ms.

Namun,*throughput* BBR pada saat melakukan*probe* RTT pada *link delay* 100 ms mengalami penurunan yang lebih drastis dibandingkan dengan *link delay* 10 ms pada Gambar 5.2. Penyebabnya adalah pada saat BBR melakukan *probe* RTT, *destination host* BBR pada *link delay* 100 ms menerima paket datalebih lama dibandingkan dengan pada saat *link delay* 10 ms. Oleh karena itu, rata-rata *throughput* BBR pada Gambar 5.3 terus mengalami sedikit penurunan seiring bertambah tingginya *link delay*.

Selanjutnya, walaupun Reno, BIC, CUBIC, dan BBRmendapatkan rata-rata *throughput* yang tinggi untuk semua variasi *link delay*, tetapi terdapat perbedaan signifikan pada rata-rata RTT Reno, BIC, CUBIC, dan BBR. Gambar 5.5 menunjukkan perbandingan rata-rata RTT antara Reno, BIC, CUBIC, dan BBR terhadap variasi *link delay* 10 ms – 100 ms. Pada variasi *link delay* 10 ms – 100 ms, rata-rata RTT Reno, BIC, CUBIC, dan BBR lebih tinggi dibandingkan dengan *link delay*. Namun, rata-rata RTT BBR pada variasi *link delay* 10 ms – 100 ms lebih rendah sekitar 26 ms – 58 ms dibandingkan dengan Reno, BIC, dan CUBIC. Fenomena rata-rata RTT Reno, BIC, CUBIC, dan BBR yang lebih tinggi dibandingkan dengan variasi*link delay* menunjukkan bahwa Reno, BIC, CUBIC, dan BBR cenderung mengirimkan paket datalebih besar dibandingkan dengan kapasitas maksimum *link*jaringan, sehingga paket dataReno, BIC, CUBIC, dan BBR masukke dalam *buffer* switch. Seperti yang telah dijelaskan pada sub bab 2.4, pengisian paket datake dalam *buffer* switch dapatmenyebabkan*queuing delay*. *Queuing delay*dapat berdampak *destination host* menerima paket datalebih lama dibandingkan dengan pada saat pengirimanpaket datatanpa *queuing delay*. Namun,*destination host* BBRselama 100 detik menerima paket datahingga 58 ms

lebih cepat dibandingkan dengan Reno, BIC, dan CUBIC karena BBR mengisi paket datake dalam *buffer* switch lebih sedikit dibandingkan dengan Reno, BIC, dan CUBIC.



Gambar5.5Perbandingan rata-rata RTT terhadap variasi *link delay*

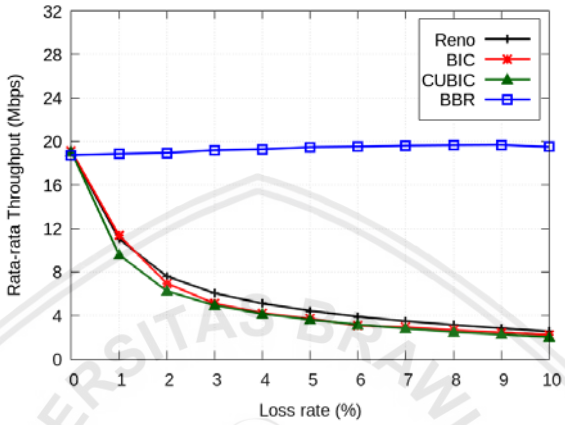
Rata-rata RTT Reno, BIC, dan CUBIC yang lebih tinggi dibandingkan dengan BBR disebabkan oleh perilaku algoritme TCP *congestion control* berbasis paket *loss* yang mendeteksi paket *loss* sebagai tanda *congestion* pada jaringan. Berdasarkan Gambar 5.4, Reno, BIC, dan CUBIC terus menaikkan ukuran *window,*walaupun ukuran *window* Reno, BIC, dan CUBIC sudah mencapai BDP, sehingga paket dataReno, BIC, dan CUBIC mulai mengisike dalam *buffer* switch. Reno, BIC, dan CUBIC terus menaikkan ukuran *window* sampai *buffer* switch terisi penuh, sehingga terjadi paket *loss*.

Sementara itu, berdasarkan Gambar 5.4, ukuran *window* BBR juga lebih tinggi dibandingkan dengan BDP. Namun, BBR tidak menaikkan ukuran *window*sampai *buffer*switch terisi penuh, sehingga pada saat BBR melakukan*probebandwidth* dan *probe* RTT detik ke 2 – 100, BBR tidak mengalami paket *loss*. Oleh karena itu, BBR mengisi paket datake dalam *buffer* switch lebih sedikit dibandingkan dengan Reno, BIC, dan CUBIC, sehingga rata-rata RTT BBR lebih rendah dibandingkan dengan Reno, BIC, dan CUBIC.

5.3 Variasi *Loss Rate*

Evaluasi ketigaadalah melakukanvariasi *loss rate* 1% – 10% pada *multiple paths*. Masing-masing jalur menggunakan variasi *loss rate* yang sama. Gambar 5.6 menunjukkan perbandingan rata-rata *throughput* antara Reno, BIC, CUBIC, dan BBR terhadap variasi *loss rate* 1% – 10%. Pada variasi *loss rate* 1% – 10%, rata-rata *throughput* Reno, BIC, dan CUBIC terus mengalami penurunan sekitar 7 Mbps – 17 Mbps dari rata-rata *throughput* 19 Mbps pada saat tanpa *loss rate*. Sebaliknya, rata-rata *throughput* BBR pada variasi *loss rate* 1% – 10% tetap stabil sekitar 19 Mbps. Fenomena rata-rata *throughput* Reno, BIC, CUBIC, dan BBR pada variasi *loss rate* 1% – 10% menunjukkan bahwa BBR tetap dapat

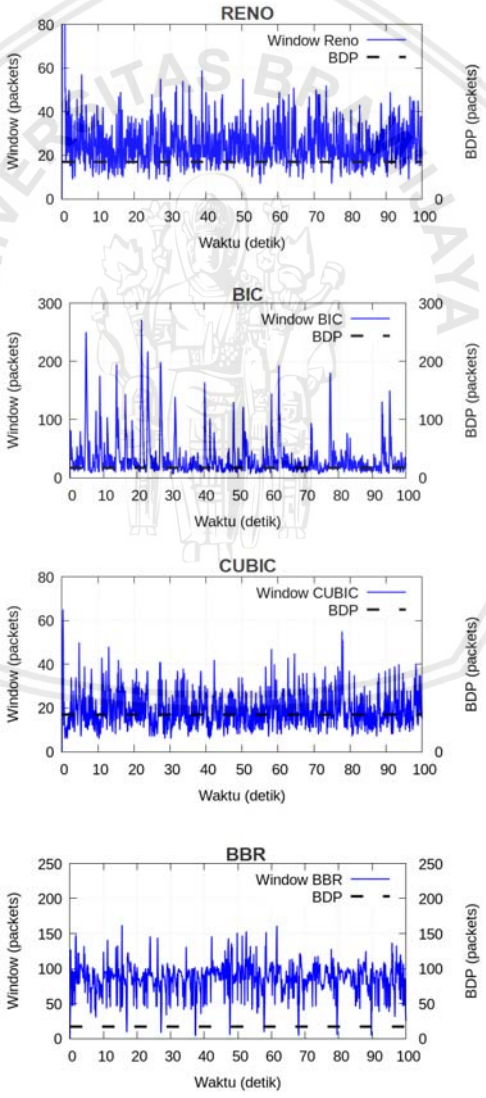
memaksimalkan sekitar 95% *bandwidth* yang tersedia pada *multiple paths,* walaupun *loss rate* semakin tinggi hingga 10%. Sebaliknya, Reno, BIC, dan CUBIC hanya dapat memaksimalkan kurang dari 55%*bandwidth* yang tersedia pada *multiple paths* pada saat terjadi *loss rate*. Pola rata-rata *throughput* BBR pada variasi *loss rate* 1% – 10% mengonfirmasi penelitian Cardwell, N., et al., (2016a) yang dilakukan pada *single path routing*. Pada penelitian Cardwell, N., et al., (2016a), BBR tetap mendapatkan rata-rata *throughput* yang tinggi pada *loss rate* 10%.



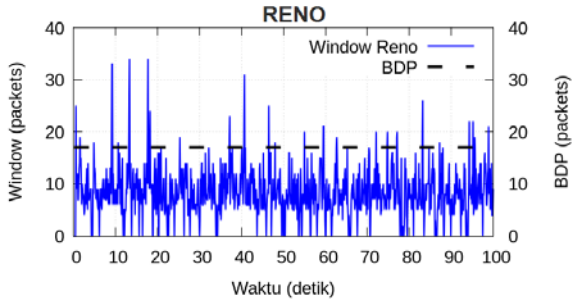
Gambar5.6Perbandingan rata-rata *throughput* terhadap variasi *loss rate*

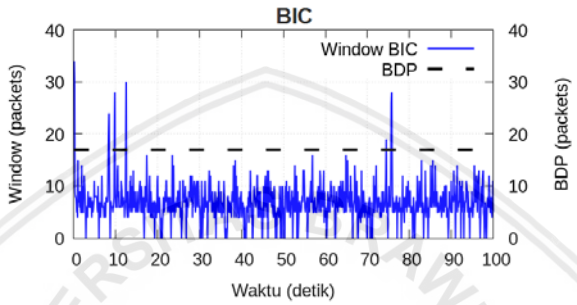
Rata-rata *throughput* Reno, BIC, dan CUBIC yang terus mengalami penurunan seiring bertambah tingginya *loss rate* disebabkan oleh Reno, BIC, dan CUBIC menurunkan ukuran *window* setiap terjadipaket *loss*. Gambar 5.7menunjukkan perubahan *window* Reno, BIC, CUBIC, dan BBR pada *loss rate* 1%. Pada *loss rate* 1%, Reno, BIC, dan CUBIC menaikkan ukuran *window* sampai BDP. Reno, BIC, dan CUBIC terus menaikkan ukuran *window*, sehinggapaket dataReno, BIC, dan CUBIC mulai mengisi ke dalam *buffer* switch. Namun, Reno, BIC, dan CUBIC mengalami paket *loss* yang disebabkan oleh *loss rate* sebelum *buffer* switch terisi penuh. Pada saat terjadi paket *loss*, Reno, BIC, dan CUBIC menurunkan ukuran *window* karena Reno, BIC, dan CUBIC mendeteksi paket *loss* sebagai tanda *congestion* pada jaringan. Namun, pada saat Reno, BIC, dan CUBIC menurunkan ukuran *window*, ukuran *window* Reno, BIC, dan CUBIC lebih rendah dibandingkan dengan BDP. Oleh karena itu,pada saat Reno, BIC, dan CUBIC menurunkan ukuran *window*, Reno, BIC, dan CUBIC tidak menggunakan seluruh kapasitas *bandwidth* jaringan yang tersedia pada *multiple paths*. Kemudian, setelah Reno, BIC, dan CUBIC selesai melakukan *retransmission*, Reno, BIC, dan CUBIC kembali menaikkan ukuran *window* danmengalami paket *loss* sebelum *buffer* switch terisi penuh. Siklus perubahan *window* Reno, BIC, dan CUBIC terjadi secara terus-menerus selama 100 detik. Oleh karena itu, berdasarkan Gambar 5.6, rata-rata *throughput* Reno, BIC, dan CUBIC pada *loss rate* 1% mengalami penurunan sekitar 7 Mbps.

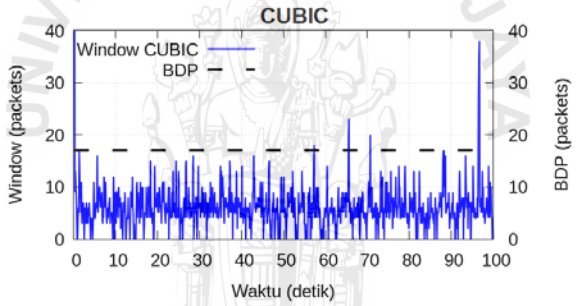
Kebalikan dari Reno, BIC, dan CUBIC, BBR tidak menggunakan paket *loss*sebagai tanda *congestion* pada jaringan. Berdasarkan Gambar 5.7, ukuran *window*BBR pada *loss rate* 1% selama 100 detik tetap lebih tinggi dibandingkan dengan BDP. Penyebabnya adalah BBR tidak menurunkan ukuran *window* sedrastis Reno, BIC, dan CUBIC pada saat terjadi paket *loss*. BBR hanya menurunkan ukuran *window* sebesar paket datayang mengalami paket *loss*(Cardwel, N., et al., 2017). Oleh karena itu, ukuran *window* BBR pada saat terjadi paket *loss*tetap lebih tinggi dibandingkan dengan BDP. Selain itu, setelah BBR selesai melakukan *retransmission*, BBR tidak menaikkan ukuran *window* secara bertahap seperti Reno, BIC, dan CUBIC. BBR langsung mengembalikan ukuran *window* ke ukuransebelum terjadi paket *loss*(Cardwel, N., et al., 2017). Mekanisme BBR yang tidak menurunkan ukuran *window* secara drastis pada saat terjadi paket *loss*, dan mekanisme BBR yang langsung mengembalikan ukuran *window* ke ukuran sebelum terjadi paket *loss,* mengakibatkan rata-rata *throughput* BBR pada *loss rate* 1% tetap stabil sekitar 19 Mbps.

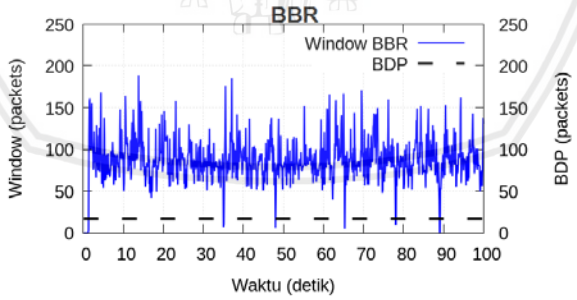


Gambar5.7Perilaku Reno, BIC, CUBIC, dan BBR terhadap *loss rate* 1%







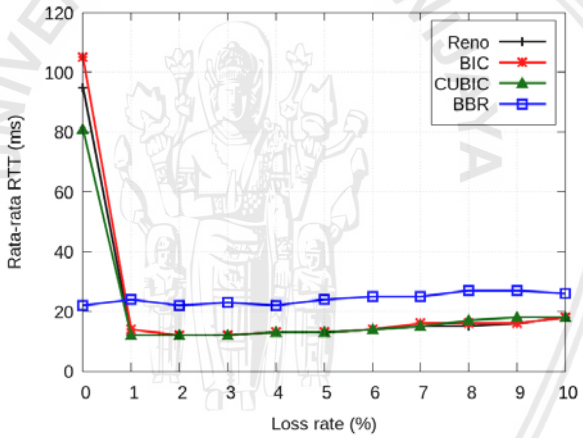


Gambar5.8Perilaku Reno, BIC, CUBIC, dan BBR terhadap *loss rate* 10%

Kemudian, semakin tinggi *loss rate*, maka semakin tinggi frekuensi Reno, BIC, CUBIC, dan BBR mengalami paket *loss*. Gambar 5.8 menunjukkan perubahan *window* Reno, BIC, CUBIC, dan BBR pada *loss rate* 10%. Pada *loss rate* 10%, rata-rata ukuran *window* Reno, BIC, dan CUBIC selama 100 detik lebih rendah

dibandingkan dengan BDP. Penyebabnya adalah pada saat Reno, BIC, dan CUBIC menaikkan ukuran *window*, Reno, BIC, dan CUBIC mengalami paket *loss* sebelum kapasitas *bandwidth* jaringan terisi penuh. Kemudian, Reno, BIC, dan CUBIC menurunkan ukuran *window* dan kembali menaikkan ukuran *window*. Namun, Reno, BIC, dan CUBIC kembali mengalami paket *loss* sebelum mencapai kapasitas maksimum *bandwidth* jaringan, sehingga Reno, BIC, dan CUBIC kembali menurunkan ukuran *window*. Siklus perubahan *window* Reno, BIC, dan CUBIC terjadi secara terus-menerus selama 100 detik. Oleh karena itu, semakin tinggi *loss rate,* semakin rendah ukuran maksimum*window* Reno, BIC, dan CUBIC, sehinggarata-rata *throughput* Reno, BIC, dan CUBIC pada Gambar 5.6 terus mengalami penurunan hingga *loss rate* 10%.

Sebaliknya, pada *loss rate* 10%, ukuran *window* BBRselama 100 detik tetap lebih tinggi dibandingkan dengan BDP. Ukuran *window* BBR yang lebih rendah dibandingkan dengan BDP disebabkan oleh BBR melakukan *probe* RTT untuk melakukan estimasi RTT minimum. Oleh karena itu, rata-rata *throughput* BBR pada Gambar 5.6 tetap stabil sekitar 19 Mbps, walaupun *loss rate* semakin tinggi hingga 10%.



Gambar5.9Perbandingan rata-rata RTT terhadap variasi *loss rate*

Selanjutnya,dilakukan pengukuran kinerja rata-rata RTT pada variasi *loss rate* 1% – 10% untuk mengetahui rata-rata waktu penerimaan paket dataReno, BIC, CUBIC, dan BBRselama 100 detik. Gambar 5.9 menunjukkan perbandingan rata-rata RTT antara Reno, BIC, CUBIC, dan BBR terhadap variasi *loss rate*1% – 10%. Pada *loss rate* 1%, rata-rata RTT Reno, BIC, dan CUBIC mengalami penurunan drastis menjadi sekitar 12 ms. Sedangkan rata-rata RTT BBRpada *loss rate* 1% mengalami kenaikan menjadi sekitar 24 ms. Fenomena rata-rata RTT Reno, BIC, CUBIC, dan BBR pada *loss rate* 1% menunjukkan bahwa *loss rate* mengakibatkan *destination host* Reno, BIC, dan CUBIC menerima paket datalebih cepat dibandingkan dengan tanpa *loss rate*. Sebaliknya, *loss rate* mengakibatkan *destination host* BBR menerima paket datalebih lama dibandingkan dengan

tanpa *lossrate*. Oleh karena itu, *destination host* Reno,BIC, dan CUBIC menerima paket datalebih cepat sekitar 12 ms dibandingkan dengan BBR. Kemudian, pada variasi *loss rate* 2% – 10%, rata-rata RTT Reno, BIC, CUBIC, dan BBR sama-sama mengalami sedikit kenaikan. Fenomena rata-rata RTT Reno, BIC, CUBIC, dan BBR pada variasi *loss rate* 2% – 10% menunjukkan bahwa semakin tinggi *loss rate*, semakin lama *destination host* Reno, BIC, CUBIC, dan BBR menerima paket data.

Rata-rata RTT Reno, BIC, dan CUBIC pada *loss rate*1% – 10% yang lebih rendah dibandingkan dengan tanpa *loss rate* disebabkan oleh Reno, BIC, dan CUBIC mengalami paket *loss* sebelum *buffer* switch terisi penuh. Berdasarkan Gambar 5.2, ukuran maksimum*window* Reno, BIC, dan CUBIC sampai *buffer* switch terisi penuh pada saat tanpa *loss rate*adalah sekitar 217 paket data. Namun, berdasarkan Gambar 5.7, rata-rata ukuran *window* Reno, BIC, dan CUBIC selama 100 *detik* pada *loss rate* 1% hanya sekitar 100 paket data. Oleh karena itu, paket dataReno, BIC, dan CUBIC yang masuk ke dalam *buffer* switch pada *loss rate* 1% lebih sedikit dibandingkan dengan tanpa *loss rate*, sehingga *queuing delay* Reno, BIC, dan CUBIC pada *loss rate* 1% lebih rendah dibandingkan dengan tanpa *loss rate*.

Sementara itu, berdasarkan Gambar 5.7, BBR hampir selalu mengisi paket datake dalam *buffer* switch selama 100 detik. Pada *loss rate* 1%, rata-rata ukuran *window* BBR selama 100 detik lebih tinggi dibandingkan dengan Reno, BIC, dan CUBIC. Oleh karena itu, *queuing delay* BBR lebih tinggi dibandingkan dengan Reno, BIC, dan CUBIC, sehingga rata-rata RTT BBR pada Gambar 5.9 lebih tinggi dibandingkan dengan Reno, BIC, dan CUBIC.

Selanjutnya, rata-rata RTT Reno, BIC, CUBIC, dan BBR yang semakin tinggi seiring bertambah tingginya *loss rate* disebabkan oleh *delay* pada saat Reno, BIC, CUBIC melakukan *retransmission*. Pada saat terjadi paket *loss*, Reno, BIC, CUBIC, dan BBR melakukan *retransmission* untuk paket datayang mengalami paket *loss*.Oleh karena itu, *destination host* Reno, BIC, CUBIC, dan BBR membutuhkan waktu yang lebih lama untuk menerima paket datayang mengalami paket *loss*. Semakin tinggi *loss rate*, semakin banyak paket datayang mengalami paket *loss*, sehingga semakin tinggi frekuensi Reno, BIC, CUBIC, dan BBR melakukan *retransmission*. Oleh karena itu, rata-rata RTT Reno, BIC, CUBIC, dan BBR semakin tinggi seiring bertambah tingginya *loss rate*.

5.4 Inter TCP *Protocol Fairness*

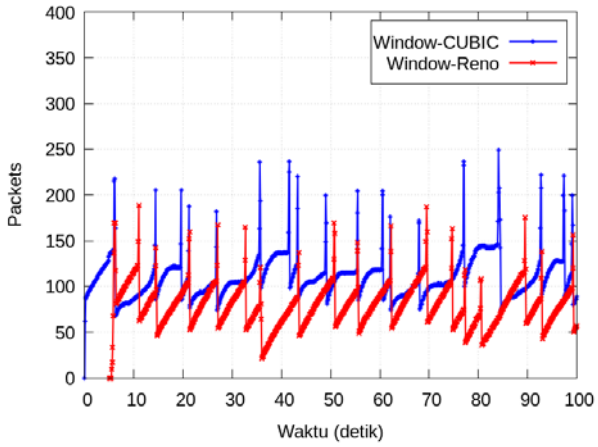
Evaluasi keempat adalah mengirimkan dua TCP *flow*melalui*multiple paths*dengan masing-masing TCP *flow* menggunakan algoritme TCP *congestion control* yang berbeda. Tabel 5.2 menunjukkan *fairness* kedua TCP *flow*berdasarkan*Jain's fairness index*yang dihitung dari rata-rata *throughput* kedua TCP *flow*. Berdasarkan kolom *Jain's fairness index*, Reno, BIC, CUBIC, dan BBR mendapatkan *fairness* yang tinggi karena *Jain's fairness index* pada semua evaluasi *inter* TCP *protocol fairness* mencapai 0.9. Namun, *fairness*CUBIC lebih tinggi dibandingkan dengan Reno, BIC, dan BBR karena *Jain's fairness index* antara CUBIC dengan Reno, CUBIC dengan BIC, dan CUBIC dengan BBR paling

mendekati nilai 1 dibandingkan dengan evaluasi *inter* TCP *protocol fairness* yang lainnya. Fenomena *Jain's fairness index* pada evaluasi *inter* TCP *protocol fairness* menunjukkan bahwa jika dua algoritme TCP *congestion control* yang berbeda mengirimkan paket datamelalui*multiple paths*, CUBIC dapat berbagi *throughput* lebih adil dibandingkan dengan Reno, BIC, dan BBR.

Tabel 5.2Perbandingan *inter* TCP *protocol fairness*

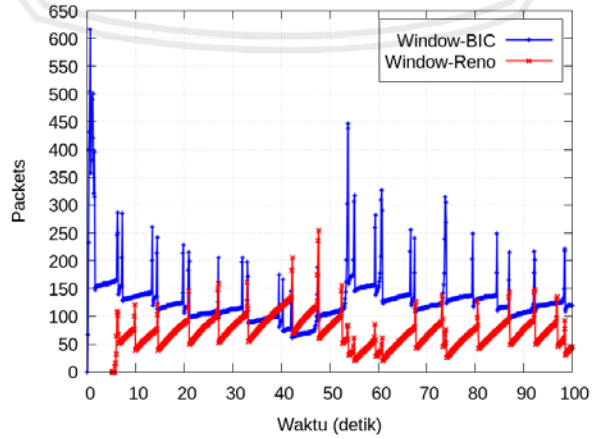
***Source host* 1**		***Source host* 2**		***Jain's fairness index***
TCP	***Throughput* (Mbps)**	**TCP**	***Throughput* (Mbps)**	
CUBIC	10.61	Reno	8.27	0.985
BIC	10.89	CUBIC	8.05	0.978
BIC	12.22	Reno	6.93	0.929
BBR	7.73	CUBIC	11.04	0.969
BBR	7.11	Reno	11.42	0.949
BBR	7.27	BIC	11.29	0.955

Berdasarkan Tabel 5.2, *fairness* tertinggi didapatkan pada saat evaluasi *inter* TCP *protocol fairness* antara CUBIC dengan Reno. Gambar 5.10 menunjukkan perubahan *window* antara CUBIC dengan Reno selama 100 detik. *Fairness* antara CUBIC dengan Reno tinggi karena CUBIC menggunakan mode TCP*friendliness* pada saat CUBIC dan Reno mengirimkan paket datake dalam jaringan yang sama. Seperti yang telah dijelaskan pada sub bab 2.2.3.3, pada mode TCP *friendliness,* CUBIC menaikkanukuran *window*sebesar ukuran*window* Reno. Oleh karena itu,ukuran *window* CUBIC dan Reno selama 100 detik hampir sama. Namun, pada saat terjadi paket *loss*, CUBIC menurunkan ukuran *window* tidak sedrastis Reno, sehingga rata-rataukuran window CUBIC selama 100 detik lebih tinggi dibandingkan dengan Reno. Oleh karena itu,rata-rata *throughput* CUBICpada Tabel 5.2 lebih tinggi dibandingkan dengan Reno.



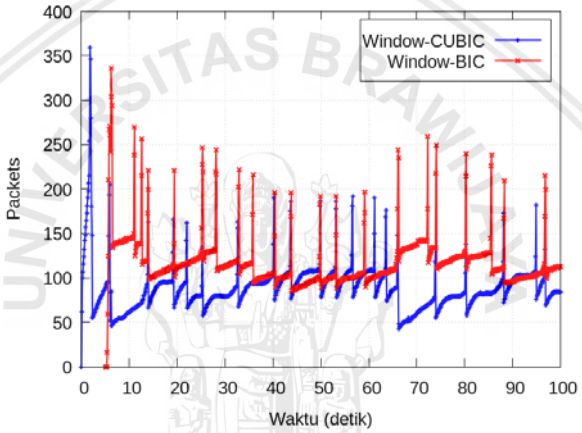
Gambar 5.10Perilaku *fairness* antaraCUBIC dengan Reno

Namun, berdasarkan Tabel 5.2, *fairness* antara CUBIC dengan BIC lebih tinggi dibandingkan dengan *fairness* antara Reno dengan BIC, walaupun CUBIC menggunakan mode TCP *friendliness*. Gambar 5.11 menunjukkan perubahan *window* antara BIC dengan Reno. Pada saat BIC mengirimkan paket datasecara bersamaan dengan Reno di detik ke 6 – 100, BIC menaikkan ukuran *window* lebih agresif dibandingkan dengan Reno. Penyebabnya adalah BIC menggunakan dua mode untuk menaikkan ukuran *window*. Apabila ukuran *window* BIC kurang dari 14 paket data, maka BIC menggunakan mode TCP *friendliness*. Pada mode TCP *friendliness*, BIC menaikkan ukuran *window* dengan mekanisme yang sama seperti Reno. Namun, jika ukuran *window* BIC lebih tinggi dari 14 paket data, maka BIC menggunakan mode *scalability*(Ha, S., Rhee, I. dan Xu, L., 2008). Pada mode *scalability*, BIC menaikkan *window* secara agresif. Berdasarkan Gambar 5.11, ukuran *window* BIC pada detik ke 6 – 100 lebih dari 14 paket data. Oleh karena itu, BIC menaikkan ukuran *window* dengan menggunakan mode *scalability*, sehingga BIC menaikkan ukuran *window* lebih agresif dibandingkan dengan Reno. Akibatnya adalah rata-rata *throughput* BIC pada Tabel 5.2 lebih tinggi dibandingkan dengan Reno.



Gambar 5.11Perilaku *fairness* antaraBIC dengan Reno

Pola perubahan *window* antara BIC dengan Reno juga sama dengan perubahan *window* antara BIC dengan CUBIC. Gambar 5.12 menunjukkan perubahan *window* antara CUBIC dengan BIC. BIC juga menaikkan ukuran *window* lebih agresif dibandingkan dengan CUBIC. Penyebabnya adalah CUBIC menaikkan ukuran *window* dengan mode TCP *friendliness*, sedangkan BIC menaikkan ukuran *window* dengan mode *scalability*. Namun, rata-rata ukuran *window* CUBIC selama 100 detik lebih tinggi dibandingkan dengan Reno pada Gambar 5.11. Penyebabnya adalah CUBIC tidak menurunkan ukuran *window* sedrastis Reno pada saat terjadi paket *loss*. Oleh karena itu, *fairness* antara CUBIC dengan BIC pada Tabel 5.2 lebih tinggi dibandingkan dengan *fairness* antara Reno dengan BIC.



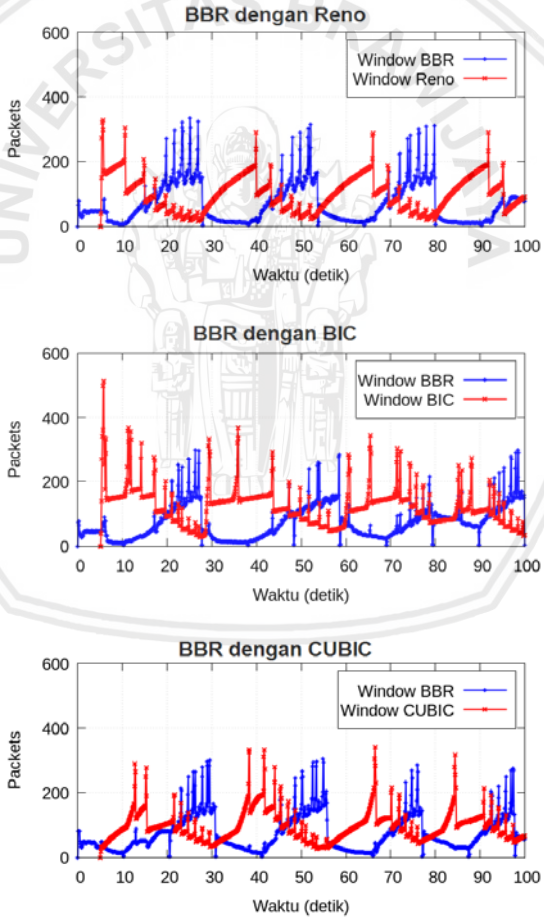
Gambar 5.12Perilaku *fairness* antaraCUBIC dengan BIC

Sementara itu, *inter* TCP *protocol fairness* BBR lebih ditentukan oleh ukuran *buffer* switch. Apabila ukuran *buffer*switch lebih tinggi dari 1.5 BDP, BBR cenderung mendapatkan rata-rata *throughput* yang lebih rendah dibandingkan dengan algoritme TCP *congestion control* berbasis paket *loss*(Cardwell, N., et al., 2016b). Apabila dihitung berdasarkan BDP, total ukuran *buffer* kedua switch pada evaluasi *inter* TCP *protocol fairness*mencapai sekitar 11 BDP. Oleh karena itu, rata-rata *throughput* BBR untuk semua evaluasi *inter* TCP *protocol fairness* BBR pada Tabel 5.2 lebih rendah dibandingkan dengan Reno, BIC, dan CUBIC.

Gambar 5.13 menunjukkan perubahan *window* antara BBR dengan Reno, BIC, dan CUBIC. Berdasarkan Gambar 5.13, terdapat kesamaan pada pola perubahan *window* antara BBR dengan Reno, BIC, dan CUBIC. Seperti yang telah dijelaskan pada sub bab 2.2.4, BBR menentukan ukuran *window* berdasarkan estimasi BDP. Estimasi BDP didapatkan dari estimasi *bottleneck bandwidth* maksimum melalui *probe bandwidth*, dan estimasi RTT minimummelalui*probe*

RTT. Sebagian besar mekanisme BBR adalah melakukan*probe bandwidth* untuk melakukan estimasi *bottleneck bandwidth*maksimum. BBR hanya melakukan *probe* RTT jika BBR tidak mendapatkan estimasi RTT yang lebih rendah selama 10 detik. Estimasi RTT BBR pada saat *probe* RTT akan menjadi estimasi RTT BBR yang baru.

Berdasarkan Gambar 5.13, pada detik ke 6 BBR mendapatkan estimasi *bottleneck bandwidth*yang lebih rendah dibandingkan dengandetik ke 1 – 5 karena pada detik ke 6 Reno, BIC, dan CUBIC mengirimkan paket datake dalam*linkmultiple paths* yang sama dengan BBR. Namun, estimasi RTT minimum BBR pada detik ke 6 tetap karena BBR tidak mendapatkan estimasi RTT minimum yang lebih rendah dibandingkan dengan detik ke 1.Oleh karena itu, estimasi BDP BBR mengalami penurunan, sehingga pada detik ke 6 BBR menurunkan ukuran *window*. BBR terus menurunkan *window* sampai sekitar detik ke 10 karena Reno, BIC, dan CUBIC terus menaikkan ukuran *window*, sehingga estimasi BDP BBR pada saat *probe bandwidth*terus mengalami penurunan.



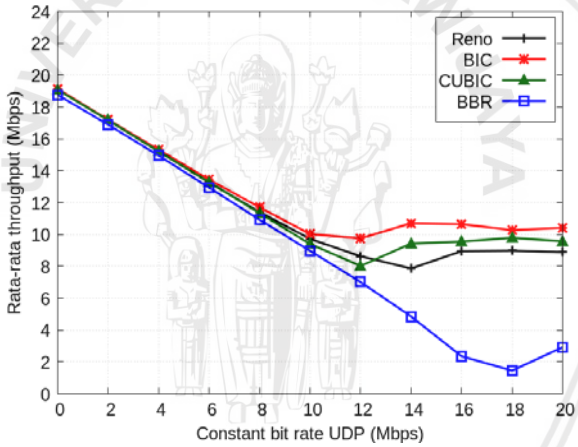
Gambar 5.13Perilaku *fairness* antara BBR dengan Reno, BIC, dan CUBIC

Kemudian, pada sekitar detik ke 10, BBR melakukan *probe* RTT untuk melakukan estimasi RTT minimum karena pada detik ke 1 – 9 BBR tidak mendapatkan perubahan pada estimasi RTT minimum. Namun, pada detik ke 10BBR mendapatkan estimasi RTT minimum yang lebih tinggi dibandingkan dengan detik ke 1 – 9. Penyebabnya adalah pada detik ke 10*link bandwidth* pada *multiple paths* didominasi oleh Reno, BIC, dan CUBIC, sehingga *destination host* BBR pada detik ke 10 menerima paket datalebih lama dibandingkan dengan detik ke 1 – 9. Oleh karena itu, estimasi BDP BBR pada detik ke 10 lebih tinggi dibandingkan dengan detik ke 6 – 9, walaupun pada detik ke 10 estimasi *bottleneck bandwidth* BBR tetap rendah. Pada saat BBR menaikkan ukuran *window* untuk menyesuaikan dengan estimasi BDP yang terbaru, Reno, BIC, dan CUBIC mengalami paket *loss* yang disebabkan oleh *buffer* switch terisi penuh. Oleh karena itu, Reno, BIC, dan CUBIC menurunkan ukuran *window*. Pada saat Reno, BIC, dan CUBIC menurunkan ukuran *window*, BBR mendapatkan estimasi *bottleneck bandwidth* yang lebih tinggi, sehingga estimasi BDP BBR juga lebih tinggi. Oleh karena itu, pada sekitar detik ke 10 – 30, BBR terus menaikkan ukuran *window*, sedangkan Reno, BIC, dan CUBIC terus menurunkan ukuran *window* karena paket *loss*yang disebabkan oleh *buffer* switch terisi penuh terjadi secara terus menerus. Seperti yang telah dijelaskan pada sub bab 5.2, BBR tidak menurunkan ukuran *window* sedrastis Reno, BIC, dan CUBIC pada saat terjadi paket *loss*. Oleh karena itu, BBR terus menaikkan ukuran *window* pada sekitar detik ke 10 – 30, walaupun BBR juga mengalami paket *loss*.

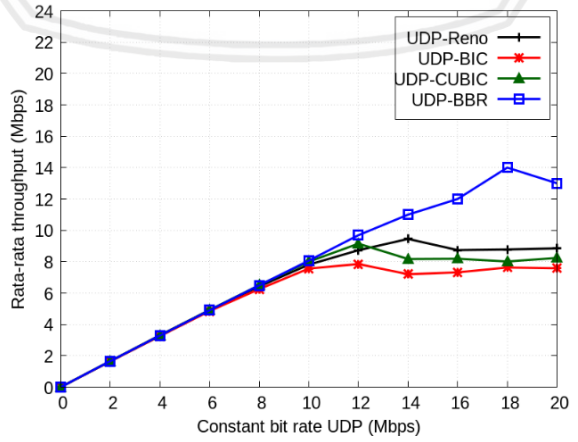
Selanjutnya, pada sekitar detik ke 30, BBR kembali melakukan *probe* RTT. Pada saat BBR melakukan *probe* RTT, estimasi RTT minimum BBR lebih rendah dibandingkan dengan pada saat BBR melakukan *probe* RTT detik ke 10. Penyebabnya adalah ukuran *window* Reno, BIC, dan CUBIC pada detik ke 30 lebih rendah dibandingkan dengan pada saat detik ke 10, sehingga pada detik ke 30 *destination host* BBR menerima paket datalebih cepat dibandingkan dengan detik ke 10. Oleh karena itu, estimasi BDP BBR pada detik ke 30 lebih rendah dibandingkan dengandetik ke 29, sehingga BBR menurunkan ukuran *window* untuk menyesuaikan dengan estimasi BDP yang terbaru. Pada saat BBR menurunkan ukuran *window*, Reno, BIC, dan CUBIC mendapatkan ruang untuk menaikkan ukuran *window*, sehingga Reno, BIC, dan CUBIC terus menaikkan ukuran *window*. Pada saat Reno, BIC, dan CUBIC menaikkan ukuran *window*, BBR mendapatkan estimasi *bottleneck bandwidth* yang lebih rendah pada saat BBR melakukan *probe bandwidth*. Oleh karena itu, pada sekitar detik ke 30 – 40, BBR terus menurunkan ukuran *window*, sedangkan Reno, BIC, dan CUBIC terus menaikkan ukuran *window*. Siklus perubahan *window* antara BBR dengan Reno, BIC, dan CUBIC terjadi secara terus-menerus sampai detik ke 100. Namun, ukuran minimum dan maksimum *window* BBR pada detik ke 6 – 100 lebih rendah dibandingkan dengan Reno, BIC, dan CUBIC. Oleh karena itu, rata-rata *throughput* Reno, BIC, dan CUBIC pada Tabel 5.2 lebih tinggi dibandingkan dengan BBR.

5.5 *Fairness* Antara TCP dengan UDP

Evaluasi terakhir adalah mengirimkan dua *flow*melalui*multiple paths*, dimana *flow* pertama adalah TCP *flow* dan *flow* kedua adalah UDP *flow*. Pada UDP *flow* dilakukan variasi pada CBR sebesar 2 – 20 Mbps. Gambar 5.14 menunjukkan perbandingan rata-rata *throughput*antara Reno, BIC, CUBIC, BBR, dan UDP terhadap variasi CBR UDP 2 Mbps – 20 Mbps. Gambar 5.14 (a) menunjukkan rata-rata *throughput* TCP, sedangkan Gambar 5.14 (b) menunjukkan rata-rata *throughput* UDP. Pada Gambar 5.14 (a), rata-rata *throughput* Reno, BIC, CUBIC, dan BBR pada CBR UDP 2 Mbps – 10 Mbps terus mengalami penurunan hampir sebesar variasi CBR. Sebaliknya, pada Gambar 5.14 (b), rata-rata *throughput* UDP pada CBR UDP 2 Mbps – 10 Mbps terus mengalami kenaikan, walaupun tidak selalu sebesar variasi CBR karena UDP tidak memiliki mekanisme pendeteksian paket *loss* dan*congestion control*. Fenomena rata-rata *throughput* pada CBR 2 Mbps – 10 Mbps menunjukkan bahwa Reno, BIC, CUBIC, dan BBR berbagi *throughput* secara adil dengan UDPpada saat UDP mengirimkan paket datadengan kecepatan kurang dari atau sama dengan setengah total *bandwidth* pada *multiple paths*. Reno, BIC, CUBIC, dan BBR berbagi *throughput* dengan UDP sesuai dengan variasi CBR UDP.



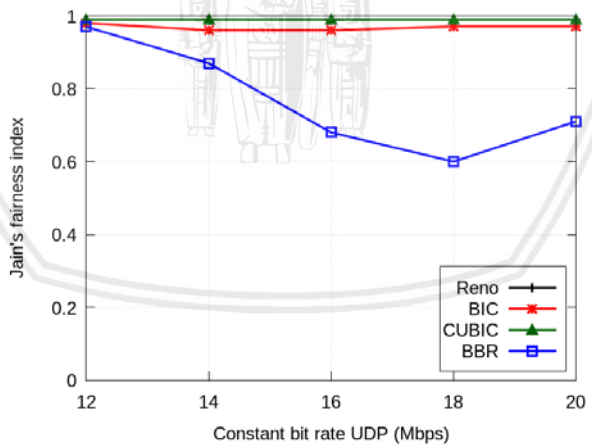
(a) Perbandingan rata-rata throughput TCP terhadap variasi CBR



(b) Perbandingan rata-rata *throughput* UDP terhadap variasi CBR

Gambar 5.14Perbandingan rata-rata *throughput* antara TCP dengan UDP

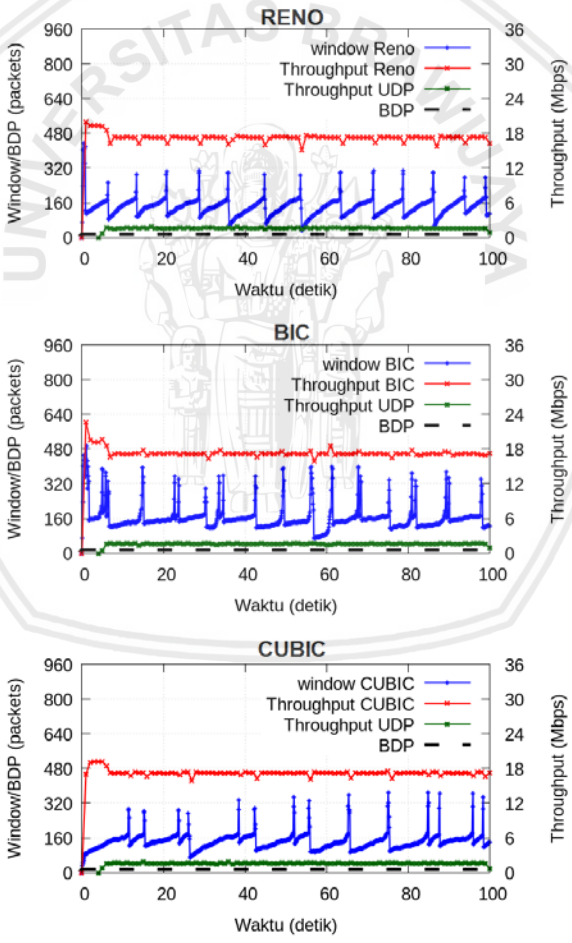
Kemudian, berdasarkan Gambar 5.14 (a) dan (b), rata-rata *throughput* Reno, BIC, dan CUBIC pada CBR UDP 12 Mbps – 20 Mbps cenderung stabil dan hampir sama dengan UDP. Sebaliknya, rata-rata *throughput* BBR pada CBR UDP 12 Mbps – 20 Mbps terus mengalami penurunan hampir sebesar variasi CBR, sedangkanrata-rata *throughput* UDP terus mengalami kenaikan. Apabila rata-rata *throughput*antara Reno, BIC, CUBIC, dan BBR dengan UDP pada CBR UDP 12 Mbps – 20 Mbps dihitung dengan menggunakan *Jain's fairness index* yang ditunjukkan pada Gambar 5.15, *Jain's fairness index* Reno, BIC, dan CUBIC stabil pada 0.9. Sebaliknya, *Jain's fairness index* BBR pada variasi CBR UDP 12 Mbps – 20 Mbps terus mengalami penurunan hingga 0.6. Fenomena rata-rata *throughput* dan *Jain's fairness index* Reno, BIC, dan CUBIC pada CBR UDP 12 Mbps – 20 Mbps menunjukkan bahwa pada saat UDP mengirimkan paket datadengan kecepatan lebih dari setengah total *bandwidth* pada *multiple paths*, *fairness* antara Reno, BIC, dan CUBIC dengan UDP tinggi. Sebaliknya, trafik UDP dapat mengakibatkan *unfairness* pada BBR pada saat UDP mengirimkan paket datadengan kecepatan lebih dari setengah total *bandwidth* pada multiple paths. Namun, *fairness* Reno dan CUBIC lebih tinggi dibandingkan dengan BIC dan BBR karena *Jain's fairness index* Reno dan CUBIC pada Gambar 5.15paling mendekati 1. Oleh karena itu, Reno dan CUBIC dapat berbagi *throughput* lebih adil dengan UDP dibandingkan dengan BIC dan BBR.



Gambar 5.15Perbandingan *fairness* antara TCP dengan UDP

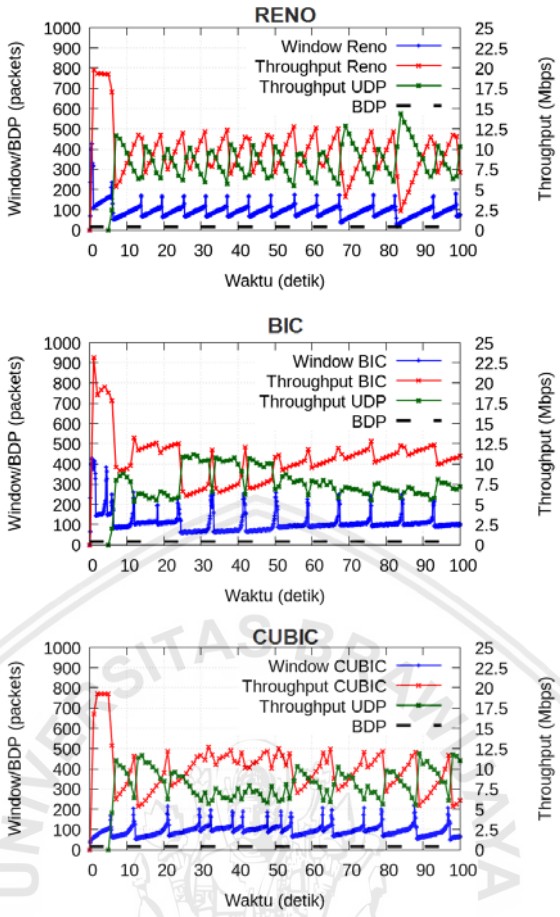
Rata-rata *throughput* Reno, BIC, dan CUBIC mengalami penurunan pada CBR UDP 2 Mbps – 10 Mbps disebabkan oleh Reno, BIC, dan CUBIC harus berbagi *link bandwidth* dan *buffer* switch dengan UDP. Gambar 5.16 menunjukkan perubahan *window* Reno, BIC, dan CUBIC terhadap *throughput* pada CBR UDP 2

Mbps.Pada saat terdapat trafik UDP detik ke 6, Reno, BIC, dan CUBIC menaikkan ukuran *window* sampai *buffer* switch terisi penuh, sehingga terjadi paket *loss*. Namun, ukuran maksimum *window* Reno, BIC, dan CUBIC sampai terjadi paket *loss* mengalami penurunan sekitar 40 paket datajika dibandingkan dengan tanpa trafik UDP pada Gambar 5.2. Penyebabnya adalah Reno, BIC, dan CUBIC harus berbagi *link bandwidth* dan *buffer* switch dengan UDP, sehingga *destination host* Reno, BIC, dan CUBIC menerima paket datalebih lama dibandingkan dengan tanpa trafik UDP. Oleh karena itu, *throughput* Reno, BIC, dan CUBIC pada saat terdapat trafik UDP detik ke 6 – 100 mengalami penurunan sekitar 2 Mbps. Semakin tinggi CBR UDP, semakin rendah ukuran maksimum *window* Reno, BIC, dan CUBIC, sehingga semakin lama *destination host* Reno, BIC, dan CUBIC menerima paket data.Olehkarena itu, berdasarkan Gambar 5.14 (a) dan (b), rata-rata *throughput* Reno, BIC, dan CUBIC pada CBR UDP 2 Mbps – 10 Mbpsterus mengalami penurunan, sedangkan rata-rata *throughput* UDP mengalami kenaikan.



Gambar 5.16Perilaku Reno, BIC, dan CUBIC terhadap trafik UDP pada CBR 2 Mbps

Namun, pada saat UDP mengirimkan paket datadengan CBR12 Mbps – 20 Mbps, Reno, BIC, dan CUBIC dapat menaikkan ukuran *window* untuk mencapai rata-rata *throughput* yang hampir sama dengan UDP. Penyebabnya adalah ukuran *buffer* yang digunakan pada evaluasi ini berukuran besar, sehingga Reno, BIC, dan CUBIC memiliki ruang yang besar untuk menaikkan ukuran *window* sampai *buffer* switch terisi penuh. Gambar 5.17 menunjukkan perubahan *window* Reno, BIC, dan CUBIC terhadap *throughput* pada CBR UDP 20 Mbps. Pada saat terdapat trafik UDP detik ke 6, Reno, BIC, dan CUBIC menaikkan ukuran *window*sampai sekitar 120 paket data,sehinggakemudian terjadi paket *loss* yang disebabkan oleh *buffer* switch terisi penuh. Pada saat Reno, BIC, dan CUBIC menaikkan ukuran *window* sampai 120 paket data, *link bandwidth* pada *multiple paths* didominasi oleh Reno, BIC, dan CUBIC. Oleh karena itu, *throughput* Reno, BIC, dan CUBIC lebih tinggi dibandingkan dengan UDP. Kemudian, pada saat terjadi paket *loss*, Reno, BIC, dan CUBIC menurunkan ukuran *window*. Pada saat Reno, BIC, dan CUBIC menurunkan ukuran *window*, *throughput* Reno, BIC, dan CUBIC mengalami penurunan, sedangkan*throughput* UDP mengalami kenaikan. Siklus perubahan *window* dan *throughput*Reno, BIC, dan CUBIC terjadi secara terus menerus sampai detik ke 100. Oleh karena itu, berdasarkan Gambar 5.14 (a) dan (b), rata-rata *throughput* Reno, BIC, dan CUBIC pada CBR UDP 12 Mbps – 20 Mbps hampir sama dengan UDP.Namun, BIC tidak menurunkan ukuran *window* sedrastis Reno dan CUBIC pada saat terjadi paket *loss*. Pada saat BIC menurunkan ukuran *window*, BIC tetap mendominasi *link bandwidth* pada *multiple paths*. Oleh karena itu, *throughput* BIC pada detik ke 6 – 100 lebih tinggi dibandingkan dengan UDP, sehingga *Jain's fairness index* BIC pada Gambar 5.15 lebih rendah dibandingkan dengan Reno dan CUBIC.

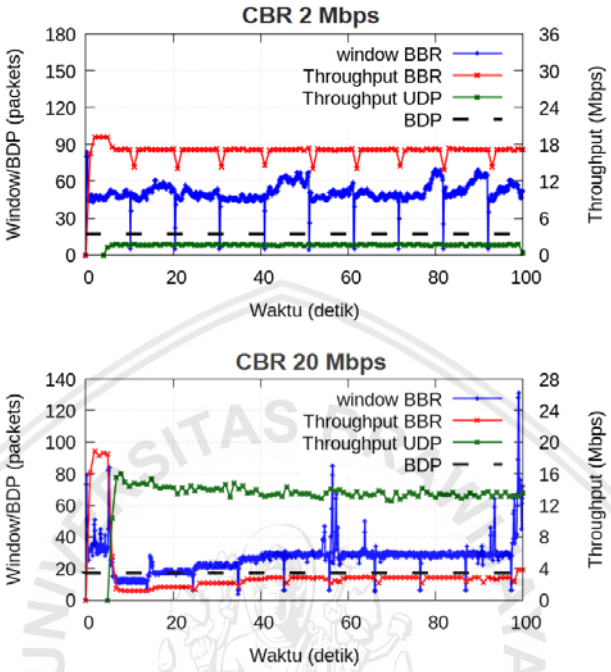


Gambar 5.17Perilaku Reno, BIC, dan CUBIC terhadap trafik UDP pada CBR 20 Mbps

Kebalikan dari Reno, BIC, dan CUBIC yang menaikkan ukuran *window* sampai *buffer* switch terisi penuh, BBR beroperasi dengan menjaga ukuran *window*pada sekitarBDP. Gambar 5.18 menunjukkan perubahan*window* BBR terhadap *throughput* pada CBR UDP 2 Mbps dan 20 Mbps. Pada CBR UDP 2 Mbps, ukuran *window* BBR pada saat terdapat trafik UDP detik ke 6 – 100 tetap sama dengan ukuran *window* BBR pada saat tanpa trafik UDP detik ke 1 – 5. Oleh karena itu, *throughput* BBR pada detik ke 6 – 100 mengalami penurunan karena *destination host* BBR menerima paket datalebih lama dibandingkan dengan tanpa trafik UDP.

Kemudian, semakin tinggi CBR UDP, ukuran *window* BBR pada saat terdapat trafik UDP tetap sama dengan ukuran *window* BBR pada saat tanpa trafik UDP. Pada CBR 20 Mbps, rata-rata ukuran *window* BBR pada saat terdapat trafik UDP detik ke 6 – 100 tetap sama dengan ukuran *window* BBR pada saat tanpa trafik UDP pada detik ke 1 – 5. Akibatnya adalah *destination host* BBR pada CBR UDP 20 Mbps menerima paket datalebih lama dibandingkan dengan CBR UDP 2 Mbps karena *link bandwidth* pada *multiple paths* didominasi oleh UDP, sehingga *throughput* BBR pada CBR UDP 20 Mbps mengalami penurunan lebih drastis

dibandingkan dengan CBR UDP 2 Mbps. Oleh karena itu, berdasarkan Gambar 5.14 (a) dan (b),rata-rata *throughput* BBR pada CBR 2 Mbps – 20 Mbps terus mengalami penurunan, sedangkan rata-rata *throughput* UDP terus mengalami kenaikan.



Gambar 5.18Perilaku BBR terhadap trafik UDP

BAB 6 PENUTUP

6.1 Kesimpulan

Berdasarkan seluruh pengukuran kinerja algoritme TCP *congestion control* Reno, BIC, CUBIC, dan BBR pada *multi-path routing*, maka didapatkan kesimpulan:

1. Rata-rata *throughput* Reno, BIC, CUBIC, dan BBR pada *multi-path routing* mengalami kenaikan sekitar 2 kali lipat dibandingkan dengan *single path routing*. Oleh karena itu, paket *reordering* yang terjadi pada *multi-path routing*tidak mengakibatkan penurunan rata-rata *throughput* Reno, BIC, CUBIC, dan BBR.
2. Pada saat hanya satu TCP *flow* yang mengirimkan paket data melalui *multiple paths*, BBR merupakan algoritme TCP *congestion control* terbaik pada *multi-path routing*. Namun, jika dua *flow* mengirimkan paket data melalui *multiple paths*, CUBIC merupakan algoritme TCP *congestion control* terbaik pada *multi-path routing*. Pada evaluasi variasi *link delay*, rata-rata RTT BBR lebih rendah hingga 58 ms dibandingkan dengan Reno, BIC, dan CUBIC. Sedangkan pada evaluasi variasi*loss rate*, rata-rata *throughput* BBR lebih dari 45% lebih tinggi dibandingkan dengan Reno, BIC, dan CUBIC. Selanjutnya, pada evaluasi *inter* TCP *protocol fairness*, *Jain's fairness index* CUBIC paling mendekati nilai 1 dibandingkan dengan Reno, BIC, CUBIC, 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.

6.2 Saran

Pada penelitian selanjutnya dapat mempertimbangkan menggunakan *cost* yang berbeda pada setiap *multiple paths* untuk melakukan evaluasi*loss rate*,*inter* TCP *protocol fairness* dan *fairness* antara TCP dengan UDP pada algoritme TCP *congestion control* Reno, BIC, CUBIC, dan BBR. Penyebabnya adalah *cost* pada *multiple paths* tidak selalu sama, sehingga paket *reordering* akan lebih banyak terjadi dibandingkan dengan pada saat *cost* pada *multiple paths* sama.

DAFTAR PUSTAKA

Afanasyev, A., Tilley, N., Reiher, P. dan Kleinrock, L., 2010. Host-to-Host Congestion Control for TCP. IEEE Communications Surveys and Tutorials, 12(3), pp. 304 – 342.

Arokkiam, J. A., Wu, X., Brown, K. N. dan Sreenan, J., 2014. Experimental Evaluation of TCP performance over 10Gb/s Passive Optical Networks (XG-PON). In:Proceedings 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.

Bridge-utils., 2019. bridge-utils-interfaces - bridge-utils extensions for the interfaces(5) file format. [online] Tersedia di <https://manpages.debian.org/jessie/bridge-utils/bridge-utils-interfaces.5.en.html> [Diakses 23 Juli 2019].

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].

Cardwel, N., Cheng, Y., Yeganeh, S. H. dan Jacobson, V., 2017. BBR Congestion Control.Internet Congestion Control Research Group - Internet Draft.

Carpa, R., Assuncao, M. D. D., Gluck, O., Lefevre, L. dan Mignot, J. C., 2017. Evaluating the Impact of SDN-Induced Frequent Route Changes on TCP Flows. In: Proceedings 13th International Conference on Network and Service Management.

Chang, H. Y., Lee, W. T. dan Wei, H. W., 2014. A Novel Protocol UDCP for Improving Fairness and Maintaining the high-Speed Rate of UDP. In: Proceedings10th International Conference on Intelligent Information Hiding and Multimedia Signal Processing, pp. 698-701.

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: Proceedings IEEE INFOCOM, pp. 2130–2138.

Floyd, S., Mahdavi, J., Mathis, M. dan Podolsky, M., 2000. An Extension to the Selective Acknowledgment (SACK) Option for TCP. RFC 2883.

Gettys, J. dan Nichols, K., 2011. Bufferbloat: Dark Buffers in the Internet.ACM Queue, 9(11), pp. 40–54.

Ha, S. dan Rhee, I., 2008. Hybrid Slow Start for High-Bandwidth and Long-Distance Networks. In: Proceedings 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.

Hasegawa, G. dan Murata, M., 2001. Survey on Fairness Issues in TCP Congestion Control Mechanisms.IEICE Transactions on Communications, E84-B(6), pp. 1461–1472.

He, J. dan Rexford, J., 2008. Toward Internet-Wide Multipath Routing. IEEE Network, 22(2), pp. 16–21.

Hock, M., Bless, R. and Zitterbart, M., 2017. Experimental Evaluation of BBR Congestion Control. In: Proceedings 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.

Jacobson, V., Braden, R. dan Borman, D., 1992. TCP Extensions for High Performance. RFC 1323.

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 ReportTR-301, Digital Equipment Corporation.

Karlsson, J., Hurtig, P., Brunstrom, A., Kassler, A., Di Stasi, G., 2012. Impact of Multi-path Routing on TCP Performance. In: Proceedings 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].

Lar, S. dan Liao, X., 2013. An initiative for a classified bibliography on TCP/IP congestion control. Journal of Network and Computer Applications. Elsevier, 36(1), pp. 126-133.

Leung, K.-C., Li, V. O. dan Yang, D., 2007. An Overview of Packet Reordering in Transmission Control Protocol (TCP): Problems, Solutions, and Challenges.IEEE Transactions on Parallel and Distributed Systems, 18(4), pp. 522–535.

Li, Y. T., Leith, D. dan Shorten, R. N., 2007. Experimental Evaluation of TCP Protocols for High-Speed Networks. IEEE/ACM Transactions on Networking, 15(5), pp. 1109–1122.

Ludwig, R. dan Katz, R. H., 2000. The Eifel Algorithm: Making TCP Robust Against Spurious Retransmissions.ACM SIGCOMM Computer Communication Review, 30(1), pp. 30 - 36.

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: Proceedings IEEE 41st Conference on Local Computer Networks, pp. 706–714.

Mathis, M., Mahdavi, J., Floyd, S., Romanow, A., 1996. TCP Selective Acknowledgment Options. RFC 2018.

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

Rhee, I., Xu, L., Ha, S., Zimmermann, A., Eggert, L., Scheffenegger, R., 2018. CUBIC for Fast Long-Distance Networks. RFC 8312.

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.

Stevens, W., 1997. TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithm. RFC 2001.

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

Villamizar, C. dan Song, C., 1994. High performance TCP in ANSNET.ACM SIGCOMM Computer Communication Review, 24(5), pp. 45–60.

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

Wagner, K., 2001.Short Evaluation of Linux's Token-Bucket-Filter (TBF) Queueing Discipline. Tersedia di: <http://www.docum.org/docum.org/docs/other/tbf02_kw.ps> [Diakses 23 Juli 2019].

Wang, J., Wen, J., Han, Y., Zhang, J., Li, C. dan Xiong, Z., 2013. CUBIC-FIT: A High Performance and TCP CUBIC Friendly Congestion Control Algorithm.IEEE Communications Letters, 17(8), pp. 1664–1667.

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

Xu, L., Harfoush, K. dan Rhee, I., 2004. Binary Increase Congestion Control for Fast, Long Distance Networks. In: Proceedings 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:ProceedingsIEEE 3rd International Conference on Software Engineering and Service Science, pp. 509–512.