PERBANDINGAN KINERJA ALGORITMA BIC, CUBIC DAN HTCP PADA TOPOLOGI DUMBBELL DAN SIMPLE NETWORK MENGGUNAKAN NS2

Rian Fahrizal[1], Wahyu Dewanto[2], Sujoko Sumaryono[3]
[1] Jurusan Teknik Elektro FT UNTIRTA
Jln. Jend. Sudirman Km 2 Cilegon Indonesia
[2,3] Jurusan Teknik Elektro FT UGM
Jln. Grafika 2 Yogyakarta 55281 INDONESIA

Abstract

High speed computer networks with a large waiting time is a common from of network in the future. In this network are commonly used TCP algorithms have difficullty in sending data. There are several algorithms that has used the BIC, CUBIC and HTCP. These algorithms needs to be tested to determine its performance when apllied to the network topology with two dumbbells, and simple network. Teh results obtained testing the algorithms is best HTCP performance by having the smallest value.

Keywords : algorithms, BIC, CUBIC, HTCP

Jaringan komputer kecepatan tinggi dengan waktu tunggu yang besar merupakan bentuk jaringan yang umum di masa depan. Pada jaringan ini algoritma TCP yang umum digunakan mengalami kesulitan di dalam melakukan pengiriman data. Ada beberapa algoritma yang telah digunakan yakni BIC, CUBIC, dan HTCP. Algoritma-algoritma ini perlu diuji untuk mengetahui kinerjanya jika diterapkan pada jaringan dengan dua topologi yakni *dumbbell,* dan *simple network*. Hasil pengujian didapatkan algoritma yang paling baik adalah HTCP dengan memiliki nilai kinerja yang paling kecil.

Kata kunci : Algoritma, BIC, CUBIC, HTCP

1. PENDAHULUAN

TCP/IP merupakan protokol yang digunakan secara luas di dalam jaringan Internet. Protokol ini memiliki beberapa lapisan yakni application layer, transport layer, network layer dan physical layer. Setiap layer ini memiliki fungsi yang berbeda antara satu dengan yang lainnya. Lapisan yang bertanggung jawab di dalam melakukan pengiriman data yang juga merupakan lapisan yang penting di dalam TCP/IP ini ialah transport layer. Ada dua motode yang digunakan di dalam layer ini yakni TCP dan UDP. Sebagian besar pengiriman data di dalam jaringan komputer menggunakan metode TCP.

Di dalam pengiriman TCP yang ada digunakan algoritma adaptive increase multiplicity decrease (AIMD). Di dalam algoritma ini penambahan jumlah data yang dikirim dalam satu kali pengiriman (window) jika data berhasil dikirim dan tidak mengalami kongesti. Apabila terjadi kongesti atau terjadi error maka pengiriman window berkurang setengahnya. Algoritma ini telah digunakan sejak tahun 1980-an hingga sekarang. Seiring perkembangan waktu jaringan Internet membutuhkan bandwidth yang jauh lebih besar. Algoritma AIMD ini mengalami masalah pada jaringan komputer dengan bandwidth lebih besar dari 1 Gbps. Hasil penelitian menunjukan dibutuhkan waktu yang lama untuk mencapai nilai badwidth 1 Gbps jika propagation delay pada jaringan tersebut memiliki nilai 100 ms.

Untuk menangani masalah di dalam algoritma AIMD dibuat algoritma alternatif yang sudah diterapkan di dalam Linux kernel terbaru yakni BIC, CUBIC dan HTCP. Ketiga algoritma ini digunakan untuk menangani masalah pada jaringan Internet dengan waktu delay yang besar. Akan tetapi masih diperlukan pengujian yang baik yang dapat membuktikan bahwa ketiga algoritma ini merupakan algoritma yang terbaik yang dapat diterapkan di dalam jaringan komputer. Pengujian yang dilakukan untuk ketiga algoritma ini dilakukan dengan menggunakan topologi yang mencoba mensimulasikan jaringan Internet yang ada dengan menggunakan topologi dumbbell dan simple network.

Parameter kinerja yang digunakan di dalam pengujian ini ialah menggunakan rata-rata throughput, stabilitas dan fairness (Raj Jain index). Kemudian dari nilai-nilai parameter yang

telah digunakan tersebut dilakukan perhitungan yang dapat menunjukan bahwa parameter tersebut bisa dibandingkan secara keseluruhan. Perhitungan tambahan ini menunjukan kinerja dari algoritma yang dibandingkan.

2. DASAR TEORI

2.1 BIC-TCP

Algoritma ini memandang pengendalian congesti merupakan sebuah masalah pencarian dimana sistem dapat memberikan jawaban ya/tidak sebagai umpan balik melalui paket yang hilang yang menunjukan bahwa nilai pengiriman lebih besar dari kapasitas jaringan. Nilai dari *window* minimum dapat ditentukan sebagai besarnya *window* yang mengalir yang tidak terdapat paket yang hilang. Apabila ukuran *window* maksimum diketahui, dapat diterapkan teknik pencarian biner untuk menentukan besarnya *window* yang dituju untuk mencapai nilai tengah maksimum dan minimum. Pada saat penambahan sampai ke nilai yang diharapkan, jika menghasilkan paket hilang, nilai *window* dapat ditentukan sebagai nilai maksimum baru dan megurangi besarnya *window* setelah paket *loss* dapat ditentukan sebagai minimum yang baru. Nilai antara kedua nilai ini merupakan nilai yang diharapkan.

Hubungan untuk pendekatan ini ialah bahwa ketika jaringan menghasilkan *loss* antara minimum baru tapi tidak terjadi terlalu dekat dengan nilai minimum yang baru, nilai yang diharapkan pasti berada diantara dua nilai. Sampai mencapai nilai yang diharapkan dan tidak menghasilkan paket hilang, maka besar nilai yang ada menjadi nilai minimum yang baru, dan nilai yang diharapkan dihitung kembali. Proses ini berulang dengan melakukan pembaharuan nilai minimum dan maksimum sampai perbedaan antara nilai-nilai tersebut mencapai nilai dibawah batas, yang disebut dengan penambahan minimum (minimum *increament* S_{min}). Teknik ini disebut dengan *binary search increase*.

Binary search increase menjadikan pencarian *bandwidth* menjadi semakin agresif ketika perbedaan antara *window* yang ada dengan *window* yang diharapkan besar, dan menjadi kurang agresif pada saat besarnya *window* yang ada mendekati besarnya *window* yang diharapkan. Teknik unik dari protokol ini ialah fungsi penambahannya dalam bentuk logaritmik, sehingga pada saat mendekati nilai saturasi besarnya penambahan *window* semakin berkurang. Protokol yang lain menambahkan nilai *bandwidth* sampai pada saat nilai saturasi sehingga penambahan nilai saturasi adalah nilai maksimum dari keadaan tersebut. Pada umumnya, jumlah paket yang hilang berbanding lurus dengan besarnya penambahan terakhir sebelum paket hilang. Oleh karena itu *binary search increase* dapat mengurangi paket yang hilang. Seperti yang diketahui, keuntungan utama dari *binary search* ialah dapat memberikan fungsi respon yang baik, yang dapat digabungkan dengan penambahan secara linear.

Untuk menjamin penyatuan yang lebih cepat dan RTT yang adil, digabungkan *binary search increase* dengan strategi penambahan linear. Pada saat jarak dari ukuran *window* dengan yang diinginkan terlalu besar, penambahan ukuran *window* secara langsung pada nilai tengah tersebut memberikan beban berat ke dalam jaringan. Pada saat jarak dari ukuran *window* yang ada dengan besarnya *window* yang diinginkan pada *binary search increase* lebih besar dari langkah maksimum yang disebut dengan penambahan maksimum (*maximum increment* S_{max}), ditambahkan nilai *window* dengan S_{max} sampai jarak menjadi kurang dari S_{max}, pada saat dimana penambahan *window* secara langsung ke dalam nilai yang diinginkan. Kemudian setelah pengurangan *window* besar, menambahkan nilai *window* secara linear dan selanjutnya penambahan secara logaritmik. Penggabungan dari *binary search increase* dan penambahan secara penjumlahan disebut dengan *binary increase*.

Penggabungan dengan strategi pengurangan secara pembagian, *binary increase* menjadi mendekati penambahan linear dalam *window* yang besar. Hal ini disebabkan *window* yang lebih besar menghasilkan pengurangan yang lebih besar dengan pengurangan dengan perkalian, sehingga membutuhkan rentang waktu yang lebih lama. Kemudian di saat ukuran *window* kecil, menjadi *binary search increase* dengan rentang waktu penambahan yang lebih sedikit.

Dapat dilihat bahwa di dalam model *loss* yang telah tersinkironisasi secara penuh, *binary search increase* digabung dengan pengurangan secara pengali bergabung menjadi sebuah nilai *throughput* yang adil. Sebagai contoh ada aliran dengan ukuran *window* yang berbeda, tapi dengan RTT yang sama. Karena *window* yang berukuran lebih besar berkurang lebih banyak pada pengurangan secara perkalian (dengan factor yang tetap β), waktu untuk mencapai target lebih lama untuk *window* yang lebih besar. Akan tetapi, penggabungannya membutuhkan waktu yang lama. Pada penambahan *binary increase*, membutuhkan log(d)-

log(S_{min}) RTT untuk mencapai *window* maksimum setelah pengurangan *window* d. karena penambahan *window* dalam fungsi log, semakin besar *window* dan semakin kecil *window* dapat mencapai kembali nilai maksimumnya dengan sangat cepat pada waktu yang hampir bersamaan. Akan tetapi *window* yang lebih kecil mengalir dengan *bandwidth* yang lebih kecil dari yang lebih besar sebelum pengurangan *window* selanjutnya. Oleh karena itu dimodifikasi *binary search increase* sebagai berikut.

Dalam binary *search increase*, setelah pengurangan *window*, nilai maksimum dan minimum ditentukan. Contohnya nilai-nilai ini adalah max_win$_i$ dan min_win$_i$ untuk aliran i (i=1,2). Apabila nilai maksimum baru lebih kecil dari nilai sebelumnya, *window* ini memiliki tren penurunan. Kemudian diatur ulang nilai maksimum baru menjadi sama seperti nilai *window* yang diinginkan baru, dan selanjutnya diatur lagi nilai tujuan. Kemudian diterapkan penambahan *binary increase* normal. Bentuk strategi ini disebut *convergence*.

BIC-TCP menggunakan bentuk algoritma pencarian biner untuk memperbaharui *cwnd*. Nilai dari w_1 dipelihara yang menentukan nilai setengah antara nilai *cwnd* sebelum dan setelah kejadian kehilangan terakhir. Aturan pembaharuan *cwnd* mencari dengan cepat penambahan *cwnd* pada saat hal ini mencapai jarak tertentu S_{max} dari w_1, dan memperbaharui *cwnd* lebih lambat ketika nilainya mendekati w_1. Perkalian *backoff* dari *cwnd* digunakan pada pendeteksian paket yang hilang, dengan asumsi faktor *backoff* β sebesar 0,8. Secara rinci,

$$Ack: \begin{cases} \delta = (w_1 - cwn \\ cwnd \leftarrow cwnd + \end{cases} \quad (2.16)$$

$$Loss: \begin{cases} w_1 = \begin{cases} \frac{1+\beta}{1} x cwnd & c \\ cwnd \end{cases} \\ w_2 = cwnd \\ cwnd \leftarrow \beta \; x \; c \end{cases} \quad (2.17)$$

Dengan

$$f_\alpha(\delta, cwnd) \begin{cases} \frac{\delta}{\sigma} & (\delta \leq 1, cwn \\ & atau \; (w_1 \leq cwnd < w_1 + \\ \delta & 1 < \delta \leq S_{max}, \; cwnc \\ \frac{w_1}{B-1} & B \leq cwnd - w_1 < S_{max} \\ S_{max} & se \end{cases} \quad (2.18)$$

dengan
σ = *distance*
cwnd = *congestion window(bytes)*
f(δ ,cwnd) = *fungsi penambahan window size*
δ = *pengurangan*
w_1 = *window(bytes)*
w_2 = *window(bytes)*
β = *backoff factor*

BIC-TCP juga menerapkan sebuah algoritma dimana pada saat utiliasasi rendah terdeteksi, akan menambahkan *window* lebih agresif. Hal ini dikendalikan dengan parameter *Low-Util* dan *Util_Check*. Untuk menentukan kompabilitas sebelumnya, hal ini menggunakan parameter pembaharu TCP standar dimana *cwnd* dibawah batas *Low-Window*.

2.3 CUBIC

Seperti namanya fungsi penambahan *window* yang digunakan ialah fungsi cubic yang memiliki kesamaan dengan fungsi penambahan BIC-TCP. CUBIC menggunakan fungsi cubic untuk waktu yang berlalu dari kejadian congesti yang terakhir. Walaupun sebagian besar algoritma alternative menggunakan fungsi penambahan convex dimana setelah kejadian paket hilang, penambahan *window* selalu bertambah, CUBIC menggunakan bentuk concave dan convex dari fungsi cubic untuk penambahan *window*.

Penjelasan lebih detail dari fungsi ini ialah setelah pengurangan *window* karena kejadian paket yang hilang, didaftarkan W_{maz} menjadi nilai *window* dengan munculnya kejadian *window* hilang dan menggunakan pengurangan dengan perkalian dari congestion *window* dengan menggunakan factor β dengan β adalah konstanta pengurangan *window* dan menggunakan algoritma recovery dan retransmit dari TCP. Kemudian setelah masuk ke algoritma congestion avoidance dari recovery, mulai ditambahkan *window* menggunakan bentuk concave pada fungsi cubic. Fungsi cubic ditentukan untuk memiliki nilai baru pada W_{max} sehingga penambahan berlanjut sampai ukuran *window* menjadi W_{max}. kemudian fungsi cubic berubah menjadi bentuk convex dan penambahan *window* convex dimulai. Bentuk penentuan *window* (concave dan selanjutnya convex) meningkatkan stabilitas protokol dan jaringan ketika mengatur utilisasi jaringan kecepatan tinggi. Hal ini disebabkan ukuran *window* hamper mendekati konstan, membentuk nilai baru dekat dengan W_{max} dengan utilisasi jaringan tertinggi dan di dalam kondisi steady, sebagian besar ukuran *window* CUBIC mendekati W_{max}, sehingga mempromosikan stabilitas utilisasi jaringan kecepatan tinggi dan protokol. Protokol dengan fungsi penambahan convex menjadi memiliki penambahan *window* terbesar sekitar nilai saturasi, dengan paket yang hilang banyak.

Algoritma ini menggabungkan ide dasar dari *High-Speed* TCP dan H-TCP. Disebut dengan penambahan *cwnd* sebagai fungsi dari waktu karena pemberitahuan terakhir dari *congesti*, dan besarnya *window* pada pemberitahuan terakhir *congesti*. Bentuk dari algoritma ini dapat disimpulkan sebagai berikut:

Perubahan *slow start*. Perubahannya ditentukan pada awal. Sekali cwnd meningkat di atas *ssthresh*, CUBIC keluar dengan menggunakan algoritma *slow start* normal dan perubahan menggunakan penambahan eksponensial yang agresif dimana *cwnd* ditambahkan sebesar satu paket untuk setiap 50 ack yang diterima atau sama dengan dua kali *cwnd* mendekati setiap 35 waktu *round-trip*.

Faktor *backodd* 0,8. Pada paket yang hilang, cwnd dikurangi dengan faktor 0,8.

Clamp pada laju penambahan maksimum. Laju penambahan pada operasi AIMD dibatasi paling tidak 20*delay_min paket setiap RTT, dimana *delay_min* adalah perkiraan *round-trip propagation* waktu tunda dari aliran data. Merubah dari paket per RTT menjadi paket per detik, *clamp* ini kira-kira sama dengan laju penambahan 20 paket/detik tidak tergantung RTT.

Fungsi penambahan cubic. Subyek pada *clamp* ini, laju penambahan adalah paket target-cwnd per RTT. Perhatikan bahwa efek dari penambahan ini ialah menyesuaikan cwnd menjadi sama dengan target pada arah dalam RTT tunggal. Nilai dari target dihitung dari

$$target = W_{max} + C\left(t - \sqrt[3]{(\beta(W_{max} - 0{,}8W))}\right)^3$$

dengan

W_{max} = *window maksimum(bytes)*
W = *window(bytes)*
t = *waktu (detik)*
C = *cubic*
β = *faktor pengurangan*

dimana t adalah waktu berlalu sejak backoff terkahir (mendekati nilai delay_min ditambahkan ke dalam nilai ini) dan W_{max} dihubungkan dengan *cwnd* pada *backoff* terkahir dan ditandai origin_point pada kode. W adalah nilai *cwnd* sebelum *backoff* terakhir, sehingga 0,8W adalah nilai *cwnd* nilai sebelum *backoff* muncul.

Adaptasi fungsi cubic. Nilai dari W_{max} ditentukan tergantung dari apakah *backoff* terakhir muncul sebelum atau sesudah cwnd tercapai pada nilai W_{max} sebelumnya. Atau jika tidak W_{max} diset sama dengan 0,9W.

Algoritma ini juga mencakup kode untuk meyakinkan bahwa algoritma ini paling tidak seagresif seperti TCP yang ada.

2.4 HTCP

Metode pengaturan *bandwidth* dilakukan dengan melakukan penambahan *window*. Awal pengiriman data algoritma *slow start* dijalankan sampai mencapai nilai batas tertentu yang telah ditentukan. Setelah nilai tersebut tercapai maka algoritma ini dijalankan. Pada saat muncul kongesti nilai ambang batas menjadi setengah dari nilai sebelumnya yang digunakan pada pengiriman paket selanjutnya.

HTCP menggunakan waktu lalu Δ sejak kejadian congesti terakhir, daripada cwnd, untuk menentukan jalur pada *bandwidth*-waktu tunda dan parameter penambahan AIMD berbeda sebagai fungsi Δ. Parameter penambahan AIMD juga ditentukan dengan jalur waktu *round-trip* untuk mengurangi ketidak adilan antara aliran dengan waktu *round-trip* yang berbeda. Pengurangan faktor AIMD ditentukan untuk meningkatkan utilisasi jalur berdasarkan pada estimasi ketentuan queue pada jalur. Secara lengkap

$$ACK: \quad cwnd \leftarrow cwnd + \frac{2!}{} \tag{2.1}$$

$$Loss: \quad cwnd \leftarrow g_\beta(B \tag{2.2}$$

Dengan

$$f_\alpha(\Delta) = \begin{cases} 1 \\ \max\left(\bar{f}_\alpha(\Delta) T_{min}, 1\right) \end{cases} \tag{2.3}$$

$$g_\beta(B) = \begin{cases} 0.5 & \left| \frac{B(k+1)-}{B(k)} \right. \\ \min\left(\frac{T_{min}}{T_{max}}, 0.8\right) & se \end{cases} \tag{2.4}$$

dengan

cwnd	= *congestion window(bytes)*
β	= *faktor backoff*
Δ	= *delta*
$f_\alpha(\Delta)$	= *fungsi penambahan congestion window*
$g_\beta(B)$	= *fungsi pengurangan congestion window*
T_{min}	= *waktu minimum(*detik*)*
T_{max}	= *waktu maksimum(*detik*)*

dimana Δ_L ditentukan batas sehingga algoritma update TCP standar digunakan dimana $\Delta \leq \Delta_L$. Fungsi penambahan kuadrat untuk f_α ialah

$$\bar{f}_\alpha(\Delta) = 1 + 10(\Delta - \Delta_L) + 0.25 (\Delta \tag{2.5}$$

dengan

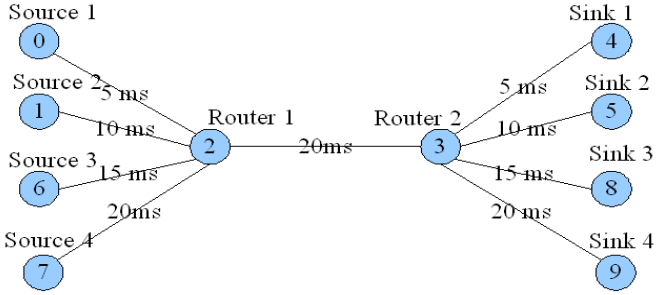
$f_\alpha(\Delta)$	= *fungsi penambahan dalam rentang waktu*
Δ_L	= *Delta limit*

T_{min} T_{max} menghitung waktu round-trip yang telah ada. β(k + 1) adalah perhitungan *throughput* yang dicapai maksimum pada masa congesti terakhir.

3. METODE PENELITIAN

3.1 Topologi dumbbell

Seperti yang terlihat pada Gambar 1., dimana *node* sumber (*node* 0,1,6,7) dan *node* penerima (*node* 4,5,8,9) terhubung melewati *node* 2 dan *node* 3 yang berfungsi sebagai *router*. *Bandwidth* antara *node* sama sebesar 1 Gbps. Pada detik ke-5 *node* 0 melakukan pengiriman data ke *node* 4, kemudian *node* 1 melakukan pengiriman data pada detik ke-15. Pada pengiriman data ke-3 dilakukan oleh *node* 6 ke *node* 8 pada detik ke-25, lalu *node* 7 melakukan pengiriman data ke *node* 9 pada detik ke-35. Pada detik ke-100 simulasi selesai.



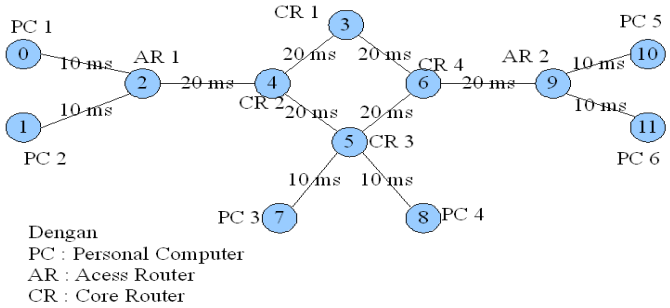
Gambar 1. Topologi *dumbbell*

Program simulasi menggunakan algoritma , H-TCP, BIC dan CUBIC pada metode pengiriman data. Pada simulasi ini ada empat aliran data mengalir melalui satu link yang sama yakni dari link antara *node* 3 dan *node* 4, seperti yang ditunjukan pada Gambar 3.

Topologi ini memiliki dua bentuk yang digunakan untuk memudahkan di dalam analisis yakni dengan menggunakan nilai yang sama pada waktu tunggu untuk semua *node* yang melakukan pengiriman. *Node* yang melakukan pengiriman memiliki waktu pengiriman yang sama dengan scenario sebelumnya.

3.2 Topologi Simple network

Topologi *simple network* dapat dilihat pada Gambar 2. Pada konfigurasi ini, *router* inti menunjukan backbone jaringan dengan *router* perantara bertanggung jawab untuk *node* pengirim dan penerima terhubung dengan jaringan.



Gambar 2. Topologi *Simple network*

Pada topologi ini *node* 0,1, dan 7 melakukan pengiriman data sedangkan *node* 8, 10, dan 11 menerima data yang dikirim. Topologi ini merupakan penyederhanaan topologi menggunakan *transit* dan *stub* [15].

Program simulasi menggunakan algoritma H-TCP, BIC dan CUBIC pada metode pengiriman data. Hubungan antara *node* dapat dilihat pada Gambar 2.

4. ANALISIS HASIL SIMULASI

4.1. Analisis Topologi *Dumbbell*.

Pada simulasi ini terdapat empat buah *node* yang melakukan pengiriman yang melakukan pengiriman secara bergiliran sampai detik ke-100. Ada dua bentuk scenario pada simulasi kali ini yang pertama meiliki waktu tunggu yang berbeda, sedangkan scenario yang kedua waktu tunggunya sama.

Simulasi ini menunjukan algoritma berhasil melakukan pengiriman, dimana keempat *node* yang melakukan pengiriman berhasil mengirim data. Sedangkan pada algortima BIC dan CUBIC hanya dua *node* yang berhasil melakukan pengiriman. Kemudian algoritma HTCP hanya berhasil melakukan pengiriman sebanyak 3 *node*.

Kemudian untuk pengujian pada topologi *dumbbell* yang meliputi dua pengujian yang sedikit berbeda dapat ditunjukan pada Tabel 1. dan Tabel 2. Walaupun pada pengujian topologi ini menggunakan sedikit metode yang sedikit berbeda hasil kinerja menunjukan perbedaan yang jauh pada algoritma tertentu.

Tabel 1. Hasil perhitungan kinerja algoritma topologi *dumbbell* pertama

Algoritma	y	r	z	u
BIC	0.98825	0.076415	0.497368	1.562033
CUBIC	0.998781	0.209982	0.541685	1.750448
HTCP	0.504797	0.128989	0.288293	0.922079

Tabel 2. Hasil perhitungan kinerja algoritma topologi *dumbbell* kedua

Algoritma	y	R	z	u

BIC	0.987156	0.071837	0.496263	1.555256
CUBIC	0.998813	0.206208	0.538122	1.743142
HTCP	0.498344	0.293928	0.316936	1.109208

Tabel 1. menunjukan algoritma yang paling baik adalah algoritma dengan nilai 0,14. Algoritma HTCP lebih baik dibandingkan algoritma BIC dan CUBIC dengan nilai 0,9. Algoritma BIC sedikit lebih baik dibandingkan dengan CUBIC dengan nilai 1,6, sedangkan algoritma CUBIC memiliki nilai kinerja 1,8.

Seperti yang terlihat dari Tabel 1. sebelumnya pada Tabel 2. ini menunjukan algoritma yang terbaik, walaupun dengan nilai kinerja 0,7. Algoritma HTCP memiliki nilai kinerja 1,1 sedangkan algoritma BIC memiliki nilai 1,6, dan algoritma CUBIC memiliki nilai 1,7.

4.2 Analisis Topologi *Simple network*

Pengujian pada simulasi ini menunjukan bahwa keempat algoritma memiliki nilai yang selalu berubah-ubah hingga akhir simulasi. Akan tetapi algoritma BIC memiliki nilai akhir *throughput* yang tidak terlalu besar perbedaannya antara *node*-*node*-nya. Kemudian algoritma CUBIC membutuhkan waktu yang lama untuk mencapai nilai *throughput* yang mendekati stabil. Sedangkan algoritma HTCP tidak mencapai nilai yang stabil atau mendekati stabil hingga akhir simulasi. Pada algoritma terdapat kegagalan di dalam pengiriman pada *node* ketiga.

Hasil analisis kinerja algoritma pada pengujian topologi *simple network* dapat ditunjukan pada Tabel 3.

Tabel 3. Hasil perhitungan kinerja algoritma topologi *simple network*

Algoritma	y	R	z	u
BIC	0.107054	0.292323	0.064604	0.463982
CUBIC	0.202931	0.259688	0.126967	0.589586
HTCP	0.137327	0.37138	0.111944	0.62065

Berdasarkan Tabel 3. menunjukan bahwa algoritma BIC memiliki kinerja yang paling baik jika dibandingkan dengan algoritma yang lainnya yakni memiliki nilai 0,46. Kemudian algoritma yang terbaik kedua ialah algoritma CUBIC dengan nilai 0,59. Algoritma HTCP memiliki kinerja yang lebih baik jika dibandingkan dengan , algoritma HTCP memiliki nilai kinerja 0,62. Algoritma memiliki kinerja yang paling buruk pada topologi ini dengan memiliki nilai 0,77.

Dari pengujian yang dilakukan dapat dikatakan bahwa algoritma HTCP merupakan algoritma yang terbaik di dalam simulasi. Nilai kinerja memiliki nilai yang terkecil untuk pengujian secara keseluruhan. Hal ini menunjukan pengujian di dalam jaringan computer yang mencoba mensimulasikan jaringan Internet algoritma HTCP lebih baik jika dibandingkan dengan algoritma BIC dan CUBIC.

DAFTAR PUSTAKA

Chiu D.M., Jain R., Analyis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks, Computer Networks and ISDN Sistems, 1989

Even B., An Experimental Investigation of TCP Performance in High Bandwidth-Delay Product Path, Thesis, Hamilton Institute, Maynooth, Ireland, 2007.

Ha S., Rhee I., *CUBIC: A New TCP-Friendly High-Speed TCP Variant,* International Workshop on Protocols for and Long Distance Networks, 2005.

Leith D., Shorten R., "H-TCP Protocol for High-Speed Long Distance Networks", PFLDnet, 2004.

Leith D., Shorten R.N., McCullagh G., "*Experimental Evaluation of CUBIC-TCP,* PFLDnet, 2007.

Rhee I., Xu L., *CUBIC: A New TCP-Friendly High-Speed TCP Variant,* North Carolina State University, University of Nebraska-Licoln, USA: 2004.

Tanenbaum A. S., *Computer Network, Fourth Edition,* Pearson Education International, Upper Saddle River, New Jersey, USA: Prentice Hall, 2003.

Xu L., Khaled H., Rhee I., *Binary Increase Congestion Control for , Long Distance Networks,* Paper, North Carolina state University, Releigh, NC, USA, 2003.

Wang G., Xia Y., Harisson D., *An NS2 TCP Evaluation Tool,* Internet Engineering Task Force, USA, 2007.

Wei D.X., Cao P., NS-2 TCP-Linux: An NS-2 TCP Implementation with Congestion Control Algorithm from Linux, ACM, 2006.

Zegura W. E., Calvert L. K., Danahoo J. M., A Quantitative Comparison of Graph-based Models for Internet Topology, GT-ITM, 1997.