ANALISIS PERFORMANSI CONGESTION CONTROL DENGAN CADPC/PTP

Yans Auliyansyah[1], Vera Suryani[2], [3]

[1]Teknik Informatika, Fakultas Teknik Informatika, Universitas Telkom

Abstrak

Pada tugas akhir ini dilakukan pengujian performansi skema congestion control yang disebut ”Congestion Avoidance with Distributed Proportional Control (CADPC) yang dikombinasikan dengan ”Performance Transparency Protocol (PTP)” atau disebut CADPC/PTP. Untuk menguji performansinya, dilakukan perbandingan dengan TCP Reno pada topologi single-bottleneck, double-bottleneck, parking-lot, dan multihop. Parameter uji yang digunakan adalah throughput, fairness, dan average queue length (AQL). Dari hasil pengujian didapatkan hasil throughput dari TCP Reno lebih baik dibandingkan CADPC/PTP jika dilihat dari rata-rata throughput pada setiap simulasi, namun fairness index yang dihasilkan tidak lebih baik daripada CADPC/PTP yang semakin naik walaupun topologinya semakin kompleks. Average queue length pada CADPC/PTP pun sangat kecil bila dibandingkan dengan TCP Reno. Secara umum, TCP Reno masih lebih baik pada realibilitas dan throughput yang dihasilkan pada masing-masing simulasi.

Kata Kunci : Congestion Control, Analisis Performansi, CADPC/PTP

Abstract

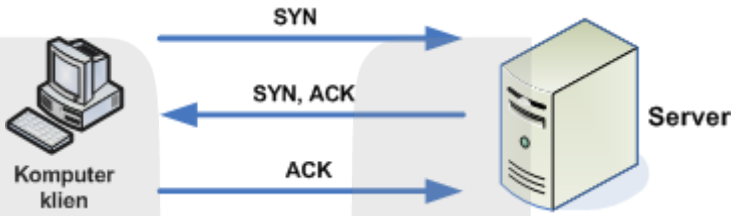
This final task, comparing the performance of congestion control scheme called "Congestion Avoidance with Distributed Proportional Control (CADPC)” combined with "Performance Transparency Protocol (PTP)" - CADPC/PTP with TCP Reno. Single-bottleneck, double-bottleneck, parking-lot, and multihop topologies are used to compare the performance of these two algorithms. The test parameters used were throughput, fairness, and average queue length (AQL). In tests performed show that the throughput of TCP Reno is better than CADPC/PTP -based on average throughput- for each simulation, but CADPC/PTP has better fairness index for each simulation and getting better in more complex topologies. Average queue length at CADPC was very small when compared with TCP Reno. Overall, TCP Reno is still better based on reliability and throughput generated from each simulation.

Keywords : Congestion Control, Performance Analysis, CADPC/PTP, TCP Reno

BAB II
LANDASAN TEORI

2.1TRANSMISSION CONNECTION PROTOCOL (TCP)

Transmission Connection Protocol (TCP) adalah protokol yang memungkinkan program-program aplikasi untuk mengakses/menggunakan layanan komunikasi bersifat *connection-oriented*. TCP mampu memberikan jasa pengiriman yang dapat diandalkan (*reliable*) sekaligus bersifat *flow-controlled*. Sifat *flow-controlled* ini memungkinkan peralatan-peralatan jaringan yang berkecepatan rendah (*slower-speed network devices*) dapat berhubungan dengan peralatan-peralatan jaringan yang berkecepatan tinggi (*higher-speed network devices*). Protokol ini menggunakan proses *three way handshake* dalam proses pembuatan koneksinya. Prosesnya antara lain:



Gambar 2.1 Three-way Handshake

1. *Host* pertama (yang ingin membuat koneksi) akan mengirimkan sebuah segmen TCP dengan *flag* SYN diaktifkan kepada *host* kedua (yang hendak diajak untuk berkomunikasi).
2. Host kedua akan meresponsnya dengan mengirimkan segmen dengan *acknowledgment* dan juga SYN kepada *host* pertama.
3. Host pertama selanjutnya akan mulai saling bertukar data dengan *host* kedua.

2.1.1 Karakteristik TCP

TCP memiliki karakteristik sebagai berikut:

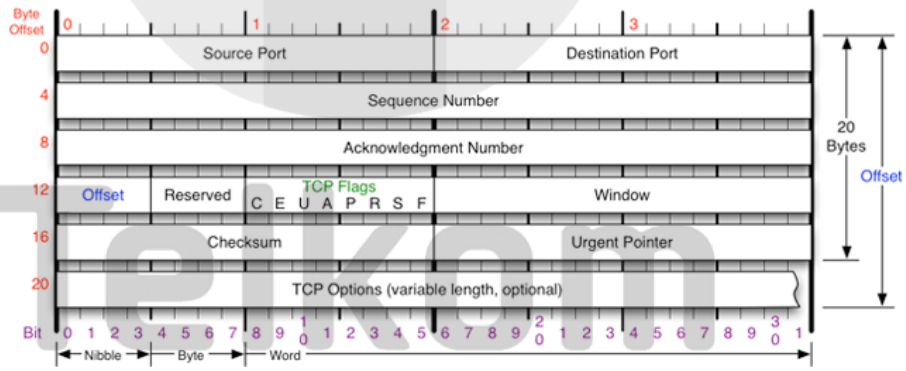
- *Connection-Oriented*: Sebelum data dapat ditransmisikan antara dua host, dua proses yang berjalan pada lapisan aplikasi harus melakukan negosiasi untuk membuat sesi koneksi terlebih dahulu. Koneksi TCP ditutup dengan menggunakan proses terminasi koneksi TCP (TCP *connection termination*).
- *Reliable*: Data yang dikirimkan ke sebuah koneksi TCP akan diurutkan dengan sebuah nomor urut paket dan akan mengharapkan paket *positive acknowledgment* dari penerima. Jika tidak ada paket *Acknowledgment* dari penerima, maka segmen TCP (*protocol data unit* dalam protokol TCP) akan ditransmisikan ulang. Pada pihak penerima, segmen-segmen duplikat akan diabaikan dan segmen-segmen yang datang tidak sesuai dengan urutannya akan diletakkan di belakang untuk mengurutkan segmen-segmen TCP. Untuk

menjamin integritas setiap segmen TCP, TCP mengimplementasikan penghitungan TCP Checksum.

- *Byte stream*: TCP melihat data yang dikirimkan dan diterima melalui dua jalur masuk dan jalur keluar TCP sebagai sebuah byte stream yang berdekatan (kontigu). Nomor urut TCP dan nomor *acknowlegment* dalam setiap header TCP didefinisikan juga dalam bentuk byte. Untuk melakukannya, hal ini diserahkan kepada protokol lapisan aplikasi (dalam DARPA Reference Model), yang harus menerjemahkan byte stream TCP ke dalam "bahasa" yang ia pahami.
- *Efficient Flow Control*: Untuk mencegah data terlalu banyak dikirimkan pada satu waktu, yang akhirnya membuat "macet" jaringan internetwork IP, TCP mengimplementasikan layanan flow control yang dimiliki oleh pihak pengirim yang secara terus menerus memantau dan membatasi jumlah data yang dikirimkan pada satu waktu.
- *Full-Duplex Operation:* TCP bisa mengirim dan menerima dalam waktu yang bersamaan.
- *Multiplexing*: komunikasi antar upper-layer yang terjadi secara simultan bisa dimultiplexikan melalui satu koneksi tunggal.

2.1.2 TCP Segment

Sebuah data unit TCP disebut sebagai *segment*. Sebuah segmen TCP terdiri atas sebuah *header* dan segmen data (*payload*), yang dienkapsulasi dengan menggunakan header IP dari protokol IP. Berikut ini merupakan format dari *segment* TCP



Gambar 2.2 TCP Header

- *Source Port* (16 bit) dan *Destination Port* (16 bit): menunjukan *port* sumber dan port tujuan untuk mengidentifikasi sambungan *end-to-end* dan aplikasi pada lapisan yang lebih tinggi.
- *Sequence Number* (32 bit): berisi nomor urut *byte* pertama dari segmen dari aliran *byte* yang dikirim. Karena nomor urut mengacu pada jumlah *byte* dan bukan jumlah segmen, nomor urut dalam TCP segmen biasanya tidak berurutan.

- *Acknowledgment Number* (32 bit): Digunakan oleh pengirim untuk memberikan *acknowledge* penerimaan data; kolom ini menunjukan nomor urut dari *byte* data selanjutnya yang diharapkan di terima.
- *Data Offset* (4 bit): Menunjuk pada byte data pertama dari segmen TCP; kolom ini menunjukan panjang *header* segmen TCP.
- *Reserved* (6 bit): Direservasikan untuk digunakan pada masa depan. Pengirim segmen TCP akan mengeset bit-bit ini ke dalam nilai 0.
- *Control Flags* (6 bit): Sekumpulan bit yang mengontrol beberapa aspek dari sambungan virtual TCP. Bit control ini termasuk:
  1. *Urgent Pointer Field Significant* (URG): Jika di set, menunjukan bahwa segmen yang dikirim berisi data *urgent* (atau prioritas tinggi) dan kolom *Urgent Pointer* adalah valid.
  2. *Acknowledgment Field Significant* (ACK): Jika di set, menunjukan bahwa nilai yang ada di kolom *Acknowledgment Number* adalah valid. Bit ini biasanya di set, kecuali pada saat pertama kali sambungan dilakukan.
  3. *Push Function* (PSH): Digunakan pada saat aplikasi sumber / pengirim menginginkan untuk memaksa TCP segera mengirimkan data yang saat ini ada di *buffer* tanpa menunggu *buffer* penuh; Hal ini sangat berguna untuk mengirimkan data pendek / kecil.
  4. *Reset Connection* (RST): Jika di set, akan segera memutuskan sambungan TCP *end-to-end*.
  5. *Synchronize Sequence Numbers* (SYN): Menset awal segmen pada saat proses / membentuk sambungan, menunjukan bahwa segmen tersebut membawa nomor urut awal.
  6. *Finish* (FIN): Memohon untuk pemutusan hubungan TCP secara normal; agar hubungan benar-benar terputus kedua belah pihak harus mengirimkan segmen FIN.
- *Window* (16 bit): Digunakan untuk *control flow*, berisi nilai dari *receive window size* yang menentukan maksimum *byte data* / paket yang dapat di terima oleh penerima sekali kirim / sekaligus.
- *Checksum* (16 bit): Memberikan bit tambahan untuk mendeteksi kesalahan pada segmen TCP (termasuk *header* dan data).
- *Urgent Pointer* (16 bit): Data urgent adalah informasi yang telah di beri tanda sebagai data dengan prioritas tinggi oleh aplikasi di lapisan atas; data ini biasanya akan mem-*bypass* urutan data TCP yang normal dan biasanya di letakan dalam segmen antara header dan data yang normal. *Urgent Pointer* yang hanya valid pada saat URG bit di set, menunjukan posisi dari octet pertama dari data yang tidak urgen dalam segmen.
- *Options* (32 bit): Digunakan pada saat negosiasi proses pembentukan hubungan. Ada banyak option yang dapat di negosiasi. *Maximum Segment Size* (MSS) adalah option yang paling sering di negosiasikan, dan jika tidak dilakukan maka nilai default MSS adalah 536. Option lain yang sering digunakan adalah *Selective Acknowledgement* (SACK), yang memungkinkan untuk menerima segmen yang tidak mengikuti urutan yang benar.

- Data: berisi tentang informasi dari *upper-layer*.

2.2 FILE TRANSFER PROTOCOL FTP

File Transfer Protocol (FTP) adalah protokol jaringan standar yang digunakan untuk menyalin file dari satu host ke yang lain melalui jaringan berbasis TCP, seperti Internet. FTP dibangun pada arsitektur *client-server*, menggunakan saluran yang berbeda untuk control (port 20) dan koneksi data antara *client* dan *server* (port 21). FTP *user* (*client*) dapat berhubungan dengan *server* menggunakan otentikasi *username* dan *password* ataupun dapat terhubung secara *anonymous* jika server dikonfigurasi untuk bisa melakukan hal tersebut.

2.3 CONGESTION CONTROL

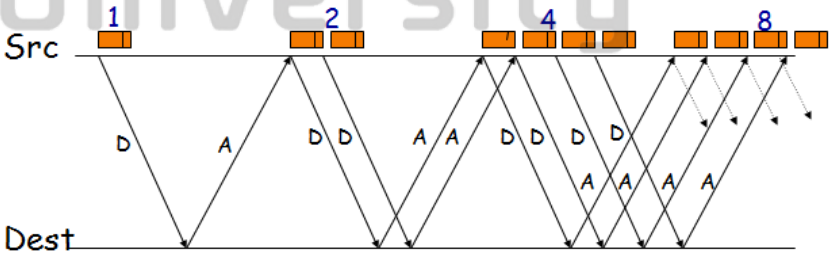
Congestion dapat didefinisikan sebagai keadaan atau kondisi yang terjadi ketika sumber daya jaringan kelebihan beban dan mengakibatkan gangguan pada pengguna jaringan. Hasilnya adalah berkurangnya utilitas jaringan [1]. Dengan kata lain, suatu jaringan dikatakan kongesti, jika kualitas layanan yang dirasakan oleh pengguna menurun karena kenaikan beban jaringan [3]. Sedangkan *congestion control* adalah algoritma yang digunakan untuk membagi sumber daya jaringan secara adil (fair) dalam lalu lintas data pada jaringan tertentu.

2.4 TCP RENO

TCP Reno didefinisikan sebagai TCP yang berisi beberapa algoritma *Congestion Control*. Algoritma ini disebut juga dengan Agoritma Kendali Kongesti TCP yang terdiri dari: *Slow Start*, *Fast Retransmit* dan *Fast Recovery*, serta *Congestion Avoidance*.

2.4.1 Algoritma *Slow Start*

Slow start mengizinkan TCP memeriksa kondisi jaringan dengan menaikkan secara perlahan data yang ditransmisikan ke dalam jaringan. Algoritma *slow start* menggunakan *congestion window* untuk mengontrol *flow data*. *Cwnd* diinisialisasi ke satu segmen, biasanya 512 *bytes*. Prinsip *slow start* sederhana, bahwa untuk setiap ACK yang diterima, menambahkan satu segmen ke cwnd. Proses slow start dapat dilihat pada Gambar dibawah



Gambar 2.3 Algoritma *Slow Start*

Pengirim dapat mengirim *congestion windows minimum*, atau *ssthresh*. *Ssthresh* diinisialisasi ke window yag diperlihatkan penerima. Saat *cwnd* lebih besar atau sama dengan nilai *ssthresh*, koneksi memasuki fase *congestion avoidance*. Jika kapasitas jaringan dapat dipenuhi sebelum *cwnd* lebih besar dari *ssthresh*, maka *gateway* akan memberi sinyal kongesti dengan membuang segmen dan TCP akan memasuki fase *retransmit* setelah tiga ACK duplikat.

2.4.2 *Congestion Avoidance*

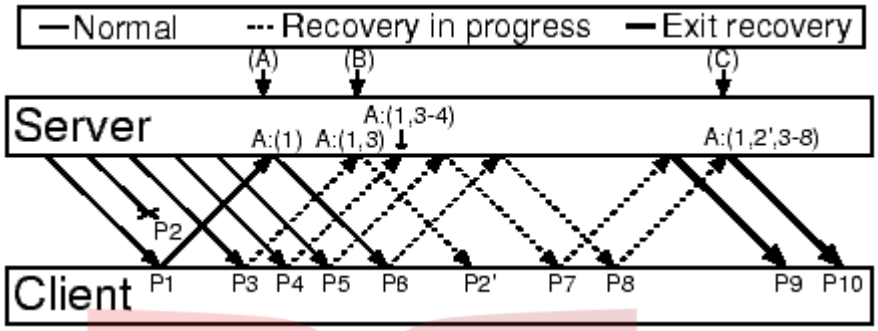
Jalur bottleneck dapat terjadi saat saluran besar terhubung ke saluran kecil. Kongesti terjadi saat volume segmen dapat melampaui *buffer space gateway*. *Gateway* akan terus membuang segmen sampai *buffer space* tersedia. Proses ini memberi sinyal kongesti pada koneksi TCP melalui ACK duplikat atau *retransmission timeout*. Saat kongesti terjadi, koneksi melakukan *recovery* lalu memasuki *congestion avoidance*. Jika *retransmission timeout* terjadi, *cwnd* diset ke satu MSS. Saat *cwnd* > *ssthresh*, fase *slow start* selesai, dan *congestion avoidance* mengambil alih. Saat fase *congestion avoidance*, cwnd tidak akan pernah dipecah lebih dari satu segmen per RTT, jika semua segmen dalam window telah di-ACK. Ini merupakan laju pertumbuhan linear bila dibandingkan dengan laju pertumbuhan eksponensial *slow start*.

2.4.3 *Fast Retransmit* dan *Fast Recovery*

Request Time Out (RTO) terjadi karena mendeteksi *lost segment* dalam TCP. Saat *timeout* terjadi, TCP akan kembali kepada fase *slow start* dan *retransmit* segmen yang hilang. Ini menimbulkan retransmisi yang tidak perlu dari segmen *out-of-order* yang diterima. Yaitu segmen yang sedang disimpan di *buffer receiver*. Jacobson mengajukan fase *fast retransmit* dan kembali ke *slow start*. *Fast retransmit* terjadi bila tiga ACK duplikat memicu retransmisi paket yang hilang.

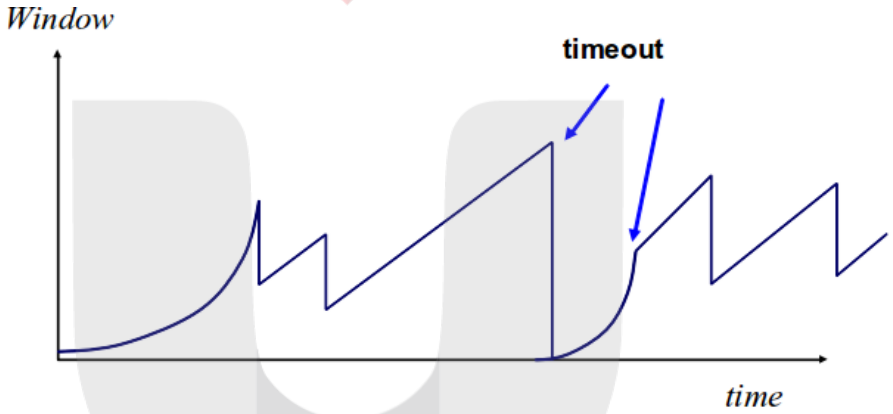
Prinsip kerja dari *fast retransmit* adalah bahwa kebanyakan segmen *out-of-order* akan muncul setelah satu atau dua segmen. Setelah tiga ACK duplikat, telah aman jika diasumsikan segmen telah hilang. Jika pengirim sedang menerima ACK duplikat, maka data yang sedang melalui jaringan dan kembali ke slow start tidak dibutuhkan, sehingga *fast recovery* dapat dilakukan.

Cwnd akan di-update dengan satu MSS untuk setiap ACK duplikat tambahan yang diterima. Jika cwnd mengijinkan, pengirim akan mentransmit segmen baru kedalam jaringan. Segmen baru ini akan memicu ACK tambahan yang mungkin mengijinkan fase fast retransmit lainnya jika segmen tambahan telah hilang. Bila sepotong data baru di ACK, *congestion window* di set sama dengan ssthresh. Pengirim kemudian memasuki fasa *congestion avoidance* Proses *fast retransmit* ini dapat dilihat pada Gambar dibawah ini



Gambar 2.4 Fast Retransmit/Recovery

Dari semua mekanisme yang ada, maka TCP Reno bisa digambarkan ke dalam grafik TCP Saw Tooth behavior, seperti gambar di bawah ini:



Gambar 2.5 TCP Saw Tooth Behaviour

2.5 CADPC/PTP

Skema ini terdiri atas dua bagian, yaitu *Congestion Avoidance with Distributed Proportional Control* (CADPC) yang berfungsi sebagai mekanisme *Congestion Control* dan *Performance Transparency Protocol* (PTP) berfungsi sebagai protokol *signalling* yang digunakan untuk mendapatkan feedback yang diharapkan – *available bandwidth*.

Cara kerja dari CADPC/PTP adalah

1. Pengirim membangkitkan paket PTP yang kemudian dikirim pada penerima. Paket ini diletakan di atas IP untuk mendeteksi keadaan pada router menggunakan *Router Alert*. Kemudian untuk *Forward Packet Stamping*, paket PTP membawa informasi yang dikirim dari sumber ke tujuan dan diperbarui oleh router-router yang berada diantara pengirim dan tujuan. Di sisi lain penerima membangun tabel entri router, mendeteksi informasi yang relevan dan umpan kembali ke pengirim.

2. CADPC membutuhkan informasi tentang *Available Bandwidth* pada jaringan. Dengan menggunakan PTP, artinya setiap intermediate router harus menambahkan beberapa informasi, yaitu:
   - Alamat dari interface jaringan
   - *Timestamp*
   - *Bandwidth nominal link* ("ifSpeed" objek "*Management Information Base* (MIB)" dari router)
   - Sebuah *counter byte* ("ifOutOctets" atau "ifInOctets" dari MIB router)
3. Pada penerima, dua paket berurutan diperlukan untuk menghitung *bandwidth* yang tersedia diantara periode pengiriman kedua paket tersebut. Kemudian, *bandwidth* (β), *traffic* (*λ)*, dan interval (δ) dari dataset yang memiliki sisa *bandwidth* terkecil akan menjadi feedback bagi pengirim.

CADPC adalah varian dari *Congestion Avoidance with Proportional Control* (CAPC). CAPC menaikan dan menurunkan *rate* dari pengiriman data secara proposional berdasarkan *total network traffic* yang ada, apakah berada di bawah atau di atas dari "*target rate*" yang telah ditetapkan sebelumnya. CADPC melakukan hal yang sama dengan membandingkan hubungan antara bandwidth yang tersedia pada jaringan dengan banyaknya aliran data. Dalam istilah matematika, tingkat x (t +1) dari pengirim dihitung sebagai

$$\boldsymbol{x(t+1) = x(t)(2 - \frac{x(t)}{\beta(t)} - \frac{\lambda(t)}{\beta(t)})} \quad (2.1)$$

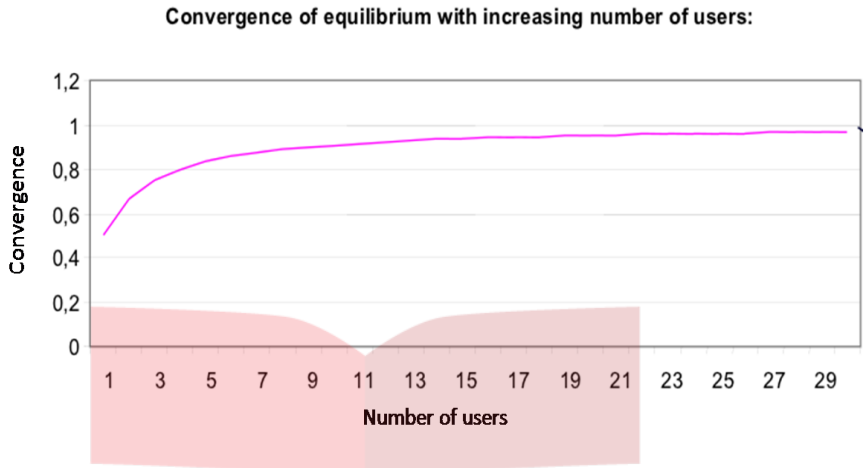
Saat feedback tiba, *old rate* x(t) dapat berubah menjadi kecil, tapi tidak mencapai nol. Dengan menggunakan model *fluid-flow*, synchronous RTTs dan n user, maka persamaan di atas menjadi

$$\boldsymbol{x(t+1) = x(t)(2 - x(t) - \lambda(t))} \quad (2.2)$$

(normalisasi B = 1), yang merupakan formula dari *logistic growth* dan mempunyai titik equilibrium tak stabil pada x = 0 (walaupun aturan mengatakan rate tidak boleh mencapai nol) dan titik ekuilibrium stabil asimtotik pada x = 1/(1+n). Dengan demikian total traffic dalam jaringan menjadi

$$\boldsymbol{nx = \frac{n}{1+n}} \quad (2.3)$$

Ini berarti rate dari CADPC hanya akan terisi setengah dari kapasitas *bottleneck* untuk satu flow, tapi akan menuju 100/101 dari kapasitas bottleneck jika terdiri dari seratus flow. Dengan kata lain semakin banyak flow maka rate akan konvergen mendekati kapasitas bottleneck yang ada pada jaringan tersebut.[3] Hal tersebut dapat digambarkan seperti gambar di bawah ini:



Gambar 2.6 Konvergensi CADPC/PTP

2.6 UJI PERFORMANSI

2.6.1 Bandwidth dan Throughput

Bandwidth adalah ukuran dari banyaknya informasi/data yang dapat mengalir dari suatu node ke node lain pada waktu tertentu. Satuan yang dipakai untuk bandwidth adalah bits per second atau disingkat bps. Umumnya *bandwidth* sudah terdefenisi pada saat pembangunan jaringan komputer biasanya sesuai dengan kartu jaringan yang dipakai (*ethernet, fiber optic*).

Bandwidth tidak menggambarkan keadaan sebenarnya yang terjadi pada jaringan. Untuk itulah muncul besaran *throughput*. *Throughput* adalah *bandwidth* aktual yang terukur pada suatu ukuran waktu tertentu dalam suatu jaringan. Walaupun memiliki satuan dan rumus yang sama dengan bandwidth, tetapi throughput lebih pada menggambarkan bandwidth yang sebenarnya (aktual) pada suatu waktu tertentu dengan ukuran data tertentu. *Bandwidth* ataupun *Throughput* dirumuskan :

$$\text{Throughput} = \frac{\text{Ukuran total data } (kilobyte)}{\text{Waktu transfer data } (second)} \quad (2.4)$$

Beberapa hal yang menyebabkan perbedaan *bandwidth* dengan *throughput* antara lain spesifikasi *hardware* dan *software*, topologi jaringan, kondisi lingkungan (listrik, cuaca), ukuran dan tipe data yang ditransfer, serta banyaknya pengguna jaringan.

2.6.2 Jain's Fairness

Index Fairness Jain berdasarkan pada

$$\boldsymbol{fairness\ index} = \frac{(\sum_{i=1}^{n} xi)^2}{n \sum_{i-1}^{n} xi^2} \quad (2.5)$$

dimana x{i} adalah *throughput* dari sesi (i) pada link bottleneck, dan (n) adalah jumlah sesi yang berbagi link bottleneck. Indeks *fairness*

berada diantara 1/n hingga 1. Partisi equal lokal dari *bandwidth* mencapai indeks = 1. Jika hanyak dari n aliran menerima bandwidth yang sama (dan yang lain tidak mendapat apa-apa) maka indeksnya adalah k/n.

Beberapa sifat dar indeks ini meliputi: tidak tergantung dari jumlah populasi, skala, dan metric, bernilai 0-1, dan kontinu.

2.6.3 Average Queue Length (AQL)

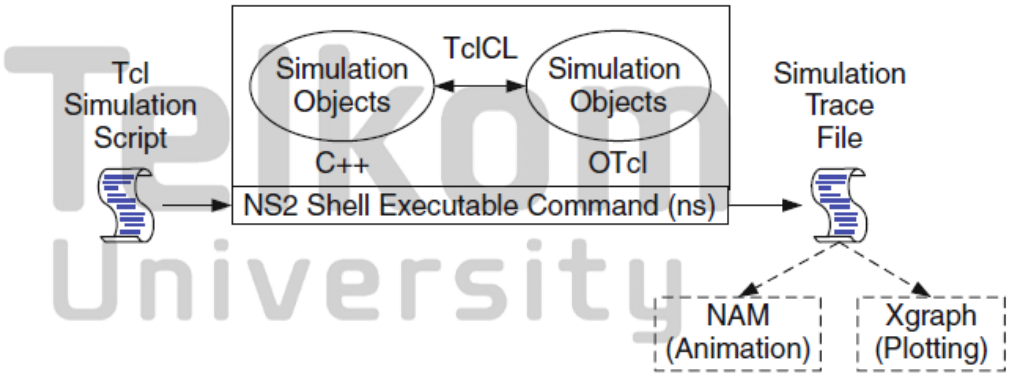
Average Queue Length adalah rata-rata dari jumlah paket dalam antrian yang sedang menunggu untuk ditransmisikan oleh *interface* jaringan. Berikut ini formula untuk menghitung AQL(t) :

$$\boldsymbol{AQL(t) = \alpha * AQL(t-1) + (1-\alpha) * QL(t)} \quad (2.6)$$

Dimana QL(t) merupakan panjang antrian pada saat (t) dan AQL(t-1) adalah rata-rata panjang antrian sebelumnya. *Metric* ini menggambarkan tingkat kesulitan node untuk mengirim paket ke suatu jaringan.

2.7 NETWORK SIMULATOR (ns-2)

NS-2 merupakan *network simulator* yang digunakan untuk melakukan simulasi terhadap *discrete event* yang berisi berbagai macam *library*, *network element*, *traffic model* dan berbagai protokol. ns-2 besifat *object-oriented* dan dibangun dengan menggunakan bahasa pemrograman C++ dan OTcl. C++ mempunyai keunggulan dalam kompilasi dan eksekusi program sedangkan OTcl mempunyai keunggulan dalam interpretasi dari Tcl *scripts* yang digunakan untuk skenario simulasi.



Gambar 2.7 Arsitektur NS-2

Hasil keluaran dari simulasi menggunakan ns adalah file

- *.nam* : merupakan keluaran dari hasil simulasi yang dapat dilihat secara grafis dengan menggunakan network animator .nam

- *.tr* : merupakan keluaran yang berbasis text. Biasa digunakan untuk melihat apa yang terjadi saat simulasi per satuan waktu, hasilnya dapat digunakan untuk menampilkan parameter lain seperti *throughput*, *fairness*, *packet loss* dan lain-lain dalam bentuk grafik menggunakan aplikasi pengolah data (xgraph, gnuplot, MS Excel).

BAB V
KESIMPULAN DAN SARAN

5.1 KESIMPULAN

Berdasarkan dari hasil proses implementasi, pengujian, dan analisis maka dapat ditarik kesimpulan sebagai berikut:

1. Hasil simulasi menunjukkan performansi antara dua algoritma *congestion control*, yaitu CADPC/PTP dan TCP Reno. Dari 3 topologi awal (*Single-bottleneck, Double-bottleneck,* dan *Parking-lot*) TCP Reno mempunyai *throughput* yang lebih baik jika dilihat dari rata-rata *throughput*-nya, namun terjadi perbedaan *throughput* yang cukup jauh (pada *node* yang mempunyai hop sedikit dan *node* yang meiliki hop lebih banyak untuk sampai ke tujuan) ketika diimplementasikan ke dalam skenario yang lebih kompleks (*Parking-Lo*t).

   Sedangkan CADPC/PTP membagi *throughput* ke dalam angka yang hampir sama besar untuk setiap aliran data pada setiap skenario, namun memiliki utilitas *bandwidth* yang lebih kecil dibandingkan TCP Reno.

   Hal tersebut disebabkan, pada TCP Reno node yang mempunyai RTT lebih kecil (sedikit hop) akan mendapat *sharing bandwidth* yang lebih besar. Sedangkan pada CADPC/PTP, adanya mekanisme PTP *signaling* yang dilakukan untuk memberikan *feedback* berupa *available bandwidth*, menjadikan setiap *flow* mendapatkan rate data yang sama dalam pengiriman datanya.

   Begitu pula pada kondisi Multihop dengan variasi sender (5-30) yang mengirimkan data melewati 10 router (sehingga mempunyai RTT yang besar), TCP Reno masih lebih unggul daripada CAPDC/PTP dalam hal *throughtput* dan utilitas *bandwidth*-nya. Walaupun CADPC/PTP terus menunjukkan tren yang meningkat para kondisi *sender* yang semakin banyak dengan konvergensi n/(n+1), tetapi secara keseluruhan TCP Reno lebih baik.

2. Sedangkan dalam Fairness, seperti ditunjukkan pada hasil simulasi, CADPC/PTP menghasilkan fairness index yang lebih baik dibandingkan dengan TCP Reno pada semua skenario. Perbedaannya kecil pada scenario yang sederhana (Single-Bottleneck) tetapi semakin menjauh saat diimplementasikan ke dalam skenario yang lebih kompleks (Parking-Lot).

   Pada topologi Multihop dengan variasi sender (5-30) CADPC/PTP masih jauh lebih baik, karena adanya PTP signaling yang terus memberikan feedback pada agent CADPC, sehingga setiap flow

memperoleh rate hampir sama, sebesar 1(n+1) dari total bandwith yang tersedia, akibatnya fairness indexnya selalu baik dan mendekati 1.
TCP Reno mempunyai fairness yang baik (di atas 0.9) namun tidak sebaik CADPC, karena adanya perbedaan *end to end throughput* dari masing-masing *flow* yang mengalir, hal tersebut dipengaruhi oleh paket yang hilang selama transmisi data, baik dipengaruhi *error model* pada *bottleneck link*, maupun karena router yang melakukan drop paket saat *buffer penuh*. Paket yang di hilang berasal dari salah satu flow secara bergantian (paket yang terakhir datang saat buffer penuh - droptail). Inilah yang menyebabkan *fairness*-nya sedikit lebih rendah dari CADPC/PTP. Secara keseluruhan CADPC/PTP lebih baik dari segi *fairness* dibandingkan dengan TCP Reno.

3. Hasil simulasi menunjukkan bahwa *Average Queue Length* (AQL) dari CADPC/PTP selalu lebih kecil daripada AQL pada TCP Reno dalam semua skenario. Ini disebabkan adanya PTP signaling yang selalu mengecek ketersediaan bandwidth dan tidak adanya skema *retransmission* pada CADPC/PTP sehingga antrian tidak pernah penuh.

   Sedangkan TCP Reno mempunyai AQL yang cukup besar karena menerapkan algoritma *Slow Start* saat awal transmisi data, kemudian *Additive Increase Multiplicative Decrease* (AIMD) dan *Fast Retransmit/Recovery* bila terjadi *duplicate ACK*, walaupun sebenarnya bisa saja paket yang ditransmisikan ulang masih berada pada jaringan. Hal itulah yang bisa membuat antrian pada router menjadi penuh. Dalam pemanfaatan *buffer router* TCP Reno lebih baik.

4. CADPC/PTP lebih baik dalam Fairness dan lebih mudah mengirimkan data (AQL-nya kecil) namun dari segi utilitas *bandwidth*, pemanfaatan *buffer router* dan *end to end throughput* yang dihasilkan, TCP Reno lebih baik. Selain itu TCP Reno pun memiliki skema ACK untuk setiap paket yang berhasil dikirim dan skema *packet retransmission* sehingga dapat menjamin reliabilitas terhadap data yang dikirimkan. Dari beberapa hasil pengujian, maka TCP Reno masih lebih baik dibandingkan CADPC/PTP pada semua kondisi simulasi yang telah dilakukan dalam tugas akhir ini.

5.2 SARAN

Saran yang dapat diajukan untuk penelitian lebih lanjut adalah:

1. Menerapkan skema retransmission pada CADPC/PTP sehingga realibilitasnya lebih baik.
2. Mengganti signaling PTP dengan SNMP yang mungkin saja lebih buruk dari dari efisiensi namun tidak perlu mengupdate software pada router. [9]
3. Menerapkan algoritma ini pada wireless link.[10]

DAFTAR PUSTAKA

[1] E. He, R. Kettimuthu, Y. Gu, S. Hegde, M. Welzl, P. Vicat-Blanc, J. Leigh, C. Xiong. Survey of Protocols and Mechanisms for Enhanced Transport over LONG FAT PIPES, 2003.

[2] J. Yves. Rate adaptation, Congestion Control and Fairness, 2008.

[3] M. Mathis, S Jefreyy. The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm, 1997.

[4] Mukul Goyal, Roch Guerin and Raju Rajan. Predicting TCP Throughput From Non-invasive Data, 2001.

[5] N. Osipova, A. Blanc, and K. Avrachenkov. Improving TCP fairness with MarkMax policy, 2004.

[6] P. Rusmin, C. Machbub, A. Harsoyo, dan Hendrawan. Sistem Kendali Kongesti Di Internet, 2008.

[7] S. Hauber, M. Scharf, J. K¨ogel, C. Suriyajan. Evaluation of Router Implementations for Explicit Congestion Control Schemes, 2010

[8] Tanenbaum, A.S., 2003, *Computer Networks, Fourth Edition*, Prentice Hall.

[9] Welzl, M. Scalable Router Aided Congestion Avoidance for Bulk Data Transfer in High Speed Networks, 2003.

[10] Welzl, M., 2005, *Network Congestion Control : Managing Internet Traffic*, John Wiley & Sons, Inc.