# ANALISIS PERBANDINGAN UNJUK KERJA ALGORITMA *CONGESTION CONTROL* PADA TCP TAHOE, RENO DAN SACK (*SELECTIVE ACKNOWLEDGMENT*)

**TUGAS AKHIR**

**Diajukan sebagai salah satu syarat untuk menyelesaikan**

**Program Sarjana Komputer**


**Oleh:**

**Nama : Yuliana Wahyu Putri Utami**

**NIM : 09.41020.0079**

**Program : S1 (Strata Satu)**

**Jurusan : Sistem Komputer**





**SEKOLAH TINGGI**

**MANAJEMEN INFORMATIKA & TEKNIK KOMPUTER**

**SURABAYA**


**2013**

*"Kejarlah mimpi setinggi mungkin"*

*"Sampai ke negeri China"*

*"Pendidikan merupakan perlengkapan paling*

*"baik untuk hari tua"*

***(Aristoteles)***



*"Bukan harta kekayaanlah, tetapi budi pekerti yang harus"*

*"ditingalkan sebagai pusaka untuk anak – anak kita"*

Kupersembahkan kepada

Allah SWT.

Rasulullah, keluarga beserta sahabat,

Alm. Bapak yang ku rindukan, Mama tercinta, kakak dan adik tersayang

Beserta semua keluarga yang mendukung

**Tugas Akhir**

# ANALISIS PERBANDINGAN UNJUK KERJA ALGORITMA *CONGESTION CONTROL* PADA TCP TAHOE, RENO DAN SACK (*SELECTIVE ACKNOWLEDGMENT*)

dipersiapkan dan disusun oleh

**Yuliana Wahyu Putri Utami**

**NIM : 09.41020.0079**

Telah diperiksa, diuji dan disetujui oleh Dewan Penguji

pada : Maret 2013

**Susunan Dewan Penguji**

Pembimbing

**I. Dr. Jusak** _____

**II. Anjik Sukmaaji, S.Kom., M.Eng.** _____

Penguji

**I. Yuwono Marta Dinata, S.T., M.Eng.** _____

**II. Ira Puspasari, S.Si., M.T.** _____

Tugas Akhir ini telah diterima sebagai salah satu persyaratan
untuk memperoleh gelar Sarjana

**Pantjawati Sudarmaningtyas, S.Kom, M.Eng, OCA**
**Pembantu Ketua Bidang Akademik**

SEKOLAH TINGGI MANAJEMEN INFORMATIKA & TEKNIK KOMPUTER SURABAYA

# PERNYATAAN

Dengan ini saya menyatakan dengan benar, bahwa Tugas Akhir ini adalah asli karya saya, bukan plagiat baik sebagian maupun apalagi keseluruhan. Karya atau pendapat orang lain yang ada dalam Tugas Akhir ini adalah semata hanya rujukan yang dicantumkan dalam Daftar Pustaka saya.

Apabila dikemudian hari ditemukan adanya tindakan plagiat pada karya Tugas Akhir ini, maka saya bersedia untuk dilakukan pencabutan terhadap gelar kesarjanaan yang telah diberikan kepada saya.

Surabaya, 20 Maret 2013





Yuliana Wahyu Putri Utami
NIM: 09.41020.0079

# ABSTRAK

Kongesti merupakan masalah yang serius dalam jaringan Internet yang dapat mengakibatkan terjadinya kenaikan jumlah paket yang hilang. Selain itu kongesti juga membebani suatu jaringan sehingga dapat memperlambat koneksi jika tidak ditangani dengan baik. Masalah terburuk mungkin akan terjadi kelumpuhan pada jaringan.

Varian TCP terbaik diantara protokol-protokol TCP adalah mampu menjamin layanan yang handal dan konektifitas yang spontan terutama di Internet dengan layanan *highspeed communication* saat ini. Penelitian ini dilakukan melalui perbandingan unjuk kerja TCP varian protokol dalam mengidentifikasi protokol terbaik untuk pengembangan protokol lapisan transport di masa mendatang. Analisis simulasi dilakukan melalui perbandingan algoritma *congestion control* TCP Tahoe, TCP Reno dan SACK di NS2 pada rentang nilai *bandwidth* 0.5Mb/30ms, 0.256Mb/30ms dan 0.128Mb/30ms.

Hasil penelitian menunjukkan bahwa SACK mempunyai nilai rata-rata CongWin tertinggi yaitu 15.3 MSS, 11.3 MSS dan 12.9 MSS untuk masing-masing *bandwidth* 0.5Mb/30ms, 0.256Mb/30ms dan 0.128Mb/30ms, sementara TCP Reno mewarisi kinerja terbaik dalam pemanfaatan *queue*, rendahnya nilai *packet drop* serta pengukuran RTT. Hal ini dapat disimpulkan bahwa SACK mampu mempertahankan nilai CongWin yang lebih baik daripada TCP varian lainnya. Di sisi lain, TCP Reno mampu mengadaptasi variasi *bandwidth* yaitu 0.5Mb/30ms, 0.256Mb/30ms dan 0.128Mb/30ms yang ditunjukkan dalam hal utilisasi *queue* dengan rata-rata nilai terendah yaitu 1219.61 bytes, 2478.54 bytes

dan 3473.33 bytes, *packet drop* dengan nilai terendah yaitu 3120 bytes, 9360 bytes dan 13940 bytes, dan RTT terkecil yaitu 0.44 s, 0.81 s dan 2.08 s. Hasil simulasi dan analisis dalam Tugas Akhir ini ditunjukkan secara grafik.

**Key Words:** TCP Tahoe, TCP Reno, SACK, *Congestion Control*, CongWin, RTT, NS2.

# DAFTAR ISI

Halaman

# DAFTAR TABEL

# DAFTAR GAMBAR

## DAFTAR LAMPIRAN

# BAB I

# PENDAHULUAN

## 1.1 Latar Belakang

Seiring perkembangan teknologi maka diiringi pula dengan banyaknya penggunaan jaringan Internet dengan layanan *highspeed communication*. Untuk mencapai tuntutan tersebut dibutuhkan peningkatan unjuk kerja protokol TCP (*Transmission Control Protocol*) dalam menangani masalah pada jaringan, salah satunya yaitu *congestion*. *Congestion* (kongesti) merupakan masalah yang serius dalam jaringan yang dapat mengakibatkan terjadinya kenaikan jumlah paket yang hilang. Selain itu kongesti juga menyebabkan lambatnya koneksi yang diakibatkan padatnya jalur sehingga apabila tidak ditangani dengan baik maka akan terjadi kelumpuhan pada jaringan tersebut. Untuk itu diperlukan suatu algoritma yang sesuai untuk mempertahankan unjuk kerja ketika terjadi kongesti, disebut dengan *congestion control*.

TCP menggunakan kontrol kongesti untuk membatasi kecepatan pengiriman data. Akan tetapi apabila pengirim mendeteksi bahwa kongesti dalam jaringan telah berkurang, maka pengirim akan meningkatkan kecepatan pengiriman data. Ketika ketiga teknik (*slow start, congestion avoidance, fast retransmit*) digunakan secara bersama-sama, implementasi ini disebut “Tahoe”. Di samping itu, TCP Reno merupakan pengembangan dari TCP Tahoe dimana menambahkan fase *fast recovery* didalamnya sehingga mempengaruhi kecepatan dalam pengiriman. (Agilent Technologies, 2008)

Berbeda dengan TCP *Selective Acknowledgment* (SACK) merupakan strategi dalam menghadapi kehilangan beberapa segmen yaitu melalui *selective acknowledgment*, dimana penerima menginformasikan kepada pengirim tentang semua segmen yang telah berhasil tiba sehingga hanya perlu mengirim ulang segmen yang benar-benar hilang. (Mathias, Mahdavi, Floyd, & Romanow, 1996)

Beberapa *paper* terkait dengan uji perbandingan antara Tahoe, Reno dan SACK oleh Feipeng (2008) hanya melakukan perbandingan algoritma *congestion control* secara umum. Menurut Lui, ketika menerima duplikasi ACK atau *time-out*, Tahoe melakukan fase *retransmit* sedangkan pada Reno melakukan fase *recovery* dan pada SACK mempertahankan informasi *selective acknowledgment* untuk me-*retransmisi* paket-paket yang belum terkirim saja. Selain itu pada *paper* yang lain (Sikdar, 2002) melakukan uji perbandingan pada sisi *latency* dan *steady-state throughput* menjelaskan Reno lebih akurat khususnya untuk transfer jarak pendek pada sisi *latency* sedangkan pada sisi *throughput*, Tahoe tampil lebih baik daripada Reno dan SACK. Dan terakhir pada *paper*(Rahman M,. Kabir, Lutfullah, & Amin, 2008) membandingkan CongWin, *queue*, *packet drop* dan RTT pada TCP Tahoe, Reno, NewReno dan Vegas di mana menjelaskan bahwa TCP Vegas mampu beradaptasi terhadap perubahan *bandwidth* dan kuat terhadap resiko fluktuasi.

Model algoritma *congestion control* TCP yang ada sekarang ditengarai masih belum memenuhi kebutuhan transaksi data pada jaringan Internet saat ini, terutama kebutuhan akan jaringan dengan *highspeed communication* dan juga aplikasi-aplikasi multimedia. Karena itu masih dibutuhkan algoritma alternative

yang dapat memenuhi kebutuhan transaksi data saat ini melalui uji perbandingan unjuk kerja pada ketiga algoritma.

Dalam tugas akhir ini penulis melakukan analisis perbandingan unjuk kerja algoritma *congestion control* TCP Tahoe, Reno dan SACK pada sisi *Congestion Window* (CongWin), *queue,packet drop* dan *Round Trip Time* (RTT). Uji perbandingan pada Tugas Akhir ini dilakukan menggunakan Network Simulator 2 (NS-2). NS-2 merupakan sebuah perangkat lunak simulasi Internet untuk kepentingan riset interaksi antar protokol dalam konteks pengembangan protokol Internet pada saat ini dan masa yang akan datang (Wirawan & Indarto, 2004).

Dengan melakukan perbandingan dan analisis terhadap ketiga algoritma, yaitu dengan membuat simulasi jaringan akan didapatkan perbandingan nilai CongWin, *queue*, *packet drop* dan RTT untuk masing-masing algoritma tersebut. Melalui Tugas Akhir ini nantinya akandiketahui unjuk kerja terbaik dari ketiga algoritma sehingga dapat dijadikan pertimbangan dalam menentukan algoritma yang tepat dalam mengatasi kongesti pada jaringan Internet saat ini, serta dapat meningkatkan unjuk kerja pada jaringan dan membantumengambil keputusan dalam pengembangan algoritma baru pada lapisan *transport* di masa yang akan datang.

## 1.2 Perumusan Masalah

Berdasarkan latar belakang di atas, dapat dirumuskan permasalahan yaitu:

1. Bagaimana membangun simulasi jaringan dengan menggunakan NS-2 untuk melakukan perbandingan unjuk kerja TCP Tahoe, Reno dan SACK.

2. Bagaimana membandingkan dan menganalisis algoritma-algoritma tersebut berdasarkan CongWin, *queue, packet drop* dan RTT.

## 1.3 Pembatasan Masalah

Dalam analisis perbandingan unjuk kerja algoritma kontrol kongesti ini, terdapat beberapa batasan masalah, antara lain:

1. Algoritma yang digunakan adalah TCP Tahoe, Reno dan SACK.
2. Perbandingan yang dilakukan berdasarkan nilai parameter yaitu CongWin, *queue, packet drop* dan RTT.
3. Aplikasi yang digunakan untuk simulasi adalah *Network Simulator 2*.
4. Model topologi yang digunakan adalah *dumb-bell* yang mempunyai dua sumber dan dua tujuan.
5. Model jaringan *unicast* yaitu pengiriman dari satu sumber ke satu tujuan.
6. Simulasi yang dilakukan tidak secara simultan secara terpisah untuk masing-masing algoritma TCP.
7. Simulasi dilakukan dengan satu model variasi berdasarkan *bandwidth* yaitu 0.5Mb/30ms, 0.256Mb/30ms dan 0.128Mb/30ms.
8. Input data trafik berasal dari NS-2 yang dibangkitkan oleh NS-2.
9. Input data trafik yang digunakan adalah aplikasi *File Tranfser Protocol* (FTP) dan trafik generator *Constant Bite Rate* (CBR).
10. Ukuran paket FTP yaitu 1000 bytes sedangkan untuk CBR yaitu 1460 bytes.
11. Monitoring *queue* dilakukan pada jalur *bottleneck* antara node 2 dan node 3.
12. *Packet drop* yang dianalisis hanya pada sisi *bottleneck*.

## 1.4 Tujuan

Dalam analisis perbandingan unjuk kerja algoritma kontrol kongesti ini terdapat beberapa tujuan penulis, antara lain:

1. Membangun simulasi jaringan dengan menggunakan NS-2 untuk melakukan perbandingan unjuk kerja algoritma TCP.
2. Membandingkan dan menganalisis algoritma-algoritma tersebut berdasarkan parameter uji antara lain CongWin, *queue, packet drop* danRTT.
3. Dapat menentukan algoritma dengan unjuk kerja terbaik.

## 1.5 Kontribusi

Analisis Perbandingan Unjuk Kerja Algoritma *Congestion Control* pada TCP Tahoe, Reno dan SACK (*Selective Acknowledgment*) merupakan analisis yang melakukan perbandingan unjuk kerja dari tiga algoritma TCP yaitu TCP Tahoe, Reno dan SACK dengan parameter uji yaitu CongWin, *queue, packet drop* dan RTT yang dilakukan menggunakan NS-2. Tugas Akhir ini diharapkan dapat membantu menentukan algoritma TCP dengan unjuk kerja terbaik sehingga nantinya dapat dijadikan pertimbangan dalam pengembangan algoritma baru lapisan *transport* di masa yang akan datang.

## 1.6 Sistematika Penulisan

Pada penulisan Laporan Tugas Akhir ini ditulis dengan sistematika penulisan sebagai berikut :

**BAB I : Pendahuluan**

Bab I meliputi latar belakang, perumusan masalah, pembatasan masalah, tujuan yang ingin dicapai, kontribusi serta sistematika penulisan laporan tugas akhir ini.

**BAB II : Landasan Teori**

Bab II akan dibahas mengenai TCP/IP model, lapisan *transport*, TCP Tahoe, Reno dan SACK, *congestion control*, *congestion window,* RTT, *packet drop, queue management* dan NS-2.

**BAB III : Metode Penelitian dan Perancangan Sistem**

Bab III akan menjelaskan tentang metode penelitian dalam merancang dan membuat simulasi jaringan pada Algoritma TCP Tahoe, Reno dan SACK serta alasan penggunaan metode penelitian tersebut dalam penelitian. Pada bab ini dijelaskan pula tentang pembuatan yaitu perancangan dan pembuatan simulasi jaringan beserta cara pengambilan data analisis menggunakan NS-2.

**BAB IV : Analisis dan Pembahasan**

Bab IV akan melakukan simulasi terhadap ketiga varian TCP, mengambil data dengan menggunakan bahasa pemograman *perl*, memplotting, membandingkan dan menganalisisnya kemudian hasil analisisnya dibahas sesuai literatur.

**BAB V : Penutup**

Bab V berisi kesimpulan dan saran. Saran yang dimaksud adalah saran terhadap kekurangan dari analisis yang ada kepada pihak

lain yang ingin meneruskan Tugas Akhir ini. Tujuannya adalah agar pihak lain tersebut dapat menyempurnakan analisis sehingga bisa menjadi lebih baik dan berguna.

# BAB II

# LANDASAN TEORI

## 2.1 Model Referensi OSI and TCP/IP

*Open System Interconnection* (OSI) model adalah model referensi pertama yang dikembangkan oleh *International Standart Organization* (ISO) untuk menyediakan kerangka kerja standar untuk menggambarkan tumpukan protokol dalam jaringan komputer. OSI terdiri dari tujuh lapisan, di mana setiap lapisan mempunyai fungsi masing-masing. Tujuh layer OSI adalah *physical layer*, *data link layer*, *network layer*, *transport layer*, *session layer*, *presentation layer*, dan *application layer*. Model OSI ini menentukan apa yang harus dilakukan setiap lapisan, namun tidak menentukan layanan yang tepat dan protokol yang akan digunakan dalam setiap lapisan.

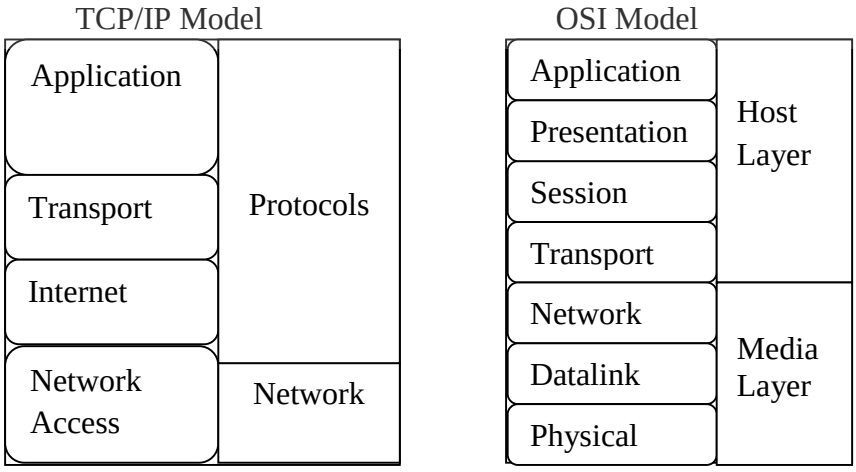
Menurut Issariyakul & Hossain (2012:4-5),*Transmission Control Protocol* (TCP) / *Internet Protocol* (IP) adalah model referensi yang didasarkan pada dua protokol utama, yaitu TCP dan IP, digunakan di *Internet* saat ini. Protokol ini telah terbukti sangat kuat, dan sebagai hasilnya telah digunakan secara luas dan sebagai implementasi pada jaringan komputer yang ada. TCP/IP dikembangkan oleh ARPANET, sebuah jaringan penelitian yang disponsori oleh Departemen Pertahanan AS, yang dianggap sebagai nenek dari semua jaringan komputer. Dalam model TCP/IP, lapisan protokol terdiri dari lima hubungan lapisan- *phyisical*, *data*, *network*, *transport*, dan aplikasi- yang masing-masing bertanggung jawab untuk layanan tertentu. Lapisan aplikasi dalam TCP/IP model

dapat dianggap sebagai kombinasi sesi, presentasi, dan lapisan aplikasi dari model OSI.

Menurut Sukmaaji & Rianto (2008), TCP/IP adalah singkatan dari *Transmission Control Protocol/Internet Protocol*. Dalam hal ini TCP bertugas menerima pesan elektronik dengan panjang sembarang dan membaginya ke dalam bagian-bagian berukuran 64 Kb. Dengan membagi pesan menjadi bagian-bagian, maka perangkat lunak yang mengontrol komunikasi jaringan dapat mengirim tiap bagian dan menyerahkan prosedur pemeriksaan bagian demi bagian.

Apabila suatu bagian mengalami kerusakan selama transmisi, maka pengirim hanya perlu mengulang transmisi bagian itu dan tidak perlu mengulang dari awal. Sedangkan IP mengambil bagian-bagian, memeriksa ketepatan bagian-bagian, pengalamatan ke sasaran yang dituju dan memastikan apakah bagian-bagian tersebut sudah dikirim sesuai dengan urutan yang benar. IP memiliki informasi tentang berbagai skema pengalamatan yang berbeda-beda.






Gambar 2.1 TCP/IP dan OSI Layer(Sukmaaji & Rianto:2008)

### 2.1.1 *Application Layer*

Menurut Issariyakul & Hossain (2012:4-5),lapisan aplikasi duduk di atas tumpukan dan menggunakan layanan dari lapisan transport. Lapisan ini mendukung protokol tingkat tinggi seperti *Hypertext Transfer Protocol* (HTTP) untuk aplikasi *World Wide Web*, *Simple Mail Transfer Protocol* (SMTP) untuk surat elektronik, TELNET untuk terminal virtual remote, *Domain Name Service* (DNS) untuk pemetaan nama host yang alamatnya dikenal jaringan, dan *File Transfer Protocol* (FTP) untuk transfer *file*.

### 2.1.2 *Transport Layer*

Menurut Issariyakul & Hossain (2012:4-5),tujuan dari lapisan transport adalah untuk melakukan kontrol aliran dan kontrol kesalahan untuk transportasi pesan. *Flow control* memastikan bahwa kecepatan transmisi *end-to-end* yang tidak terlalu cepat membuat jaringan padat atau terlalu lambat untuk *underutilize* jaringan. *Error control* memastikan bahwa paket yang dikirimkan ke tujuan dengan benar. Secara umum, ketika sebuah protokol transport menerima pesan dari lapisan yang lebih tinggi. Pesan dibagi menjadi potongan-potongan yang lebih kecil yang menghasilkan PDU-disebut-segmen dengan melampirkan kesalahan dan informasi yang diperlukan kontrol aliran, dan fase segmen ke lapisan yang lebih rendah.

Dua protokol transport yang terkenal, yaitu TCP dan *User Datagram Protocol* (UDP), yang didefinisikan dalam lapisan ini. Sementara TCP bertanggung jawab untuk komunikasi yang dapat diandalkan dan berorientasi koneksi antara dua host, UDP mendukung transportasi *connectionless* tidak dapat

diandalkan. TCP sangat ideal untuk aplikasi yang lebih memilih akurasi atas pelayanan yang cepat dan sebaliknya untuk UDP.

### 2.1.3 *Network Layer*

Menurut Issariyakul & Hossain (2012:4-5),lapisan *network layer* menentukan rute yang dilalui sebuah paket dikirimkan dari node sumber ke node tujuan. Sebuah PDU untuk lapisan jaringan disebut paket.

### 2.1.4 *Link Layer*

Menurut Issariyakul & Hossain (2012:4-5),sebuah protokol *link layer* memiliki tiga tanggung jawab utama. Pertama, kontrol aliran mengatur kecepatan transmisi dalam *link* komunikasi. Kedua, kontrol kesalahan menjamin integritas transmisi data. Ketiga, aliran *multiplexing/demultiplexing* menggabungkan beberapa arus data dan data ekstrak mengalir dari *link* komunikasi. Pilihan protokol link layer dari host ke host dan jaringan ke jaringan. Contoh protokol *link* banyak digunakan lapisan/teknologi termasuk Ethernet, *Point-to-Point Protocol* (PPP), IEEE 802.11 (yaitu, Wi-Fi), dan *Asynchronous Transfer Mode* (ATM).

Perbedaan *link layer* dari protokol lapisan transport sebagai berikut. Yang pertama berhubungan dengan *link* komunikasi tunggal. Di sisi lain, yang kedua melakukan pekerjaan yang sama untuk aliran *end-to-end* yang dapat melintasi beberapa *link.*

### 2.1.5 *Physical Layer*

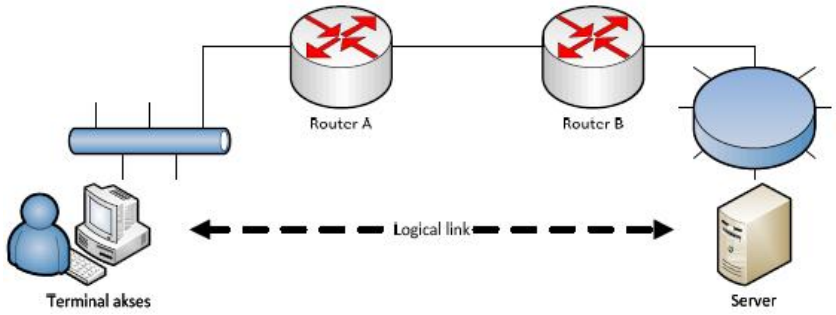
Menurut Issariyakul & Hossain (2012:4-5),kesepakatan lapisan fisik dengan transmisi bit data di *link* komunikasi. Tujuan utamanya adalah untuk memastikan bahwa parameter transmisi (misalnya, daya transmisi) ditetapkan

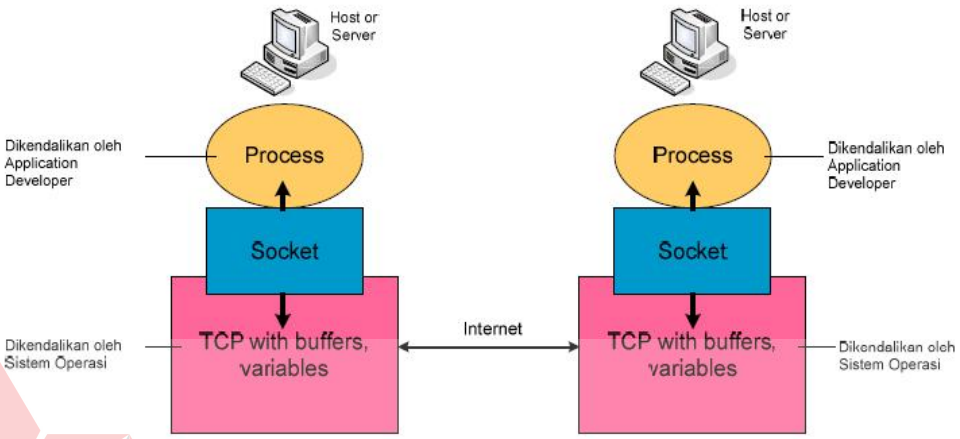
dengan tepat untuk mencapai kinerja transmisi yang diperlukan (misalnya, untuk mencapai target kinerja *bit error rate*).

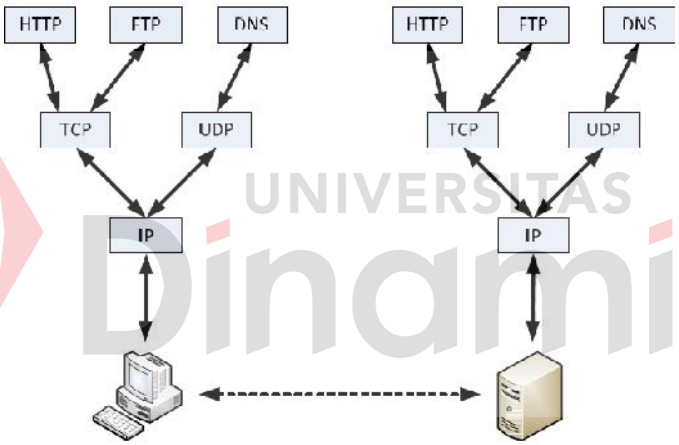
Terakhir, kami menunjukkan bahwa lima lapisan yang dibahas di atas adalah secara umum untuk lapisan OSI. Seperti telah disebutkan, model OSI berisi dua lapisan lainnya yang berada di atas lapisan transport, yaitu, sesi dan lapisan presentasi. Lapisan sesi hanya memungkinkan pengguna pada komputer yang berbeda untuk menciptakan sesi komunikasi di antara mereka sendiri. Lapisan presentasi pada dasarnya mengatur presentasi data yang berbeda yang ada di seluruh jaringan. Sebagai contoh, sistem jaringan terpadu manajemen mengumpulkan data dengan format yang berbeda dari komputer yang berbeda dan mengkonversi format mereka ke dalam format yang seragam.

## 2.2 Protokol Lapisan Transport

Protokol-protokol dalam lapisan transport telah dijelaskan sebelumnya merupakan bagian yang sangat penting dalam jaringan komunikasi. Menurut Jusak (2011), protokol-protokol dalam lapisan ini membentuk *logical communication* antara proses aplikasi yang berjalan pada sebuah *host* menuju *host* yang lain. Yang dimaksud dengan *logical communication* adalah: proses aplikasi yang berjalan pada sebuah terminal yang terhubung pada terminal yang lain (misalnya: sebuah terminal melakukan akses web munju ke web server atau ftp server) membentuk sebuah koneksi logika sedemikian sehingga seakan-akan kedua terminal tersebut langsung terhubung, padahal pada kenyataannya kedua terminal tersebut secara fisik terpisah oleh banyak sekali *router* dan *proxy*, serta perangkat-perangkat jaringan komputer yang lain, seperti terlihat dalam Gambar 2.2.







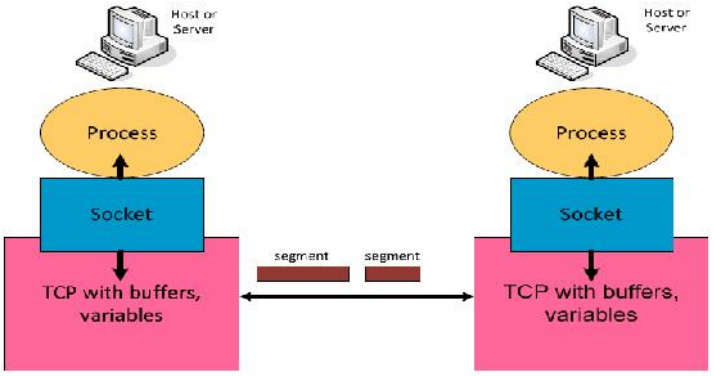
berasal dari server untuk *client*, server melakukan *setting* sebagai berikut: *flag* SYN dari segmen tersebut di set menjadi 1.

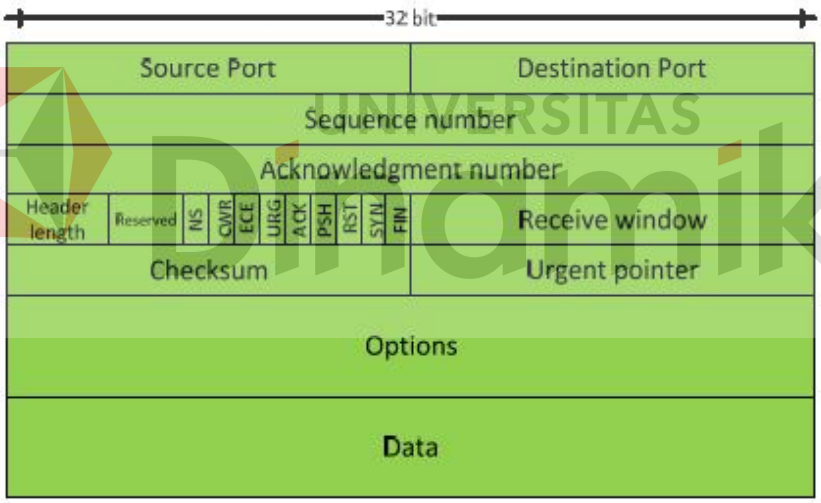
Server memilih sembarang *sequence number*, misalkan *Y*, kemudian diletakkan pada slot *sequence number* dari *header* TCP, dan terkahir server melakukan *setting* slot *acknowledgment* pada *header* TCP dengan nilai *X*+1. Segmen balasan dari server ini seringkali disebut sebagai segmen SYNACK, yang kira-kira berarti "Saya setuju untuk melakukan komunikasi dengan anda, dan nomor inisialisasi segmen saya adalah *Y*".

**Langkah 3**

Setelah menerima segmen balasan sebagai jaminan koneksi dari server, selanjutnya *client* juga melakukan alokasi memori, dan mengirimkan segmen pengakuan. Segmen pengakuan ini memiliki flag SYN yang diset menjadi 0 (karena koneksi telah dibangun), slot dari *sequence number* ini diisi dengan nilai *X*, dan slot dari *acknowledgmen*t diisi dengan nilai *Y*+1.

Setelah proses *three-way handshake* selesai dilakukan, maka berikutnya kedua terminal dapat melakukan pertukaran data pada Gambar 2.6.Proses (aplikasi) pada sisi *client* mengirimkan deretan data kepada lapis di bawahnya, yaitu TCP, melalui *socket.* Deretan data ini masuk ke dalam memory (*buffer*) yang telah diset pada saat inisialisasi *three-way handshake* terjadi, dan dikirimkan ke lapisan bawahnya. Perhatikan bahwa Gambar 2.6 tidak menggambarkan seluruh lapisan di bawah lapisan transport karena konsep *logical link* dalam bagian terdahulu.




32 bit
Source Port
Destination Port
Sequence number
Acknowledgment number
Header length
Reserved
NS
CWR
ECE
URG
ACK
PSH
RST
SYN
FIN
Receive window
Checksum
Urgent pointer
Options
Data

3. ***Acknowledgment number*** dengan panjang 32 bit merupakan nomor bagi segmen *acknowledgment*. Apabila *flag* ACK diset ke 1 maka *acknowledgment number* berisi nomor urut dari segmen berikutnya yang diharapkan oleh penerima.
4. ***Header Length*** dengan panjang 4 bit menspesifikasikan panjang *header* dari segmen TCP. Panjang *header* dapat berubah-ubah karena adanya *slot option* pada Gambar 2.7. Apabila *options* bernilai 0, maka panjang *header* hanya 20 byte.
5. ***Flag***sebanyak 9 buah dengan panjang masing-masing 1 bit. (i) FIN: tidak ada data lagi dari pengirim. (ii) SYN: sinkronisasi *sequence number*. (iii) RST: reset koneksi. (iv) PSH digunakan untuk meminta mendorong data dari *buffer* ke lapisan aplikasi. (v) ACK: untuk menunjukkan bahwa segmen ini adalah segmen *acknowledgment*. (vi) URG: mengindikasikan data yang bersifat *urgent*. (vii) ECE (ECN-Echo): untuk indikasi *Explicit Congestion Notification*. (viii) CWR: untuk indikasi *Congestion Window Reduced*. (ix) NS: untuk indikasi ECN-*nonce* dengan tujuan untuk proteksi terhadap percobaan serangan dari pengirim.
6. ***Receive window*** dengan panjang 16 bit berisi jumlah dalam bentuk byte yang mengindikasikan jumlah data yang dapat ditampung oleh penerima pada saat ini.
7. ***Checksum*** sepanjang 16 bit digunakan untuk pengecekan kesalahan bit pada *header* dan data.
8. ***Urgent data pointer*** sepanjang 16 bit menunjuk pada *sequence number* dari data (yang bersifat *urgent*) terkahir apabila *flag* URG diset menjadi 1.

9. ***Options*** digunakan oleh pengirim dan penerima untuk melakukan negosiasi MSS.

Seperti terlihat dalam struktur segmen TCP, TCP memotong-motong aliran data dalam bentuk byte ke dalam segmen yang berurutan. Mari kita lihat sebuah contoh. Dua buah terminal A dan B akan bertukar data sebesar 90.000 byte dengan menggunakan TCP. MSS adalah 1500 byte. Misalkan inisialisasi *sequence number* awal adalah 0. Maka aliran data akan dipotong-potong menjadi 60 buah segmen dengan masing-masing berisi data sebesar 1500 byte. Karena itu segmen pertama mendapat nilai *sequence number* 0, segmen kedua mendapat nilai *sequence* number 1500, segmen ketiga mendapat nilai *sequence number* 3000 dan seterusnya.

Bagaimana sekarang halnya dengan *acknowledgment number*. Dalam contoh di atas kita menyederhanakan proses pengiriman aliran data pada TCP menjadi *half-duplex*, yaitu pengiriman data satu arah (dari A ke B). Tetapi pada kenyataannya, TCP menggunakan model komunikasi *full-duplex*, artinya pada satu saat A dapat mengirim dan menerima aliran data, demikian juga B. Dengan demikian, pada saat terminal A mengirimkan aliran data lain yang berasal dari terminal B. dalam hal ini, *acknowledgment number* yang diletakkan di dalam *header* segmen data yang dikirimkan oleh terminal A berisi informasi tentang *sequence number* dari *byte* berikutnya yang diharapkan oleh A (dari terminal B). Misalkan, terminal A telah menerima semua data *byte* ke 0 sampai 1499 dari terminal B, dan sekarang terminal A akan mengirimkan segmen ke terminal B. *Bytes* berikutnya yang diharapkan oleh terminal A adalah data mulai dari nomor

1500, maka terminal A meletakkan angka 1500 ini ke dalam *acknowledgment number* pada segmen yang dikirim ke terminal B. Demikian seterusnya.

Contoh lain lagi misalkan sekarang terminal A telah menerima segmen dari terminal B berisi data byte ke 0 sampai 1499, dan terminal A juga telah menerima segmen berisi data byte ke 3000 sampai 4499. Karena suatu alasan tertentu terminal A belum menerima data byte ke 1500 sampai 2999, maka terminal A masih menunggu segmen kedua dengan byte 1500 sampai 2999 ini. Segmen ini dibutuhkan untuk proses pengurutan data di dalam terminal A. karena segmen berikutnya yang berasal dari A ke B akan memiliki *acknowledgment number* dengan nilai 1500.

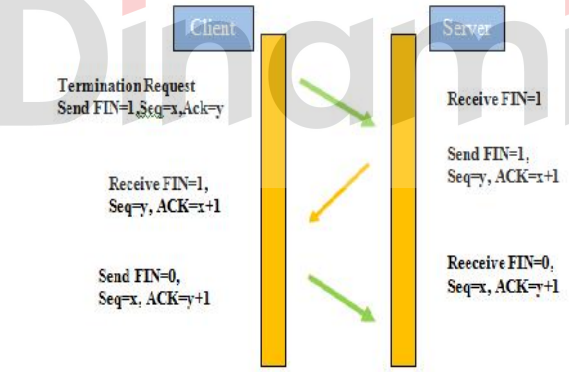
### 2.2.3 Terminasi Koneksi Pada TCP

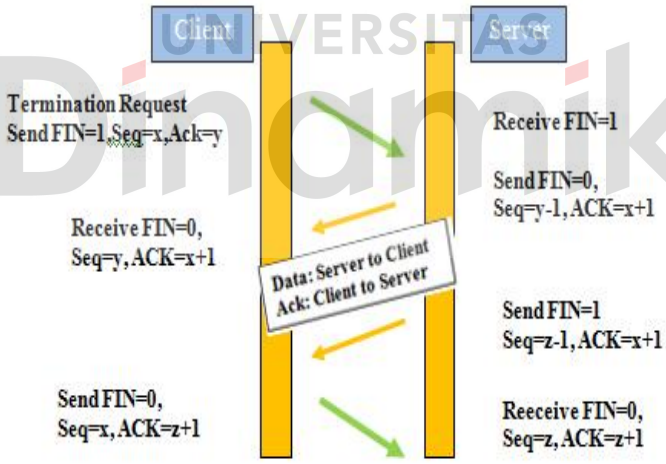
Menurut Jusak (2011), proses terminasi koneksi dapat dilakukan oleh pengirim dan penerima. Implementasi terminasi koneksi dapat dilakukan dengan menggunakan salah satu dari 2 cara, yaitu: *three-way handshaking* atau *four handshaking with a half-close option.*

**a. *Three-way handshaking***

Urutan proses dari *three-way handshaking* untuk penutupan koneksi adalah sebagai berikut: (Bisa dilihat pada Gambar 2.8)

1) Pada saat *client* akan melakukan penutupan koneksi, maka terminal tersebut mengirimkan segmen FIN kepada server. Segmen FIN adalah segmen TCP dengan nilai flag FIN diaktifkan. Segmen FIN ini dapat merupakan *file* terakhir dari data yang dikirimkan oleh *client* atau merupakan segmen sendiri dengan 1 *sequence number* saja.


Client
Server
Termination Request
Send FIN=1, Seq=x, Ack=y
Receive FIN=1
Send FIN=1,
Seq=y, ACK=x+1
Receive FIN=1,
Seq=y, ACK=x+1
Reeceive FIN=0,
Seq=x, ACK=y+1
Send FIN=0,
Seq=x, ACK=y+1


Client
Server
Termination Request
Send FIN=1,Seq=x,Ack=y
Receive FIN=1
Send FIN=0,
Seq=y-1,ACK=x+1
Receive FIN=0,
Seq=y,ACK=x+1
Data: Server to Client
Ack: Client to Server
Send FIN=1
Seq=z-1,ACK=x+1
Send FIN=0,
Seq=x,ACK=z+1
Reeceive FIN=0,
Seq=z,ACK=z+1

mengalami kongesti, maka pengirim akan menurunkan kecepatan pengiriman data. Akan tetapi pengirim mendeteksi bahwa kongesti dalam jaringan telah berkurang, maka pengirim akan meningkatkan kecepatan pengiriman data.

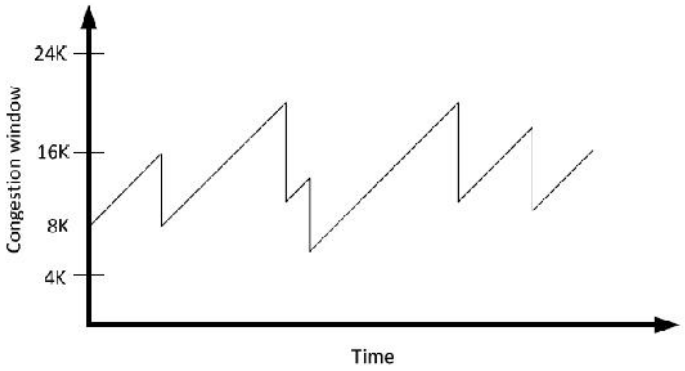
Kongesti di dalam jaringan dapat terjadi terutama karena adanya keterbatasan memori pada *router*, padahal pada saat yang sama setiap pengguna di dalam jaringan ingin menggunakan kapasitas jalur komunikasi di dalam jaringan sebanyak-banyaknya. Di sisi lain, meningkatkan jumlah memori pada router mendekati jumlah yang tak terbatas mungkin dapat mengatasi masalah kongesti dalam jaringan tetapi membawa efek negatif yang lain yaitu semakin tingginya waktu tunda (*delay*) antara pengirim dan penerima.

**a. Additive-Increase, Multiplicative-Decrease**

Apabila terjadi kondisi di mana perangkat *router* di dalam jaringan internet tidak lagi dapat menerima/memproses data, *congestion control* pada dasarnya akan meminta pengirim untuk menurunkan kecepatan pengiriman data dengan cara menurunkan ukuran CongWin, dengan cara demikian diharapkan *router* akan memiliki cukup waktu untuk memproses data. TCP mengetahui habwa jaringan sedang mengalami kongesti apabila:

1) Waktu *time-out* pada sisi pengirim terlampaui atau
2) Pengirim menerima tiga duplikasi ACK dari sisi penerima

Untuk menurunkan kecepatan pengiriman data, TCP meggunakan konsep *multiplicative decrease*, yaitu dengan cara menurunkan sebanyak setengah dari ukuran CongWin. Misalkan ukuran *congestion window* pada saat ini adalah 20 KB, maka pada saat terjadi kongesti TCP akan menurunkan ukuran *congestion window* menjadi 10 KB, demikian pula apabila terjadi *congestion window* lagi,



awal koneksi adalah 400 Kbps. Dengan demikian inisialisasi semacam ini, apabila *bandwidth* di dalam jaringan yang tersedia sangat besar, maka menaikkan kecepatan data secara linier hanya akan memperlambat proses pengiriman data secara keseluruhan. Untuk mengatasi hal ini, metoda *slow start* TCP menaikkan kecepatan pengiriman data secaraeksponensialdengan cara menaikkan nilai CongWin sebanyak dua kali setiap RTT. Selanjutnya pengirim akan menaikkan kecepatan pengiriman data secara eksponensial sampai terjadi kongesti pada jaringan. Apabila terjadi kongesti (melalui adanya waktu *time-out* atau menerima ACK sebanyak 3 kali), maka kecepatan akan diturunkan setengahnya dan kemudian meningkat perlahan-lahan secara linier.

Dengan cara seperti dijelaskan di atas, TCP mengawali koneksi dengan *slow start* kemudian meningkatkan kecepatan pengiriman data secara eksponensial sehingga kecepatan pengiriman data akan meningkat dengan sangat cepat. Pengirim meningkatkan kecepatan secara eksponensial dengan cara menaikkan nilai CongWin sebesar 1 MSS setiap kali mendapatkan ACK dari segmen yang dikirimkan. Sebagai contoh, mula-mula pengirim mentransmisikan sebuah segmen, apabila segmen tersebut mendapatkan *acknowledgment* (ACK diterima oleh pengirim) maka pengirim akan meningkatkan nilai CongWin sebesar 1 MSS dan mengirimkan 2 buah segmen lagi. Apabila kedua buah segmen tersebut mendapatkan *acknowledgment*, maka sekarang pengirim akan meningkatkan nilai CongWin sebanyak 1 MSS lagi untuk setiap segmen yang telah menerima ACK sehingga nilai CongWin saat ini telah menjadi 4 MSS dan selanjutnya pengirim mengirimkan 4 buah segmen lagi. Demikian seterusnya

sampai kongesti di dalam jaringan terjadi. Pada fase inisialisasi *slow start* semacam ini terlihat bahwa nilai CongWin meningkat dua kali lipat setiap RTT.

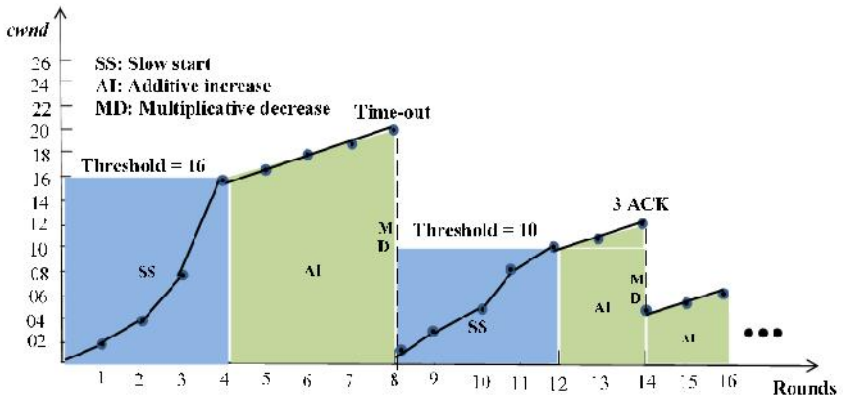
**c. Fast Recovery**

Pada bagian sebelumnya telah dijelaskan bagaimana TCP melakukan proses inisialisasi kecepatan pengiriman data dengan meenggunakan metoda *slow start*. Proses inisialisasi *slow start*akan berakhir pada saat TCP mendeteksi adanya kongesti pada jaringan, dan selanjutnya TCP akan menggunakan metoda AIMD.

Akan tetapi pada keadaan yang sebenarnya TCP bereaksi secara berbeda untuk kedua cara deteksi adanya kongesti dalam jaringan, yaitu melalui *time-out* atau pengirim menerima 3 duplikasi ACK.

i) Apabila pengirim mendeteksi adanya kongesti dalam jaringan melalui adanya 3 duplikasi ACK yang datang. Nilai dari CongWin akan diturunkan menjadi setengah dan selanjutnya meningkat secara linier seperti dijelaskan di dalam metoda AIMD.

ii) Sebaliknya, apabila pengirim mendeteksi adanya kongesti dalam jaringan melalui adanya *time-out*, maka TCP pada sisi pengirim akan menetapkan nilai CongWin sebesar 1 MSS dan pengirim memasuki fase *slow start*. Berikutnya nilai CongWin akan meningkat secara eksponensial sampai mencapai setengah dari nilai CongWin sebelum *time-out* terjadi. Selanjutnya nilai CongWin akan meningkat secara linier sebagaimana halnya terjadi pada saat pengirim duplikasi 3 ACK.



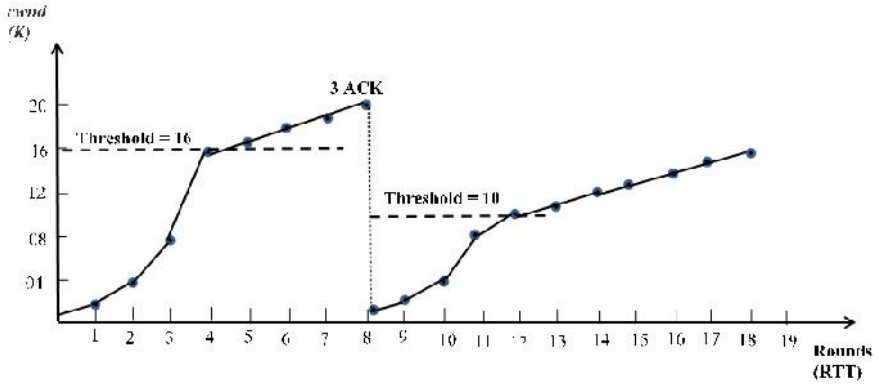
Pada Gambar 2.11 kita asumsikan bahwa maksimum *window size* adalah 32 segmen. *Threshold* di set menjadi 16 segmen (setengah dari maksimum *window size*). Pada fase slow-start *window size* dimulai dari 1 dan naik secara eksponensial sampai mencapai *threshold*. Setelah mencapai *threshold*, fase *congestion avoidance* (*additive increase*) mengijinkan *window size* naik secara linier sampai *time-out* terjadi atau maksimum *window size* tercapai. Pada Gambar 2.11 *time-out* terjadi ketika *window size* 20. Pada saat ini, fase *multiplicative decrease* dimulai dan mereduksi *threshold* sampai setengah dari *window size* sebelumnya. *Window size* sebelumnya adalah 20 ketika *time-out* terjadi maka *threshold* yang baru menjadi 10.

TCP bergerak ke fase *slow start* lagi dan mulai dengan *window size* 1, dan TCP bergerak meningkat secara aditiv ketika *threshold* baru tercapai. Ketika *window size* 12, 3 duplikasi ACK terjadi. Prosedur *multiplicative decrease* mengambil alih. *Threshold* di set menjadi 6 dan pada saat ini TCP bergerak menuju fase *additive increase*. Tetap pada fase ini sampai keluar *time-out* atau 3 duplikasi ACK selanjutnya.(A. Behrouz: 2007)



## 2.4 TCP Tahoe

Menurut Feipeng (2008) TCP Tahoe adalah algoritma yang paling sederhana dari TCP varian lainnya. TCP Tahoe didasarkan pada tiga algoritma kongesti kontrol, yaitu *slow start (SS), congestion avoidance (CA) dan fast retransmit*. Tahoe tidak menggunakan algoritma *fast recovery*. Pada fase *congestion avoidance*, Tahoe memperlakukan duplikat tiga ACK sama dengan *time-out*. Ketika duplikat tiga ACK diterima, Tahoe akan menggunakan *fast*

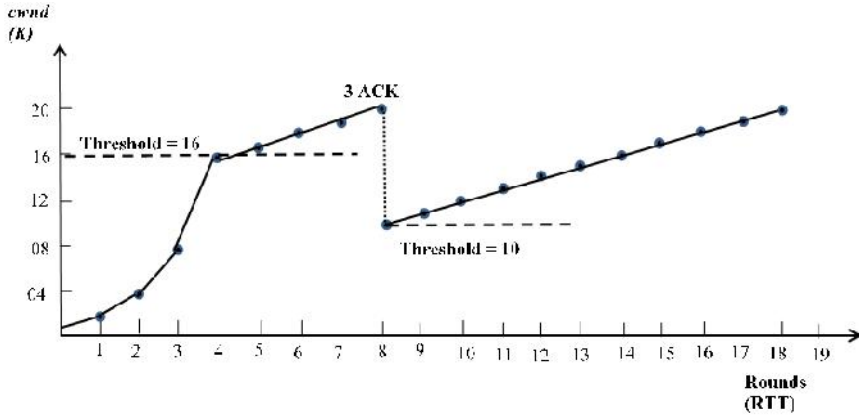


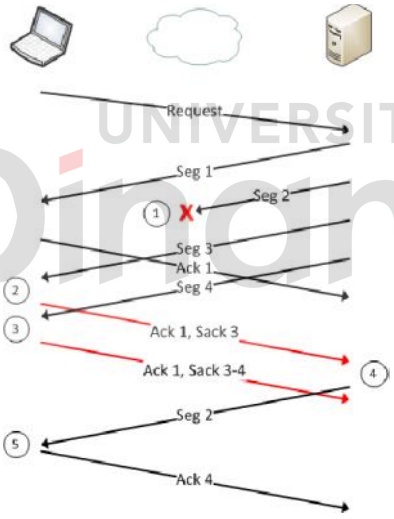
## 2.5 TCP Reno

Menurut Feipeng (2008), TCP Reno berasal dari empat buah algoritma yaitu *slow start (SS), congestion avoidance (CA), fast retransmit* dan *fast recovery*.

Menurut Jusak (2011), TCP Reno membuang fase *slow start* pada saat mendeteksi kongesti melalui diterimanya 3 duplikasi ACK. Untuk selanjutnya proses ini disebut dengan nama *fast recovery*. Pada saat pengirim menerima 3 duplikasi ACK maka nilai *threshold* akan diturunkan menjadi setengah dari nilai CongWin saat sebelum terjadi kongesti, dan nilai CongWin ditetapkan sama dengan nilai *threshold* dan selanjutnya kecepatan pengiriman data akan meningkat secara linier. Algoritma TCP Reno dapat diuraiakan sebagai berikut:

1. Pada saat pengirim menerima 3 ACK ataupun *time-out* maka nilai CongWin akan dinaikkan secara eksponensial jika berada di bawah nilai *threshold* dan dinaikkan secara linier apabila nilai di atas *threshold*.
2. Namun, saat terjadi kongesti dalam jaringan TCP Tahoe akan menurunkan nilai CongWinsetengah dari nilai *threshold* sebelumnya dan berikutnya kecepatan akan naik secara linier.




Request
Seg 1
Seg 2
1
X
Seg 3
Ack 1
Seg 4
2
3
Ack 1, Sack 3
Ack 1, Sack 3-4
4
Seg 2
5
Ack 4

menambahkan sebuah SACK yang menandakan bahwa dia telah menerima segmen #3.

**Poin 3**

*Client* menerima segmen #4 dan mengirimkan duplikasi *acknowledgment* yang lain untuk segmen #1, tetapi kali ini *client* manambahkan SACK yang menunjukkan bahwa *client* telah menerima segmen #3 dan #4.

**Poin 4**

Server menerima duplikasi ACK dari *client* untuk segmen #1 dan SACK untuk segmen #3(yang keduanya terdapat pada paket TCP yang sama). Dari sini server dapat menduga bahwa *client* telah kehilangan segmen #2, yang kemudian akan dikirimkan ulang segmen #2. SACK yang berikutnya diterima oleh server merupakan pemberitahu bahwa *client* juga telah berhasil menerima segmen #4, jadi telah tidak ada lagi segmen yang harus dikirimkan.

**Poin 5**

*Client* menerima segmen #2 dan mengirimkan sebuah ACK yang memberitahukan bahwa dia telah menerima seluruh data termasuk segmen #4.

## 2.7 Packet Drop

Menurut Jonathan & Hermawan (2011), *packet drop* atau *packet loss* merupakan suatu parameter yang menggambarkan suatu kondisi yang menunjukkan jumlah total paket yang hilang, dapat terjadi karena *collision* dan *congestion* pada jaringan dan hal ini berpengaruh pada semua aplikasi karena retransmisi akan mengurangi efisiensi jaringan secara keseluruhan meskipun jumlah *bandwidth* cukup tersedia untuk aplikasi-aplikasi tersebut. Umumnya

perangkat jaringan memiliki *buffer* untuk menampung data yang diterima. Jika terjadi kongesti yang cukup lama, *buffer* akan penuh, dan data baru tidak akan diterima. Kategori *packet loss* dapat dilihat pada Tabel 2.1.

Tabel 2.1 Nilai yang menggambarkan *packet loss*

| Kategori Degredasi | Packet Loss |
|---|---|
| Sangat bagus | 0 |
| Bagus | 3 % |
| Sedang | 15 % |
| Jelek | 25 % |

## 2.8 Round Trip Time (RTT)

Menurut Jusak (2011), untuk mengetahui etimasi terhadap waktu *time-out* yang pertama-tama dilakukan adalah baik *client* ataupun server harus mengetahui terlebih dahulu berapa waktu yang dibutuhkan oleh segmen data melintas dari *client* menuju ke server kemudian dari server kembali ke *client*. Waktu lintasan ini disebut dengan nama *Round Trip Time* (RTT). Dalam setiap koneksi nilai RTT akan berbeda-beda dikarenakan *client* dan *server* dalam posisi yang berbeda-beda pula. Apabila koneksi dilakukan oleh *client* dan *server* yang sama, kemungkinana besar lintasan yang dilewati berbeda-beda dari waktu ke waktu. Apanila koneksi yang dilakukan oleh *client* dan *server* yang sama dan melalui lintasan yang sama, kemungkinan besar kondisi jaringan berbeda dari waktu ke waktu. Oleh karena itu RTT pasti berbeda dari waktu ke waktu.

Protokol TCP dalam implementasinya selalu menghitung waktu yang dibutuhkan dari saat sebuah segmen dikirimkan sampai *acknowledgment* dari

segmen tersebut diterima. Mari kita notasikan sampel RTT ini sebagai $RTT_{sampel}$(t). TCP hanya menghitung segmen yang dikirimkan sekali saja, TCP tidak menghitung segmen yang dikirimkan ulang. Karena nilai $RTT_{sampel}$(t) ini berbeda-beda dari waktu ke waktu, maka TCP tidak dapat menggunakan $RTT_{sampel}$(t) sebagai acuan. Dalam hal ini TCP lebih tertarik pada estimasi dari RTT dalam bentuk rata-rata sampel RTT. Estimasi RTT ini dinotasikan sebagai f$RTT_{estimasi}$(t). Estimasi terhadap RTT dilakukan secara online dengan menggunakan rumusan (Jacobson's Algorithm):

$$RTT_{estimasi}(t) = (1 - \alpha) * RTT_{estimasi}(t - 1) + \alpha * RTT_{sampel}(t)$$

Berdasarkan rekomendasi dalam RFC 2988, nilai α = 0.125. kalau kita perhatikan dengan seksama persamaan di atas memberikan bobot lebih besar pada sampe yang paling baru daripada pada sampel yang lama. Persamaan matematika di atas dalam bidang staatistik seringkali dikenal dengan nama *exponential weighted moving average* (EWMA).

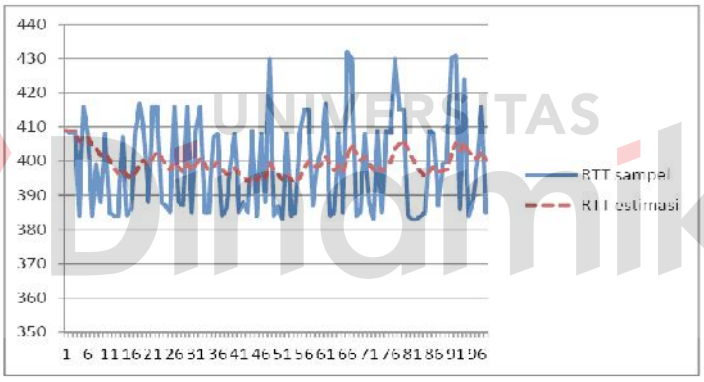
Karena nilai dari $RTT_{estimasi}$(t) merupakan nilai rata-rata, secara statistik kita juga dapat menghitung variasi dari sampel RTT. Sesuai dengan RFC 2988, variasi dari RTT yang dinotasikan dengan simbol $RTT_{var}$(t) dirumuskan sebagai:

$$RTT_{var}(t) = (1 - \beta) * RTT_{var}(t - 1) + \beta * | RTT_{sampel}(t) - RTT_{estimasi}(t)|$$

Nilai rekomendasi dari parameter β = 0.25. Gambar 2.15 menunjukkan grafik dari nilai $RTT_{sampel}$(t) dan $RTT_{estimasi}$(t), nilai dari RTT pada Gambar 2.15, diambil dengan menggunakan aplikasi PING yang berasal dari domain stikom.edu menuju ke sebuah perguruan tinggi di Australia dengan alamat situs http://www.rmit.edu.au.

ık menentukan waktu *time out*:

$$RTT_{estimasi}(t) + 4 * RTT_{var}(t)$$



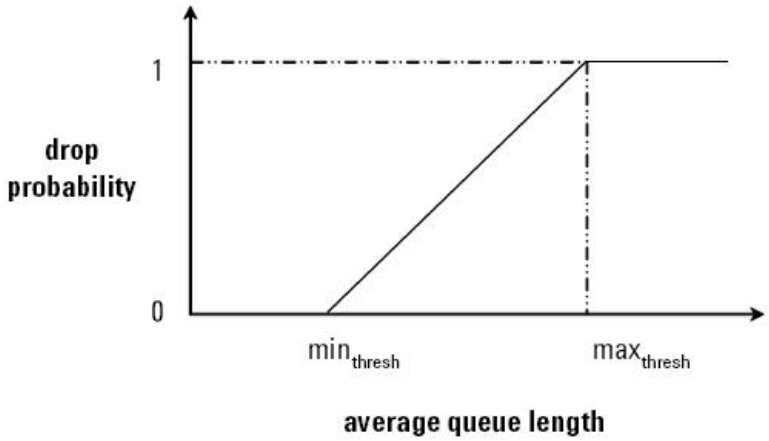
**a. *Drop Tail***

Menurut Agilent Technologies (2008), *Drop Tail* adalah manajemen sederhana antrian algoritma, dimana ketika antrian penuh, paket akan di*drop*. Karena paket dapat di*drop* pada semua koneksi TCP secara bersamaan, banyak koneksi TCP pada *link* bisa dipaksa untuk masuk ke mode *slow start*. Sebagai koneksi TCP yang berjalan secara eksponensial, kemacetan akan terjadi lagi, menyebabkan paket data dapat di*drop* secara bersama-sama pada banyak koneksi. Efek ini disebut "TCP Sinkronisasi Global". Penurunan *tail* dapat diklasifikasikan sebagai antrian pasif manajemen algoritma karena pada dasarnya mekanisme sederhana FIFO dimana paket dibuang ketika panjang antrian melebihi panjang penyangga.

**b. *Random Early Discard* (RED)**

RED dirancang dengan 4 objektivitas: (1) meminimalkan *packet loss* dan *delay*, (2) menghindari sinkronisasi global dari sumber TCP, (3) memelihara pemanfaatan *link* yang tinggi, dan (4) menghilangkan efek terhadap *bursty* sumber.

RED merupakan efektif algoritma pengontrol kemacetan dengan menjatuhkan paket sebelum terjadinya kemacetan, yang menyebabkan TCP menurunkan laju transmisinya. RED menentukan apa yang harus dilakukan pada paket tersebut berdasarkan rata-rata panjangnya antrian. Gambar 2.16 mengilustrasikan proses RED dasar. Jika rata-rata dibawah ambang batas minimum, paket-paket tersebut akan antri. Jika rata-rata di atas ambang batas maksimum, paket akan dibuang.



asumsikan demikian. Dengan demikian rumusan di atas berarti bahwa jumlah data yang akan dikirimkan yang belum mendapatkan *acknowledgment* dibatasi oleh variabel CongWin, dengan cara demikian kecepatan pengiriman data menjadi terbatas. Pada setiap awal koneksi CongWin diset sebesar MSS byte.

## 2.11 Network Simulator

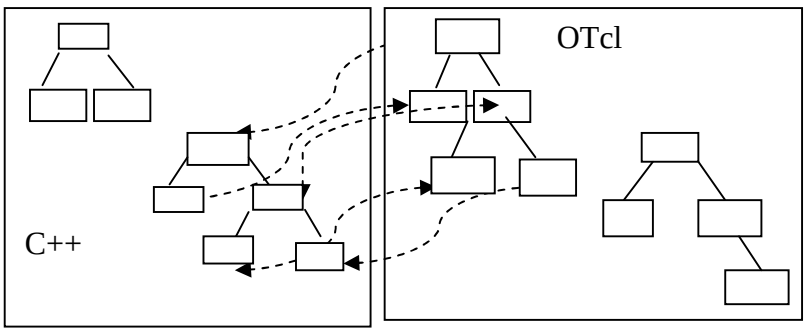
### 2.11.1 Sejarah NS

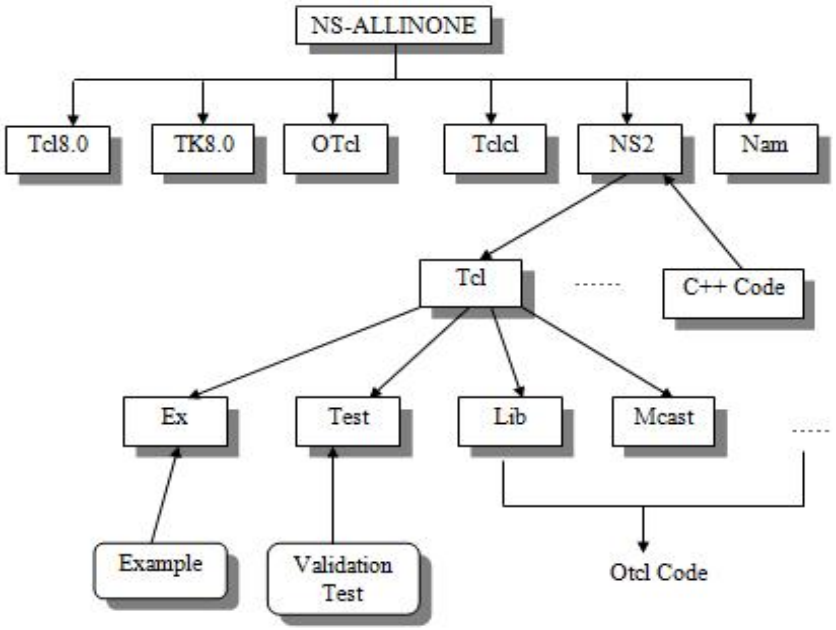
Menurut Wirawan & Indarto (2004), Network Simulator (NS) pertama kali dibangun sebagai varian dari REAL Network Simulator pada tahun 1989 di UCB (University of California Berkeley). Pada tahun 1995 pembangunan *Network Simulator* didukung oleh DARPA (*Defense Advanced Research Project Agency*) melalui VINT (*Virtual Internet Testbed*) *Project*, yaitu sebuah tim riset gabungan yang beranggotakan tenaga ahli dari LBNL (*Lawrence Berkeley of National Laboratory*), Xerox PARC, UCB dan UCS/ISI (*University of Southern California School of Engineering/Information Science Institue*). Tim gabungan ini membangun sebuah perangkat lunak simulasi jaringan Internet untuk kepentingan riset interaksi antar protokol dalam konteks pengembangan protokol pada saat ini dan masa yang akan datang.

### 2.11.2 Kelebihan NS

Kelebihan dari NS adalah sebagai perangkat lunak simulasi pembantu analisis dalam riset atau penelitian. NS-2 dilengkapi dengan *tool* validasi. *Tool* validasi digunakan untuk menguji validitas pemodelan yang ada pada NS2.

Pembuatan simulasi dengan menggunakan NS-2 jauh lebih mudah daripada menggunakan *software developer* seperti Delphi atau C++. Pada





Keterangan:

TCL: *Tool Command Language*

Otcl: *Object TCL*

TK: *Tool Kit*

Tclcl: Tcl/C++ *Interface*

NS2: NS versi 2

Nam: *Network Animator*

### 2.11.5 Cara Membuat dan Menjalankan Skrip NS

Membuat skrip simulasi NS sangat mudah. Skrip simulasi bisa Anda buat dengan menggunakan program teks editor yang ada pada linux Anda, dan disimpan dalam sebuah folder dengan ekstensi .tcl.

Contoh:

```
simulasitcp.tcl
```

Untuk menjalankan simulasi yang telah Anda buat, Anda tinggal masuk ke folder tersebut dan mengaktifkan NS serta nama *file* tcl simulasi yang Anda jalankan.

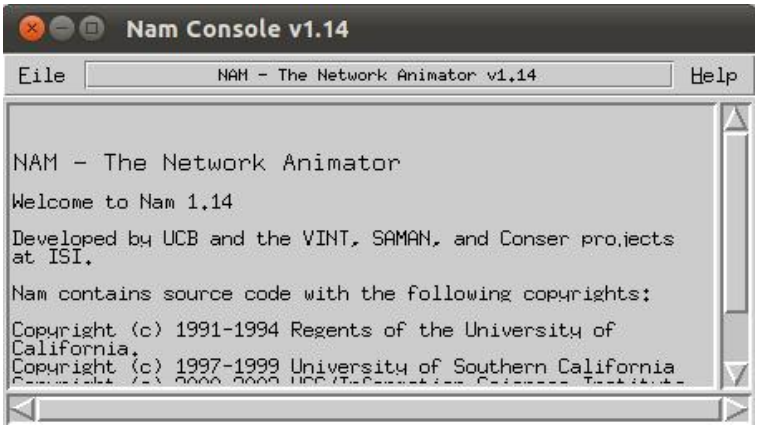
Contoh:

```
[root @ accessnet your_folder]# ns
simulasitcp.tcl
```

### 2.11.6 Output Simulasi NS

Pada saat satu simulasi berakhir, NS membuat satu atau lebih *file outputtext-based* yang berisi detail simulasi jika dideklarasikan pada saat membangun simulasi. Ada dua jenis *output* NS, yaitu: *file trace* yang akan digunakan untuk analisa numerik dan simulasi yang disebut *network animator* (nam). Jika Anda ingin menampilkan tampilan grafis simulasi Anda harus

menjalankan system X Windows pada linux Anda yang bisa berupa KDE atau Gnome. Pada buku ini penulis menggunakan KDE sebagai X Windows system.




Gambar 2.19 Nam Konsole (Wirawan & Indarto: 2004)

Nam memiliki *interface* grafis yang mirip dengan CD *player* (*play*, *rewind*, *fast forward*, *pause*, dan lain-lain) dan juga memiliki kontroler kecepatan tampilan simulasi.

### 2.11.7 *Tool Command Line* (TCL)

Tcl (*Tool Command Language*) adalah *string-based command language*. Bahasa Tcl diciptakan oleh John Ousterhout pada akhir tahun 1980-an sebagai *command language* dengan tool interaktif. Tcl didesain untuk menjadi semacam "perekat" yang membangun *software building block* menjadi suatu aplikasi.

Berikut ini akan dijelaskan dasar-dasar bahasa Tcl yang berguna dalam membangun simulasi Anda nantinya.

**Syntax Dasar**

Syntax dasar perintah tcl yaitu:

```
Command arg1 arg2 arg3 …
```

Command tersebut bisa berupa nama dari *built in command*, atau sebuah prosedur Tcl.

Contoh:

```
expr 2*3
puts *ini adalah contoh command*
```

**Variable dan Array**

Untuk membuat variable pada tcl, digunakan perintah set.

Contoh:

```
set x *ini adalah contoh variable*
set y 20
```

Pemanggilan variabel dilakukan dengan menggunakan tanda $ seperti contoh dibawah ini:

```
puts “$x, semuanya berjumlah $y “
```

NS juga mensupport penggunaan array. Array ditandai dengan menggunakan tanda kurung setelah nama array tersebut.

Contoh:

```
set opts(bottlenecklinkrate) 1Mb
set opts(ENC) *on*
set n(0) [$ns node]
set n(1) [$ns node]
```



### 2.11.8 *Object Oriented Tcl* (Otcl)

Sekarang kita akan mulai mengenal Otcl (*Object Oriented TCl*). Otcl adalah ekstensi tambahan pada Tcl yang memungkinkan fungsi *object oriented* (OO) pada Tcl. Ini memungkinkan pendefinisian dan penggunaan *class* Otcl. Di bawah ini adalah contoh *function* sederhana Otcl. Perhatikan perbedaan antara Tcl dengan Otcl:

#function sapaan

```
Class ibu
```

#create object class ibu

```
ibu instproc sapaan {} {
```

#mendefinisikan object class yang sudah didefinisikan

```
$self instvar umur_

#cek variabel referensi

puts " : $umur_ibu berkata : Apa yang kamu kerjakan?"
}

#membuat child class dari class ibu

Class anak –superclass ibu

#inheritance class dari class ibu

$self instvar umur_
puts "$umur _ anak berkata: tidak apa-apa"
}

#buat object ibu dan anak

set a [new ibu]
$a set umur_50
set b [new anak]
$b set umur_17

#panggil function 'sapaan'

$a sapaan
```

Struktur perintah dalam NS mirip dengan contoh program di atas. NS telah menyediakan objek simulasi yang berupa class pemrograman dalam OTCL. Anda tinggal memanggil objek simulasi yang telah didefinisikan dalam *library* NS dengan menggunaakn TCL. (Wirawan & Indarto, 2004)

### 2.11.9 Tahap-Tahap dalam Membangun Simulasi

**Langkah 1:** Inisialisasi Simulasi

Menurut Wirawan & Indarto (2004), untuk memulai pembuatan simulasi sederhana, Anda dapat membukasalah satu teks editor yang ada pada Linux Anda. Kemudian simpan *file* tersebut dalam folder Anda sebagai contoh.tcl.

Simulasi NS dimulai dengan menuliskan skrip Tcl seperti di bawah ini. Skrip ini merupakan inisialisasi simulasi dan harus ada dalam setiap semulasi yang Anda buat.

Ingatlah bahwa baris yang dimulai dengan tanda # dianggap sebagai komentar yang digunakan untuk menjelaskan masing-masing perintah.

#memanggil simulator *object*

```
set ns [new Simulator]
```

#open file handle untuk simulator nam trace data

```
set nf [open out.nam w]
$ns namtrace-all $nf
```

#prosedur finish berguna untuk menyelesaikan simulasi

```
proc finish {} {
```

#menutup file dan memulai nam (*network animator*)

```
global ns nf
$ns flush-trace
close $ns
exec nam out.nam &
exit 0
}
```

#mengeksekusi prosedur finish pada saat detik ke 5.0

```
$ns at 5.0 “finish”
```

#menjalankan simulasi

```
$ns run
```

**Langkah 2:** Pembuatan Topologi

Topologi dibangun oleh node dan link. Berikut ini dijelaskan pembuatan node dan link yang membangun topologi simulasi.

**Node**

Sebuah objek node pada NS didefinisikan dengan command $ns node. Perintah pembuatan node pada NS adalah sebagai berikut:

```
set node [$ns node]
```

**Link**

Ada dua jenis link yang bisa digunakan pada NS, yaitu *simplex link* dan *duplex link*. Berikut ini adalah perintah pembuatan link beserta parameternya:

1. Untuk simplex link:

   $ns simplex-link <node1><node2><bw><delay><qtype>

   Link satu arah dari <node1> ke <node2>.

2. Untuk duplex link:

   $ns_ duplex-link <node1><node2><bw><delay><qtype>

   Link dua arah dari <node1> ke <node2> dan sebaliknya.

**Langkah 3:** Sending Data

Proses sending data pada NS dilakukan dengan membuat *transport agent* dan aplikasi di atasnya. Transport agent dibuat berpasangan, satu berfungsi sebagai sumber data dan pasangannya sebagai tujuan.

Pada simulasi ini, kita akan mencoba mengirim UDP paket dengan menggunakan *traffic* generator CBR antara kedua node. Kita awali dengan membuat *agent* pengirim data.

#membuat objek simulator yang berupa UDP agent

```
set udp0 [new Agent/UDP]
```

#attach-agent berfungsi untuk mengambil object agent yang sudah didefinisikan yaitu UDP0 sebagai pengirim pada node0

```
$ns attach-agent $n0 $udp0
```

Setelah kita mempunyai sumber data, sekarang coba Anda buat tujuan dari paket data yang kita kirimkan dan sekaligus Anda hubungkan keduanya.

#membuat *object* simulator berupa *agent null*

```
set null0 [new Agent/Null]
```

#attach-agent berfungsi untuk mengambil *object agent* yang sudah didefinisikan yaitu null0 sebagai penerima pada node n1

```
$ns attach-agent $n1 $null0
```

#perintah untuk menghubungkan antara *agent* pengirim dengan penerima

```
$ns connect  $udp0 $null0
```

Kemudian buat aplikasi yang berjalan di atas *transport agent* tersebut. Pada contoh ini, kita gunakan generator trafik dengan fungsi CBR (*Constant Bit Rate*).

#inisialisasi cbr0

```
set cbr0 [new Application/Traffic/CBR]
```

#ukuran paket datanya 500 byte

```
$cbr0 set packetSize_ 500
```

#interval pengiriman antar paket 0.005 s

```
$cbr0 set interval_ 0.005
```

#paket UDP0 dikirim dengan fungsi CBR

```
$cbr0 attach-agent $udp0
```

Bagaimana kita mengatur pproses sending data?

#sending data dimulai pada 0.5 detik

```
$ns at 0.5 “$cbr0 start”
$ns at 4.5 “$cbr0 stop”
```

Artinya bahwa data akan dikirim setelah 0.5 detik simulasi dari node n0 ke node n1 dan berakhir pada detik 4.5.

### 2.11.10 Transport Agent

Menurut Wirawan & Indarto (2004), protokol lapisan transport data yang didukung *network simulator* antara lain:

1. TCP (*Transport Control Protocol*)
2. UDP (*Universal Datagram Protocol*)
3. RTP (*Real Time Transport Protocol*)

**TCP (*Tranport Control Protocol*)**

1. *Two way* TCP *agent* men*support* proses *handshaking* pada saat *connection setup*, sehingga koneksi dapat dibangun atau di*drop* tergantung pada kondisi jaringannya. *One way* TCP tidak men*support* proses *handshaking*. Pertukaran data menggunakan *agent* ini diasumsikan telah melewati proses handshaking.
2. *Two way* TCP agent men*support* data transfer dua arah.
3. Penomoran pada jumlah *byte* yang ditransfer, bukan pada jumlah paket.



**One Way TCP**

Simulasi koneksi pada *one way* TCP dilakukan dengan menggunakan 2 *agents* yang berpasangan, yaitu TCP *sender* dan TCP *Sink.*

**TCP Sender**

Network simulator 2 mendukung beberapa jenis TCP sender agent yaitu:

1. TCP Sender base (Tahoe TCP)

   Agent/TCP
2. Reno TCP

   Agent/TCP/Reno

3. New Reno TCP

   Agent/TCP/NewReno

4. Vegas TCP

   Agent/TCP/Vegas

5. SACK (Selective ACK) TCP

   Agent/TCP/SACK1

6. FACK (Forward ACK) TCP

   Aget/TCP/Fack

Masing-masing *sender* memeiliki *sender agent* pada NS memiliki perilaku yang sama dengan implementasi masing-masing jenis TCP di dunia nyata.

**TCP Sink Agent**

TCP Sink bertugas mengirimkan ACK per paket yang diterima pada TCP *sender* pasangannya. Beberapa macam TCP Sink yang disupport NS yaitu:

1. Base TCP Sink

   Agent/TCPSink

2. Delayed ACK

   Agent/TCPSink/DelAck

3. Sack TCP Sink

   Agent/TCPSink/Sack

4. Delayed ACK dengan SACK

   Agent/TCPSink/Sack1/DelAck

*Agent* TCP pengirim dan penerima yang bermacam-macam ini dimaksudkan agar NS *user* dapat memahami dan dapat melakukan riset yang berkenaan dengan karakteristik *congestion control* dan *error control* masing-masing jenis TCP.

Untuk mendekatkan dengan kondisi sebenarnya, Anda dapat mengatur parameter TCP flow. Konfigurasi parameter TCP flow yang umum dilakukan yaitu:

```
$tcp set window_ <wind size>
$tcp set packetSize_ <packet size>
```

### 2.11.11 Pengambilan Data Simulasi

Menurut Wirawan & Indarto (2004), simulasi dengan menggunakan NS-2 dapat menghasilkan visualisasi dari *tracing* dengan menggunakan NAM dan juga *file tracing* dalam bentuk *text* (ASCII) yang berkaitan dengan setiap kejadian dalam jaringan. *File trace* sangat dibutuhkan jika menginginkan keluaran berupa data numerik yang kemudian dianalisis secara matematis atau ditampilkan dalam bentuk grafik.

**1. File Trace**

*File trace* merupakan pencatatan seluruh *event* (kejadian) yang dialami oleh suatu simulasi paket pada simulai yang telah dibangun. Pembuatan *file trace* dilakukan dengan memanggil objek *trace* pada *library*. Sama seperti *file* nam, pembuatan *output trace file* dinyatakan pada inisialisasi simulasi seperti dibawah ini:

```
set ns [new Simulator]
set nf [open out.nam w]
set tf [open out.tr w]
#Buka trace files
set tf [open out.tr w]
set windowVsTime2 [open WindowsVsTimeNReno w]
```

```
$ns trace-all $tf
#Tracing file
set nf [open out.nam w]
$ns namtrace-all $nf
proc finish {} {
global ns tf nf
$ns flush-trace
close $tf
close $nf
exec nam out.nam &
exit 0
}
```

Sehingga keluaran NS yang dihasilkan berupa *file* out.nam yang berfungsi sebagai *input*-an pada nam dan *file trace* out.tr. File trace NS telah tersusun dengan format yang ditentukan. Gambar 2.20 merupakan isi masing-masing kolom hasil *record*.

| Event | Time | From Node | To Node | Pkt Type | Pkt Size | Flags | Fid | Src Addr | Dst Addr | Seq Num | Pkt Id |
|---|---|---|---|---|---|---|---|---|---|---|---|

Gambar 2.20 Format kolom hasil record

Keterangan:

1. Event (kejadian)

   Kejadian yang dicatat oleh NS yaitu:

   r : receive (paket diterima oleh to node)

   + : enqueue (paket masuk ke dalam antrian atau keluaran dari from node)

   - : dequeue (paket keluar dari antrian)

   d : drop (paket drop dari antrian)

2. Time: Yaitu waktu terjadinya suatu kejadian dalam detik.
3. From node
4. To node: From node dan to node menyatakan keberadaan paket. Saat pencatatan kejadian, paket berada pada link di antara from node dan to node.
5. Pkt type: Adalah tipe paket yang dikirim seperti ud,p tcp, ack atau cbr.

6. Pkt size: Adalah ukuran paket dalam byte.

7. Flag: *Flag* digunakan sebagai penanda. Macam-macam *flag* yang bisa digunakan yaitu:

   a. E untuk terjadinya kongesti (*Congestion Experienced*/*CE*)

   b. N untuk indikasi ECT (ECN-*Capable-Transport*) pada header IP.

   c. C untuk ECN-*Echo*.

   d. A untuk pengurangan window kongesti pada *header* TCP.

   e. P untuk prioritas

   f. F untuk TCP *fast start.*

8. Fid: Adalah penomoran unik dari tiap aliran data.

9. Src_addr: Adalah alamat asal paket. Format src_addr adalah: node.port, misalnya 3.0.

10. Dst_addr: Adalah alamt tujuan paket. Sama dengan srcc_addr, format dst_addr adalah:

    node.port misalnya 0.0.

11. Sequence number: Adalah nomor urut tiap paket.

12. Packet id: Adalah penomoran unik dari tiap paket.

Berikut ini adalah contoh isi dari suatu file trace:

```
- 1.699424 4 3 ack 40 ------- 1 4.0 0.0 6 15
r 1.707744 4 3 ack 40 ------- 1 4.0 0.0 6 15
- 1.707744 3 2 ack 40 ------- 1 4.0 0.0 6 15
```

**2. Monitoring Queue**

Menurut Jusak (2011), salah satu *object*yang cukup penting dalam NS adalah monitor-*queue*. Objek ini berisi banyak sekali informasi terkait dengan karakteristik proses antrian, misalnya panjang antrian, jumlah paket yang datang,

jumlah paket yang keluar dari antrian, jumlah paket yang di*drop* dan sebagainya. Untuk mengaktifkan proses *queue* monitor pada link antara node2 dan node3 digunakan perintah:

```
set qmon [$ns monitor-queue $n2 $n3 [open qm.out w] 0.1];
[$ns link $n2 $n3] queue-sample-timeout;
$ns at 120 "finish"
$ns run
```

Seperti terlihat dalam perintah di atas, objek monitor *queue* memiliki 4 buah argumen. Dua argumen pertama berkaitan dengan letak *link* di antara dua buah node yan gakan diobservasi, argumen berikutnya adalah file *output* yaitu qm.out, dan argumen terakhir merupakan frekuensi dari proses pengamatan, dalam skrip di atas penulis akan mengambil sampel setiap 0.1 detik.

Salah satu contoh isi dari *file* qm.out:

```
6.2999999999999936 2 3 3689.5334400002166 6.2323200000003665 246
240 0 148344 144792 0
6.3999999999999932 2 3 3647.4777599999325 6.1612799999998868 252
245 0 151896 147752 0
6.4999999999999929 2 3 3644.0678400002553 6.1555200000004318 257
251 0 154856 151304 0
6.5999999999999925 2 3 3637.2480000000851 6.1440000000001449 262
256 0 157816 154264 0
6.6999999999999922 2 3 4477.2480000000141 6.9840000000000728 268
261 0 161776 157224 0
```

*Fileoutput* berisi informasi yang terdiri atas 11 kolom seperti berikut:

1. Waktu pengambilan sampel
2. Node *input*,
3. Node *output*,
4. Ukuran *queue* dalam satuan byte (berkorespondensi dengan atribut size_ pada objek monitor-*queue*),
5. Ukuran *queue* dalam satuan byte (berkorespondensi dengan atribut pkt_ pada objek monitor-*queue*),

6. Jumlah paket yang telah datang (berorespondesi dengan atribut parrival_ pada objek monitor-*queue*),
7. Jumlah paket yang telah keluar dari *queue* (berorespondesi dengan atribut pdepartures_ pada objek monitor-*queue*),
8. Jumlah paket yang di*drop* (dibuang) oleh *queue* (berorespondesi dengan atribut pdrops_ pada objek monitor-*queue*),
9. Jumlah data dalam byte yang telah datang(berorespondesi dengan atribut barrivals_ pada objek monitor-*queue*),
10. Jumlah data dalam byte yang telah keluar dari *queue* (berorespondesi dengan atribut bdepartures_ pada objek monitor-*queue*),
11. Jumlah data dalam byte yang telah di*drop* (dibuang) oleh *queue*(berorespondesi dengan atribut bdrops_ pada objek monitor-*queue*).

**3. Pemrosesan file data dengan Perl**

Menurut Jusak (2011),Perl singkatan dari *Practical Extraction and Report Language* merupakan bahasa pemorgraman yang banyak digunakan untuk pemrosesan data *file* ASCII pada sistem UNIX. Bahasa pemrograman ini dibuat oleh Larry Wall dengan tujuan untuk memudahkan tugas-tugas administrasi sistem UNIX. Namun perl saat ini telah berevolusi menjadi bahasa pemrograman dan merupakan salah satu alat yang dapat digunakan untuk pemrograman web. Di bawah ini merupakan contoh prosedur queue.pl untuk mencari *queue.*

```
$infile=$ARGV[0];
open (DATA,"<$infile") || die "Can't open $infile $!";
while (<DATA>)
{
      @x = split(' ');
      print STDOUT "$x[0] $x[3]\n";
}
close DATA;
exit(0);
```

Selanjutnya, jalankan prosedur queue.pl di atas dengan menggunakan *file* qm.out sebagai *input*, dan simpan hasil pengolahan ke dalam file bernama QueueByte_tahoe.

```
perl queue.pl qm.out 4 1 > QueueByte_tahoe
```

Perintah di atas menunjukkan bahwa analisis *queue* akan dilakukan pada node 4 dengan nilai granularity sebesar 1s.

**4. Menggambar Grafik dengan Gnuplot**

Setelah mendapatkan nilai *queue* dari proses pada perl, maka langkah selanjutnya adalah menggambarkan *queue* tersebut ke dalam bentuk grafik. Salah satu cara menggambarkan grafik dengan menggunakan Gnuplot. Di bawah ini merupakan contoh skrip gnuplot.

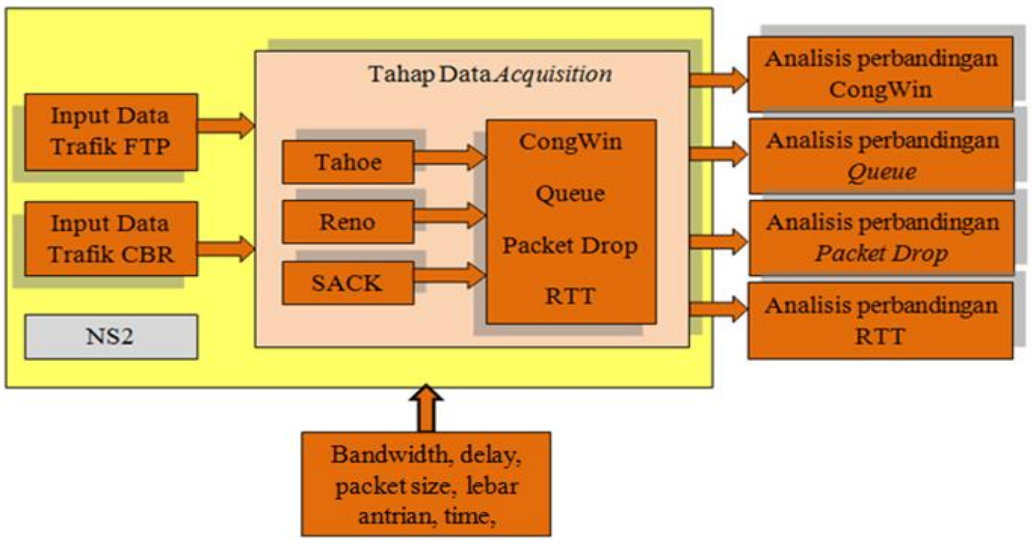
gnuplot

```
set size 0.8,0.8
set key right bottom
set xlabel “waktu (s)”
set ylabel “queue”
plot ‘QueueByte_tahoe’ with lines 1
```

Ukuran *congestion window* pada TCP juga dapat digambarkan dengan menggunakangnuplot dengan memanggil *file* WinFile seperti pada contoh skrip di bawah ini:

gnuplot

```
set size 0.8,0.8
set key right bottom
set xlabel “waktu (s)”
set ylabel “CongWin”
plot ‘WinFile’ with lines 1
```



Pada Gambar 3.1 dikelompokkan menjadi tiga bagian utama, yaitu input data trafik, tahap data *acquisition* dan output yang berupa hasil analisis perbandingan. Tiga bagian utama di atas akan dijelaskan pada sub bab berikutnya.

### 3.1.1 Input Data Trafik

Bagian input data trafik adalah proses di mana paket data mulai berjalan dari sumber menuju tujuan. Ada 2 tipe dasar aplikasi yang disimulasikan pada NS, yaitu *simulated application* dan generator trafik. Pada *simulated application*, penulis menggunakan FTP (*File Transport Protocol)* sebagai input data dan pada generator trafik, penulis menggunakan CBR (*Contstant Bit Rate*). Sehingga pada bagian input memiliki 2 sumber yaitu berupa FTP dan CBR. FTP dibangun untuk mensimulasikan *bulk data transfer* sedangkan CBR untuk membangkitkan data secara kontinyu dengan bit rate yang konstan. Trafik generator CBR pada simulasi ini digunakan untuk menciptakan terjadinya kongesti dan tidak diikutkan dalam pengolahan data.



### 3.1.2 Data *Acquisition* Algoritma TCP Tahoe

Data *acquisition* algoritma TCP Tahoe adalah proses yang dilakukan pada algoritma Tahoe mulai dari pembuatan skrip *.tcl, melakukan pengaturan parameter seperti *access-link bandwidth*, *delay access-link*, *bottleneck-link bandwidth*, *delay bottleneck-link*, lebar antrian, ukuran paket, maksimum ukuran *window*, dan model *queue* yang digunakan, kemudian tahap selanjutnya menjalankan skrip dan melakukan parsing data parameter uji berdasarkan CongWin, *queue*, *packet drop* dan RTT sesuai dengan desain topologi simulasi serta menggambarkan grafik berdasarkan hasil data yang telah didapatkan. Uji

coba perbandingan simulasi ini dilakukan secara tidak simultan artinya skrip *.tcl TCP Tahoe, Reno dan SACK dijalankan secara terpisah. Setelah proses menggambarkan grafik selesai, langkah selanjutnya adalah menganalisisnya sesuai dengan variasi *bandwidth* yaitu 0.5Mb/30ms, 0.256Mb/30ms dan 0.128Mb/30ms. Ketiga variasi *bandwidth* ini digunakan untukmengetahui perubahan *bandwidth* terhadap unjuk kerja algoritma TCP Tahoe sesuai dengan parameter uji yang telah ditentukan.

### 3.1.3 Data *Acquisition*Algoritma TCP Reno

Data *acquisition* algoritma TCP Reno adalah proses yang dilakukan pada algoritma Reno mulai dari pembuatan skrip *.tcl, melakukan pengaturan parameter seperti *access-link bandwidth, delay access-link, bottleneck-link bandwidth, delay bottleneck-link*, lebar antrian, ukuran paket, maksimum ukuran *window*, dan model *queue* yang digunakan, kemudian tahap selanjutnya menjalankan skrip dan melakukan parsing data parameter uji berdasarkan CongWin, *queue*, *packet drop* dan RTT sesuai dengan desain topologi simulasi serta menggambarkan grafik berdasarkan hasil data yang telah didapatkan. Uji coba perbandingan simulasi ini dilakukan secara tidak simultan artinya skrip *.tcl TCP Tahoe, Reno dan SACK dijalankan secara terpisah. Setelah proses menggambarkan grafik selesai, langkah selanjutnya adalah menganalisisnya sesuai dengan variasi *bandwidth* yaitu 0.5Mb/30ms, 0.256Mb/30ms dan 0.128Mb/30ms. Ketiga variasi *bandwidth* ini digunakan untuk mengetahui perubahan *bandwidth* terhadap unjuk kerja algoritma TCP Reno sesuai dengan parameter uji yang telah ditentukan.

### 3.1.4 Data *Acquisition* AlgoritmaSACK

Data *acquisition* algoritma SACK adalah proses yang dilakukan pada algoritma SACK mulai dari pembuatan skrip *.tcl, melakukan pengaturan parameter seperti *access-link bandwidth, delay access-link, bottleneck-link bandwidth, delay bottleneck-link*, lebar antrian, ukuran paket, maksimum ukuran *window*, dan model *queue* yang digunakan, kemudian tahap selanjutnya menjalankan skrip dan melakukan parsing data parameter uji berdasarkan CongWin, *queue*, *packet drop* dan RTT sesuai dengan desain topologi simulasi serta menggambarkan grafik berdasarkan hasil data yang telah didapatkan. Uji coba perbandingan simulasi ini dilakukan secara tidak simultan artinya skrip *.tcl TCP Tahoe, Reno dan SACK dijalankan secara terpisah. Setelah proses menggambarkan grafik selesai, langkah selanjutnya adalah menganalisisnya sesuai dengan variasi *bandwidth* yaitu 0.5Mb/30ms, 0.256Mb/30ms dan 0.128Mb/30ms. Ketiga variasi *bandwidth* ini digunakan untuk mengetahui perubahan *bandwidth* terhadap unjuk kerja algoritma SACK sesuai dengan parameter uji yang telah ditentukan.

### 3.1.5 Analisis Perbandingan CongWin

Analisis perbandingan CongWin merupakan hasil akhir yang ingin dicapai dalam penelitian ini. Pada bagian ini yang dilakukan adalah melakukan analisis berdasarkan parameter uji CongWin. CongWin berisi tentang kecepatan pengiriman data yang dapat dilakukan oleh sisi pengirim, dan dirumuskan sebagai jumlah data yang akan dikirimkan yang belum mendapat *acknowledgment* tidak boleh melebihi jumlah CongWin atau *RcvWindow* (Jusak, 2011). Menurut Haugdahl (2007), jumlah paket yang beredar sangat besar akan mengakibatkan

jaringan bisa lebih cepat dan ukuran jendela juga lebih besar. Ada dua tahap dalam melakukan analisis, pertama yaitu menganalisis hasil grafik yang dilakukan menggunakan gnuplot dan yang kedua adalah menganalisis hasil grafik yang sudah digabungkan ke dalam satu grafik antara TCP Tahoe, Reno dan SACK. Pada penelitian ini unjuk kerja terbaik pada CongWin dilihat berdasarkan pada nilai CongWin yang tinggi.

### 3.1.6 Analisis Perbandingan *Queue*

Analisis perbandingan *queue* merupakan hasil akhir yang ingin dicapai dalam penelitian ini. Pada bagian ini yang dilakukan adalah melakukan analisis berdasarkan parameter uji penggunaan *queue*. Analisis *queue* merupakan analisis terhadap penggunaan *queue* di mana penggunaan *queue* yang rendah menggambarkan paket yang mengantri menuju jalur *bottleneck* juga rendah. (Rahman, Kabir, Lutfullah, & Amin, 2008). Ada dua tahap dalam melakukan analisis, pertama yaitu menganalisis hasil grafik yang dilakukan menggunakan gnuplot dan yang kedua adalah menganalisis hasil grafik yang sudah digabungkan ke dalam satu gambar antara TCP Tahoe, Reno dan SACK.

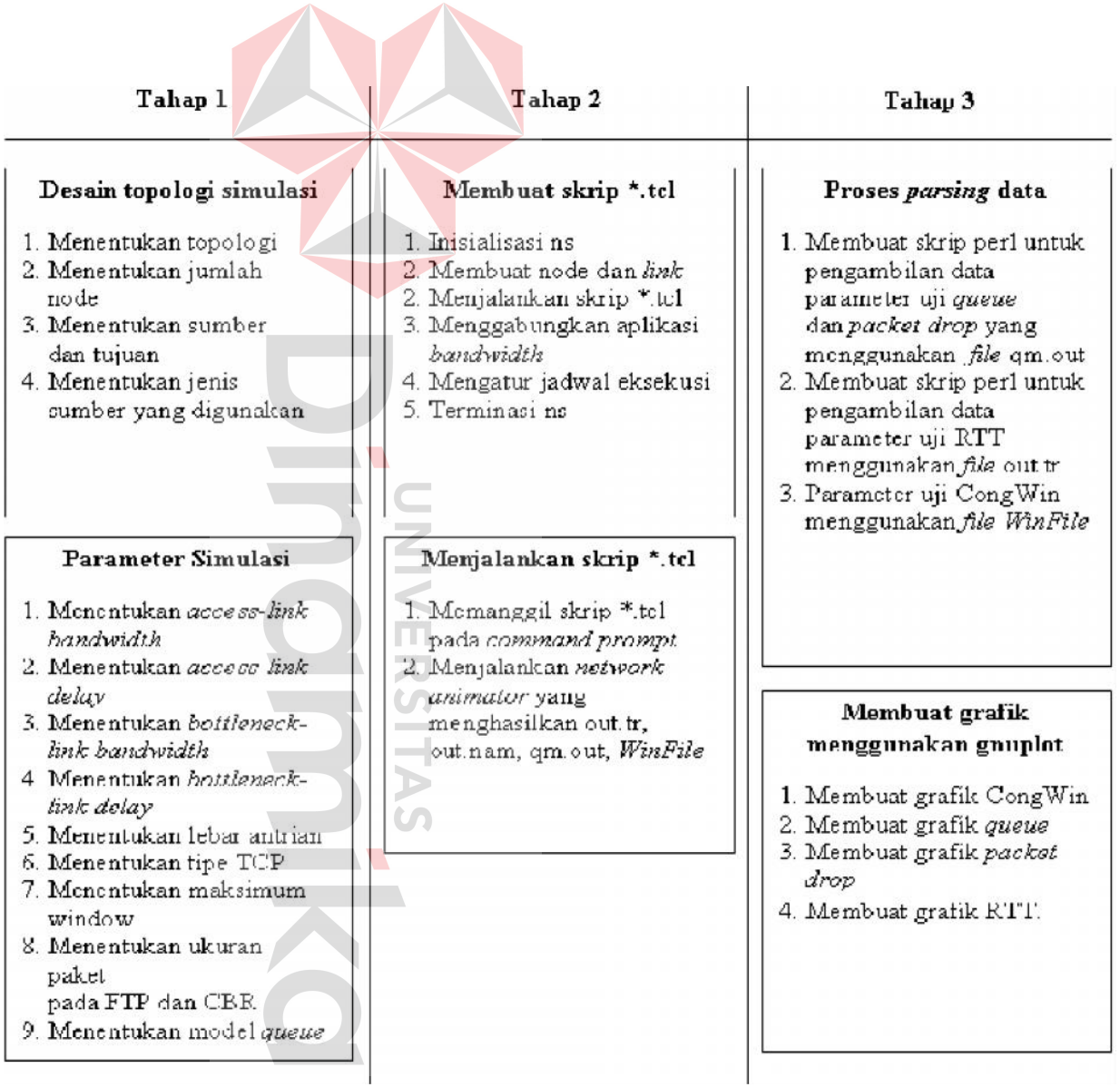
### 3.1.7 Analisis Perbandingan *Packet Drop*

Analisis perbandingan *packet drop* merupakan hasil akhir yang ingin dicapai dalam penelitian ini. Pada bagian ini yang dilakukan adalah melakukan analisis berdasarkan parameter uji *packet drop*. *Packet drop* dilihat berapa banyak paket yang telah dibuang pada antrian untuk menuju jalur *bottleneck* yang dihasilkan dalam suatu simulasi jaringan dengan menggunakan suatu algoritma tertentu(Rahman, Kabir, Lutfullah, & Amin, 2008). Ada dua tahap dalam

melakukan analisis, pertama yaitu menganalisis hasil grafik yang dilakukan menggunakan gnuplot dan yang kedua adalah menganalisis hasil grafik yang sudah digabungkan ke dalam satu gambar antara TCP Tahoe, Reno dan SACK.

### 3.1.8 Analisis Perbandingan RTT

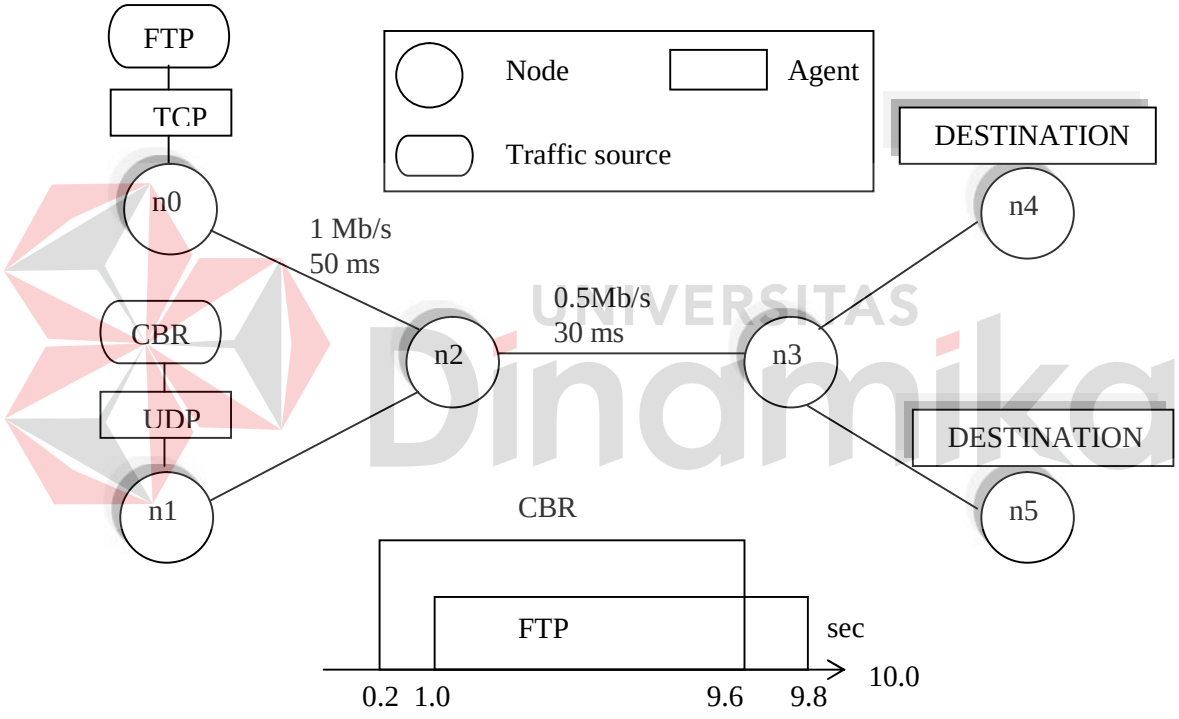
Analisis perbandingan RTT merupakan hasil akhir yang ingin dicapai dalam penelitian ini. Pada bagian ini yang dilakukan adalah melakukan analisis berdasarkan parameter uji RTT. RTTmerupakan waktu yang dibutuhkan dalam melakukan pengiriman paket dari node sumber menuju node tujuan hingga paket ACK dari tujuan yang dikirim ke sumber sebagai informasi pengiriman(Rahman, Kabir, Lutfullah, & Amin, 2008). Ada dua tahap dalam melakukan analisis, pertama yaitu menganalisis hasil grafik yang dilakukan menggunakan gnuplot dan yang kedua adalah menganalisis hasil grafik yang sudah digabungkan ke dalam satu gambar antara TCP Tahoe, Reno dan SACK.


Tahap 1
Tahap 2
Tahap 3
Desain topologi simulasi
1. Menentukan topologi
2. Menentukan jumlah node
3. Menentukan sumber dan tujuan
4. Menentukan jenis sumber yang digunakan
Membuat skrip *.tcl
1. Inisialisasi ns
2. Membuat node dan link
2. Menjalankan skrip *.tcl
3. Menggabungkan aplikasi bandwidth
4. Mengatur jadwal eksekusi
5. Terminasi ns
Proses parsing data
1. Membuat skrip perl untuk pengambilan data parameter uji queue dan packet drop yang menggunakan file qm.out
2. Membuat skrip perl untuk pengambilan data parameter uji RTT menggunakan file out.tr
3. Parameter uji CongWin menggunakan file WinFile
Parameter Simulasi
1. Menentukan access-link bandwidth
2. Menentukan access link delay
3. Menentukan bottleneck-link bandwidth
4 Menentukan bottleneck-link delay
5. Menentukan lebar antrian
6. Menentukan tipe TCP
7. Menentukan maksimum window
8. Menentukan ukuran paket pada FTP dan CBR
9. Menentukan model queue
Menjalankan skrip *.tcl
1. Memanggil skrip *.tcl pada command prompt
2. Menjalankan network animator yang menghasilkan out.tr, out.nam, qm.out, WinFile
Membuat grafik menggunakan gnuplot
1. Membuat grafik CongWin
2. Membuat grafik queue
3. Membuat grafik packet drop
4. Membuat grafik RTT

dengan pengaturan dan parameter yang ditentukan. Tahap yang ketiga adalah melakukan parsing data atau pengambilan data yang diinginkan yaitu nilai CongWin, *queue*, *packet drop* dan RTT. Dan proses terkahir yaitu membuat grafik menggunakan gnuplot. Penjelasan perancangan lebih jelasnya dijelaskan pada sub bab berikutnya.

### 3.2.1 Desain Topologi Simulasi

Gambar3.3 merupakan desain topologi yang akan digunakan dalam simulasi.




Gambar 3.3 Topologi Simulasi

Keterangan:

1. Ada 6 buah node yang disimulasikan: node n0, n1, n2, n3, n4, n5.
2. Hubungan *duplex* terjadi pada node n0-n2, n1-n2,n3-n4 dan n3-n5.
3. Hubungan *simplex* terjadi pada node n2-n3 dan n3-n2.
4. *Access link* memiliki *bandwidth* 1 Mb/s dengan delay time 50 ms.
5. TCP adalah agent untuk n0 dan UDP adalah *agent* untuk n1.

6. Node n0 mengirimkan trafik FTP dengan tujuan node n4 mulai dari 1.0 s simulasi dan diakhiri pada 9.8 s simulasi,
7. Node n1 mengirimkan trafik CBR dengan tujuan node n5 mulai dari 0.2 s simulasi dan diakhiri pada 9.6 s simulasi.

Pada Gambar 3.3 merupakan model jaringan untuk simulasi TCP yang digunakan. Pada simulasi ini digunakan model topologi Dumb-bell. Menurut Welzl (2008), model topologi ini digunakan khususnya untuk melakukan evaluasi dasar perilaku *end-to-end* dan *friendliness* pada TCP varian. Model ini adalah model standar dalam penelitian yaitu terdapat dua sumber dan dua tujuan dimana paket data akan dilewatkan melalui jalur *bottleneck* yang ditunjukkan pada node 2 dan node 3. Topologi jaringan yang digunkan dalam simulasi umumnya sama yaitu menggunkan jaringan *unicast* di mana proses pengiriman dilakukan dari satu sumber ke satu tujuan. Selain itu sumber memiliki kecepatan akses lebih besar daripada *jalur bottleneck* yaitu 1 Mb/50ms sedangkan pada jalur *bottleneck* hanya 0.5 Mb/30ms.

Sumber disimbolkan dengan n0 dan n1, *bottleneck link* n2 dan n3 serta tujuan n4 dan n5. Pada n2 dan n3 biasanya berupa router. Pada Gambar 3.3, sumber menggunakan FTP *application* yang mewakili aplikasi yang berbasis nrt-VBR (*non real time Variable Bit* Rate) yang bersifat *bursty* dan tidak sensitif terhadap *delay*, serta trafik generator yaitu CBR (*Constant Bit Rate*) yang mewakili *real-time traffic* dengan *bit-rate* yang tetap untuk mengirim data menuju tujuan(Budiardjo & Thiotrisno, 2003). Sehingga untuk menggunakan FTP, kita membutuhkan *transport layer protocol* untuk mentransfer data yaitu TCP dan untuk menggunakan CBR kita membutuhkan UDP.

Topologi ini disimulasikan pada ketiga varian TCP yaitu TCP Tahoe, Reno dan SACK secara terpisah atau tidak simultan. Pada simulasi ini sumber n0 mengirim paket menuju tujuan pada n4, sedangkan n1 menuju n5. Node 0 dimulai pada detik ke 1.0 dan berakhir setelah detik 9.8 tercapai. Sedangkan node 1 dimulai pada detik ke 0.2 dan berakhir pada detik 9.6. Adapun parameter unjuk kerja yang dibandingkan dan dianalisis yaitu CongWin, *queue*, *packet drop* dan RTT.

### 3.2.2 Parameter Simulasi

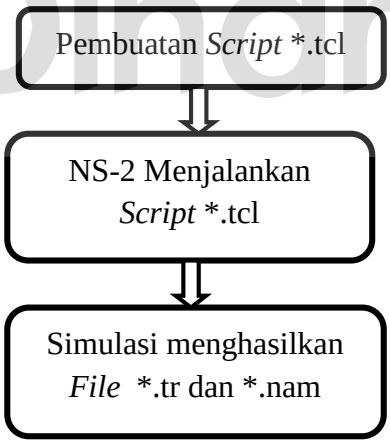
Parameter simulasi adalah parameter yang dibuat bertujuan untuk menjalankan percobaan simulasi. Parameter terdiri dari variasi-variasi yang digunakan dalam mengatur jalannya simulasi sehingga setiap variasi mempunyai hasil yang berbeda-beda. Parameter simulasi dalam penelitian ini dibuat berdasarkan referensi (Rahman, Kabir, Lutfullah, & Amin, 2008) yaitu berdasarkan ukuran paket untuk CBR 1460 bytes yakni MTU Ethernet=1500-(Header+Payload) = 1500-(20+20) = 1500-40 = 1460 bytes. Sedangkan TCP menggunakan secara *default* yaitu 1000 bytes(Jusak, 2011), *queue management* yaitu Drop Tail pada *access-link* dan RED pada *bottleneck-link*, maksimum *window size*= 30 dan *queue limits* 25. Dalam implementasinya, nilai maksimum *window* diatur oleh sisi pengirim dan penerima melalui negosiasi pada saat inisialisasi pertama kali. Dari nilai maksimum *window* akan didapatkan nilai *threshold*. *Threshold* atau *slowstart threshold* adalah nilai yang digunakan pada saat pengirim mengalami fase *slowstart* yang didapatkan dari setengah nilai CongWin.

Pada parameter *bandwidth*, penulis menggunakan nilai pada *access-link* 1Mb dengan *delay propagation* 50ms. Dari percobaan yang telah dilakukan yang mendukung penelitian ini yaitu tentang *bandwidth* di sisi *bottleneck*, di mana nilai di atas 0.5 Mb/30ms menghasilkan nilai *packet drop* hanya 2000 bytes yang artinya hanya dua paket yang mengalami *packet drop* karena ukuran paket FTP=1000 bytes, sehingga penulis hanya menentukan tiga variasi *bandwidth* yaitu 0.5Mb/30ms, 0.256Mb/30ms dan 0.128/30ms untuk menghasilkan skenario yang buruk. Dan untuk parameter lain yang tidak disebutkan disesuaikan dengan *default* pada NS-2.34.

### 3.2.3 Membuat Skrip *.tcl

Gambar 3.4 merupakan langkah-langkah pembuatan simulasi jaringan pada NS-2 yang akan dilakukan.






Gambar 3.4 Langkah-langkah pembuatan simulasi

Langkah pertama yaitu membuat *script* *.tcl sesuai dengan skenario yang telah ditentukan sebelumnya. Berikut ini struktur dasar dari *script* *.tcl menurut Jusak (2011):

**1. Inisialisasi dan terminasi ns**

Simulasi dengan NS selalu dimulai dengan mendefinisikan sebuah variabel atau *object* sebagai *instance* dari kelas Simulator dengan cara sebagai berikut:

```
set ns [new Simulator]
```

Variabel ns dalam perintah di atas dapat diganti dengan nama lain, tetapi secara umum yang digunakan adalah ns seperti di atas. Untuk menyimpan data keluaran hasil dari simulasi (trace files) dan juga sebuah *file* lagi untuk kebutuhan simulasi (nam files) akan dibuat dua buah *file* dengan perintah “open” seperti berikut:

```
#Buka trace files

set tracefile1 [open out.tr w]
$ns trace-all $tracefile1

#Buka NAM trace files

set namfile [open out.nam w]
$ns namtrace-all $namfile
```

Skrip di atas akan membuat *file* out.tr yang akan digunakan untuk menyimpan data hasil simulasi dan *file* out.nam untuk menyimpan data hasil visualisasi. Deklarasi ‘w’ pada bagian akhir dari perintah *open* adalah perintah tulis.

Skrip di bawah ini digunakan untuk membuka *file WinFile* yang akan digunakan untuk memanggil dan menampilkan grafik CongWin.

```
#Buka file winfile

set winfile [open WinFile w]
```

Selanjutnya terminasi terhadap program dilakukan dengan cara mendeklarasikan prosedur “finish”.

#Mendefiniskan prosedur 'finish'

```
proc finish {} {
global ns tracefile1 namfile
$ns flush-trace
close $tracefile1
close $namfile
exec nam out.nam &
exit 0
}
```

Perhatikan bahwa prosedur tersebut menggunakan variabel global ns, tracefile1 dan namfile. Perintah *flush-trace* digunakan untuk menyimpan semua data hasil simulasi ke dalam tracefile1 dan namfile. Perintah *exit* akan mengakhiri aplikasi dan mengembalikan status dengan angka 0 ke sistem. Perintah *exit* 0 adalah perintah *default* untuk membersihkan memori dari sistem, nilai yang lain dapat digunakan misalnya untuk memberikan status gagal.

Pada bagian akhir dari program, prosedur ‘finish’ harus dipanggil dengan indikasi waktu (dalam detik) terminasi dari program, misalnya:

```
$ns at 10.0 finish
```

Selanjutnya untuk memulai simulasi atau menjalankan program dapat dilakukan dengan menggunakan perintah:

```
$ns run
```

**2. Membuat node dan link**

Mendefinisikan sebuah node pada NS pada dasarnya adalah membuat sebuah variabel, sebagai berikut:

```
set n0 [$ns node]
```

Selanjunya untuk menggunakan node n0 dilakukan dengan cara memanggil variabel $n0. Demikian pula node-node yang lain dapat dibuat dengan cara yang sama dengan kebutuhan dalam simulasi.

```
set n1 [$ns node]
set n2 [$ns node]
```

```
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
```

Setelah node terbuat, maka langkah selanjutnya adalah membuat *link* yang akan membuat hubungan antar node. Sebuah *link* untuk menghubungkan node $n0 dan $n2 dengan *bidirectional link* berkapasitas 1Mb dan dengan waktu tunda akibat propagasi sebesar 50ms dapat dibuat dengan cara seperti di bawah ini, begitu pula pada *link* antar node lainnya.

#membuat link antar node

```
$ns duplex-link $n0 $n2 1Mb 50ms DropTail
$ns duplex-link $n1 $n2 1Mb 50ms DropTail
```

Pada NS, antrian keluaran dari sebuah node didefinisikan sebagai bagian dari sebuah *link*. Karena itu kita perlu mendefinisikan karakteristik dari *queue* ini apabila terjadi lebihan data yang datang pada sebuah node. Opsi “DropTail” berarti bahwa data terakhir yang datang akan dibuang apabila kapasitas dari memori telah penuh.

Karena *link* dianggap memiliki memori (buffer), maka kita perlu mendefinisikan kapasitas antrian dari *link* sebagai berikut:

#Melakukan setting ukuran memori pada link n2-n3 sebesar 25

```
$ns queue-limit $n2 $n3 25
```

**3. Menggabungkan Aplikasi**

Protokol TCP adalah protokol yang mengutamakan realibilitas dalam pengiriman data, baik melalui *congestion control*, *flow control*, dan *error control*. Karena setiap kali pengiriman data TCP membutuhkan konfirmasi dalam bentuk *acknowledgment*, maka jelas terlihat di sini bahwa TCP membutuhkan *link* yang bersifat dua arah (*bidirectional link*).

#Menggabungkan aplikasi FTP pada protokol TCP

```
set ftp [new Application/FTP]
```

#Membentuk koneksi melalui protokol TCP

```
set tcp [new Agent/TCP]
```

Protokol UDP adalah lalu lintas data tidak *bursty* melainkan cenderung konstan karena protokol UDP tidak memiliki *congestion control*. Karena itu pada dasarnya protokol UDP tidak memiliki batasan kecepatan pengiriman data, kecepatan pengiriman data akan linier dengan ketersediaan *bandwidth* yang ada pada jalur komunikasi.

#Menggabungkan aplikasi CBR pada protokol UDP

```
set cbr [new Application/Traffic/CBR]
```

#Membentuk koneksi melalui protokol UDP

```
set udp [new Agent/UDP]
```

Selanjutnya untuk menggabungkan protokol ini pada sebuah node dari sumber yaitu n0 ke tujuan yaitu n4 dan dari n1 ke n5, menggunakan perintah di bawah ini:

```
$ns attach-agent $n0 $tcp
$ns attach-agent $n4 $tcp
$ns attach-agent $n1 $udp
$ns attach-agent $n5 $udp
```

Tujuan akhir dari mengalirnya aplikasi TCP didefinisikan dengan pointer yang disebut dengan *sink* sebagai berikut:

```
set sink [new Agent/TCPSink]
set null [new Agent/Null]
```

TCP *sink* bertugas mengirimkan ACK per paket yang diterima pada TCP *sender* pasangannya. Pada Tugas Akhir ini penulis menggunakan *Base TCP Sink* yang didefinisikan dengan Agent/TCPSink. Koneksi antara sumber dan tujuan akhir selanjutnya didefinisikan dengan:

```
$ns connect $tcp $sink
```

```
$ns connect $udp $null
```

Ukuran dari paket TCP secara default adalah 1000 bytes, tetapi NS memungkinkan kita mengubah ukuran paket TCP dengan nilai yang lain.

```
$tcp set packetSize_ 1000
```

Untuk menentukan nilai maksimum pada window, skrip yang digunkaan yaitu:

```
$tcp set window_ 30
```

Pada aplikasi CBR dengan ukuran paket sebesar 1000 byte dan dengan kecepatan pengiriman data sebesar 0.01 Mbps dapat ditulis sebagai berikut.

```
$cbr set packetSize_ 1460
$cbr set rate_ 0.01Mb
```

Karena TCP mampu mengalirkan beberapa aplikasi dengan menggunakan konsep *multiplexing* dan *demultiplexing*, maka kita tahu bahwa akan ada beberapa aliran data di dalam *link* yang telah kita bangun. Untuk membedakan aliran satu dengan yang lain dibutuhkan sesuatu sebagai identifikasi bagi masing-masing aliran data. Pemberian identitas pada aplikasi TCP dilakukan dengan memberikan perintah berikut:

```
$tcp set fid_ 1
$udp set fid_ 2
```

Langkah terakhir adalah menentukan jenis aplikasi dan menggabungkan dengan protokol TCP yang telah didefinisikan sebagai berikut:

#Menggabungkan aplikasi FTP pada protokol TCP

```
$ftp attach-agent $tcp
```

#Menggabungkan aplikasi CBR pada protokol UDP

```
$cbr attach-agent $udp
```

Karakter lain dari CBR adalah adanya sebuah flag random_ untuk mengindikasikan adanya *random noise* pada proses transmisi atau tidak. Kita asumsikan tidak ada *random noise*, maka:

```
$cbr set random_ false
```

Secara keseluruhan, perintah-perintah untuk mendefinisikan protokol dan aplikasi UDP dan aplikasi CBR ditujukkan dalam skrip berikut ini:

#Membentuk koneksi melalui protokol UDP

```
set udp [new Agent/UDP]
$ns attach-agent $n1 $udp
set null [new Agent/Null]
$ns attach-agent $n5 $null
$ns connect $udp $null
$udp set fid_ 2
```

#Menggabungkan aplikasi CBR pada protokol UDP

```
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set packetSize_ 1460
$cbr set rate_ 0.01Mb
$cbr set random_ false
```

Secara keseluruhan, perintah-perintah untuk mendefinisikan protokol dan aplikasi TCP dan aplikasi FTP ditujukkan dalam skrip berikut ini:

#Membentuk koneksi melalui protokol TCP

```
set tcp [new Agent/TCP]
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n4 $sink
$ns connect $tcp $sink
$tcp set window_ 30
$tcp set packetSize_ 1000
$tcp set fid_ 1
```

#Menggabungkan aplikasi FTP pada protokol TCP

```
set ftp [new Application/FTP]
$ftp attach-agent $tcp
```

**4. Mengatur jadwal eksekusi pada skrip**

Karena NS merupakan simulator kejadian diskrit, maka skrip yang telah dibuat dengan Tcl perlu mendefinisikan waktu eksekusi dari setiap kejadian. Setiap kejadian pada skrip Tcl dapat didefinisikan dengan perintah:

```
$ns at <waktu><kejadian>
```

Sehingga untuk menentukan kapan aplikasi FTP pada n0 dan n1 mulai mengirimkan data dan kapan selesai mengirimkan data digunakan perintah berikut:

```
$ns at 0.2 "$cbr start"
$ns at 1.0 "$ftp start"
$ns at 9.6 "$cbr stop"
$ns at 9.8 "$ftp stop"
```

Dengan menggunakan perintah tersebut, aplikasi FTP pada `$ftp` akan mulai berjalan pada detik ke 1.0 dan berhenti pada detik 9.8 dan untuk `$cbr` akan mulai berjalan pada detik ke 0.2 dan berhenti pada detik 9.6.

### 3.3.4 Menjalankan skrip*.tcl

Setelah menyimpan *file* topologi_tahoe.tcl pada home direktori, selanjutnya eksekusi untuk menjalankan skrip tahoe_topologi.tcl pada *command prompt* NS simulator dilakukan sebagai berikut:

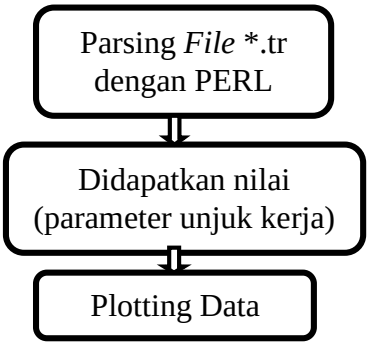
```
ns tahoe_topologi.tcl
```

Untuk pengujian apakah skrip berjalan sesuai dengan yang diinginkan dan dapat berjalan dengan benar atau tidak, yaitu dengan mengetikkan perintah di atas, apabila hasil yang ditampilkan berupa gambar visualisasi nam, maka skrip yang dibuat sudah benar dan sesuai dengan konfigurasi *.tcl.

Saat proses sedang berjalan akan menghasilkan dua buah *file* yaitu out.tr dan *file* out.nam. *File* out.tr adalah tempat untuk menyimpan data hasil simulasi sedangkan *file* out.nam untuk menyimpan data hasil visualisasi.

### 3.3.5 Parsing Data

Pada tahap analisis akan dilakukan parsing data dan plotting yang digambarkan dengan langkah-langkah pada Gambar 3.5.




Gambar 3.5 Langkah-langkah pengambilan data

Proses *parsing* kemudian dilakukan terhadap *file* *.tr dengan menggunakan Perl sehingga didapatkan nilai CongWin,*queue*, *packet drop* dan RTT. Berikut ini merupakan kerangka dasar dari skrip perl yang digunakan:

1. Inisialisasi awal: Pertama dilakukan pengecekan *file* input dengan perintah:

```
open((variabel dari file input),
$ARGV[0]) or die “cannot open the trace file”;
```

Kemudian dilakukan deklarasi variabel-variabel yang digunakan dengan perintah:

```
my $(nama variabel) = (nilai vaariabel)
```

2. *Parsing file* *.tr: *Parsing* dilakukan dengan melakukan pembacaan perbaris dari *file input* menggunakan perintah:

```
while(<(variabel dari file input)>){
```

Kemudian baris tersebut dipisah-pisahkan dengan spasi sebagai pemisahnya melalui fungsi:

```
my @line = split;
```

3. Penghitungan nilai dari parameter unjuk kerja atau QoS. Berikut ini salah satu contoh perhitungan yang digunakan:

```
@x = split(' ');
```

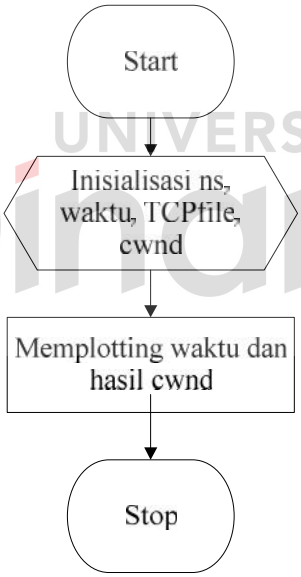
4. Menampilkan output dari hasil perhitungan dengan perintah:

```
printf(“%f”, $(nama variabel));
```

Program perl yang digunakan untuk mencari parameter uji perbandingan yaitu CongWin, *queue*, *packet drop* dan RTT. Program perl yang digunakan dalam penelitian ini dapat dijelaskan dalam bentuk flowchart berikut ini.

**1. Flowchart CongWin**

Pada Gambar 3.6 merupakan flowchart untuk menghasilkan nilai CongWin. Pertama kali yang dilakukan adalah inisialisasi *network simulator* (ns), inisialisasi waktu, mengambil *file data source* yang sedang berjalan melalui variabel `$chan`. Setelah data terkumpul langkah selanjutnya adalah melakukan plotting dengan menggunakan function `expr`.

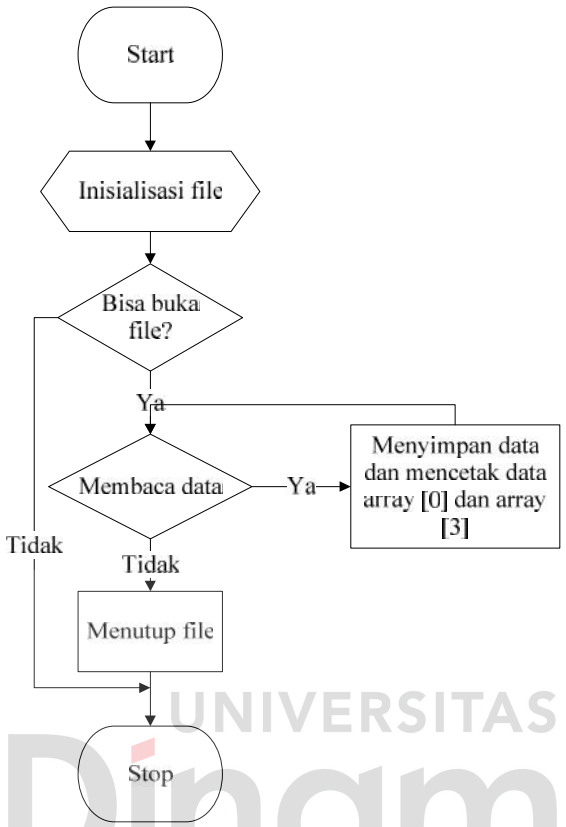



Gambar 3.6 Flowchart CongWin

**2. Flowchart *queue***

Pada Gambar 3.7 merupakan flowchart untuk menghasilkan nilai *queue* yang dianalisis pada sisi *bottleneck*. Yang pertama dilakukan adalah membuka data dari hasil keluaran *procedure qmon* yang dikonfigurasi pada *.tcl yaitu qm.out. Setelah itu membaca data dan selanjutnya dilakukan proses penyaringan

berdasarkan kolom 0 yang berarti waktu pengambilan sampel dan kolom 3 yaitu ukuran *queue* dalam satuan byte.

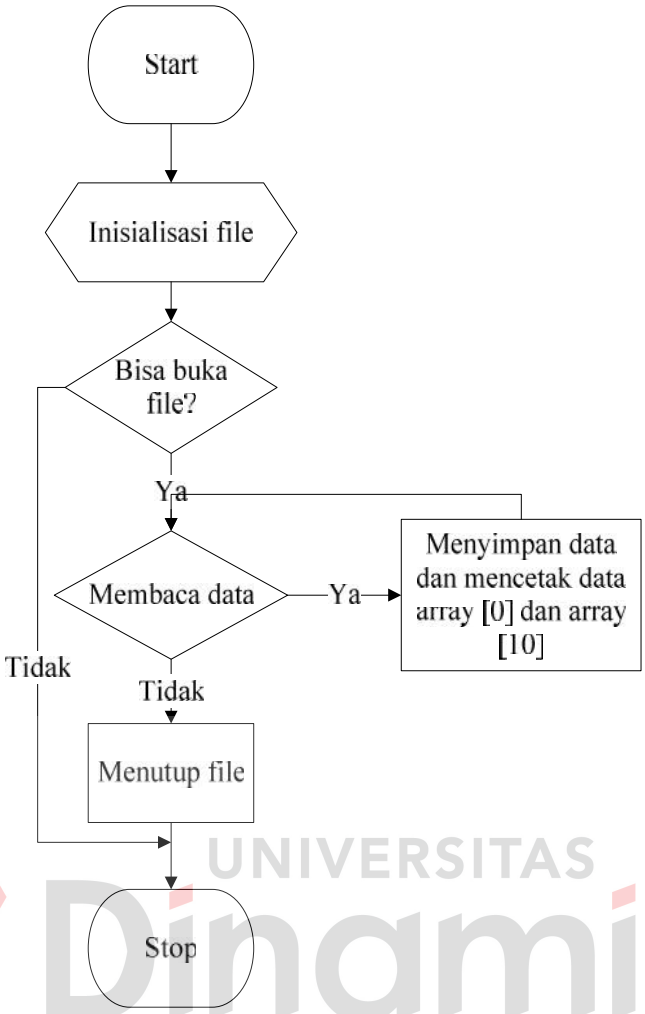



Gambar 3.7 Flowchart Queue

**3. Flowchart *packet drop***

Gambar 3.8 merupakan flowchart untuk menghasilkan nilai *packet drop* yang dianalisis pada sisi *bottleneck*. Karena TCP mempunyai *congestion control* dan bersifat *reliable connection*, maka seluruh pengiriman dipastikan tiba di node tujuan, sehingga *packet drop* hanya dapat dianalisis pada sisi *bottleneck*.

Pertama-tama yang dilakukan yaitu membuka data dari hasil keluaran *procedure qmon* yang dikonfigurasi pada *.tcl yaitu qm.out. Setelah itu membaca data dan selanjutnya dilakukan proses penyaringan berdasarkan kolom 0 yang berarti waktu pengambilan sampel dan kolom10 yaitu jumlah data yang di*drop* oleh *queue* dalam satuan byte.

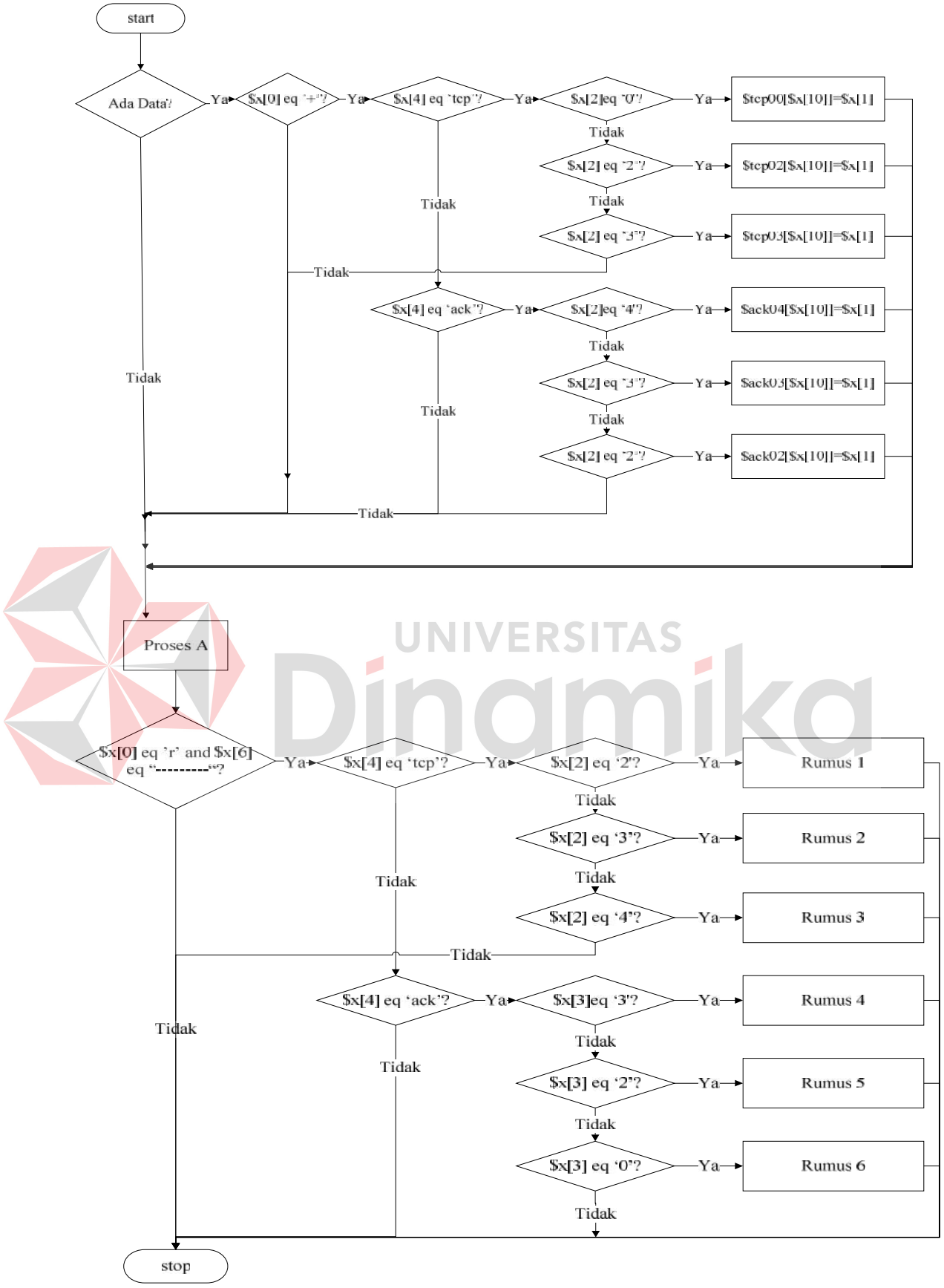





Gambar 3.8 Flowchart Packet Drop

**4. Flowchart RTT**

Pada Gambar 3.9 merupakan flowchart untuk menghasilkan nilai RTT. RTT merupakan waktu yang dibutuhkan oleh sumber dalam melakukan pengiriman ke tujuan dan kemudian kembali ke sumber lagi. Sehingga, RTT = selisih waktu berangkat + selisih waktu kembali




Gambar 3.9 Flowchart RTT

Pertama-tama yang dilakukan adalah membuka data hasil keluaran dari out.tr. Selanjutnya dilakukan pembacaan terhadap seluruh data yang ada dan dilakukan penyaringan flag (+) yang artinya adalah proses pengiriman paket. Kemudian proses penyaringan tipe paket TCP dan kemudian penyaringan pada node 0 dan menyimpan nilai waktu pengiriman ke dalam suatu variabel. Begitu pula untuk node 2 dan node 3 juga dilakukan proses yang sama.

Flag (+) dimiliki pula oleh tipe paket ACK sehingga perlu dilakukan penyaringan terhadap tipe paket tersebut. Kemudian ACK melakukan penyaringan node yang dimulai dari node 4, 3 dan 2. Kemudian menyimpan masing-masing waktu tentang pengiriman ACK setiap node. Setelah itu, proses penyaringan event (r) yang artinya adalah flag penerimaan paket yang bertipe TCP dan ACK. Di mana prosesnya juga sama pada proses penyaringan pada flag (+) di atas.

Rumus 1

```
$tcp10[$x[10]]=$x[1];
$rtttcp[$x[10]]=$rtttcp[$x[10]]+($tcp10[$x[10]]-$tcp00[$x[10]]);
```

Rumus 2

```
$tcp12[$x[10]]=$x[1];
$rtttcp[$x[10]]=$rtttcp[$x[10]]+($tcp12[$x[10]]-$tcp02[$x[10]]);
```

Rumus 3

```
$tcp13[$x[10]]=$x[1];
$rtttcp[$x[10]]=$rtttcp[$x[10]]+($tcp13[$x[10]]-$tcp03[$x[10]]);
```

Rumus 4

```
$ack13[$x[10]]=$x[1];
$rttack[$x[10]]=$rttack[$x[10]]+($ack13[$x[10]]-$ack04[$x[10]]);
```

Rumus 5

```
$ack12[$x[10]]=$x[1];
$rttack[$x[10]]=$rttack[$x[10]]+($ack12[$x[10]]-$ack03[$x[10]]);
$tmp1[$x[10]]=$tcp2[$x[10]]-$tcp1[$x[10]]
```

Rumus 6

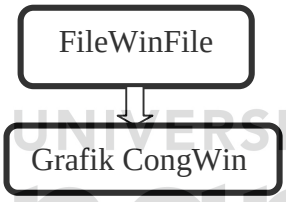
```
ack10[$x[10]]=$x[1];
$rttack[$x[10]]=$rttack[$x[10]]+($ack10[$x[10]]-$ack02[$x[10]]);
$rtt = $rtttcp[$x[10]]+$rttack[$x[10]];
print STDOUT "$tcp00[$x[10]] $rtt\n";
```

### 3.3.6 Ploting Data

Plotting data yang dilakukan ada dua tahap, yaitu yang pertama menggunakan gnuplot, dan kedua menggunakan *Microsoft Excel 2007* untuk mendapatkan susunan grafik yang diinginkan sehingga memudahkan untuk melakukan perbandingan antara TCP varian.

Plotting datamenggunakan gnuplot dijelaskan sebagai berikut:

1) Pemanggilan data *congestion window* (*CWND*)

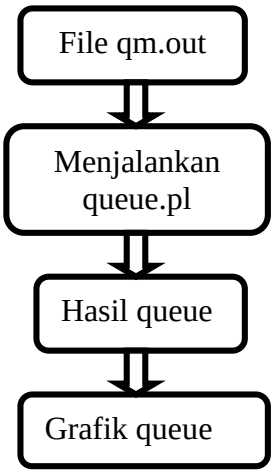



Gambar 3.10 Proses pemanggilan CongWin

Pada proses pemanggilan data CongWin seperti Gambar 3.10, tidak perlu menggunakan program perl untuk mendapatkan data *fileWinFile*. *File WinFile* telah dipanggil ketika topologi dijalankan pertama kali yaitu melalui *procedureWinFile*. Setelah itu *file WinFile* dapat dipanggil dan disajikan secara grafik dengan menggunakan gnuplot.

2) Pemanggilan data *queue*

Saat proses topologi berjalan maka seluruh *event*(kejadian) disimpan pada out.tr. Selain itu terdapat out.nam dan qm.out. Untuk memanggil data *queue* maka kita membutuhkan data yang berada pada *file* qm.out. Sebelumnya data qm.out didapat dari *procedure* qmon. Proses pemanggilan digambarkan pada Gambar 3.11.




Gambar 3.11 Proses pemanggilan *queue*

Selanjutnya dari *file* tersebut kita dapat memodifikasi *output* apa yang akan diinginkan melalui pemrograman perl. Selanjutnya hasilnya dipanggil dan disajikan menggunakan gnuplot menjadi sebuah grafik. Skrip queue.pl adalah skrip untuk mencari nilai *queue*. Kemudian hasilnya disimpan dalam QueuePacket_tahoe. Setelah itu QueuePacket_tahoe dipanggil untuk disajikan dalam bentuk grafik.

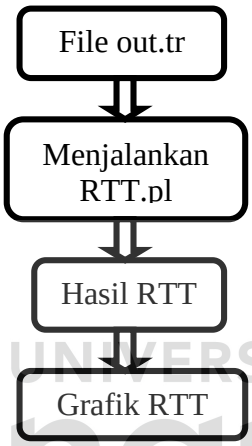
3) Pemanggilan data *packet drop*

Untuk proses pemanggilan *packet drop* hampir sama dengan proses pemanggilan *queue* pada Gambar 3.11. Perbedaannya adalah untuk mengambil data pada program queue.pl, *packet drop* memiliki kolom yang berbeda dengan *queue*. Sehingga harus dilakukan perubahan skrip perl sesuai dengan kolom untuk menampilkan *packet drop.*

Setelah program perl dipanggil, maka hasilnya dapat disimpan pada PacketDrop_tahoe dan kemudian disajikan dalam bentuk grafik menggunakan gnuplot. Setelah disajikan dalam bentuk grafik.

4) Pemanggilan data RTT

Pada proses pemanggilan data RTT seperti Gambar 3.12, hampir sama dengan pemanggilan *queue* atau *packet drop*. Saat proses topologi berjalan maka seluruh *event*(kejadian) disimpan pada out.tr. Selain itu terdapat out.nam dan qm.out. Untuk menghasilkan RTT, kita membutuhkan data yang berada pada *file* out.tr. Sebelumnya *file* out.tr didapat dari hasil simulasi yang terekam dan disimpan di dalam out.tr.




Gambar 3.12 Proses pemanggilan RTT

Setelah mendapatkan *file* out.tr, maka selanjutnya menjalankan RTT.pl untuk menghasilkan nilai RTT yang diinginkan. Setelah menjalankan skrip RTT.pl, maka akan didapatkan hasil berupa *file* rtt_tahoedan langkah selanjutnya yaitu melakukan plotting pada *file* rtt_tahoe.

# BAB IV

# HASIL DAN PEMBAHASAN

## 4.1 Kebutuhan Sistem

Untuk dapat menjalankan sistem simulasi yang dibuat, diperlukan perangkat keras dan perangkat lunak dengan spesifikasi tertentu. Adapun kebutuhan perangkat keras dan perangkat lunak untuk sistem ini adalah sebagai berikut:

1. Perangkat Keras

Perangkat keras yang digunakan dalam tugas akhir ini adalah komputer atau laptop dengan spesifikasi berikut ini:

a. Prosesor: Intel Core 2 Duo U7300
b. Memori: 3 GB
c. Sistem Operasi: Linux Ubuntu 10.04 dan Windows 7

2. Perangkat Lunak

Perangkat lunak yang digunakan dalam penelitian ini antara lain:

a. *Network Simulator 2* (NS-2) versi 2.34. Aplikasi ini merupakan aplikasi utama yang digunakan untuk menjalankan proses simulasi.
b. *Perl*. Aplikasi ini digunakan untuk mengolah *file* *.tr yang merupakan data output dari simulasi dengan menggunakan NS-2.
c. Gnuplot. Aplikasi ini digunakan untuk melakukan plotting dengan menggunakan NS-2.
d. *Microsoft Excel 2007*. Aplikasi ini digunakan untuk mengolah data dan membuat grafik dari data hasil simulasi.

## 4.2 Parameter Simulasi

Parameter simulasi bertujuan untuk menjalankan percobaan-percobaan simulasi. Parameter ini diatur di dalam skrip *.tcl. Parameter yang digunakan dalam konfigurasi ditunjukkan pada Tabel 4.1.

Tabel 4.1 Parameter Simulasi

| Percb. ke- | Access bandwidth / link delay | Bottleneck bandwidth/ link delay | Lebar *Queue* | Ukuran *Window* | Ukuran paket (bytes) | | Model *Queue* | |
|---|---|---|---|---|---|---|---|---|
| | | | | | FTP | CBR | Access-link | Bottleneck-link |
| 1. | 1Mb/50ms | 0.5Mb/30ms | 25 | 30 | 1000 | 1460 | Drop Tail | RED |
| 2. | 1Mb/50ms | 0.256Mb/30ms | 25 | 30 | 1000 | 1460 | Drop Tail | RED |
| 3. | 1Mb/50ms | 0.128Mb/30ms | 25 | 30 | 1000 | 1460 | Drop Tail | RED |



## 4.3 Menjalankan Simulasi

### 4.3.1 TCP Tahoe

Konfigurasi skrip *.tcl dengannilai *bandwidth* yang bervariasi akan menghasilkan nilai yang bervariasi pula. Skrip *.tcl berjalan pada NS dan akan menghasilkan *trace file* dan *nam file*, kemudian dengan menambahkan beberapa skrip perl untuk mendapatkan *trace file* yang diinginkan seperti *queue, packet drop* dan RTT. Di bawah ini merupakan skrip percobaan 1 yaitu tahoe_topologi.tcl berdasarkan variasi ukuran *bandwidth*=0.5 Mb/30ms dan hasil grafik sesuai dengan percobaan yang telah dilakukan.

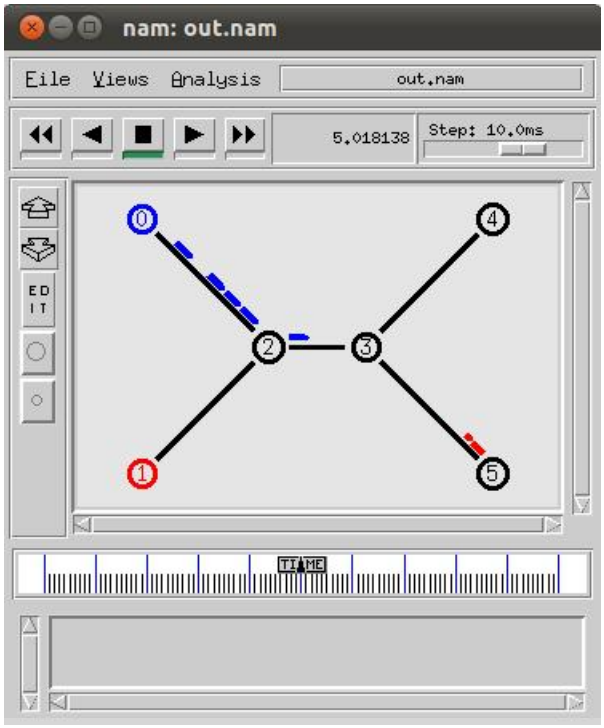
Pada skrip di bawah ini juga ditambahkan prosedur untuk menghasilkan *file WinFile* yang digunakan dalam mencari nilai CongWin dan prosedur*qmon* yang menghasilkan qm.out di mana dengan menggunakan perl dihasilkan nilai

*queue* dan *packet drop*. Selain itu terdapat out.truntuk mencari nilaiRTT melalui pemrograman perl.

```
set ns [new Simulator]
$ns color 1 Blue
$ns color 2 Red
set tracefile1 [open out.tr w]
$ns trace-all $tracefile1
set namfile [open out.nam w]
$ns namtrace-all $namfile
proc finish {} {
global ns tracefile1 namfile
$ns flush-trace
close $tracefile1
close $namfile
exec nam out.nam &
exit 0
}
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
$n0 color Blue
$n1 color Red
$ns duplex-link $n0 $n2 1Mb 50ms DropTail
$ns duplex-link $n1 $n2 1Mb 50ms DropTail
$ns simplex-link $n2 $n3 0.5Mb 30ms RED
$ns simplex-link $n3 $n2 0.5Mb 30ms RED
$ns duplex-link $n3 $n4 1Mb 50ms DropTail
$ns duplex-link $n3 $n5 1Mb 50ms DropTail
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns simplex-link-op $n2 $n3 orient right
$ns simplex-link-op $n3 $n2 orient left
$ns duplex-link-op $n3 $n4 orient right-up
$ns duplex-link-op $n3 $n5 orient right-down
$ns queue-limit $n2 $n3 25
set tcp [new Agent/TCP]
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n4 $sink
$ns connect $tcp $sink
$tcp set window_ 30
$tcp set packetSize_ 1000
$tcp set fid_ 1
set ftp [new Application/FTP]
$ftp attach-agent $tcp
set udp [new Agent/UDP]
$ns attach-agent $n1 $udp
set null [new Agent/Null]
$ns attach-agent $n5 $null
$ns connect $udp $null
$udp set fid_ 2
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
```

```
$cbr set packetSize_ 1460
$cbr set rate_ 0.01Mb
$cbr set random_ false
$ns at 0.2 "$cbr start"
$ns at 1.0 "$ftp start"
$ns at 9.6 "$cbr stop"
$ns at 9.8 "$ftp stop"
#############################################################
#prosedur untuk menghasilkan file WinFile
proc plotWindow {tcpSource file} {
global ns
set time 0.1
set now [$ns now]
set cwnd [$tcpSource set cwnd_]
set chan [open $file a]
puts $chan "$now $cwnd"
close $chan
$ns at [expr $now+$time] "plotWindow $tcpSource $file"
}
#############################################################
#prosedur yang menghasilkan file qm.out untuk mencari nilai queue
set qmon [$ns monitor-queue $n2 $n3 [open qm.out w] 0.1];
[$ns link $n2 $n3] queue-sample-timeout;
$ns at 0.1 "plotWindow $tcp WinFile"
$ns at 10.0 finish
$ns run
```

Untuk skrip pada percobaan 2 dan 3 algoritma TCP Tahoe, hampir sama dengan skrip pada percobaan 1. Bedanya yaitu pada pengaturan parameter *bandwidth*. Pada percobaan 2, *bandwidth* diatur 0.256Mb/30ms dan pada percobaan 3, *bandwidth* diatur 0.128Mb/30ms. Sehingga pada kali ini skrip percobaan 2 dan percobaan 3 tidak ditampilkan.

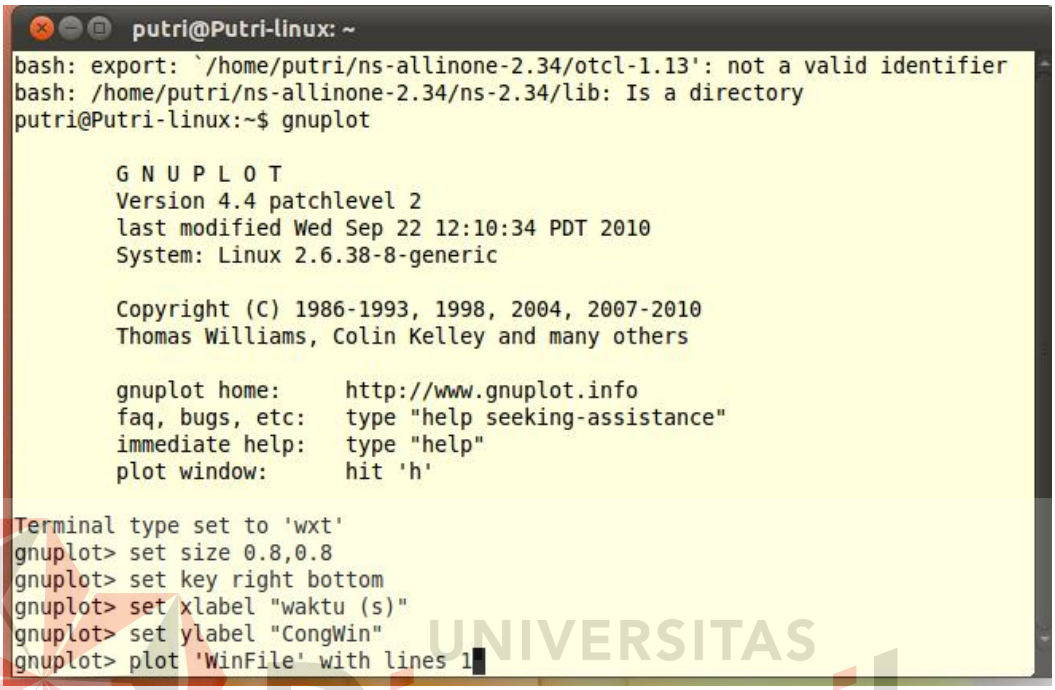



Gambar 4.1 Tahoe_topologi.nam

Gambar 4.1 merupakan gambar *network animator*(nam) dari konfigurasi tahoe_topologi yang telah dijalankan. Gambar topologi telah sesuai dengan konfigurasi yang dilakukan, baik besar *bandwidth*, banyaknya node, banyaknya sumber dan tujuan, kemana arah tujuan pengiriman *packet*, waktu yang digunakan dalam proses pengiriman, banyaknya *packet size* yang dikirim serta pengaturan pada ukuran *queue* dan lain sebagainya.

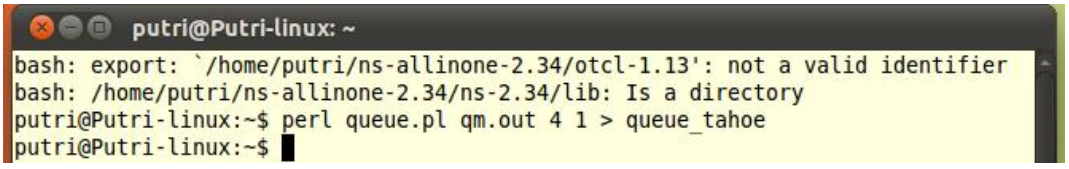
Gambar 4.1 menunjukkan terdapat 6 buah node di mana node 0 merupakan sumber menggunakan aplikasi FTP yang diwakili warna biru dengan tujuan node 4 sedangkan node 1 sumber menggunakan generator trafik CBR yang diwakili warna merah dengan tujuan node 5. Jika tombol *play* dijalankan maka paket data mulai berjalan dan seluruh kejadian akan tersimpan di *file* out.tr.

Selain menghasilkan out.tr dan out.nam, proses simulasi yang sedang berjalan juga menghasilkan *file WinFile* yang berisi tentang nilai CongWin dan *file* qm.out untuk mencari nilai *queue*, *packet drop* danRTT.




Gambar 4.2 Plotting WinFile Tahoe

Gambar 4.2 adalah skrip untuk memplotting *fileWinFile* yang berisi nilai CongWin yang dihasilkan melalui *procedure plotWindow*. Plotting yang dilakukan yaitu menggunakan gnuplot seperti pada Gambar 4.2.

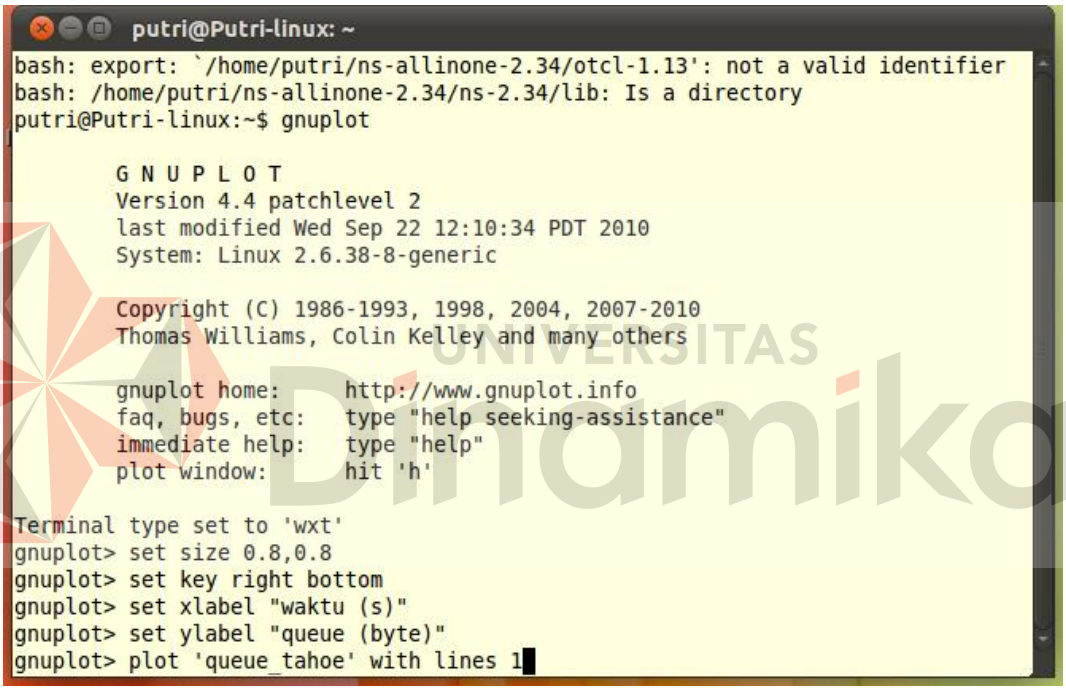



Gambar 4.3 Skrip memanggil perl queue_tahoe

Gambar 4.3 menunjukkan skrip perl untuk mencari nilai queue_tahoe. *File* qm.out diambil dari prosedur yang ditulis di dalam *file* tahoe_topologi.tcl. Setelah *file* qm.out didapatkan, maka untuk mencari nilai *queue* harus menggunakan perl dan kemudian memanggilnya seperti yang ditunjukkan pada

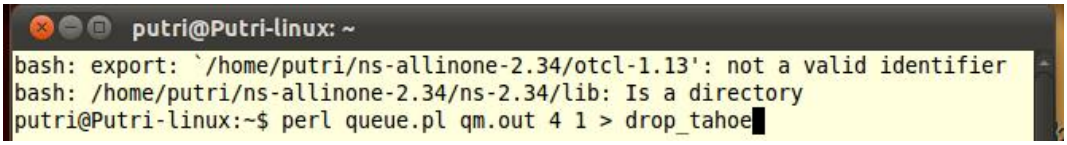
Gambar 4.3 tersebut. Di bawah ini merupakan skrip perl untuk mencari nilai *queue*, di mana nilai *queue* berada pada array [3].

```
#skrip perl untuk mencari nilai queue, di mana nilai queue berada
pada array [3]
$infile=$ARGV[0];
open (DATA,"<$infile") || die "Can't open $infile $!";
while (<DATA>)
{
      @x = split(' ');
      print STDOUT "$x[0] $x[3]\n";
}
close DATA;
exit(0);
```




Gambar 4.4 Plotting queue_tahoe pada gnuplot

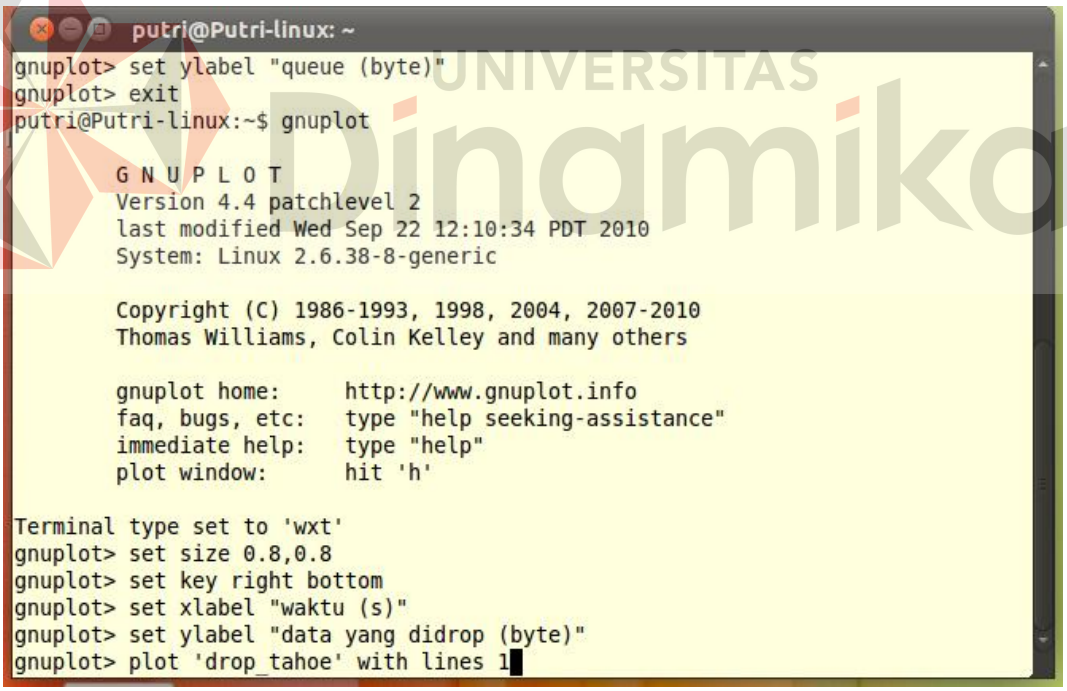
Setelah menjalankan skrip untuk mendapatkan nilai *queue* yang diinginkan dan kemudian menyimpan hasilnya ke dalam queue_tahoe, maka selanjutnya *file* queue_tahoe diplotting menggunakan gnuplot seperti pada Gambar 4.4.




Gambar 4.5 Skrip memanggil perl drop_tahoe

Gambar 4.5 adalah skrip perl untuk mendapatkan nilai *packet drop* yang kemudian disimpan ke dalam *file* drop_tahoe. Karena nilai *packet drop* berada pada array [10], maka array [3] diganti menjadi [10].

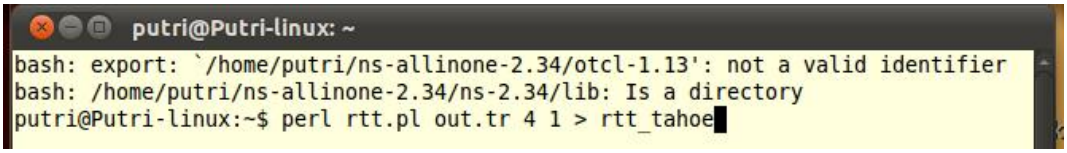
```
#skrip perl untuk mencari nilai packet drop
$infile=$ARGV[0];
open (DATA,"<$infile") || die "Can't open $infile $!";
while (<DATA>)
{
      @x = split(' ');
      print STDOUT "$x[0] $x[10]\n";
}
close DATA;
exit(0);
```




Gambar 4.6 Plotting drop_tahoe pada gnuplot

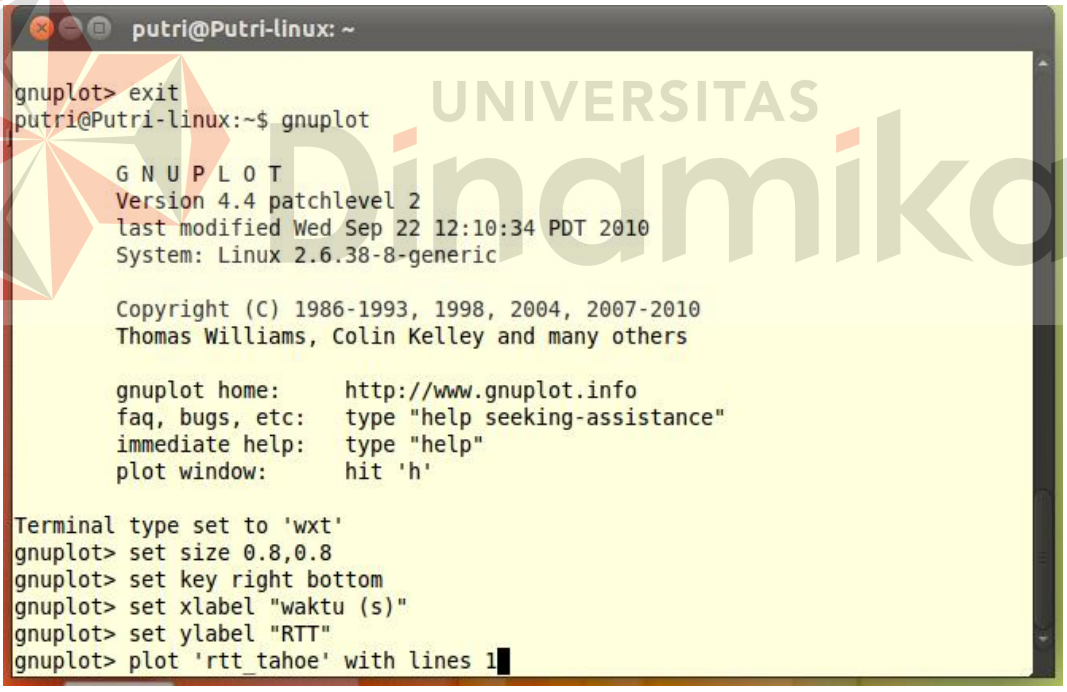
Setelah menjalankan skrip untuk mendapatkan nilai *packet drop* yang diinginkan dan kemudian menyimpan hasilnya ke dalam drop_tahoe, maka

selanjutnya *file* drop_tahoe diplotting menggunakan gnuplot seperti pada Gambar 4.6.




Gambar 4.7 Skrip memanggil perl rtt.pl

Gambar 4.7 merupakan skrip memanggil perl rtt.pl pada Tahoe yang kemudian hasilnya disimpan pada rtt_tahoe. Skrip perl rtt.pl menyortir data out.tr untuk mendapatkan nilai RTT. Skrip perl rtt.pl yang digunakan dilampirkan pada Lampiran 3. Setelah hasil RTT tersimpan dalam rtt_tahoe, maka selanjutnya dibuat grafik dengan perintah sesuai pada Gambar 4.8:



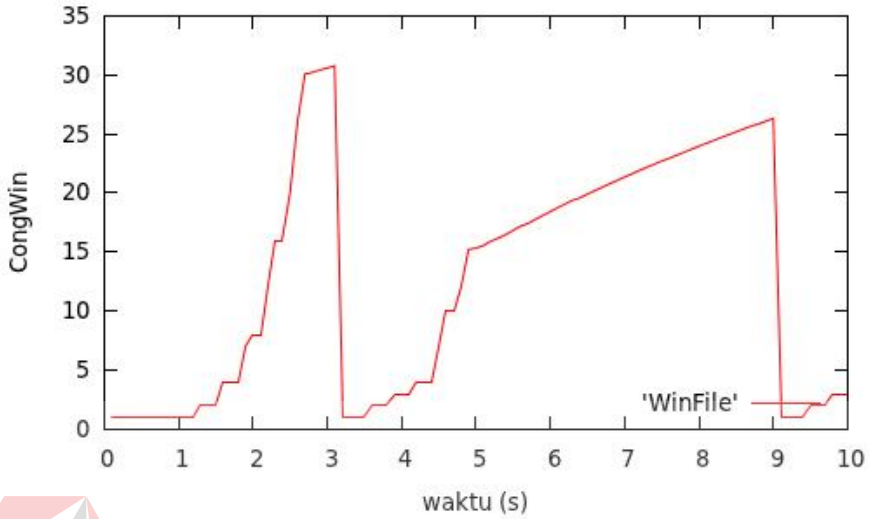

Gambar 4.8 Plotting rtt_tahoe pada gnuplot

Hasil simulasi pada percobaan 1 berdasarkan variasi *bandwidth* dengan menggunakan TCP Tahoe ditampilkan pada sub bab berikut ini.

### 4.3.2 Hasil Simulasi TCP Tahoe

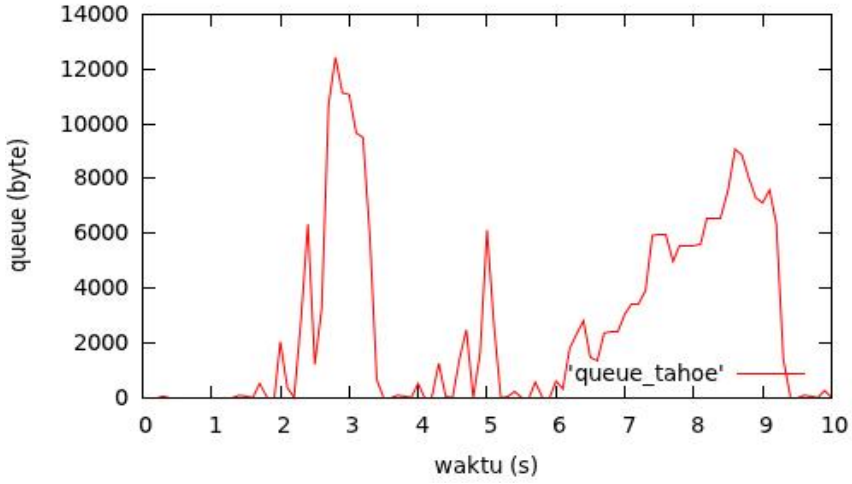
Di bawah ini merupakan grafik yang dihasilkan dari tahoe_topologi.tcl dan telah diplotting keseluruhannya yang disajikan berdasarkan percobaan.

**a. Percobaan 1**



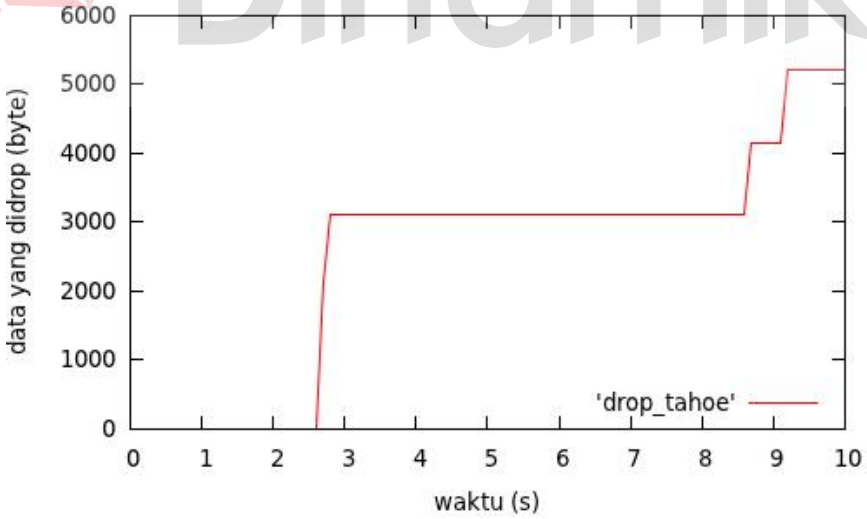

Gambar 4.9 CongWin TCP Tahoe

Gambar 4.9 merupakan grafik CongWin yang dihasilkan dari percobaan 1. CongWin digunakan untuk mengetahui kecepatan pengiriman data. Dari Gambar 4.9 diketahui bahwa Tahoe mengalami kongesti sebanyak dua kali. Dan saat terjadi kongesti, Tahoe menurunkan nilai CongWin menjadi 1 MSS kemudian naik secara eksponensial dan begitu seterusnya. Tahoe mencapai puncak CongWinyaitu mencapai nilai lebih dari 30 MSSpada pertengahan detik 2 dan 3. Dan pada akhir simulasi Tahoe kembali berada di puncak CongWinpada detik ke-9.




Gambar 4.10 *Queue* TCP Tahoe

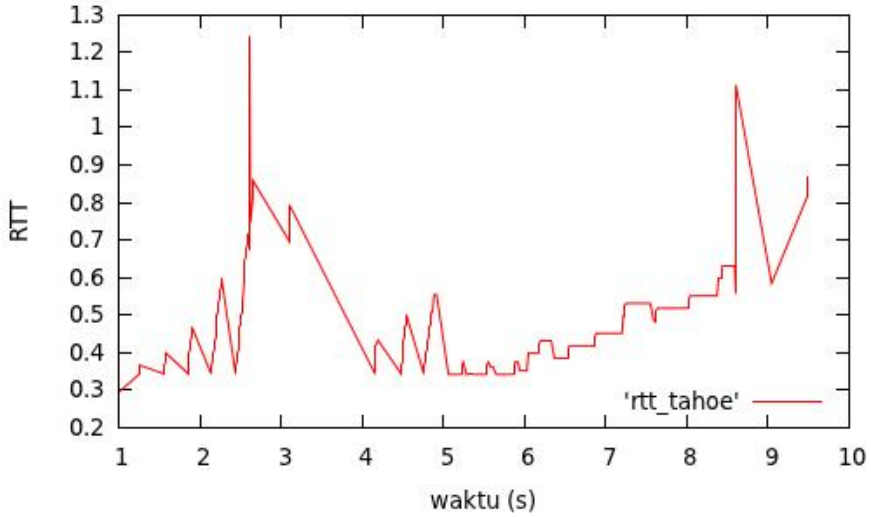
Gambar 4.10 merupakan grafik *queue* yang menunjukkan penggunaan *queue* yang cukup tinggi yaitu mencapai 12000 bytes pada detik ke-3 dan kembali tinggi pada akhir simulasi sekitar 9000 bytes pada detik ke-9. Penggunaan *queue* yang tinggi disebabkan oleh banyaknya paket yang sedang mengantri munuju jalur *bottleneck* sehingga memenuhi *buffer* yang tersedia.




Gambar 4.11 *Packet Drop* kumulatif TCP Tahoe

Gambar 4.11 menunjukkan grafik kumulatif dari *packet drop* Tahoe yang terus naik hingga mencapai nilai lebih dari 5000 bytes hingga akhir simulasi.

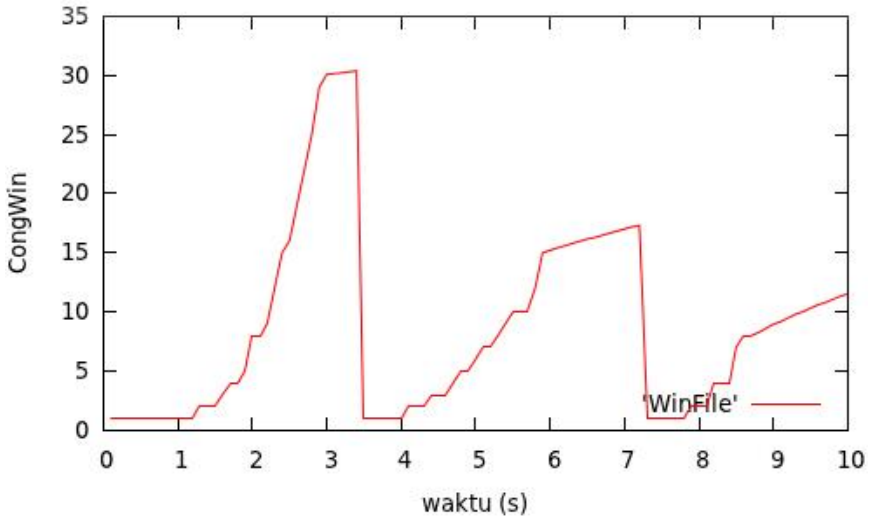
*Packet drop* kumulatif dengan nilai yang signifikan pada pertengahan detik ke-2 dan ke-3 mencapai nilai 3000 bytes.




Gambar 4.12 RTT TCP Tahoe

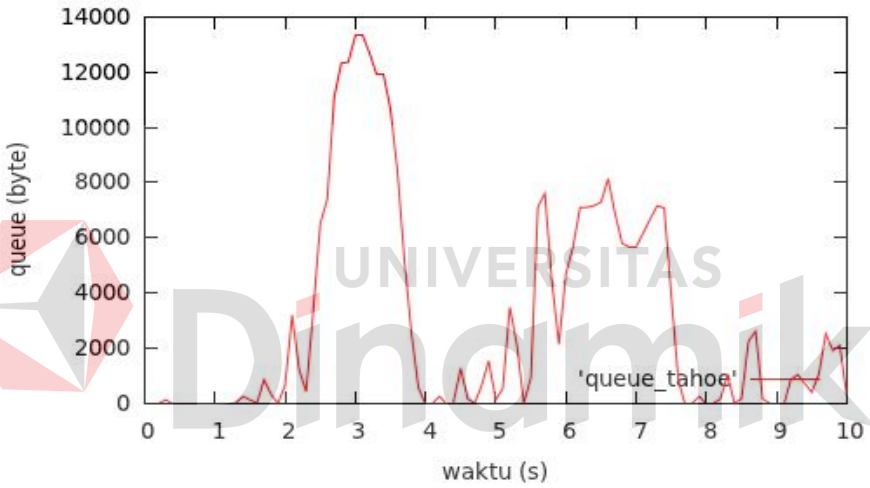
Gambar 4.12 menunjukkan grafik RTT yang menunjukkan nilai RTT yang tinggi pada pertengahan detik ke-2 dan ke-3 yang berada pada nilai 1.2 s. Dan di akhir simulasi juga mengalami kenaikan pada detik ke-9 dengan nilai 1.1 s.

**b. Percobaan 2**



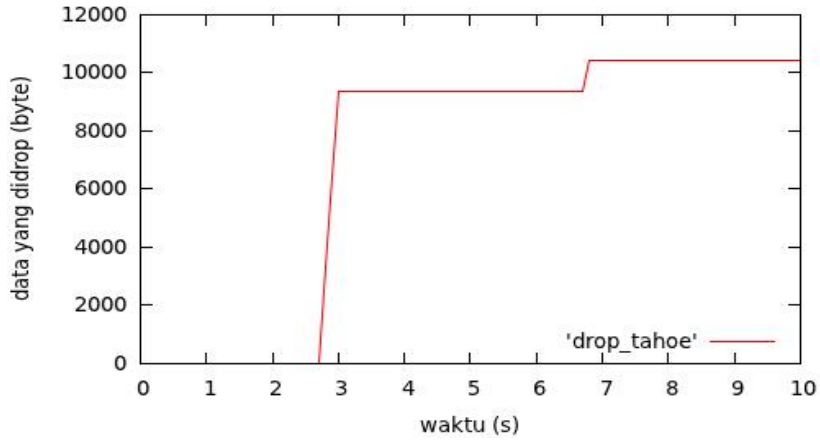

Gambar 4.13 CongWin TCP Tahoe

Dari Gambar 4.13 merupakan grafik CongWin yang naik mencapai puncak CongWin yaitu pada nilai lebih dari 30 MSS pada detik ke-3. Kemudian saat mengalami kongesti, Tahoe turun hingga 1 MSS pada pertengahan 3 dan 4. Kemudian naik secara ekponensial dan setelah mencapai nilai *threshold* yakni 15 MSS maka Tahoe naik secara linier dan puncaknya pada detik ke-7 dengan nilai sekitar 17 MSS. Sempat mengalami penurunan akibat kongesti yang kedua pada pertengahan detik 7 dan 8, kemudian naik secara eksponensial hingga akhir simulasi dan begitu seterusnya.



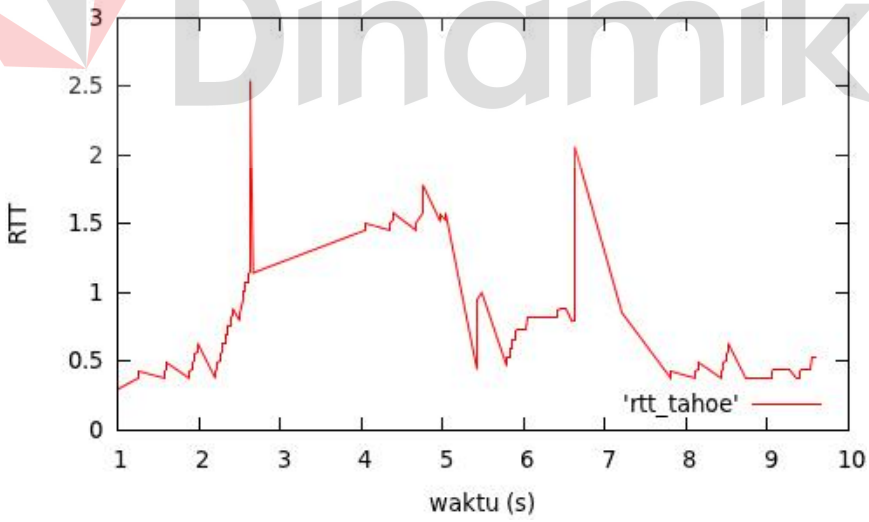

Gambar 4.14 *Queue* TCP Tahoe

Gambar 4.14 menunjukkan grafik penggunaan *queue* pada Tahoe yang tinggi. Dari grafik diketahui bahwa *queue* mencapai nilai 14000 bytes pada detik ke-3. Selanjutnya *queue* mengalami kenaikan hanya sampai nilai 6000 bytes yaitu antara detik ke-6 dan ke-7.




Gambar 4.15 *Packet Drop* kumulatif TCP Tahoe

Pada Gambar 4.15 merupakan grafik *packet drop* kumulatif dari TCP Tahoe. Antara detik ke-2 dan ke-3 nilai *packet drop* terus naik secara signifikan hingga mencapai 9000 bytes, kemudian tetap hingga detik ke-7. Selanjutnya naik hingga 11000 bytes hingga simulasi berakhir. Hal ini disebabkan penggunaan *queue* pada TCP Tahoe yang tinggi.

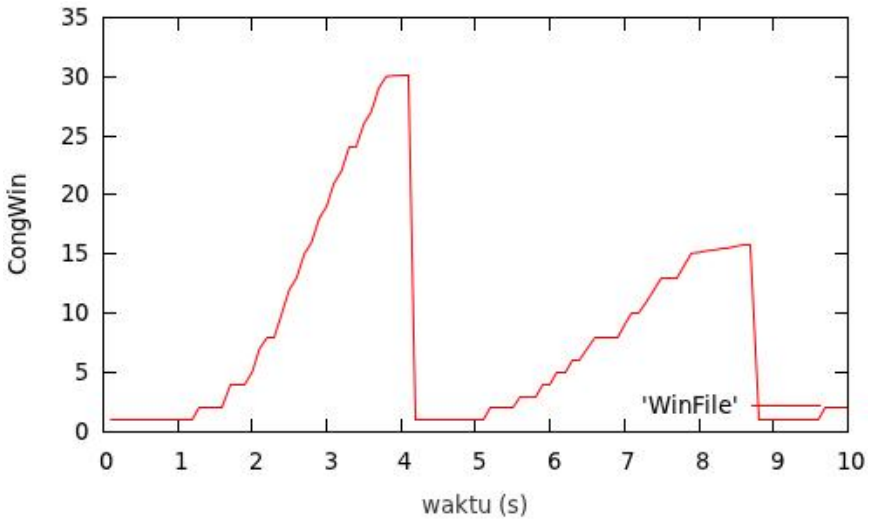



Gambar 4.16 RTT TCP Tahoe

RTT pada Gambar 4.16 menunjukkan nilai yang tinggi pada pertengahan detik ke-2 dan ke-3 yakni mencapai nilai 2.5 s. Kemudian pada pertengahan detik ke-6 dan ke-7 yang mencapai nilai 2 s. Dan pada akhir simulasi nilai RTT mulai

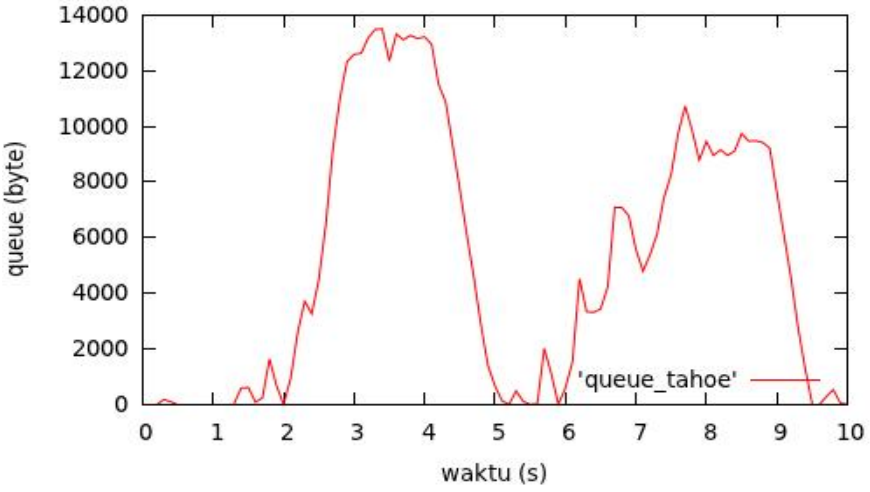
turun dan stabil dengan nilai rata-rata 0.5 s. Ini dipengaruhi oleh tingginya *packet drop* dan penurunan *bandwidth* pada TCP Tahoe.

**c. Percobaan 3**



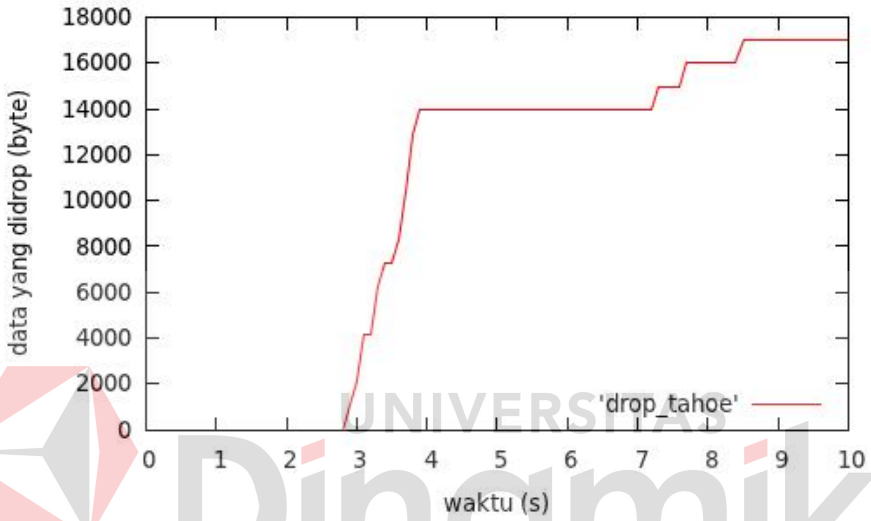

Gambar 4.17 CongWin TCP Tahoe

Gambar 4.17 merupakan CongWin Tahoe naik secara eksponensial mencapai 30 MSS pada detik ke-4. Kemudian mengalami kongesti dan turun ke nilai 1 MSS. Selanjutnya naik secara eksponensial dan ketika mencapai nilai *threshold* yakni 15 MSS, Tahoe kemudian naik secara linier hingga mencapi nilai lebih dari 15 MSS pada detik ke-9, kemudian turun hingga simulasi berakhir.



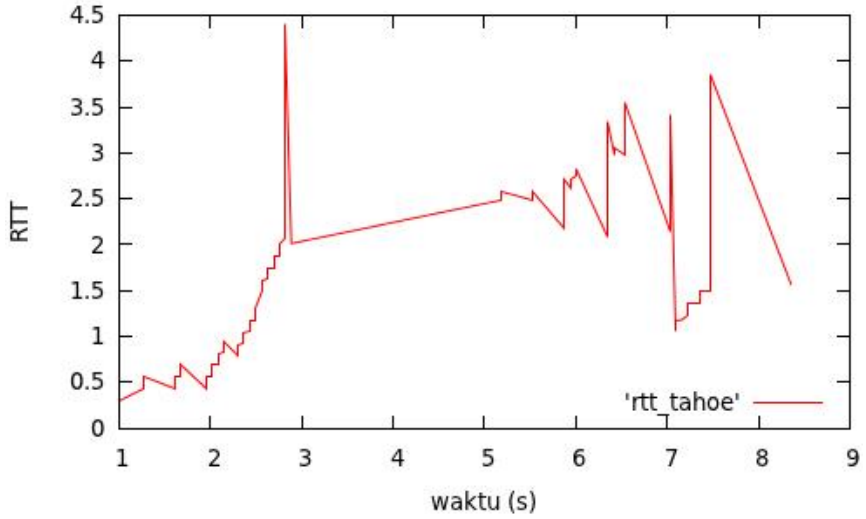

Gambar 4.18 *Queue* TCP Tahoe

Gambar 4.18 menunjukkan bahwa penggunaan *queue* pada TCP Tahoe yang sangat tinggi yaitu mencapai 14000 bytes pada detik ke-4 dan bertahan selama kurang lebih 3 detik yaitu antara detik ke-2 sampai ke-5. Kemudian naik hingga mencapai 11000 bytes pada detik ke-8 di mana berlangsung mulai dari detik ke-6 hingga pertengahan detik ke-9 dan 10. Dan mengalami penurunan penggunaan *queue* hingga akhir simulasi.




Gambar 4.19 *Packet Drop* kumulatif TCP Tahoe

Pada Gambar 4.19 merupakan grafik *packet drop* kumulatif Tahoe yang mengalami kenaikan signifikan dan sangat tinggi hingga mencapai 14000 bytes pada detik ke-4. Kemudian konstan hingga detik ke-7, selanjutnya naik secara perlahan hingga mencapai 17000 bytes pada detik ke-9 hingga akhir simulasi. *Packet drop* pada percobaan ini merupakan *packet drop* dengan nilai tertinggi dari percobaan Tahoe sebelumnya. *Packet drop* Tahoe pada percobaan ke-3 ini disebabkan oleh ukuran *bandwidth*=0.128Mb/30ms di mana yang paling kecil dibandingkan percobaan 2 dan 3.




Gambar 4.20 RTT TCP Tahoe

Gambar 4.20 menunjukkan RTT Tahoe yang cukup tinggi yaitu mencapai 4.5 s pada pertengahan detik ke-2 dan ke-3. Kemudian pada detik ke-6 hingga ke-8 mempunyai nilai RTT yang cukup stabil namun tetap tinggi yaitu mencapai 4 s. RTT pada percobaan ini merupakan RTT dengan nilai tertinggi dari percobaan-percobaan Tahoe sebelumnya yang dipengaruhi oleh tingginya *packet drop* yang diuraikan pada Gambar 4.19.

### 4.3.3 TCP Reno

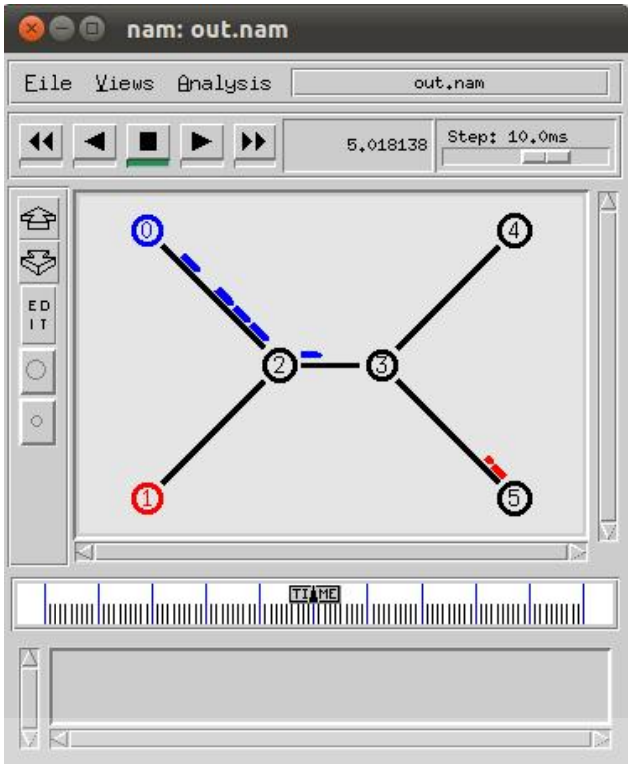
Konfigurasi skrip *.tcl dengannilai *bandwidth* yang bervariasi akan menghasilkan nilai yang bervariasi pula. Skrip *.tcl berjalan pada NS dan akan meghasilkan *trace file* dan *nam file*, kemudian dengan menambahkan beberapa skrip perl untuk mendapatkan *trace file* yang diinginkan seperti *queue*, *packet drop* dan RTT. Di bawah ini merupakan skrip percobaan 1 yaitu reno_topologi.tcl berdasarkan variasi ukuran *bandwidth* dan hasil grafik sesuai dengan percobaan yang telah dilakukan.

Pada skrip reno_topologi.tcl di bawah ini merupakan percobaan 1. Pada skrip di bawah ini juga ditambahkan prosedur untuk menghasilkan *file WinFile* yang digunakan dalam mencari nilai CongWin dan prosedur*qmon* yang menghasilkan qm.out di mana dengan menggunakan perl dihasilkan nilai *queue* dan *packet drop*. Selain itu terdapat out.truntuk mencari nilai RTT melalui pemrograman perl.

```
set ns [new Simulator]
$ns color 1 Blue
$ns color 2 Red
set tracefile1 [open out.tr w]
$ns trace-all $tracefile1
set namfile [open out.nam w]
$ns namtrace-all $namfile
proc finish {} {
global ns tracefile1 namfile
$ns flush-trace
close $tracefile1
close $namfile
exec nam out.nam &
exit 0
}
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
$n0 color Blue
$n1 color Red
$ns duplex-link $n0 $n2 1Mb 50ms DropTail
$ns duplex-link $n1 $n2 1Mb 50ms DropTail
$ns simplex-link $n2 $n3 0.5Mb 30ms RED
$ns simplex-link $n3 $n2 0.5Mb 30ms RED
$ns duplex-link $n3 $n4 1Mb 50ms DropTail
$ns duplex-link $n3 $n5 1Mb 50ms DropTail
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns simplex-link-op $n2 $n3 orient right
$ns simplex-link-op $n3 $n2 orient left
$ns duplex-link-op $n3 $n4 orient right-up
$ns duplex-link-op $n3 $n5 orient right-down
$ns queue-limit $n2 $n3 25
set tcp [new Agent/TCP/Reno]
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n4 $sink
$ns connect $tcp $sink
$tcp set window_ 30
$tcp set packetSize_ 1000
$tcp set fid_ 1
```

```
set ftp [new Application/FTP]
$ftp attach-agent $tcp
set udp [new Agent/UDP]
$ns attach-agent $n1 $udp
set null [new Agent/Null]
$ns attach-agent $n5 $null
$ns connect $udp $null
$udp set fid_ 2
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set packetSize_ 1460
$cbr set rate_ 0.01Mb
$cbr set random_ false
$ns at 0.2 "$cbr start"
$ns at 1.0 "$ftp start"
$ns at 9.6 "$cbr stop"
$ns at 9.8 "$ftp stop"
#############################################################
#prosedur untuk mendapatkan file WinFile
proc plotWindow {tcpSource file} {
global ns
set time 0.1
set now [$ns now]
set cwnd [$tcpSource set cwnd_]
set chan [open $file a]
puts $chan "$now $cwnd"
close $chan
$ns at [expr $now+$time] "plotWindow $tcpSource $file"
}
#############################################################
#prosedur yang menghasilkan file qm.out untuk mencari nilai queue
set qmon [$ns monitor-queue $n2 $n3 [open qm.out w] 0.1];
[$ns link $n2 $n3] queue-sample-timeout;
$ns at 0.1 "plotWindow $tcp WinFile"
$ns at 10.0 finish
$ns run
```

Untuk skrip pada percobaan 2 dan 3 algoritma TCP Reno, hampir sama dengan skrip pada percobaan 1. Bedanya yaitu pada pengaturan parameter *bandwidth.* Pada percobaan 2, *bandwidth* diatur 0.256Mb/30ms dan pada percobaan 3, *bandwidth* diatur 0.128Mb/30ms. Sehingga pada kali ini skrip percobaan 2 dan percobaan 3 tidak ditampilkan.

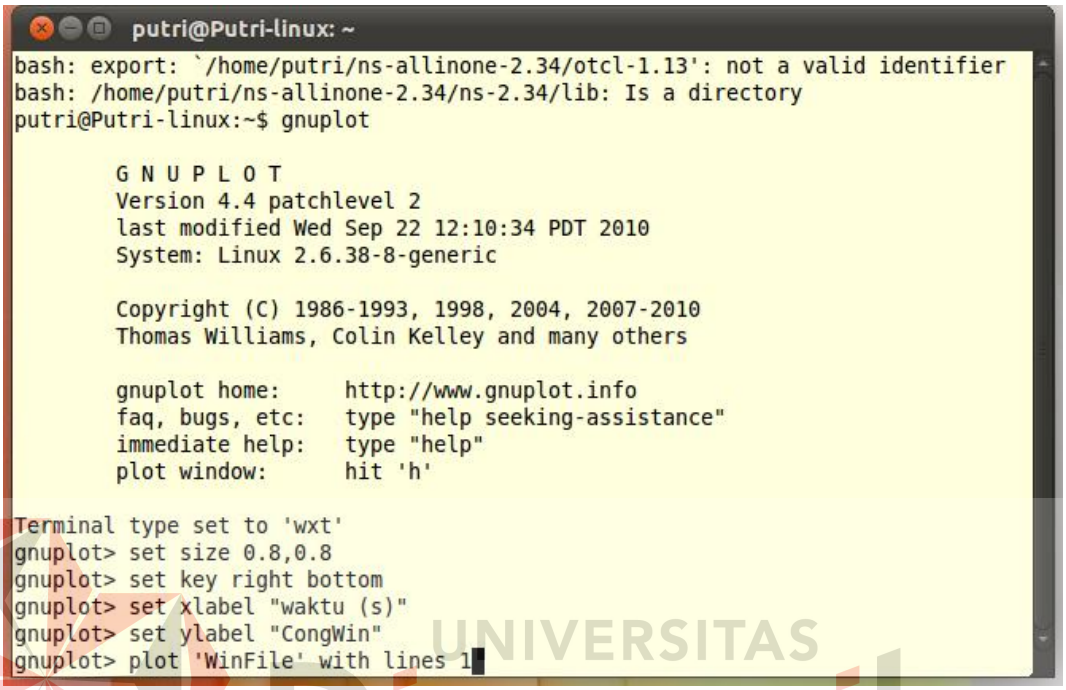



Gambar 4.21 Reno_topologi.nam

Gambar 4.21 merupakan gambar *network animator*(nam) dari konfigurasi reno_topologi yang telah dijalankan. Gambar topologi telah sesuai dengan konfigurasi yang dilakukan, baik besar *bandwidth*, banyaknya node, banyaknya sumber dan tujuan, kemana arah tujuan pengiriman *packet*, waktu yang digunakan dalam proses pengiriman, banyaknya *packet size* yang dikirim serta pengaturan pada ukuran *queue* dan lain sebagainya.

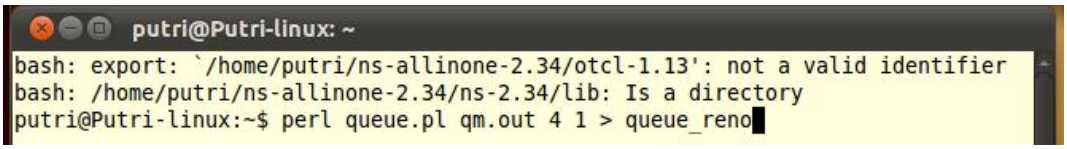
Gambar 4.21 menunjukkan terdapat 6 buah node di mana node 0 merupakan sumber menggunakan aplikasi FTP yang disimbolkan dengan warna biru dengan tujuan node 4 sedangkan node 1 sumber menggunakan generator trafik CBR yang disimbolkan dengan warna merah dengan tujuan node 5. Jika tombol *play* dijalankan maka paket data mulai berjalan dan seluruh kejadian akan tersimpan di *file* out.tr.

Selain menghasilkan out.tr dan out.nam, proses simulasi yang sedang berjalan juga menghasilkan *fileWinFile* yang berisi tentang nilai CongWin, qm.out untuk mencari nilai *queue*, *packet drop* dan RTT.




Gambar 4.22 Plotting WinFile Reno

Gambar 4.22 adalah skrip untuk memplotting *file WinFile* yang berisi nilai CongWin yang diinginkan. Plotting yang dilakukan yaitu menggunakan gnuplot.

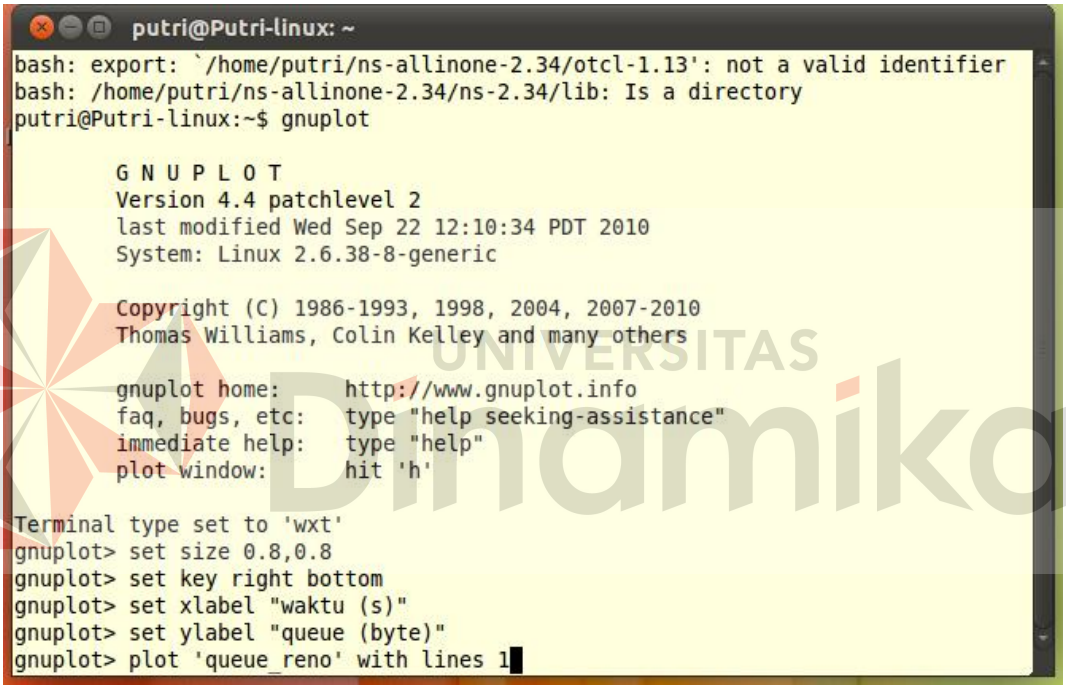



Gambar 4.23 Skrip perl queue_reno

Gambar 4.23 menunjukkan skrip perl untuk mencari nilai queue_reno. *File* qm.out diambil dari prosedur yang ditulis didalam *file* *.tcl. Setelah *file* qm.out didapatkan, maka untuk mencari nilai *queue* harus menggunakan perl dan kemudian memanggilnya seperti yang ditunjukkan pada Gambar 4.23 tersebut. Di

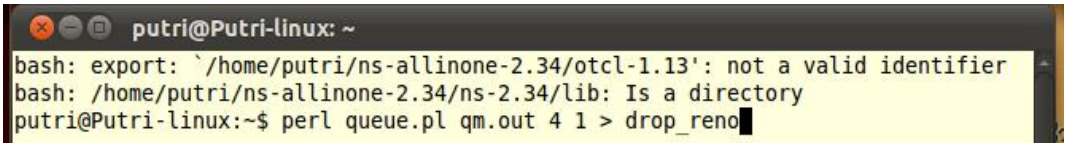
bawah ini merupakan skrip perl untuk mencari nilai *queue*, di mana nilai *queue* berada pada array [3].

```
#skrip perl untuk mencari nilai queue, di mana nilai queue berada
pada array [3]
$infile=$ARGV[0];
open (DATA,"<$infile") || die "Can't open $infile $!";
while (<DATA>)
{
      @x = split(' ');
      print STDOUT "$x[0] $x[3]\n";
}
close DATA;
exit(0);
```




Gambar 4.24 Plotting queue_reno pada gnuplot

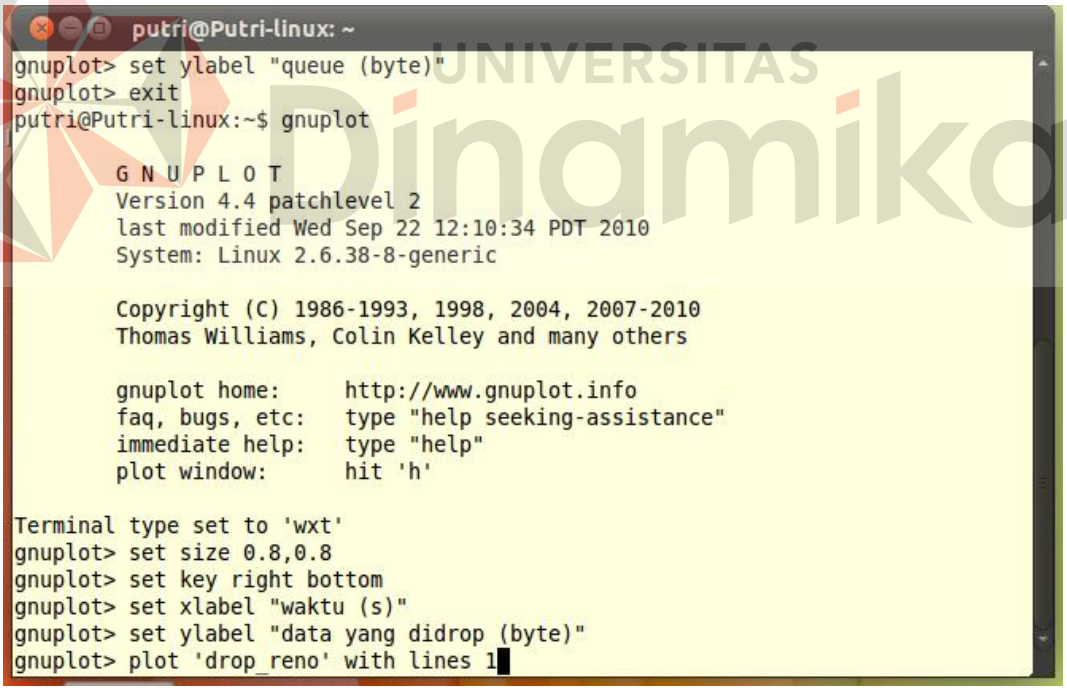
Setelah menjalankan skrip untuk mendapatkan nilai *queue* yang diinginkan dan kemudian menyimpan hasilnya ke dalam queue_reno, maka selanjutnya *file* queue_reno diplotting menggunakan gnuplot seperti pada Gambar 4.24.




Gambar 4.25 Skrip perl drop_reno

Gambar 4.25 adalah skrip perl untuk mendapatkan nilai *packet drop* yang kemudian disimpan ke dalam *file* drop_reno. Karena nilai *packet drop* berada pada array [10], maka array [3] diganti menjadi [10].

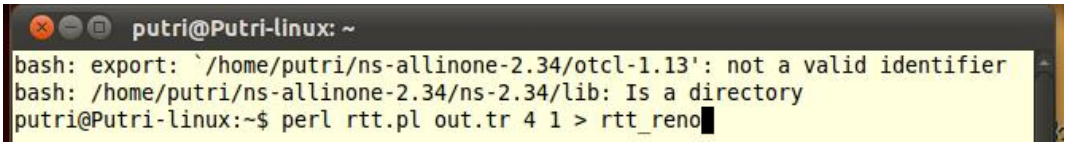
```
#skrip perl untuk mencari nilai packet drop
$infile=$ARGV[0];
open (DATA,"<$infile") || die "Can't open $infile $!";
while (<DATA>)
{
      @x = split(' ');
      print STDOUT "$x[0] $x[10]\n";
}
close DATA;
exit(0);
```




Gambar 4.26 Plotting drop_reno pada gnuplot

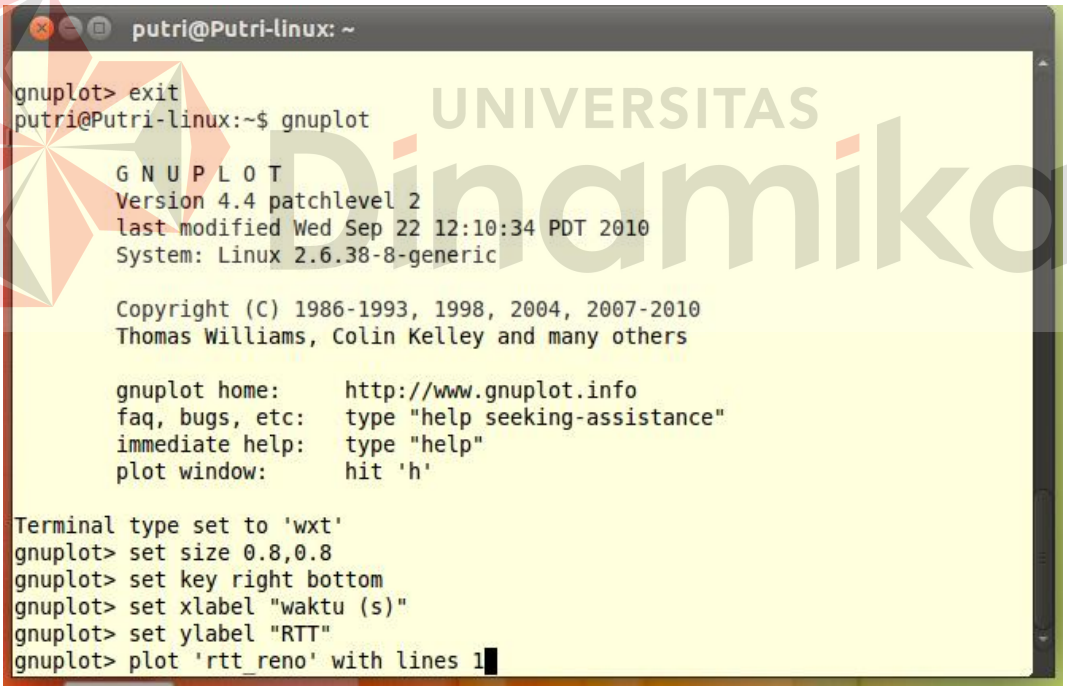
Setelah menjalankan skrip untuk mendapatkan nilai *packet drop* yang diinginkan dan kemudian menyimpan hasilnya ke dalam drop_reno, maka

selanjutnya *file* queue_reno diplotting menggunakan gnuplot seperti pada Gambar 4.26.




Gambar 4.27 Skrip memanggil rtt.pl

Gambar 4.27 merupakan skrip memanggil perl rtt.pl pada Reno yang kemudian hasilnya disimpan pada rtt_reno. Skrip perl rtt.pl menyortir data out.tr untuk mendapatkan hasil RTT yang diinginkan. Skrip perl rtt.pl yang digunakan dilampirkan pada Lampiran 3. Setelah hasil RTT tersimpan dalam rtt_tahoe, maka selanjutnya dibuat grafik dengan skrip sesuai pada Gambar 4.28 berikut ini:



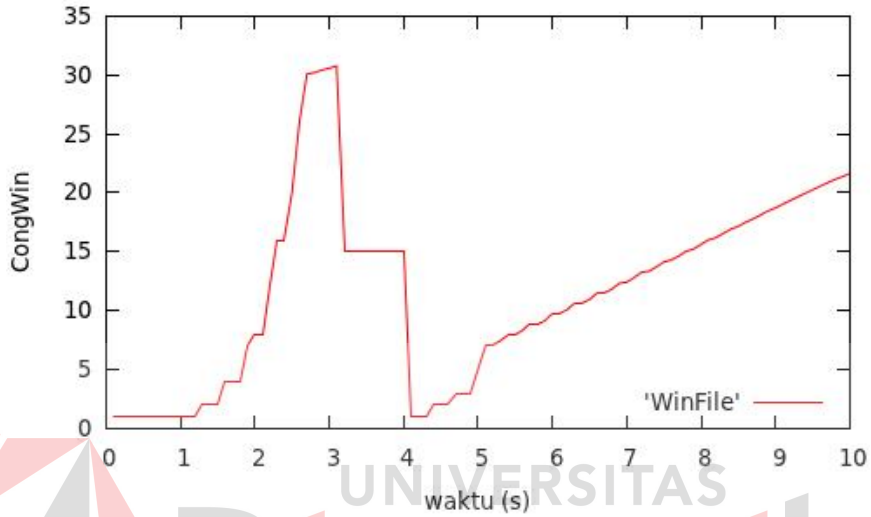

Gambar 4.28 Plotting rtt_reno pada gnuplot

Hasil simulasi pada percobaan 1 berdasarkan variasi *bandwidth* dengan menggunakan TCP Reno ditampilkan pada sub bab berikut ini.

### 4.3.4 Hasil Simulasi TCP Reno

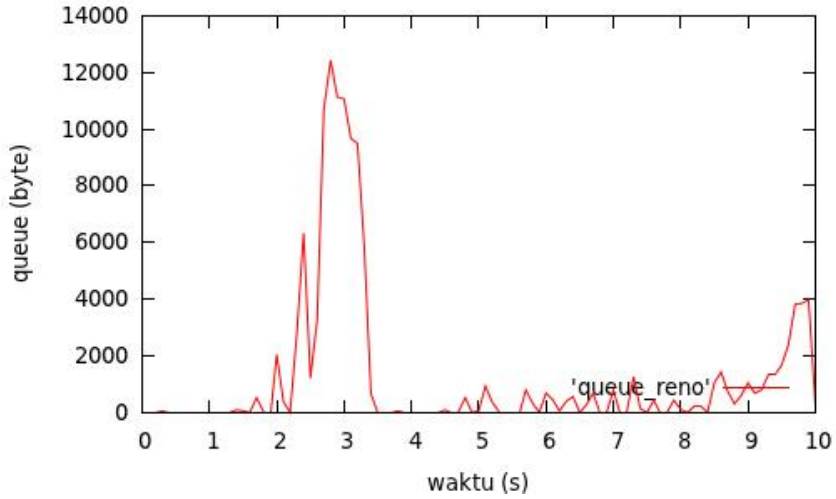
Di bawah ini merupakan grafik yang dihasilkan dari reno_topologi.tcl dan telah diplotting keseluruhannya yang disajikan berdasarkan skenario percobaan.

**a. Percobaan 1**



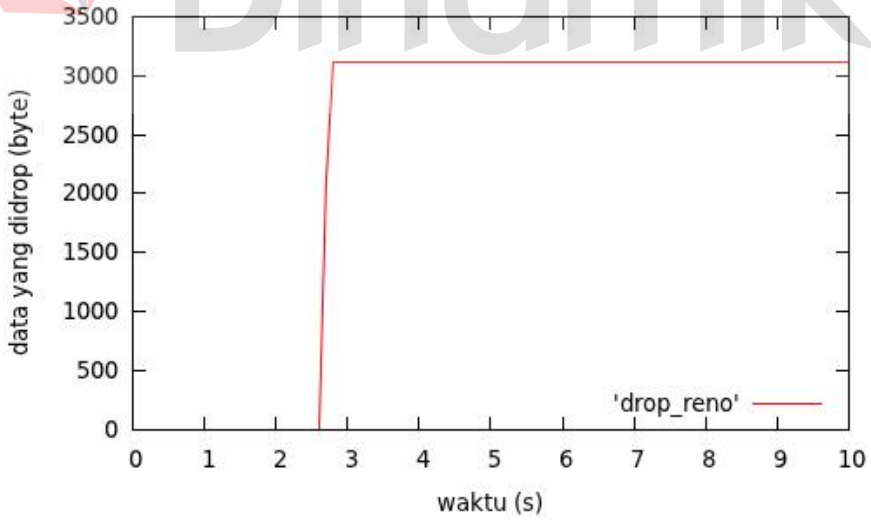

Gambar 4.29 CongWin TCP Reno

Gambar 4.29 merupakan grafik CongWin yang dihasilkan dari percobaan 1 dengan *bandwidth*=0.5Mb/30ms. Dari gambar ditunjukkan bahwa Reno mencapai puncak CongWin antara detik ke-2 dan ke-3 dengan nilai mencapai lebih dari 30 MSS. Kemudian saat terjadi kongesti, Reno menurunkan nilai CongWin menjadi setengah dari nilai *threshold* menjadi nilai 15 MSS pada detik ke-3. Kemudian sempat mengalami kongesti dan turun hingga 1 MSS kemudian naik secara eksponensial setelah itu naik secara linier hingga simulasi berakhir.



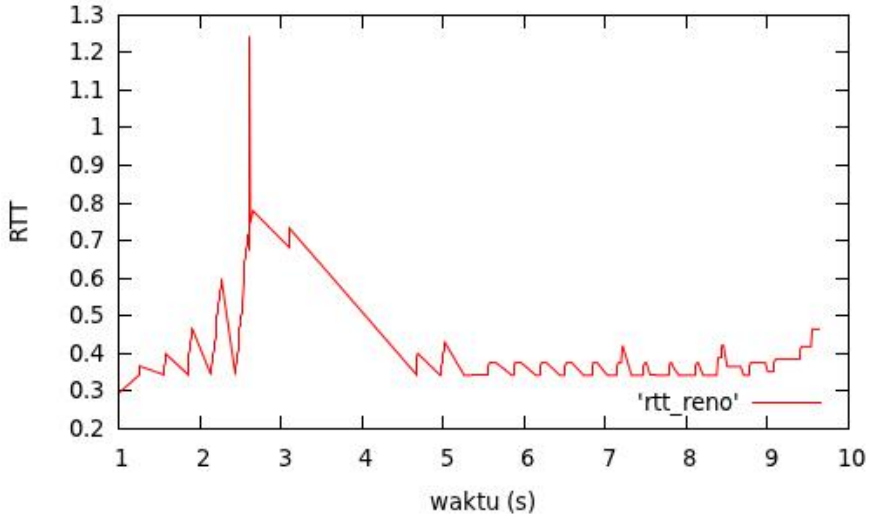

Gambar 4.30 *Queue* TCP Reno

Gambar 4.30 merupakan grafik *queue* yang menunjukkan penggunaan *queue* pada Reno yang cukup rendah hingga akhir simulasi, namun sempat tinggi pada detik ke-3 yaitu mencapai 12000 bytes akibat CongWin yang tinggi pada awal simulasi dan bertahan sekitar 1.5 s pada detik ke-2 sampai pertengahan detik ke-3 dan 4.




Gambar 4.31 *Packet Drop* kumulatif TCP Reno

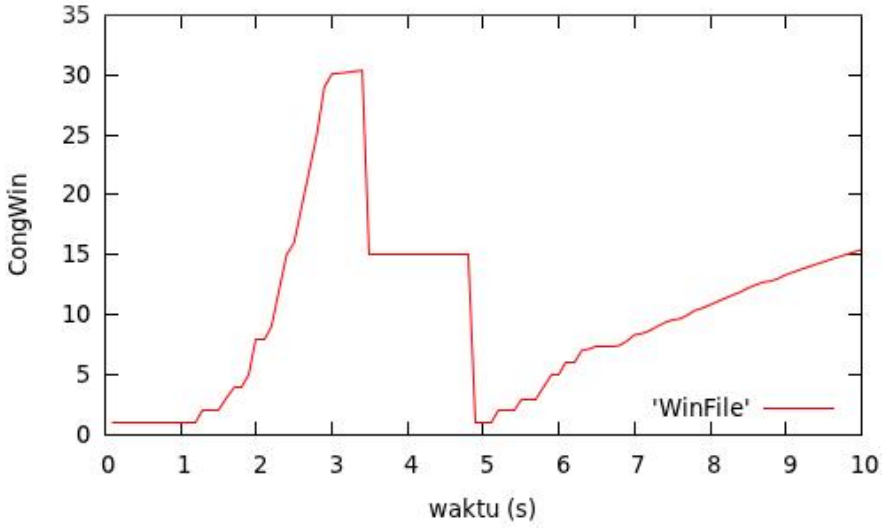
Gambar 4.31 menunjukkan grafik kumulatif *packet drop* yang naik signifikan tinggi akibat tingginya antrian hingga mencapai lebih dari 3000 byte pada pertengahan detik ke-2 dan ke-3 dan tetap hingga akhir simulasi.




Gambar 4.32 RTT TCP Reno

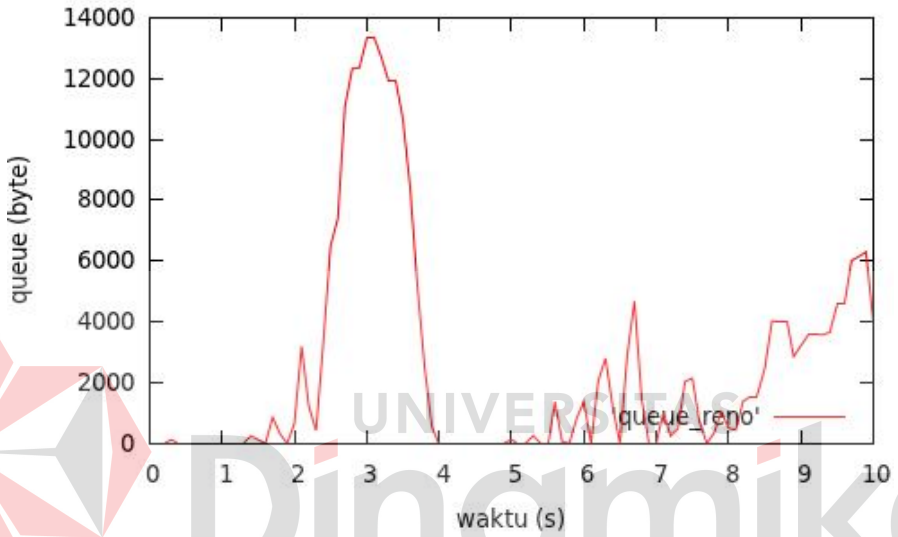
Gambar 4.32 menunjukkan grafik RTT TCP Reno. Pada pertengahan detik ke-2 dan ke-3 nilai RTT mengalami kenaikan yaitu mencapai 1.2 s yang disebabkan oleh banyaknya *packet drop* yang terjadi. Dan pada detik ke-4 hingga akhir simulasi RTT mengalami kestabilan dengan nilai yang rendah yaitu kurang dari 0.4 s.

**b. Percobaan 2**



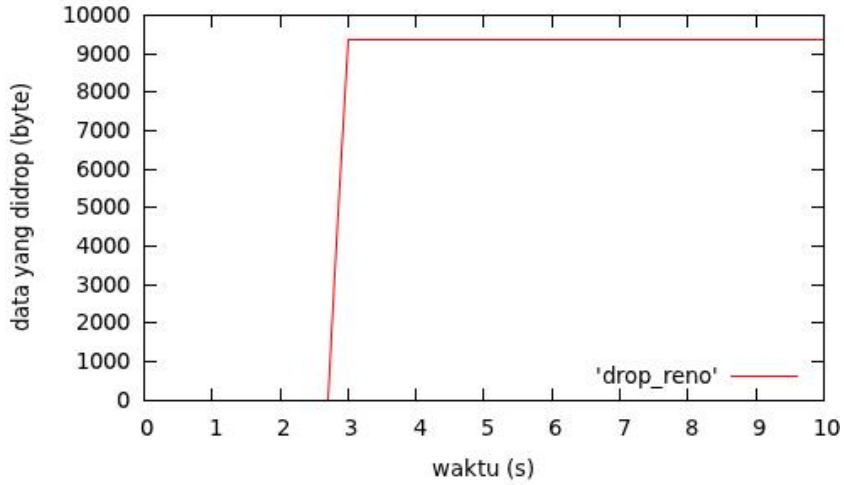

Gambar 4.33 CongWin TCP Reno

Pada Gambar 4.33 merupakan grafik CongWin pada TCP Reno. Grafik di atas menunjukkan Reno mencapai CongWin tertinggi dengan nilai lebih dari 30 MSS pada detik ke-3. Kemudian saat terjadi kongesti, Reno menurunkan nilai CongWin setengah dari nilai *threshold* yaitu 15 MSS. Kemudian nilai 15 MSS bertahan hingga detik ke-5, kemudian turun ke nilai 1 MSS. Selanjutnya naik secara eksponensial hingga di akhir simulasi mencapai nilai 15 MSS.



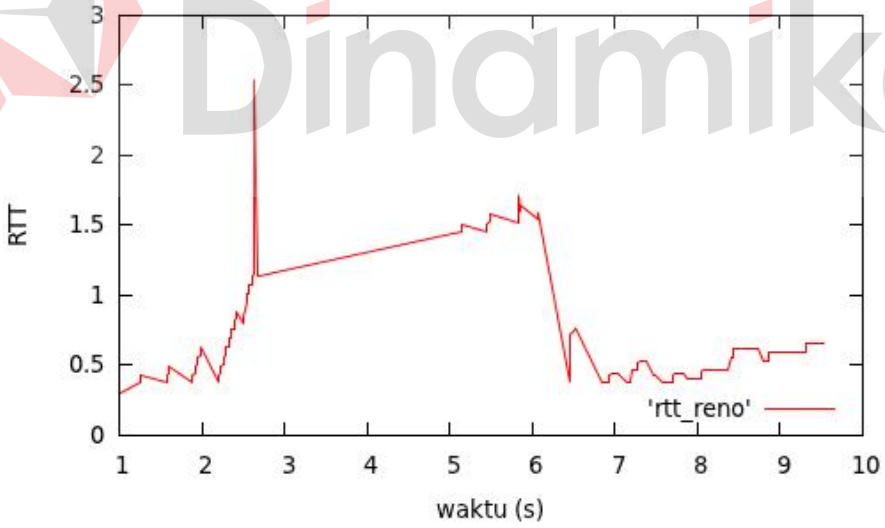

Gambar 4.34 *Queue* TCP Reno

Gambar 4.34 merupakan grafik penggunaan *queue* TCP Reno. Grafik di atas menunjukkan penggunaan *queue* pada detik ke-3 yang mencapai 14000 bytes yang bertahan hingga 2 detik antara detik ke-2 dan 4. Ini artinya banyaknya antrian yang terjadi pada interval waktu detik ke-2 hingga ke-4. Kemudian mulai detik ke-6 hingga akhir simulasi penggunaaan terbesar hanya mencapai 6000 bytes.




Gambar 4.35 *Packet Drop* kumulatif TCP Reno

Pada Gambar 4.35 adalah grafik *packet drop* kumulatif TCP Reno di mana nilai *packet drop* meningkat pada pertengahan detik ke-2 dan ke-3 yaitu sebesar 9000 bytes, dan tetap hingga akhir simulasi. Peningkatan ini diiringi oleh tinggi antrian paket menuju jalur *bottleneck*.

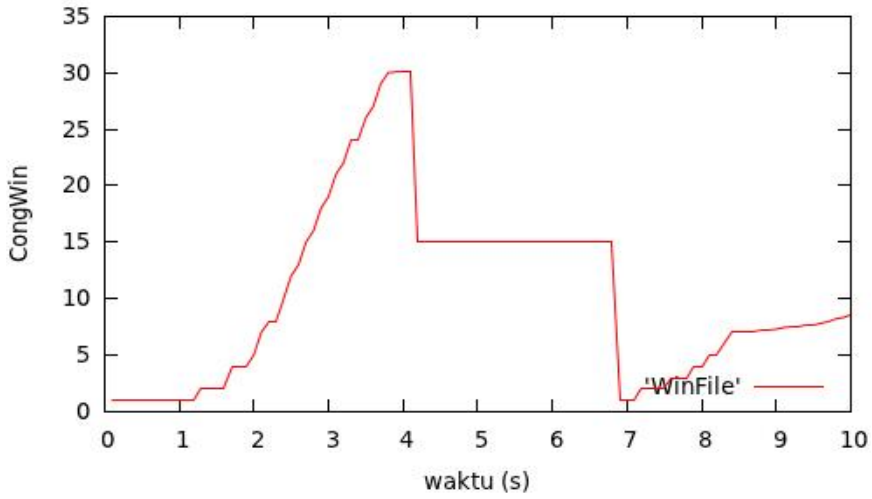



Gambar 4.36 RTT TCP Reno

Gambar 4.36 menunjukkan grafik RTT yang mencapai nilai tertinggi pada pertengahan detik ke-2 dan ke-3 yaitu mencapai 2.5 s. Hal ini disebabkan oleh tingginya *packet drop* yang terjadi. RTT mulai stabil dengan nilai 1.5 s pada

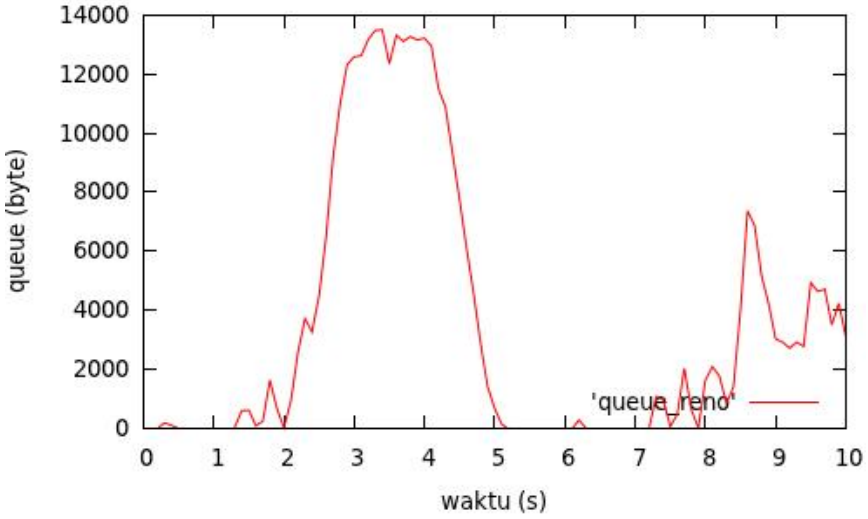
detik ke-6. Sedangkan pada akhir simulasi mengalami penurunan dan terus stabil di mana nilai RTT kurang dari 1 s.

**c. Percobaan 3**



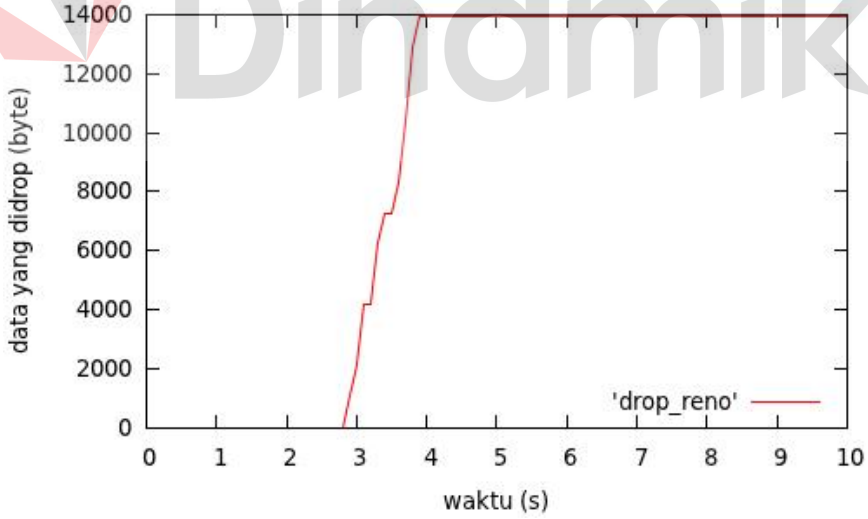

Gambar 4.37 CongWin TCP Reno

Pada Gambar 4.37 menunjukkan grafik CongWin TCP Reno di mana awalnya naik secara eksponensial hingga mencapai nilai 30 MSS mulai dari pertengahan detik ke-3 dan ke-4 sampai mengalami kongesti pada detik ke-4. Kemudian Reno menurunkan nilai menjadi setengah yaitu 15 MSS. Selanjutnya tetap berada pada nilai 15 MSS hingga detik ke-7, kemudian turun menjadi 1 MSS dan naik secara eksponensial hingga mencapai nilai sekitar 8 MSS sampai akhir simulasi.



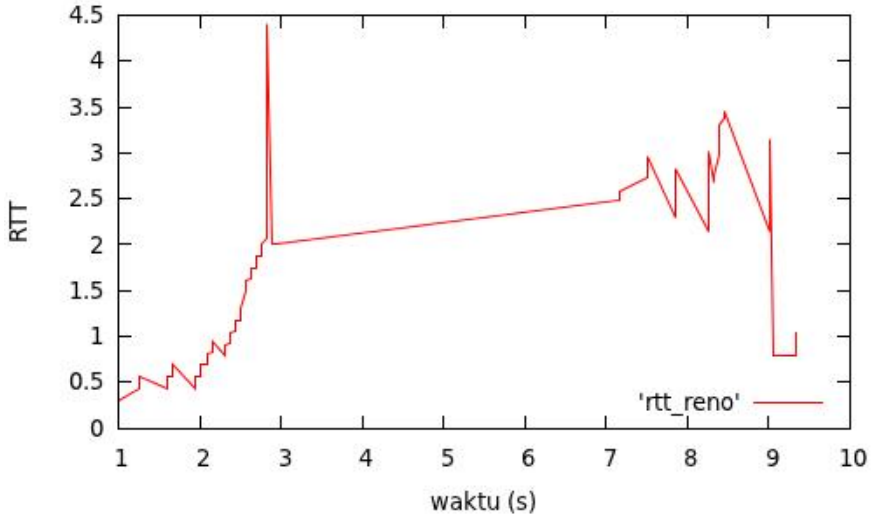

Gambar 4.38 *Queue* TCP Reno

Gambar 4.38 merupakan grafik penggunaan *queue* TCP Reno di mana mencapai penggunaan terbesar pada detik ke-2 hingga ke-5 dengan nilai tertinggi mencapai hampir 14000 bytes. Dan di akhir simulasi penggunaan yang dimulai pada detik ke-8 hingga akhir simulasi dengan nilai tertinggi mencapai 8000 bytes.




Gambar 4.39 *Packet Drop* kumulatif TCP Reno

Pada Gambar 4.39 merupakan grafik *packet drop* kumulatif TCP Reno dengan peningkatan *packet drop* yang tinggi di detik ke-4 hingga mencapai 14000 bytes hingga akhir simulasi.




Gambar 4.40 RTT TCP Reno

Gambar 4.40 merupakan grafik RTT TCP Reno yang tinggi hingga mencapai 4.5 s. Sempat mengalami penurunan pada detik ke-3 hingga akhir simulasi dengan nilai RTT rata-rata mencapai 3 s. Nilai RTT pada percobaan ini merupakan RTT tertinggi dibandingkan percobaan Reno lainnya. Hal ini dipengaruhi oleh tingginya *packet drop* dan juga rendahnya *bandwidth* yakni hanya 0.128Mb/30ms.

### 4.3.5 TCP SACK

Konfigurasi skrip *.tcl dengannilai *bandwidth* yang bervariasi akan menghasilkan nilai yang bervariasi pula. Skrip *.tcl berjalan pada NS dan akan meghasilkan *trace file* dan *nam file*, kemudian dengan menambahkan beberapa skrip perl untuk mendapatkan *trace file* yang diinginkan seperti *queue, packet drop* dan RTT. Di bawah ini merupakan skrip percobaan 1 yaitu sack_topologi.tcl dan hasil sesuai dengan percobaan yang telah dilakukan.

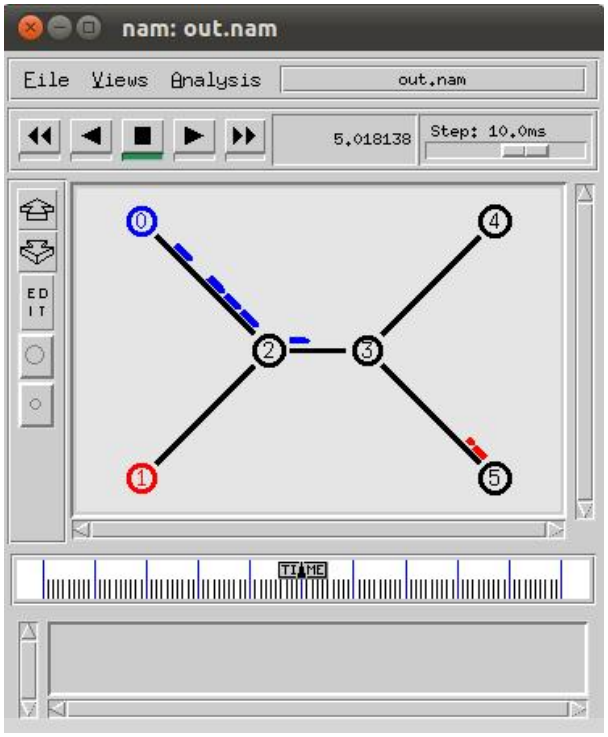
Pada skrip sack_topologi.tcl di bawah ini merupakan percobaan 1 dengan ukuran *bandwidth*= 0.5Mb/30ms. Skrip ini juga ditambahkan prosedur untuk menghasilkan *file WinFile* yang digunakan dalam mencari nilai CongWin dan

prosedur *qmon* yang menghasilkan qm.out di mana dengan menggunakan perl dihasilkan nilai *queue* dan *packet drop*. Selain itu terdapat out.truntuk mencari nilai RTT melalui pemrograman perl..

```
set ns [new Simulator]
$ns color 1 Blue
$ns color 2 Red
set tracefile1 [open out.tr w]
$ns trace-all $tracefile1
set namfile [open out.nam w]
$ns namtrace-all $namfile
proc finish {} {
global ns tracefile1 namfile
$ns flush-trace
close $tracefile1
close $namfile
exec nam out.nam &
exit 0
}
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
$n0 color Blue
$n1 color Red
$ns duplex-link $n0 $n2 1Mb 50ms DropTail
$ns duplex-link $n1 $n2 1Mb 50ms DropTail
$ns simplex-link $n2 $n3 0.5Mb 30ms RED
$ns simplex-link $n3 $n2 0.5Mb 30ms RED
$ns duplex-link $n3 $n4 1Mb 50ms DropTail
$ns duplex-link $n3 $n5 1Mb 50ms DropTail
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns simplex-link-op $n2 $n3 orient right
$ns simplex-link-op $n3 $n2 orient left
$ns duplex-link-op $n3 $n4 orient right-up
$ns duplex-link-op $n3 $n5 orient right-down
$ns queue-limit $n2 $n3 25
set tcp [new Agent/TCP/Sack1]
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n4 $sink
$ns connect $tcp $sink
$tcp set window_ 30
$tcp set packetSize_ 1000
$tcp set fid_ 1
set ftp [new Application/FTP]
$ftp attach-agent $tcp
set udp [new Agent/UDP]
$ns attach-agent $n1 $udp
set null [new Agent/Null]
$ns attach-agent $n5 $null
$ns connect $udp $null
$udp set fid_ 2
```

```
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set packetSize_ 1460
$cbr set rate_ 0.01Mb
$cbr set random_ false
$ns at 0.2 "$cbr start"
$ns at 1.0 "$ftp start"
$ns at 9.6 "$cbr stop"
$ns at 9.8 "$ftp stop"
#############################################################
#prosedur untuk menghasilkan file WinFile
proc plotWindow {tcpSource file} {
global ns
set time 0.1
set now [$ns now]
set cwnd [$tcpSource set cwnd_]
set chan [open $file a]
puts $chan "$now $cwnd"
close $chan
$ns at [expr $now+$time] "plotWindow $tcpSource $file"
}
#############################################################
#prosedur yang menghasilkan file qm.out untuk mencari nilai queue
set qmon [$ns monitor-queue $n2 $n3 [open qm.out w] 0.1];
[$ns link $n2 $n3] queue-sample-timeout;
$ns at 0.1 "plotWindow $tcp WinFile"
$ns at 10.0 finish
$ns run
```

Untuk skrip pada percobaan 2 dan 3 algoritma TCP SACK, hampir sama dengan skrip pada percobaan 1. Bedanya yaitu pada pengaturan parameter *bandwidth*. Pada percobaan 2, *bandwidth* diatur 0.256Mb/30ms dan pada percobaan 3, *bandwidth* diatur 0.128Mb/30ms. Sehingga pada kali ini skrip percobaan 2 dan percobaan 3 tidak ditampilkan.

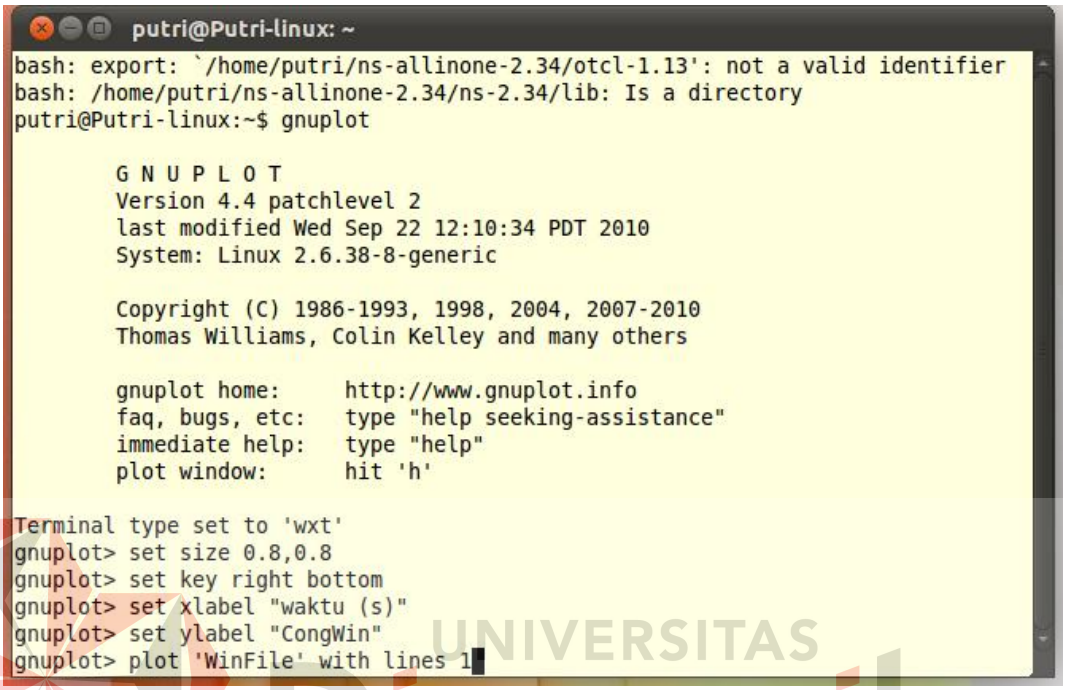



Gambar 4.41 SACK_topologi.nam

Gambar 4.41 merupakan gambar *network animator*(nam) dari konfigurasi sack_topologi yang telah dijalankan. Gambar topologi telah sesuai dengan konfigurasi yang dilakukan, baik besar *bandwidth*, banyaknya node, banyaknya sumber dan tujuan, kemana arah tujuan pengiriman *packet*, waktu yang digunakan dalam proses pengiriman, banyaknya *packet size* yang dikirim serta pengaturan pada ukuran *queue* dan lain sebagainya.

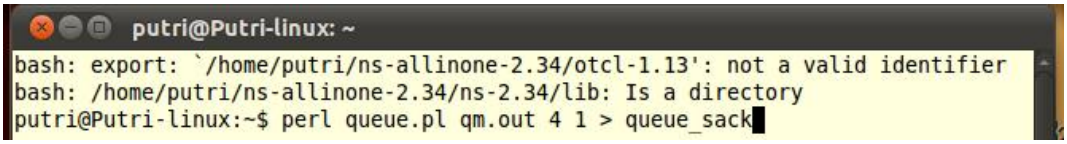
Gambar 4.41 menunjukkan terdapat 6 buah node di mana node 0 merupakan sumber menggunakan aplikasi FTP yang diwakili warna biru dengan tujuan node 4 sedangkan node 1 sumber menggunakan generator trafik CBR yang diwakili warna merah dengan tujuan node 5. Jika tombol *play* dijalankan maka paket data mulai berjalan dan seluruh kejadian akan tersimpan di *file* out.tr.

Selain menghasilkan out.tr, out.nam, proses simulasi yang sedang berjalan juga menghasilkan *fileWinFile* yang berisi tentang nilai CongWin, qm.out untuk mencari nilai *queue*, *packet drop* dan RTT.




Gambar 4.42 Plotting WinFile SACK

Gambar 4.42 adalah skrip untuk memplotting *file WinFile* yang berisi nilai CongWin yang diinginkan. Plotting yang dilakukan yaitu menggunakan gnuplot.

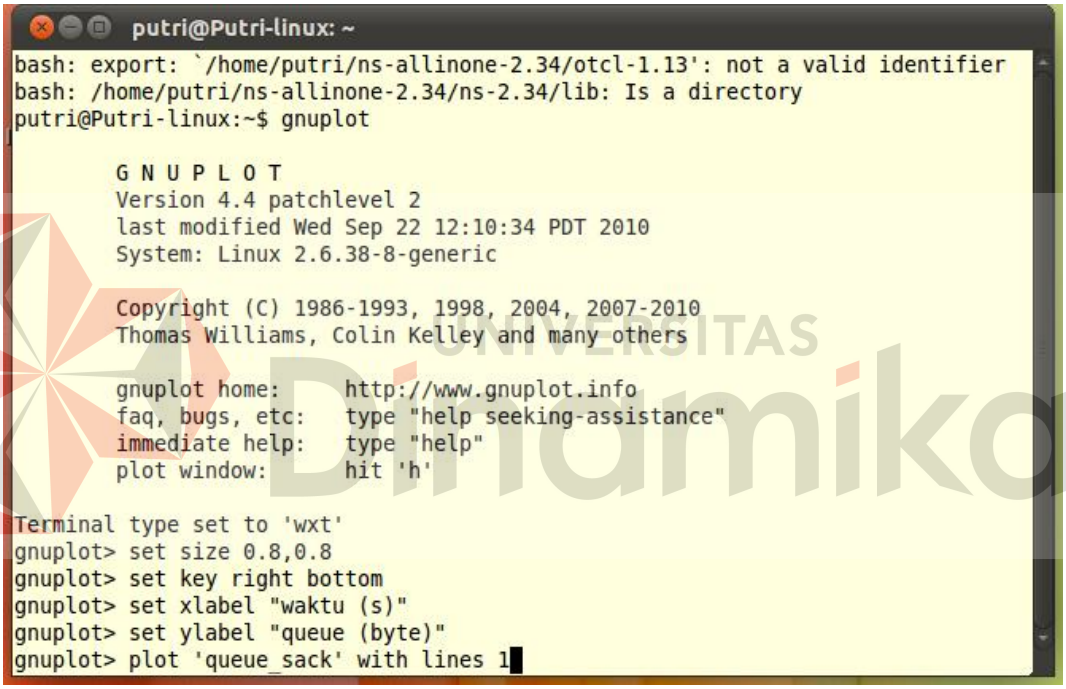



Gambar 4.43 Skrip perl queue_sack

Gambar 4.43 menunjukkan skrip perl untuk mencari nilai queue_sack. *File* qm.out diambil dari prosedur yang ditulis didalam *file* tcl. Setelah *file* qm.out didapatkan, maka untuk mencari nilai *queue* harus menggunakan perl dan kemudian memanggilnya seperti yang ditunjukkan pada Gambar 4.43 tersebut. Di

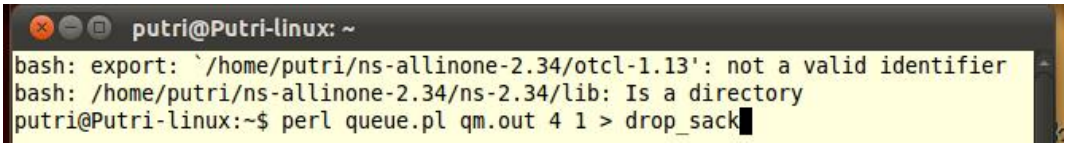
bawah ini merupakan skrip perl untuk mencari nilai *queue*, di mana nilai *queue* berada pada array [3].

```
#skrip perl untuk mencari nilai queue, di mana nilai queue berada
pada array [3]
$infile=$ARGV[0];
open (DATA,"<$infile") || die "Can't open $infile $!";
while (<DATA>)
{
      @x = split(' ');
      print STDOUT "$x[0] $x[3]\n";
}
close DATA;
exit(0);
```




Gambar 4.44 Plotting queue_sack pada gnuplot

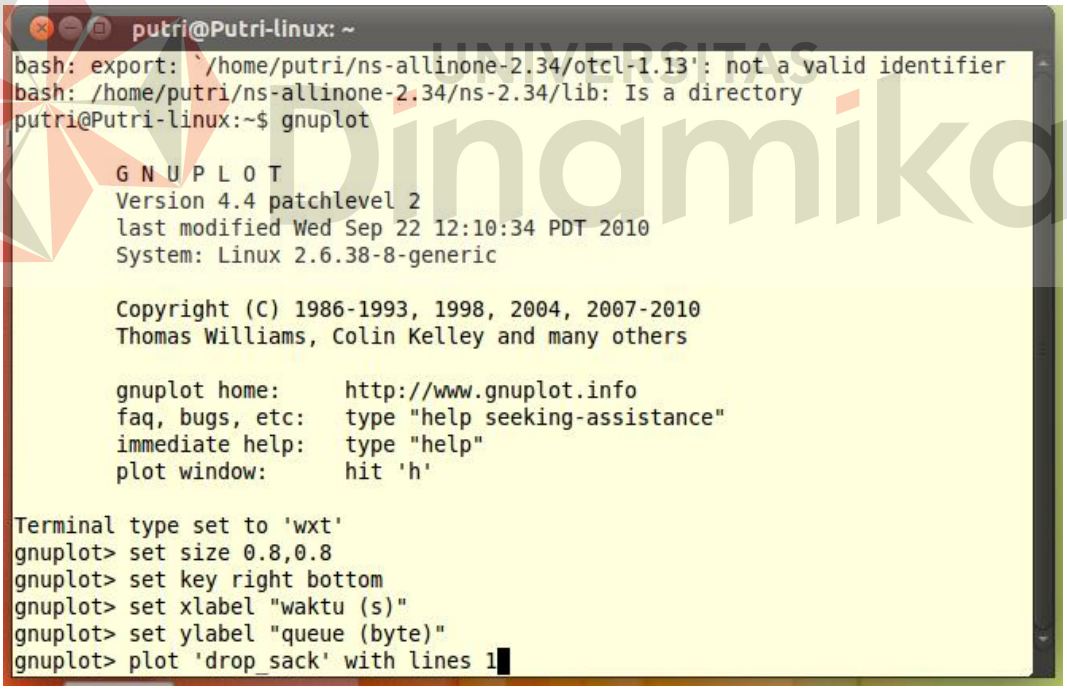
Setelah menjalankan skrip untuk mendapatkan nilai *queue* yang diinginkan dan kemudian menyimpan hasilnya ke dalam queue_reno, maka selanjutnya *file* queue_reno diplotting menggunakan gnuplot seperti pada Gambar 4.44.




Gambar 4.45 Skrip perl drop_sack

Gambar 4.45 adalah skrip perl untuk mendapatkan nilai *packet drop* yang kemudian disimpan ke dalam *file* drop_sack. Karena nilai *packet drop* berada pada array [10], maka array [3] diganti menjadi [10].

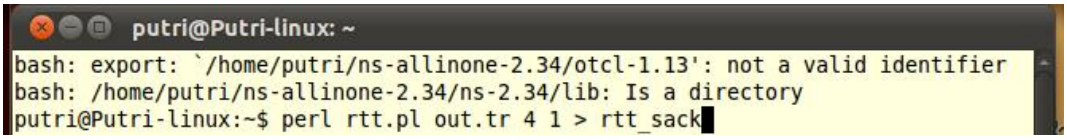
```
#skrip perl untuk mencari nilai packet drop
$infile=$ARGV[0];
open (DATA,"<$infile") || die "Can't open $infile $!";
while (<DATA>)
{
      @x = split(' ');
      print STDOUT "$x[0] $x[10]\n";
}
close DATA;
exit(0);
```




Gambar 4.46 Plotting drop_sack pada gnuplot

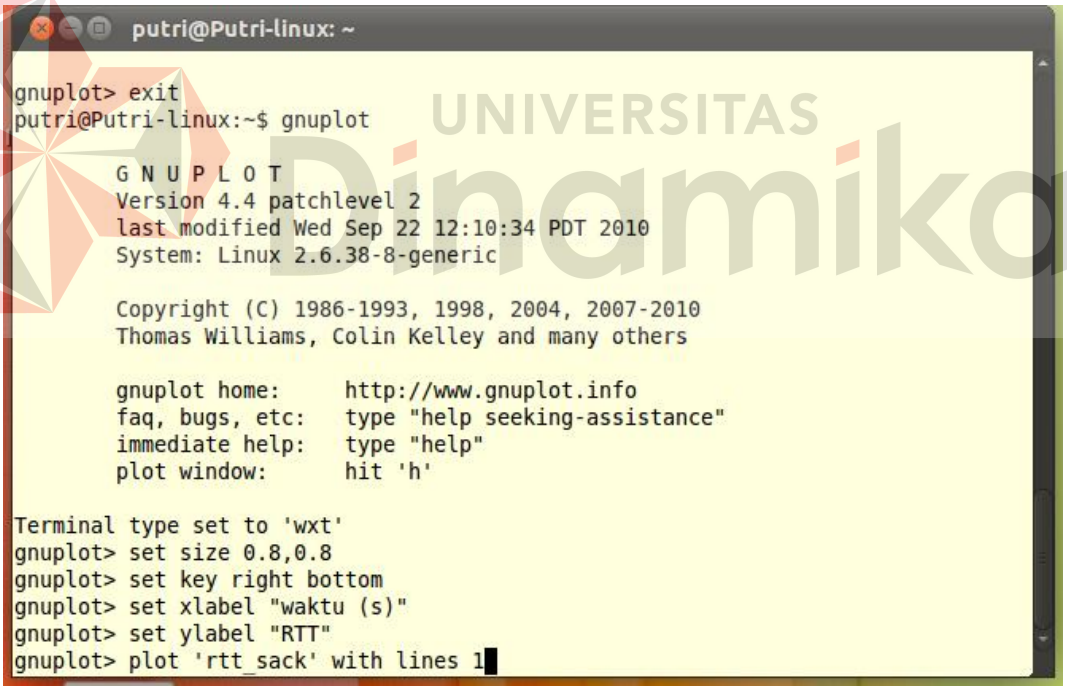
Setelah menjalankan skrip untuk mendapatkan nilai *packet drop* yang diinginkan dan kemudian menyimpan hasilnya ke dalam drop_sack, maka

selanjutnya *file* drop_sack diplotting menggunakan gnuplot seperti pada Gambar 4.46.




Gambar 4.47 Skrip memanggil rtt.pl

Gambar 4.47 merupakan skrip memanggil perl rtt.pl pada SACK yang kemudian hasilnya disimpan pada rtt_reno. Skrip perl rtt.pl menyortir data out.tr untuk mendapatkan hasil RTT yang diinginkan. Skrip perl rtt.pl yang digunakan dilampirkan pada Lampiran 3. Setelah hasil RTT tersimpan dalam rtt_sack, maka selanjutnya dibuat grafik dengan skrip sesuai pada Gambar 4.48 berikut ini:



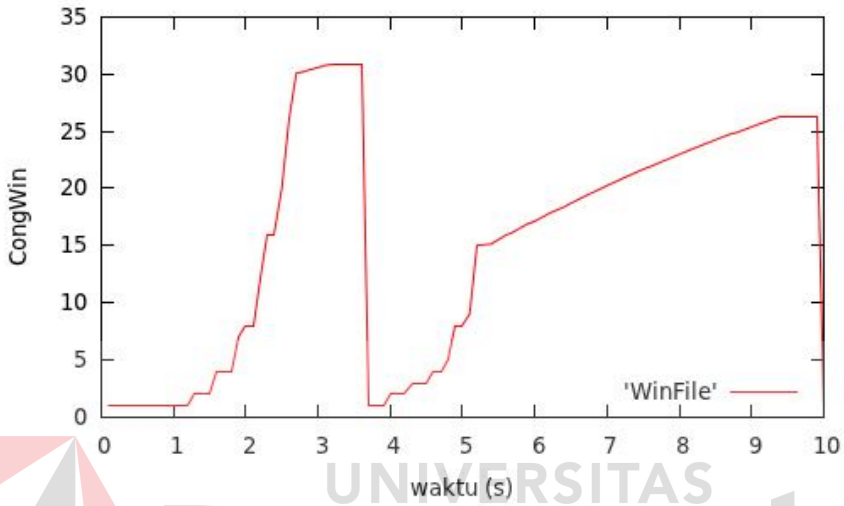

Gambar 4.48 Plotting rtt_sack pada gnuplot

Hasil simulasi pada percobaan 1 berdasarkan variasi *bandwidth* dengan menggunakan TCP SACK ditampilkan pada sub bab berikut ini.

### 4.3.4 Hasil Simulasi TCP SACK

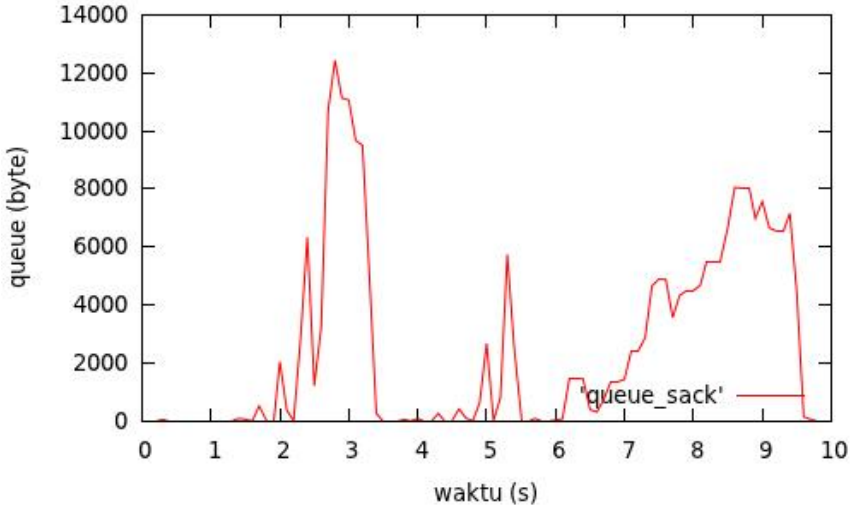
Di bawah ini merupakan grafik yang dihasilkan dari sack_topologi.tcl dan telah diplotting keseluruhannya yang disajikan berdasarkan skenario percobaan.

**a. Percobaan 1**



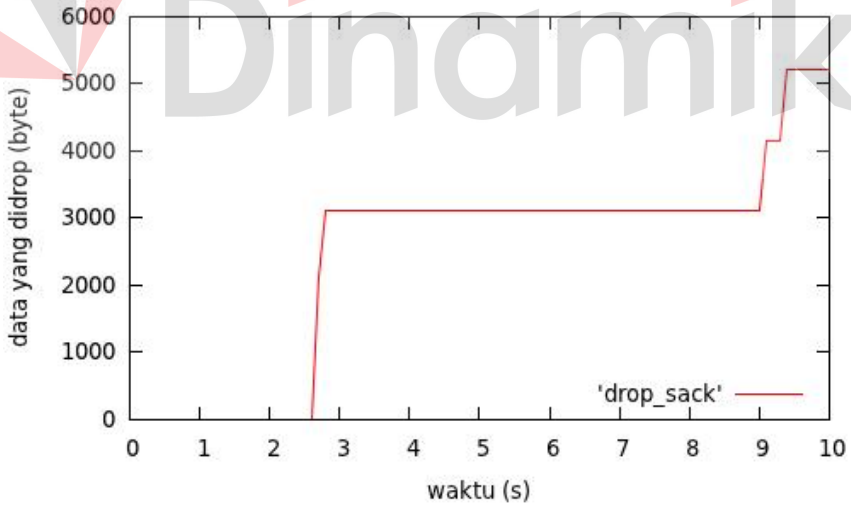

Gambar 4.49 CongWin TCP SACK

Gambar 4.49 merupakan grafik CongWin yang dihasilkan dari percobaan 1 berdasarkan variasi ukuran *bandwidth*=0.5 Mbps/30 ms. Dari Gambar 4.49 ditunjukkan bahwa SACK mencapai puncak CongWin dengan nilai di atas 30 MSS pada detik ke-3 dan bertahan mulai detik ke-2 hingga mendekati detik ke-4. Saat terjadi kongesti, SACK menurunkan nilai CongWin menjadi 1 MSS. Kemudian naik secara eksponensial dan setelah mencapai nilai *threshold* pada detik ke-5, SACK naik secara linier hingga akhir simulasi.




Gambar 4.50 *Queue* TCP SACK

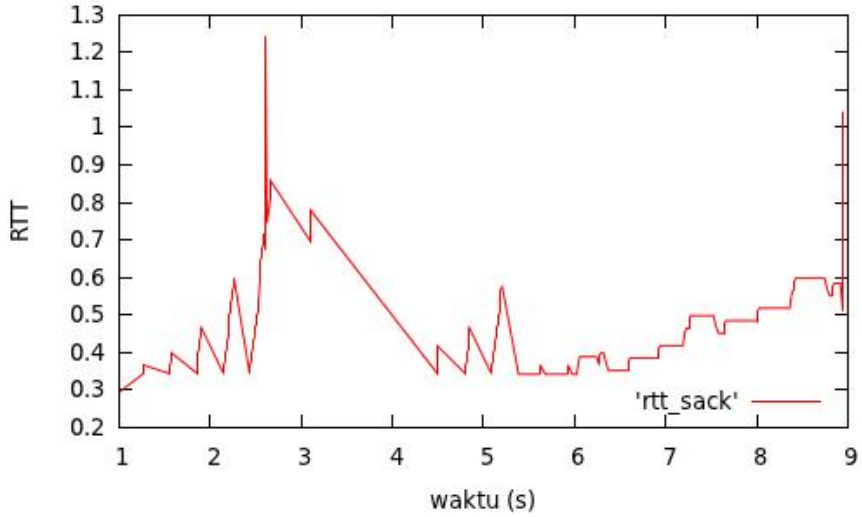
Gambar 4.50 merupakan grafik *queue* yang menunjukkan penggunaan *queue* pada SACK yang paling tinggi pada detik ke-3 yaitu mencapai 12000 bytes. Kemudian sempat tinggi di detik ke-5 yaitu sebesar 6000 bytes. Dan di akhir simulasi kembali tinggi pada detik ke-9 yaitu 8000 bytes.




Gambar 4.51 *Packet Drop* Kumulatif TCP SACK

Gambar 4.51 menunjukkan grafik *packet drop* yang terus naik hingga melewati nilai 3000 bytes dalam pada pertengahan detik ke-2 dan ke-3. Kemudian naik terus sampai di akhir simulasi hingga mencapai 5100 bytes pada akhir

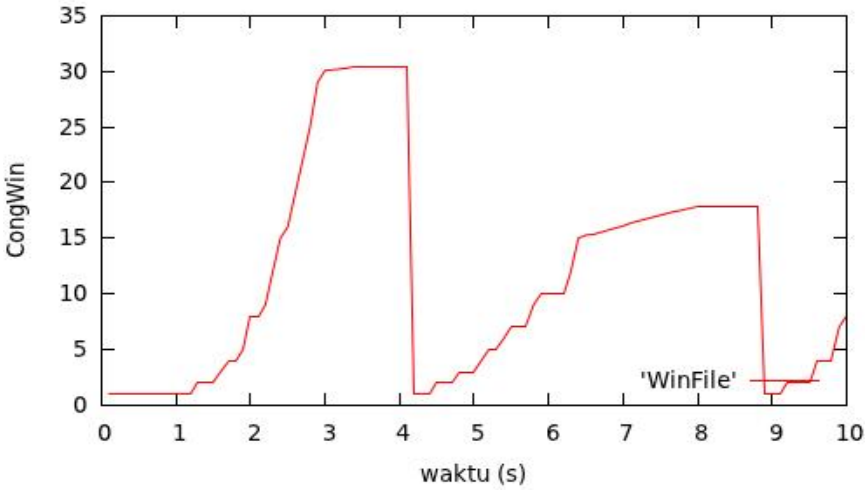
simulasi. Tingginya nilai *packet drop* dipengaruhi oleh tingginya paket yang mengantri menuju jalur *bottleneck.*




Gambar 4.52 RTT TCP SACK

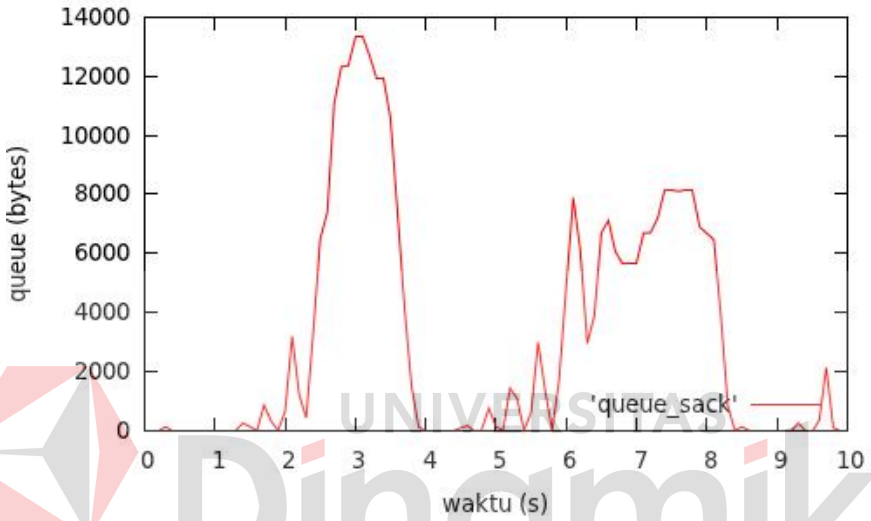
Gambar 4.52 menunjukkan grafik RTT yang dihasilkan mengalami kenaikan pada pertengahan detik ke-2 dan ke-3 yaitu 1.2 s. Kenaikan ini disebabkan oleh tingginya *packet drop* yang terjadi pada TCP SACK. Namun sempat mengalami penurunan menuju detik ke-5 dan kemudian stabil hingga akhir simualasi dengan nilai kurang dari 0.6 s.

**b. Percobaan 2**



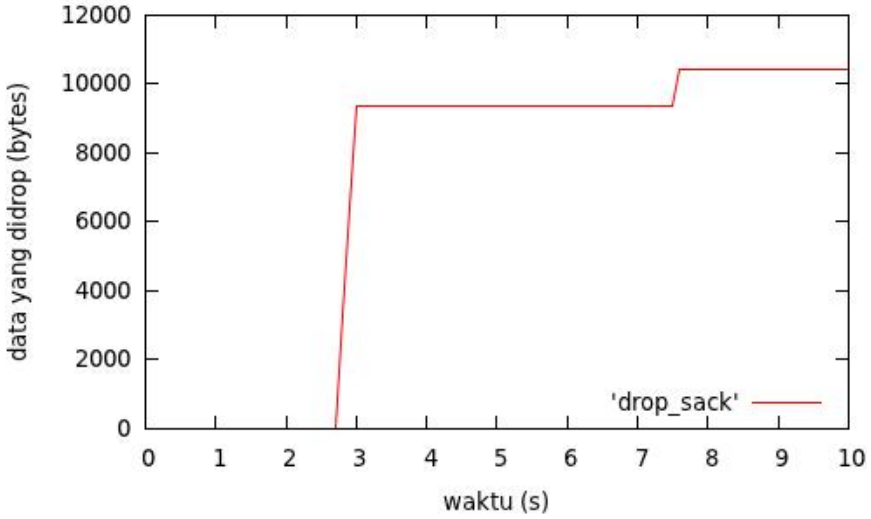

Gambar 4.53 CongWin TCP SACK

Pada Gambar 4.53 merupakan grafik CongWin TCP SACK di mana pada detik ke-3 telah mencapai puncak CongWin dengan nilai 30 MSS yang bertahan mulai detik ke-2 hingga detik ke-4. Kemudian terjadi kongesti dan turun hingga 1 MSS. Setelah mengalami kongesti, SACK mampu naik secara eksponensial hingga mencapai nilai 15 MSS mulai dari detik ke-6 sampai detik ke-9, setelah itu turun dan kembali naik secara eksponensial hingga akhir simulasi.



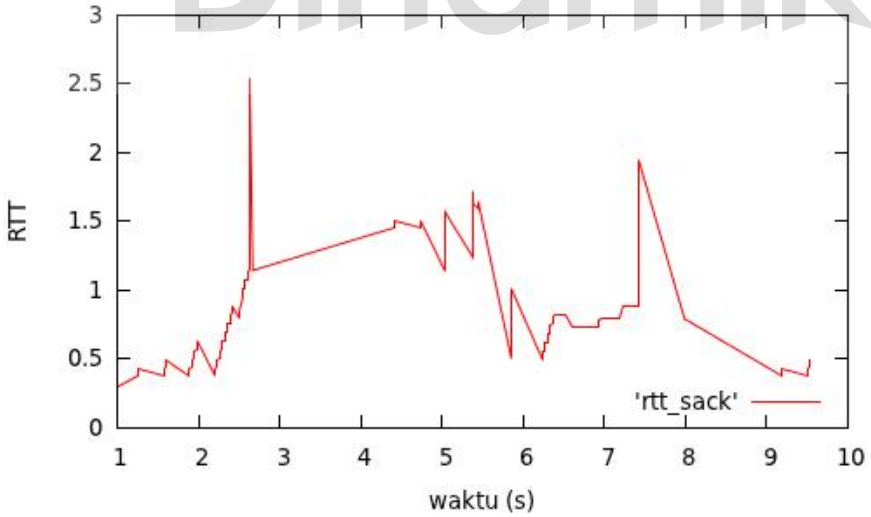

Gambar 4.54 *Queue* TCP SACK

Gambar 4.54 menunjukkan penggunaan *queue* SACK yang mengalami kenikan pada detik ke-3 dengan nilai yang hampir mencapai 14000 bytes. Pada awal simulasi menunjukkan tingginya antrian paket menuju jalur *bottleneck*. Kemudian mengalami penurunan namun masih relatif besar yaitu pada detik ke-6 hingga detik ke-8 dengan nilai mencapai 8000 bytes. Selebihnya penggunaannya hanya sedikit di mana kurang dari 4000 bytes yaitu pada detik ke-2, antara detik ke-5 dan ke-6 dan di akhir simulasi.




Gambar 4.55 *Packet Drop* kumulatif TCP SACK

Gambar 4.55 merupakan grafik *packet drop* kumulatif SACK di mana nilainya terus mengalami kenaikan akibat tingginya paket yang mengantri pada antrian terutama antara detik ke-2 dan ke-3 yaitu mencapai 9000 bytes. Kemudian nilainya tetap hingga menjelang detik ke-8 dan naik sedikit menjadi 10500 bytes hingga akhir simulasi.

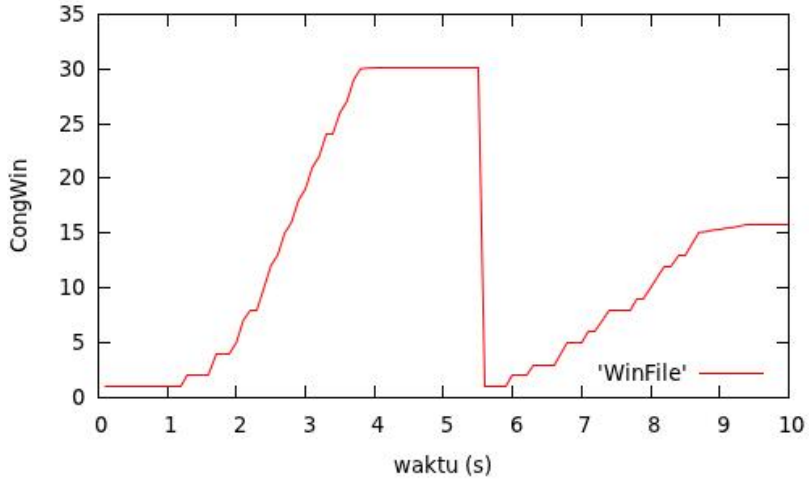



Gambar 4.56 RTT TCP SACK

Pada Gambar 4.56 adalah RTT yang dihasilkan dari TCP SACK. Pada pertengahan detik ke-2 hingga ke-3 RTT mengalami kenaikan yang tinggi akibat banyaknya *packet drop* hingga mencapai nilai 2.5 s. Dan pada detik selanjutnya

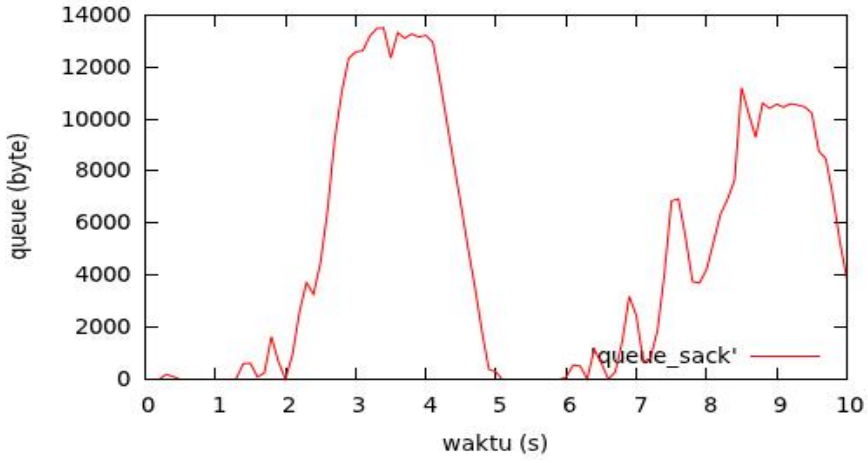
hingga akhir simulasi tetap mengalami nilai RTT yang tinggi dengan nilai mencapai 2 s.

**c. Percobaan 3**



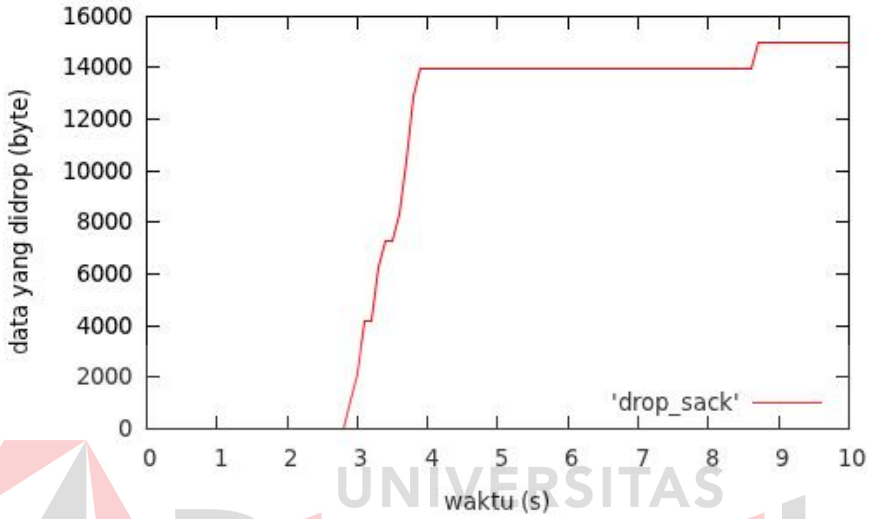

Gambar 4.57 CongWin TCP SACK

Gambar 4.57 menunjukkan grafik CongWin SACK di mana mencapai nilai 30 MSS yang berada diantara detik ke-3 dan ke-4 dan bertahan mulai dari detik ke-2 hingga pertengahan detik ke-5 hingga 6. Kemudian mengalami kongesti dan nilai diturunkan menjadi 1 MSS, selanjutnya naik secara eksponensial hingga CongWin mencapai nilai *threshold* yaitu 15 MSS hingga akhir simulasi.



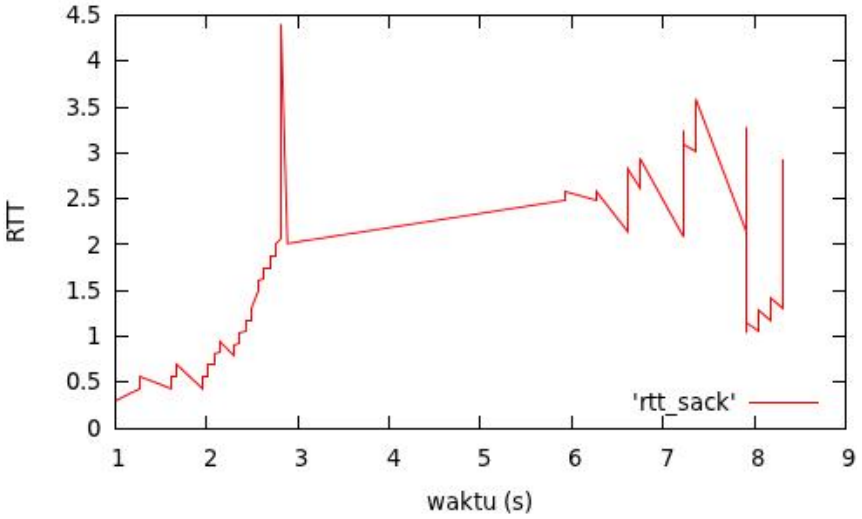

Gambar 4.58 *Queue* TCP SACK

Pada Gambar 4.58 merupakan grafik penggunaan *queue* SACK. Penggunaan terbesar terjadi pada detik antara detik ke-2 sampai ke-5 dengan puncak tertinggi hampir mencapai 14000 bytes. Dan penggunaan *queue* yang cukup besar juga terjadi di akhir simulasi antara detik ke-7 hingga detik ke-10 yaitu mencapai 11000 bytes.




Gambar 4.59 *Packet Drop* kumulatif TCP SACK

Gambar 4.59 merupakan grafik *packet drop* kumulatif pada TCP SACK. Pada gambar tesebut menunjukkan grafik *packet drop* meningkat hingga 14000 bytes akibat tingginya antrian pada detik ke-4. Meski tetap hingga pada detik ke-8, namun naik sedikit sehingga menjadi 15000 bytes hingga akhir simulasi.

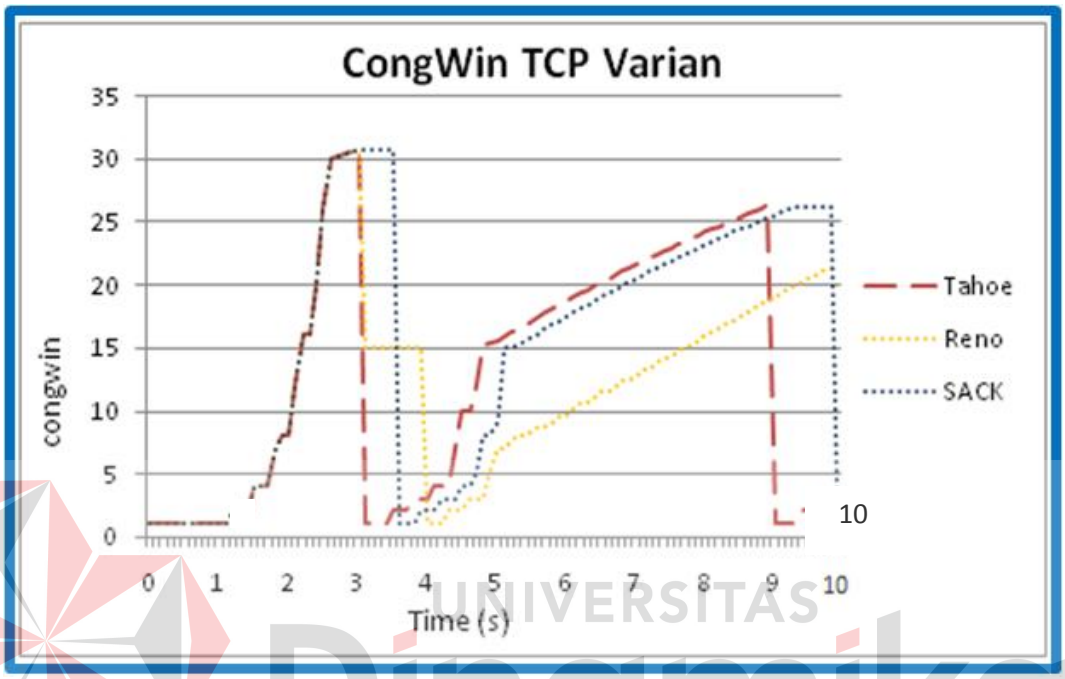



Gambar 4.60 RTT TCP SACK

Pada Gambar 4.60 di atas merupakan grafik RTT SACK dengan kenaikan di awal simulasi yang tinggi pada detik ke-3 yakni mencapai 4.5 s. Dan pada detik selanjutnya hingga akhir simulasi stabil meski tetap dengan nilai RTT yang tinggi yaitu 3.5 s. RTT pada variasi *bandwidth*=0.128Mb/30 ms merupakan RTT tertinggi diantara percobaan lainnya pada SACK. Tingginya nilai RTT akibat tingginya jumlah *packet drop* yang disebabkan oleh rendahnya *bandwidth* yaitu 0.128Mb/30ms.

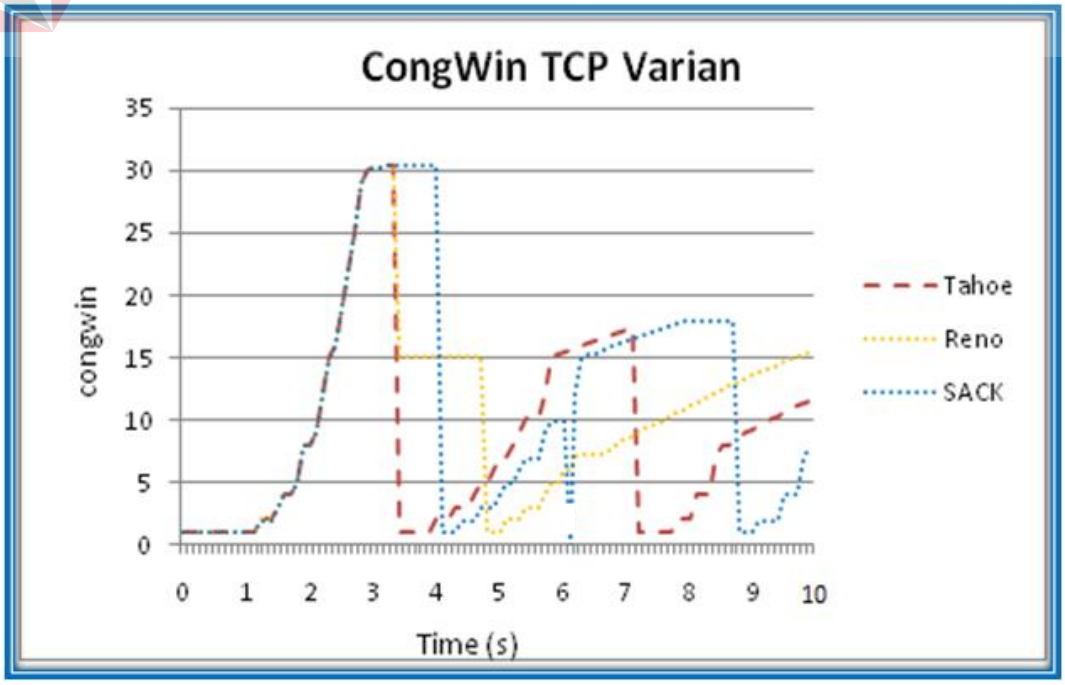
## 4.4 Analisis dan Pembahasan Hasil Simulasi

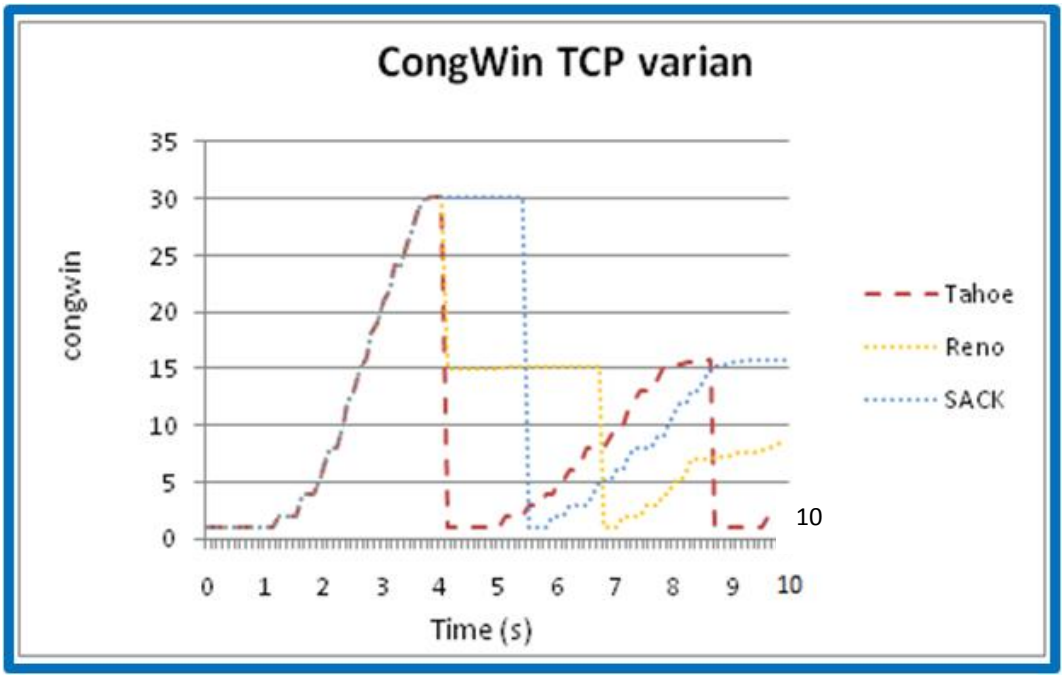
Seluruh hasil simulasi akan dianalisis dan dibahas berdasarkan 4 parameter uji perbandingan yang telah ditetapkan yaitu *congestion window*(CongWin), *queue*, *packet drop* dan RTT. Hasil grafik disajikan dalam satu gambar supaya dapat diketahui perbandingannya secara langsung dan dibahas berdasarkan teori-teori yang mendukung. Selain melalui grafik, analisis perbandingan dapat dilakukan melalui data yang berupa angka di mana telah diolah sehingga memudahkan kita dalam melakukan perbandingan unjuk kerja.




10





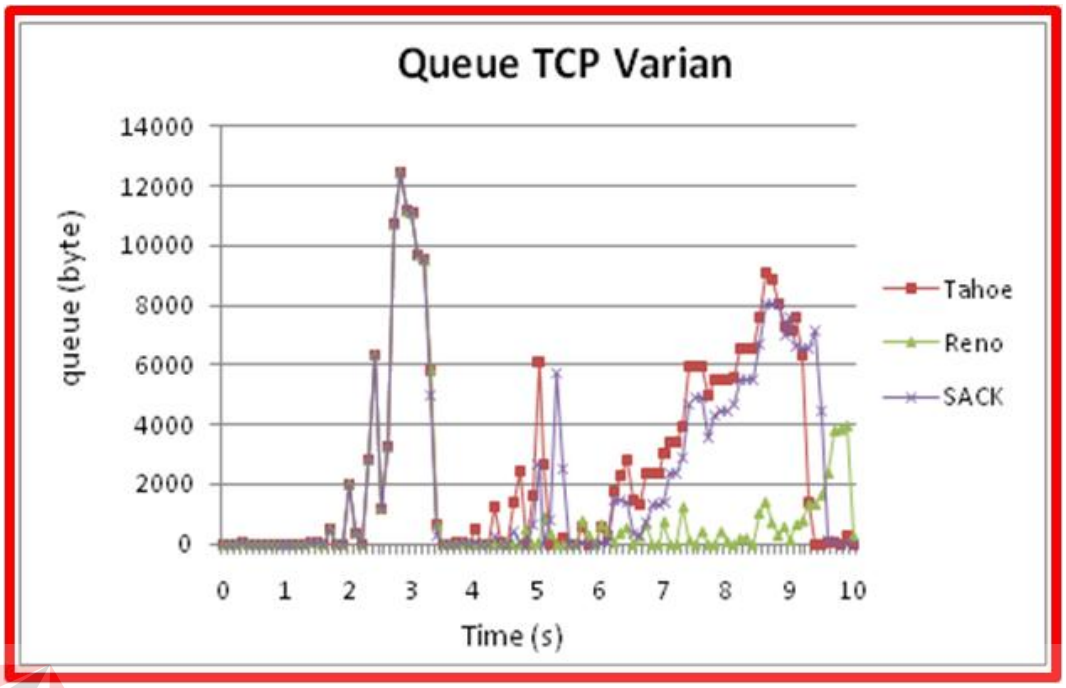


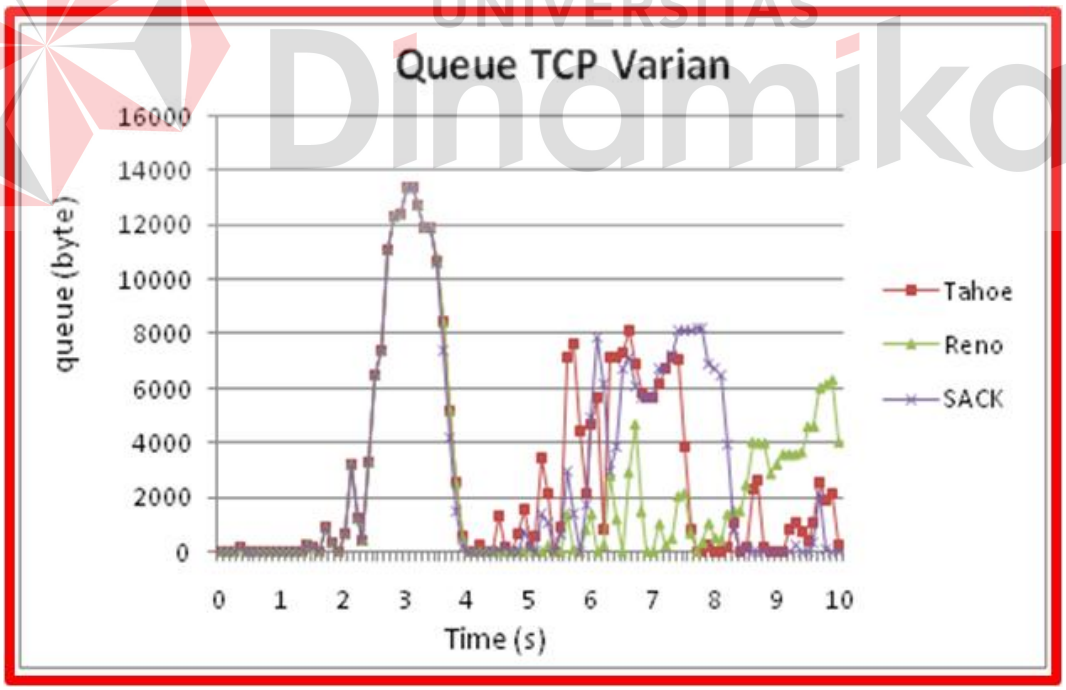

10

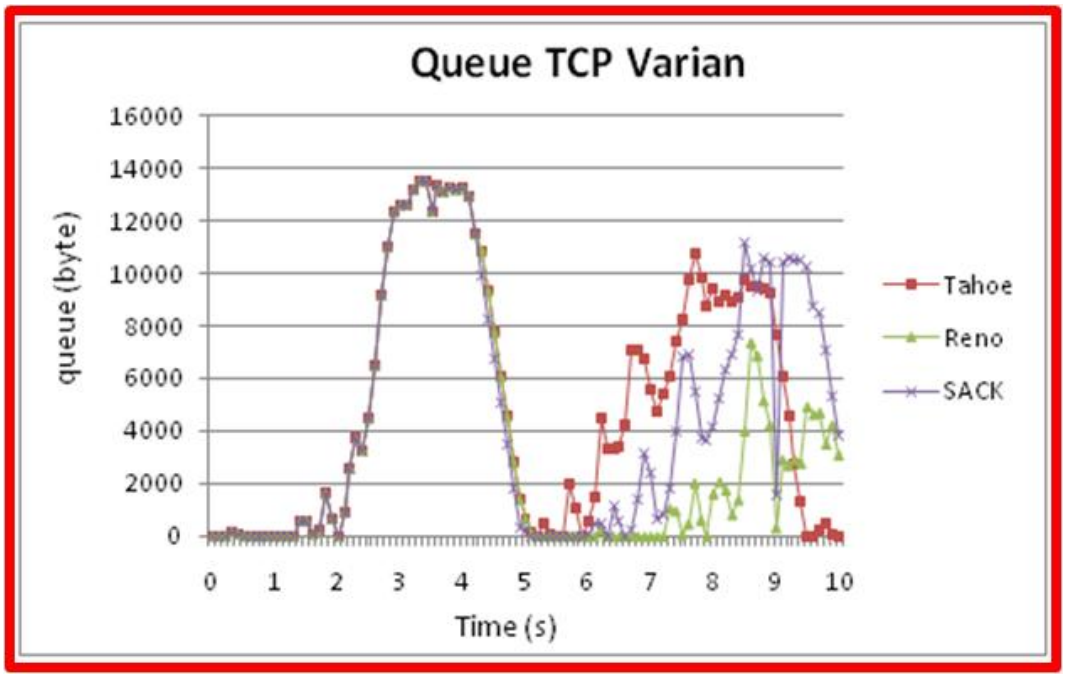
turun bahkan tidak mampu mencapai 20 MSS dan pada Gambar 4.63 dengan nilai *bandwidth*=0.128Mb/30ms di akhir simulasi hanya mencapai 15 MSS.

Pada percobaan ini, SACK memberikan unjuk kerja yang baik dengan mampu mempertahankan nilai CongWin pada nilai 30 MSS dibandingkan dua TCP lainnya karena TCP SACK tidak menurunkan langsung nilai CongWin pada saat kongesti. Selain itu SACK menggunakan fitur pengakuan *selective acknowledgments*, mengirim dengan cepat duplikat ACK segera setelah adanya kongesti serta mengakui segmen yang saat ini tiba hingga segmen terakhir. Ketiga hal ini menyebabkan pengirim tidak hanya mengirim kembali segmen yang hilang, tetapi semua segmen yang akan dikirim berikutnya. Oleh karena itu mengakibatkan jumlah paket yang beredar sangat besar sehingga jaringan bisa lebih cepat dan ukuran jendela juga lebih besar. (Haugdahl, 2007).

Terbukti dengan nilai rata-rata CongWin yang dihasilkan oleh ketiga TCP varian, di mana SACK memiliki rata-rata CongWin yang tinggi yaitu 15.3 MSS, 11.3 MSS dan 12.9 MSS, Tahoe yaitu 12.6 MSS, 8.6 MSS dan 8.1 MSS sedangkan Reno yaitu 11.6 MSS, 10.3 MSS dan 10.3 MS.



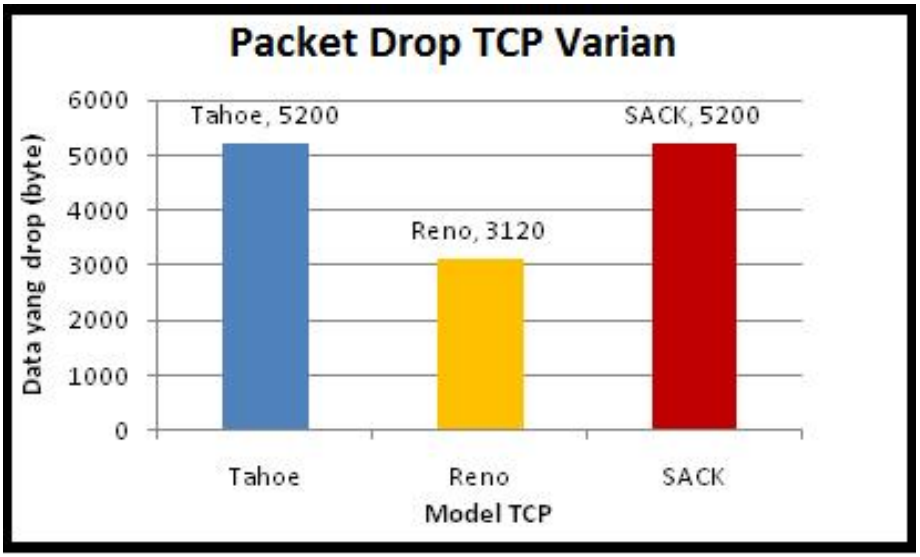





*bandwidth*=0.256Mb/30ms penggunaan *queue* naik mencapai 14000 bytes. Dan pada Gambar 4.66 dengan *bandwidth*=0.128Mb/30ms, penggunaan *queue* masih tetap yaitu 14000 bytes, namun dengan interval waktu yang lebih lama yakni sekitar 3 detik.

Pada uji parameter *queue* di atas dapat kita lihat bahwa penggunaan *queue* yang paling rendah adalah TCP Reno. Ini terlihat pada akhir simulasi di mana Reno menggunakan *queue* lebih rendah dibandingkan dua TCP varian lainnya yang berarti bahwa sistem pada Reno tidak begitu sibuk dalam melakukan proses pengiriman paket. Sehingga keterlambatan pelayanan juga rendah. Hal ini dipengaruhi oleh lebar antrian yang besar yakni 25 dan model antrian RED yang mampu meminimalisir *delay* dan menghilangkan efek *bursty*. Selain itu saat mengalami kongesti, Reno hanya menurunkan nilai CongWin menjadi setengah yang berarti bahwa Reno tidak mengosongkan jalur dan tetap melakukan pengiriman. Hal ini mempengaruhi penggunaan *queue* pada Reno yang tetap bekerja sehingga penggunaan *queue* dalam interval waktu tertentu lebih stabil daripada TCP varian lainnya.

### 4.4.3 Analisis *Packet Drop* TCP Varian

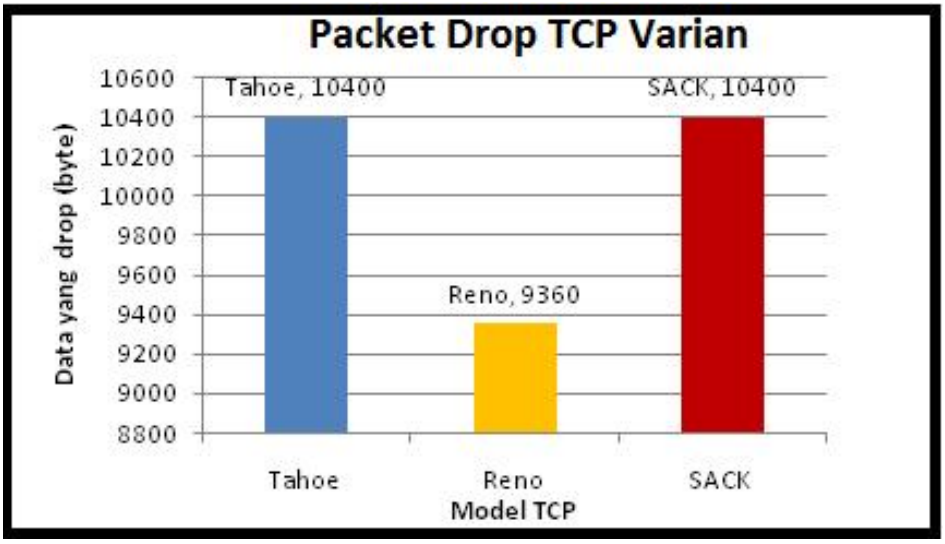



Gambar 4.67 Perbandingan *Packet Drop* dengan *bandwidth*=0.5Mb/30ms

Pada Tabel 4.2 merupakan Tabel *packet drop* TCP varian yang dengan bandwidth=0.5Mb/30ms menunjukkan Reno mempunyai *packet drop* yang paling rendah dengan 0.9%. Packet *drop* dalam prosentasi (% ) dihitung berdasarkan jumlah *packet drop* dibagi jumlah paket datang dikali 100%.

Tabel 4.2 *Packet Drop* dengan *bandwidth*=0.5Mb/30ms

| Tipe TCP | Paket datang | Packet Drop | Packet Drop |
|---|---|---|---|
| Tahoe | 397980bytes | 5200 bytes | 1.3 % |
| Reno | 334540bytes | 3120 bytes | 0.9 % |
| SACK | 383420bytes | 5200 bytes | 1.4 % |

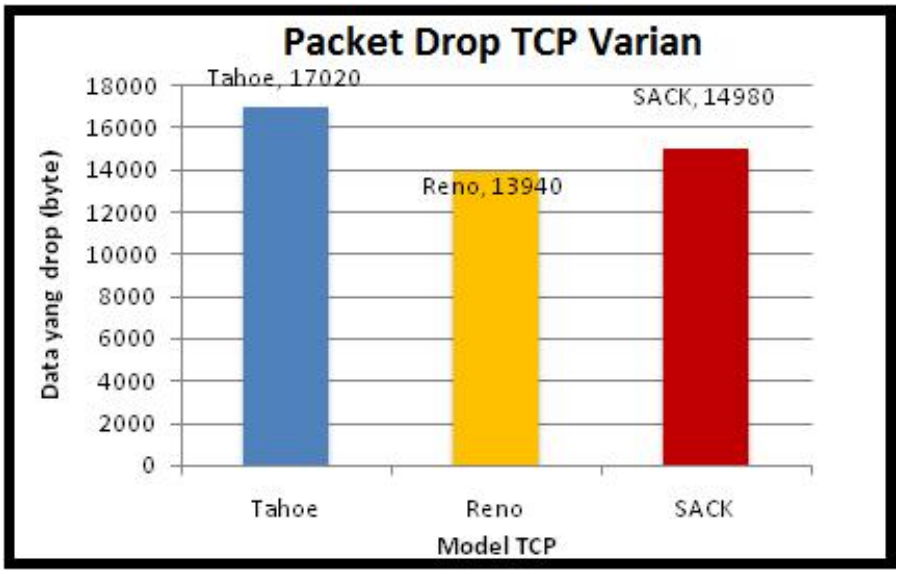



Gambar 4.68 Perbandingan *Packet Drop* dengan *bandwidth*=0.256Mb/30ms

Pada Tabel 4.3 merupakan Tabel *packet drop* TCP varian yang dengan bandwidth=0.256Mb/30ms menunjukkan Reno mempunyai *packet drop* yang paling rendah dengan 4.4%. Angka ini naik 3.5% dari *packet drop* pada *bandwidth*=0.5Mb/30ms. Sehingga, besarnya *bandwidth* sangat berpengaruh pada banyaknya *packet drop*.

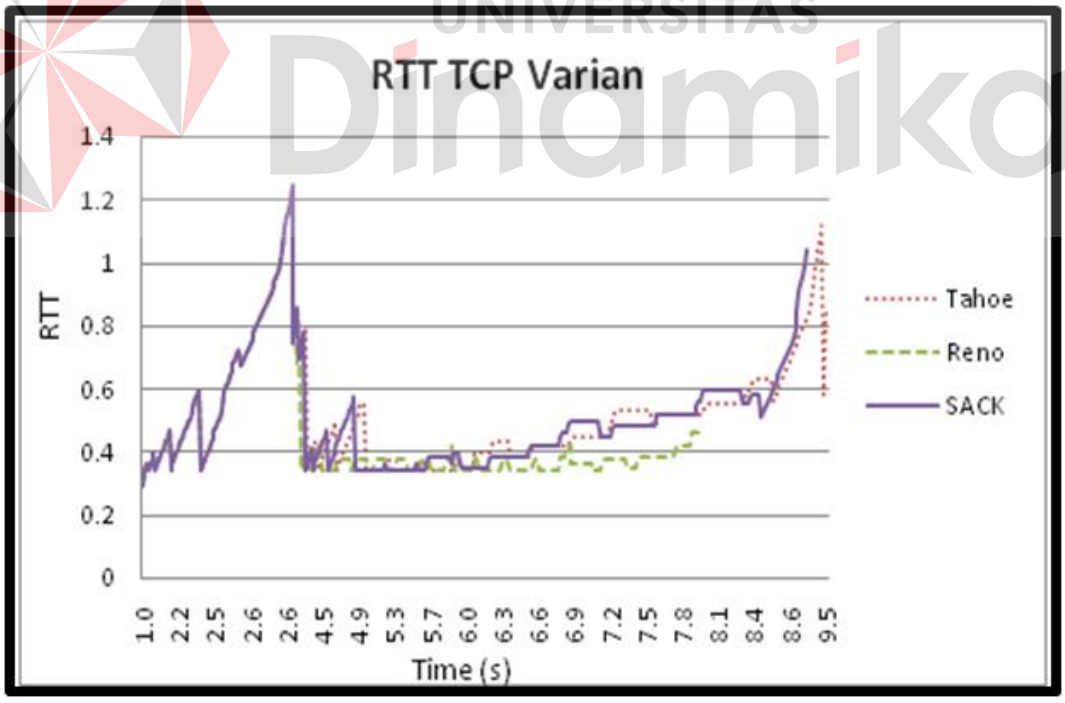
Tabel 4.3 *Packet Drop* dengan *bandwidth*=0.256Mb/30ms

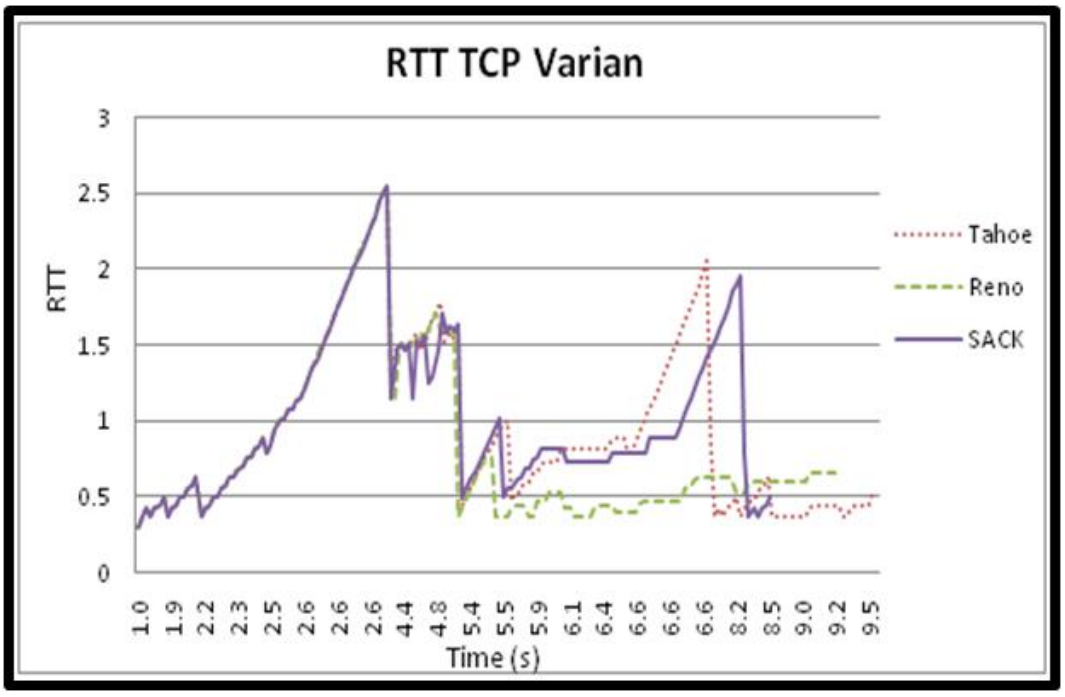
| Tipe TCP | Paket Datang | Packet Drop | Packet Drop |
|---|---|---|---|
| Tahoe | 225340 bytes | 10400 bytes | 4.6% |
| Reno | 214940 bytes | 9360 bytes | 4.4% |
| SACK | 191020 bytes | 10400 bytes | 5.4% |

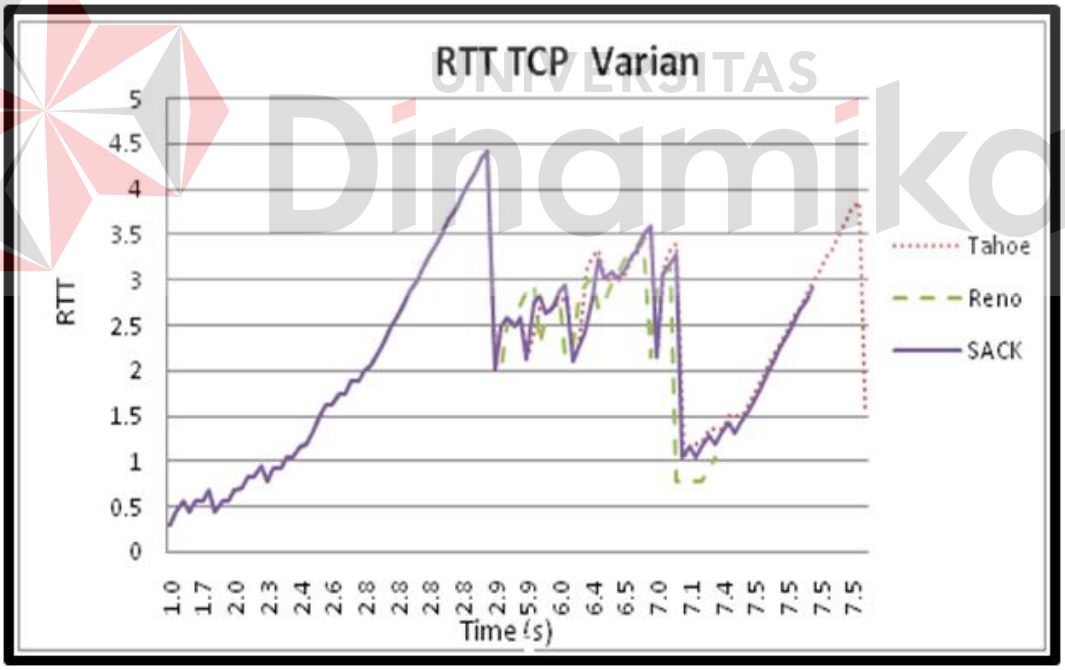





| Tipe TCP | Paketdatang | Packet Drop (byte) | Packet Drop |
|---|---|---|---|
| Tahoe | 141100bytes | 17020bytes | 12.1% |
| Reno | 118220bytes | 13940bytes | 11.8% |
| SACK | 135900bytes | 14980bytes | 11.0% |


RTT TCP Varian
RTT
1.4
1.2
1
0.8
0.6
0.4
0.2
0
1.0
2.2
2.5
2.6
2.6
4.5
4.9
5.3
5.7
6.0
6.3
6.6
6.9
7.2
7.5
7.8
8.1
8.4
8.6
9.5
Time (s)
Tahoe
Reno
SACK



pertengahan detik ke-2 dan ke-3. Hal ini disebabkan tingginya *packet* yang di*drop* (Lihat sub bab mengenai analisis *packet drop*) sehingga perlu melakukan pengiriman ulang yang mengakibatkan tingginya nilai RTT. Pada Gambar 4.70, nilai RTT terendah daripada RTT pada percobaan pada Gambar 4.71 dan Gambar 4.72. Di mana setiap penurunan *bandwidth*, RTT mengalami kenaikan hingga dua kali lipat dilihat pada titik terjadinya kongesti. Hal ini dibuktikan tinggi RTT pada Gambar 4.74, RTT tertinggi dengan nilai 1.3 s pada detik ke-2 sampai ke-3 dan turun 1.1 s di akhir simulasi. Pada Gambar 4.72, RTT tertinggi pada nilai 2.6 s pada detik ke-2 sampai ke-3 dan turun menjadi 2 s di akhir simulasi. Sedangkan percobaan pada Gambar 4.72 dengan ukuran *bandwidth* terendah yaitu 0.128Mb/30 ms, menghasilkan nilai RTT tertinggi dengan nilai 4.5 s pada detik ke-2 sampai ke-3 dan turun pada nilai 4 s hingga akhir simulasi.

Pada grafik, dapat kita lihat bahwa nilai RTT terendah adalah TCP Reno dengan rata-rata nilai RTT adalah 0.44 s, 0.82 s dan 2.08 s. RTT pada Tahoe yaitu 0.52 s, 0.93 s dan 2.22 s. Sedangkan pada SACK yaitu 0.51 s, 1.02 s dan 2.12 s. Hal ini berarti Reno membutuhkan waktu paling sedikit dalam melakukan pengiriman paket dari node sumber menuju node tujuan hingga paket ACK dari tujuan yang dikirim ke sumber sebagai informasi pengiriman. (Rahman M., Kabir, Lutfullah, & Amin, 2008. Hasil dari seluruh percobaan yang telah dilakukan berdasarkan parameter uji dapat dirangkum dan ditampilkan pada Tabel 4.5.

Tabel 4.5 Hasil Perbandingan Percobaan

| **Perc. ke-** | **TCP** | **Rata-rata CongWin** | **Rata-rata *Queue*** | **ΣPacket Drop** | **Packet Drop** | **Rata-rata RTT** |
|---|---|---|---|---|---|---|
| 1. | Tahoe | 12.6 MSS | 2647.502bytes | 5200bytes | 1.3% | 0.52 s |
| | Reno | 11.6 MSS | 1219.61 bytes | 3120bytes | 0.9% | 0.44 s |
| | SACK | 15.3 MSS | 2469.229bytes | 5200bytes | 1.4% | 0.51 s |

| | | | | | | |
|---|---|---|---|---|---|---|
| 2. | Tahoe | 8.6 MSS | 2969.594 bytes | 10400bytes | 4.6% | 0.93 s |
| | Reno | 10.3 MSS | 2478.54 bytes | 9360bytes | 4.4% | 0.81 s |
| | SACK | 11.3 MSS | 3087.85 bytes | 10400 bytes | 5.4 % | 1.02 s |
| 3. | Tahoe | 8.1 MSS | 5083.411 bytes | 17020 bytes | 12.1 % | 2.22 s |
| | Reno | 10.3 MSS | 3474.328 bytes | 13940 bytes | 11.8 % | 2.08 s |
| | SACK | 12.9 MSS | 4757.306 bytes | 14980 bytes | 11.0 % | 2.12 s |

Berdasarkan Tabel 4.5 menunjukkan rata-rata CongWin dari seluruh percobaan yang dilakukan dan berdasarkan perbandingan hasil analisis menunjukkan rata-rata CongWin tertinggi yaitu TCP SACK dengan nilai 15.3 MSS, 11.3 MSS dan 12.9 MSS. Hal ini disebabkan oleh algoritma SACK dalam menghadapi kongesti tidak segera menurunkan nilai CongWin sehingga mampu mempertahankan nilai CongWin saat mencapai maksimum dengan lebih lama dibandingkan dua TCP varian lainnya.

Pada unjuk kerja penggunaan *queue*, pada Tabel 4.5 disimpulkan bahwa Reno menunjukkan penggunaan *queue* yang paling rendah yaitu dengan nilai rata-rata *queue* 1219.61 bytes, 2478.54 bytes dan 3474.328 bytes. Penggunaan *queue* yang rendah berarti paket yang sedang mengantri pada *queue* sebelum menuju *jalur bottleneck* juga pasti rendah sehingga probabilitas mengalami *packet drop* juga rendah. Dan dari rendahnya *packet drop* maka dipastikan nilai RTT juga rendah. Hal ini terbukti pada Tabel 4.5 nilai rata-rata *packet drop* pada Reno berturut-turut adalah 3120 bytes, 9360 bytes dan 13940 bytes. Meskipun dalam prosentase pada variasi *bandwidth*=0.128 Mb/30ms *packet drop* pada Reno lebih tinggi 0.8% dari SACK, namun hal tersebut tidak terlalu signifikan karena jika dirata-rata secara keseluruhan Reno tetap memiliki rata-rata RTT terendah yaitu 0.44 s, 0.81 s dan2.08 s.

# BAB V

# KESIMPULAN DAN SARAN

## 5.1 Kesimpulan

Kesimpulan pada hasil analisis perbandingan unjuk kerja algoritma *congestion control* pada TCP Tahoe, Reno dan SACK (*Selective Acknowledgment*) adalah sebagai berikut ini:

1. Pembangunan simulasi jaringan dengan menggunakan NS-2 untuk melakukan perbandingan unjuk kerja algoritma TCP telah dilakukan dengan hasil yaitu konfigurasi *.tcl ketiga TCP varian dapat menghasilkan grafik *Network Animator*(NAM) sesuai dengan desain topologi, parameter dan skenario yang menggunakan variasi *bandwidth* 0.5Mb/30ms, 0.256Mb/30ms dan 0.128Mb/30ms yang sesuai dengan tujuan dilakukannya analisis ini.
2. Hasil perbandingan dan analisis algoritma TCP berdasarkan CongWin, *queue*, *packet drop* dan RTT adalah sebagai berikut:
   a. Pada sisi CongWin, SACK mempunyai unjuk kerja terbaik dengan rata-rata CongWin sebesar 15.3 MSS pada *bandwidth*=0.5Mb/30ms, 11.3 MSS pada *bandwidth*=0.256Mb/30ms dan 12.9 MSS pada *bandwidth*=0.128Mb/30ms
   b. Pada sisi penggunaan *queue*, Reno memiliki unjuk kerja terbaik yaitu dengan rata-rata penggunaan *queue* hanya sebesar 1219.6 bytes pada *bandwidth*=0.5Mb/30ms, 2478.5 bytes pada *bandwidth*=0.256Mb/30ms dan 3474.3 bytes pada *bandwidth*=0.128Mb/30ms.

c. Pada sisi *packet drop*, Reno juga mempunyai unjuk kerja terbaik dengan rata-rata *packet drop* yang rendah yakni untuk 3120 bytes(0.9%) pada *bandwidth*=0.5Mb/3ms, 9360 bytes (4.4%) pada *bandwidth*=0.256Mb/30 ms dan 13940 bytes (11.8%) pada *bandwidth*=0.128Mb/30ms.

d. Pada sisi RTT, Reno mempunyai unjuk kerja terbaik dengan rata-rata RTT terendah dengan nilai 0.44 s pada *bandwidth*=0.5Mb/30 ms, 0.81 s pada *bandwidth* = 0.256Mb/30ms dan 2.08 s pada *bandwidth* = 0.128Mb/30ms.

3. Dari hasil analisis, dapat disimpulkan bahwa tidak ada algoritma TCP *congestion control* dengan unjuk kerja terbaik di semua parameter uji yang telah dilakukan. Oleh karena itu, untuk karakteristik *network* yang berbeda-beda, maka nilai unjuk kerja algoritma-algoritma TCP memiliki *behavior* yang berbeda pula. Sehingga masih terus diperlukan penelitian dan pengembangan protokol TCP untuk menjawab tantangan-tantangan ke depan, terutama untuk kebutuhan *high speed communication*.

## 5.2 Saran

Agar diperoleh hasil analisis perbandingan unjuk kerja yang lebih efektif lagi dan sebagai pengembangan dalam penelitian selanjutnya, perbandingan unjuk kerja algoritma TCP Tahoe, Reno dan SACK sebaiknya disimulasikan secara simultan melewati satu *link bottleneck* dengan tujuan:

1. Dapat memahami *behavior congestion control* ketiga TCP yang disimulasikan secara simultan untuk mendekati kondisi nyata.
2. Dapat membandingkan karakteristik *fairnesss* dari utilisasi *bandwidth* ketiga TCP yang menggunakan satu jalur *bottleneck*.

# DAFTAR PUSTAKA


A. Behrouz, F. (2007). *Data Communication and Networking Fourth Edition.* Singapore: McGraw-Hill Education.

Agilent, T. (2008). *TCP and Queue Management.* Retrieved December 2, 2012, from www.agilent.com: www.agilent.com/find/n2x

Budiardjo, B., & Thiotrisno, M. (2003). *Simulasi Pengukuran Intrafairness dan Interfairness Protocol-Protocol Stream COntrol Transmission Protocol (SCTP) dan Transmission Control Protocol pada Jaringan Unicast.* Indonesia: Universitas Indonesia.

Feipeng, L. (2008). *TCP Tahoe, Reno, New Reno and SACK-a Brief Comparison.* Retrieved September 23, 2012, from www.roman10.net: www.roman10.net/tcp-tahoe-reno-newreno-and-sacka-brief-comparison/

Haugdahl, J. S. (2007, October 4). *TCP Selective ACK (SACK) Packet Recovery Analysis: Part 1 - The Problem*. Retrieved February 7, 2013, from http://thenetworkguy.typepad.com/: http://thenetworkguy.typepad.com/nau/2007/10/one-of-the-most.html

Issariyakul, T., & Hossain, E. (2012). *Introduction to Network Simulator NS2 Second Edition.* New York: Springer.

Jonathan, P., & Hermawan, D. (2011). *Network Traffic Management, Quality Of Service (QoS), Congestion Control dan Frame Relay.* Jakarta: Universitas Gunadarma.

Jusak, D. (2011). *Diktat Bab 3 dan 4 Lapisan Transport* . Surabaya: STIKOM.

Mathias, M., Mahdavi, J., Floyd, S., & Romanow, A. (1996). TCP Selective Acknowledgment Options (SACK). *RFC 2018* .

Rahman, M., Kabir, A., Lutfullah, K.& Amin, M. (2008). Fair Comparisons of Different TCP Variants for Future Deployment of Networks. *ICECC* , 260-263.

Sikdar, B., Kalyanaraman, S., & Vastola, K. (2002). *Analytic Models and Comparative Study of the Latency and Steady-State Throughput of TCP Tahoe, Reno and SACK.* New York: Dept. of ECSE, Rensselaer Polytechnic Institute Troy.

Stretch, J. (2010, Juny 17). *TCP Selective Acknowledgments.* Retrieved September 21, 2011, from www.packetlife.net:

www.packetlife.net/blog/2010/jun/17/tcp-selective-acknowledgments-sack/

Sukmaaji, A., & Rianto. (2008). *Jaringan Komputer, Konsep Dasar Pengembangan Jaringan & Keamanan Jaringan.* Yogyakarta: Andi.

Welzl, M. (2008). *The ns-2 Network Simulator*. Retrieved December 10, 2012, from www.welzl.at: www.welzl.at

Wirawan, A. B., & Indarto, E. (2004). *Mudah Membangun Simulasi dengan Network Simulator-2.* Yogyakarta: Andi.