BAB II

LANDASAN TEORI

2.1 *Transimission Control Protocol* (TCP)

TCP merupakan protokol pada lapisan transport yang bersifat dapat diandalkan (*reliable*). Dalam hal ini TCP akan menjamin bahwa aliran data dari pengirim menuju ke penerima tidak mengandung kesalahan bit, tidak ada duplikasi *segment* dan diterima sesuai dengan urutan. Dengan kata lain setiap aliran data yang diterima oleh terminal penerima persis sama dengan aliran data yang dikirimkan oleh terminal pengirim. Namun untuk menjamin aliran data yang *reliable* dibutuhkan proses yang cukup rumit, apalagi pada kenyataannya protokol TCP berjalan di atas protokol IP yang bersifat *unreliable*, dimana protokol IP menggunakan konsep *best effort service*. (Jusak, 2010)

TCP memiliki karakteristik sebagai berikut:

1. Berorientasi sambungan (*connection-oriented*)

   Sebelum data dapat ditransmisikan antara dua *host*, dua proses yang berjalan pada lapisan aplikasi harus melakukan negosiasi untuk membuat sesi koneksi terlebih dahulu. Koneksi TCP ditutup dengan menggunakan proses terminasi koneksi TCP (TCP *connection termination*).

2. *Full-duplex*

   Untuk setiap *host* TCP, koneksi yang terjadi antara dua *host* terdiri atas dua buah jalur, yakni jalur keluar dan jalur masuk. Dengan menggunakan teknologi lapisan yang lebih rendah yang mendukung *full-duplex*, maka data pun dapat secara simultan diterima dan dikirim. *Header* TCP berisi nomor urut

(TCP *sequence number*) dari data yang ditransmisikan dan sebuah *acknowledgment* dari data yang masuk.

3. Dapat diandalkan (*reliable*)

   Data yang dikirimkan ke sebuah koneksi TCP akan diurutkan dengan sebuah nomor urut paket dan akan mengharapkan paket *positive acknowledgment* dari penerima. Jika tidak ada paket *Acknowledgment* dari penerima, maka *segment* TCP (protocol data unit dalam protokol TCP) akan ditransmisikan ulang. Pada pihak penerima, *segment* duplikat akan diabaikan dan *segment* yang datang tidak sesuai dengan urutannya akan diletakkan di belakang untuk mengurutkan *segment* TCP. Untuk menjamin integritas setiap *segment* TCP, TCP mengimplementasikan penghitungan TCP *Checksum*.

4. Mempunyai *byte stream*

   TCP melihat data yang dikirimkan dan diterima melalui dua jalur masuk dan jalur keluar TCP sebagai sebuah *byte stream* yang berdekatan. Nomor urut TCP dan nomor *acknowlegment* dalam setiap *header* TCP didefinisikan juga dalam bentuk *byte*. Meski demikian, TCP tidak mengetahui batasan pesan-pesan di dalam *byte stream* TCP tersebut. Untuk melakukannya, hal ini diserahkan kepada protokol lapisan aplikasi (dalam DARPA *Reference Model*), yang harus menerjemahkan *byte stream* TCP ke dalam bahasa yang ia pahami.

5. Mempunyai *congestion control*

   TCP menggunakan *congestion control* untuk membatasi kecepatan pengiriman data pada saat terjadi kongesi di dalam jaringan. Secara umum, pada saat pengirim mendeteksi bahwa jaringan sedang mengalami kongesi, maka pengirim akan menurunkan kecepatan pengiriman data. Akan tetapi

apabila pengirim mendeteksi bahwa kongesi dalam jaringan telah berkurang, maka pengirim akan meningkatkan kecepatan pengiriman data.

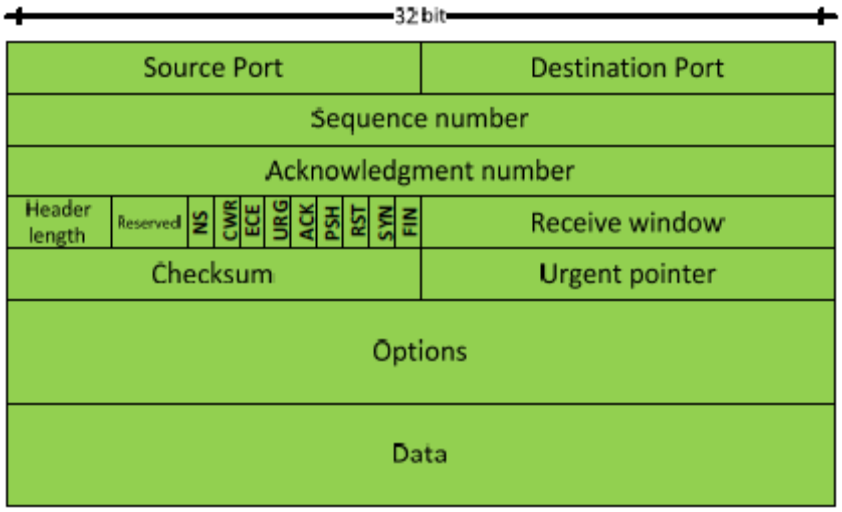
6. Memiliki layanan *flow control*

   Untuk mencegah data terlalu banyak dikirimkan pada satu waktu, yang akhirnya membuat kemacetan jaringan *internetwork* IP, TCP mengimplementasikan layanan *flow control* yang dimiliki oleh pihak pengirim yang secara terus menerus memantau dan membatasi jumlah data yang dikirimkan pada satu waktu. Untuk mencegah pihak penerima memperoleh data yang tidak dapat disangganya (*buffer*), TCP juga mengimplementasikan *flow control* dalam pihak penerima, yang mengindikasikan jumlah *buffer* yang masih tersedia dalam pihak penerima.

7. Mengirimkan paket secara *one-to-one*

   Hal ini karena memang TCP harus membuat sebuah sirkuit logis antara dua buah protokol lapisan aplikasi agar saling dapat berkomunikasi. TCP tidak menyediakan layanan pengiriman data secara *one-to-many*.

Protokol TCP mempunyai struktur *segment* yang terdiri atas dua bagian besar, yaitu *header* dan data/*payload*. *Payload* adalah data yang berasal dari lapisan aplikasi. Struktur dari sebuah *segment* TCP secara detail ditunjukkan dalam Gambar 2.1. Penjelasan dari masing-masing slot adalah sebagai berikut:



Gambar 2.1. Struktur S*egment* pada TCP

a) Bagian pertama dari *header* TCP adalah *source port* sepanjang 16 bit dan *destination port* sepanjang 16 bit, keduanya digunakan oleh TCP untuk menunjukkan nomor *port* sumber dan nomor *port* tujuan sebagai identifikasi dari jenis aplikasi yang sedang disetujui oleh *client* dan *server*, dalam hal ini keduanya berkaitan erat dengan proses *multiplexing* dan *demultiplexing*.

b). *Sequence number* dengan panjang 32 bit merupakan nomor urut bagi *segment* TCP. Apabila *flag* SYN bernilai 1 maka *sequence number* berisi nomor inisialisasi bagi segment, tetapi apabila *flag* SYN bernilai 0 *sequence number* berisi nomor segment yang merupakan akumulasi panjang data dari *byte* yang paling awal sampai *byte* terakhir yang ada di dalam *segment* ini.

c). *Acknowledgment number* dengan panjang 32 bit merupakan nomor bagi *segment acknowledgment*. Apabila *flag* ACK di set ke 1 maka *acknowledgment number* berisi nomor urut dari *segment* berikutnya yang diharapkan oleh penerima.

d). *Header length* dengan panjang 4 bit menspesifikasikan panjang *header* dari *segment* TCP. Panjang *header* dapat berubah-ubah karena adanya *slot options*

pada Gambar 2.1. Apabila *options* bernilai 0, maka panjang *header* hanya 20 *byte*.

e). *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 *segment* ini adalah *segment acknowledgement*. (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.

f). *Receive window* dengan panjang 16 bit berisi jumlah dalam bentuk *byte* yang mengindikasikan jumlah data yang dapat ditampung oleh penerima pada saat ini.

g). *Checksum* sepanjang 16 bit digunakan untuk pengecekan kesalahan bit pada *header* dan data.

h). *Urgent data pointer* sepanjang 16 bit menunjuk pada *sequence number* dari data (yang bersifat urgent) terakhir apabila *flag* URG di set menjadi 1.

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

2.2 *Datagram Congestion Control Protocol* (DCCP)

DCCP mempunyai dua tugas utama. Satu untuk menetapkan, memelihara dan mengakhiri koneksi *unreliable* serta yang lain digunakan untuk mekaninsme

congestion-control pada koneksi *unreliable.* (Lai, 2008).

Karakteristik utama DCCP adalah sebagai berikut:

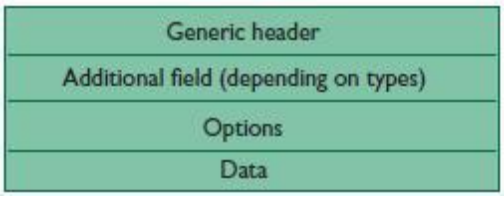
1. *Unreliable data transfer*. DCCP tidak mengirim ulang paket data yang hilang selama transmisi.
2. Penetapan koneksi yang dapat dipercaya dan kemampuan bernegoisasi.

   Selama proses penetapan koneksi, mengakhiri koneksi dan proses negoisasi, DCCP menggunakan transmisi yang *reliable* sehingga tetap mengirim ulang paket kontrol sampai yang satu menerima balasan sisi yang lain.
3. Pilihan paket yang cukup.

   DCCP menawarkan banyak pilihan paket. Pilihan *ack vector* misalnya, merekam *receiving status* penerima dan pilihan *data dropped* mencatat penyebab paket hilang.
4. Pilihan *congestion control.*

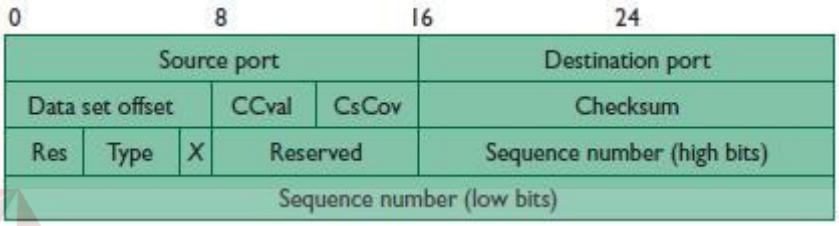
   Sekarang ini DCCP menawarkan tiga mekanisme *congestion control. Congestion control ID 2 (*CCID 2*)* seperti TCP, CCID 3 menggunakan *TCP Friendly Rate Control* (TFRC) dan CCID 4 menggunakan TFRC-SP (*TFRC for Small Packet*). Suatu koneksi dapat memilih *congestion control* sesuai dengan permintaannya.
5. Pencegahan dari serangan *SYN flooding.*

   Selama proses penetapan koneksi, DCCP menggunakan *cookies* untuk mencegah serangan *SYN flooding* yang mana sering dialami TCP.



Gambar 2.2. Format Paket DCCP (Lai, 2008)

a). Bagian pertama adalah *header* paket secara umum. Format paket DCCP terlihat pada Gambar 2.2.



Gambar 2.3. DCCP *Generic Header* (Lai, 2008)

b). *Source* dan *destination port:* memiliki panjang masing-masing sebesar 16 bit. *Field* ini menunjukkan koneksi, serupa dengan *field* pada TCP dan UDP.

c). *Source port* menunjukkan *port* yang berkaitan dengan *endpoint* yang mengirim paket ini dan *destination port* menunjukkan port di sisi yang lain. Ketika menginisialisasi sebuah koneksi, *client* seharusnya memilih *source port* secara acak untuk mengurangi kemungkinan serangan.

d). *Data offset*: menunjukkan lokasi *offset* dari awal header sampai awal data – ukuran *header*. Memiliki panjang 8 bit.

e). CCval: menunjukkan *congestion control* yang digunakan (CCID2, CCID3 atau CCID4). Panjang *field* sebesar 4 bit.

f). CsCov (*Checksum Coverage*) menentukan bagian paket yang dicakup oleh *checksum field*. Memiliki panjang *field* sebesar 4 bit.

g). Checksum: 16 bit. *Checksum* internet *header* paket DCCP (termasuk *options*), pseudoheader *network-layer* dan tergantung pada cakupan *checksum*,

h). *Reserved* (Res): 3 bit. Pengirim harus mengisi *field* ini bernilai nol pada paket yang dibangkitkan dan penerima harus mengabaikan nilai tersebut.

i). *Type*: 4 bit. *Field type* menunjukkan jenis paket. Sekarang ini, DCCP memiliki sepuluh tipe paket: *DCCP-Request, DCCP-Response, DCCP-Data, DCCP-Ack, DCCP-DataAck, DCCP-CloseReq, DCCP-Close, DCCP-Reset, DCCP-Sync, DCCP-SynAck.*

j). *Extended Sequence Numbers* (X) : 1 bit. Nilai satu menunjukkan penggunaan sebuah *header* tambahan dengan 48-bit nomor *sequence* dan *acknowledgement*.

k). *Sequence number*: 48 atau 24 bit. Menunjukkan paket secara unik pada urutan (*sequence*) semua paket. *Sequence number* bertambah satu demi satu selama pengiriman paket, termasuk paket seperti DCCP-Ack yang tidak membawa data aplikasi.

2.2.1 *CONGESTION CONTROL ID 2* (CCID 2)

CCID 2 menggunakan tiga variabel utama yaitu cwnd (*congestion window*), nilai maksimum yang diijinkan pada paket yang beredar. Ssthresh (*slow start threshold*) adalah permulaan dari cwnd yang memisahkan *slow start* dari *congestion avoidance.* Dan *pipe* yaitu jumlah rata-rata paket yang beredar. Tiga variabel ini hampir sama dengan variabel di dalam *Selective Acknowledgement* (SACK) TCP. (Lai, 2008).

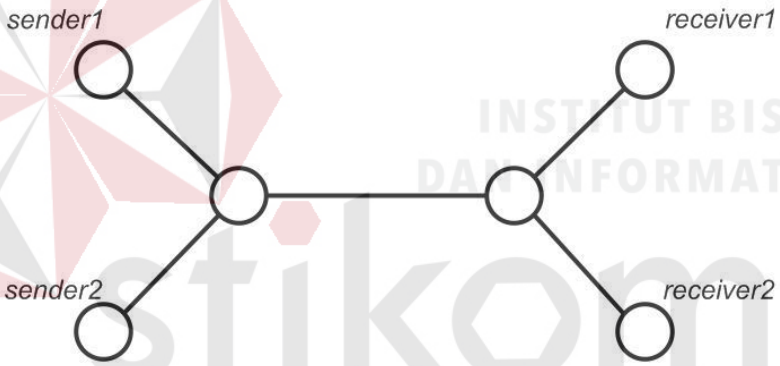
2.2.2 *CONGESTION CONTROL ID 3* (CCID 3)

CCID 3 menggunakan *congestion control* TFRC. Di dalam tahap awal,

slow start, nilai transmisi naik dua kali lipat setelah *round-trip time* (RTT) sampai penerima melaporkan adanya paket hilang atau ECN tertandai. Setelah itu pengirim meninggalkan tahap *slow start* dan menghitung kecepatan transmisi paket yang diijinkan pada periode selanjutnya. CCID 3 mengharapkan untuk bisa mendapatkan *throughput* yang sama dengan TCP dan mendapatkan *smooth transmission rate.* (Lai, 2008).

2. 3. Topologi *Dumb-Bell*

Model topologi yang digunakan adalah topologi *dumb-bell*. Terlihat pada Gambar 2.3



Gambar 2.4. Topologi *Dumb-bell*

Penggunaan model ini dikarenakan topologi *dumb-bell* mempunyai *single bottleneck link* dengan jumlah lebih dari satu pengirim dan lebih dari satu penerima. Sama seperti desain jaringan di dunia nyata yang terdiri dari banyak pengguna yang mengakses internet sehingga pada suatu titik akan bertemu di jalur *backbone*.

2.4. *Quality of Service* (Qos)

Quality of Service (QoS) merupakan metode pengukuran tentang

seberapa baik jaringan dan merupakan suatu usaha untuk mendefinisikan karakteristik dan sifat dari satu servis. Ada beberapa alasan mengapa diperlukan QoS, yaitu:

1. Untuk memberikan prioritas untuk aplikasi-aplikasi yang kritis pada jaringan.
2. Untuk memaksimalkan penggunaan investasi jaringan yang sudah ada.
3. Untuk meningkatkan performansi untuk aplikasi-aplikasi yang sensitif terhadap *delay*.
4. Untuk merespon terhadap adanya perubahan pada aliran trafik di jaringan.

2.4.1. Utilisasi *Bandwidth*

Faktor-faktor ini akan dianalisa dengan asumsi, utilisasi sumber daya atau sering disebut sebagai utilisasi. Utilisasi adalah ukuran seberapa sibuk sumber daya yang ada. Dalam teori antrian utilisasi direpresentasi sebagai waktu server melakukan layanan dan didefinisikan dengan persamaan 2. (Riadi & Wicaksono, 2011).

Perhitungan utilisasi *bandwidth* memerlukan nilai *throughput*. Rumus *throughput* ditunjukkan pada persamaan 2.1.

$$\boldsymbol{Throughput} = \frac{\textbf{Jumlah data yang berhasil lewat (bps)}}{\textbf{Lama waktu pengamatan (s)}} \quad (2.1)$$

$$\boldsymbol{Utility} = \frac{\boldsymbol{throughput}}{\boldsymbol{bandwidth}} \times 100\% \quad (2.2)$$

dimana:

Throughput = Jumlah paket yang berhasil melewati jalur dalam waktu tertentu.

Bandwidth = Jumlah besaran *bandwidth* yang tersedia (bps)

2.4.2 *Packet loss*

Packet loss adalah jumlah paket yang hilang saat pengiriman paket data ke tujuan, kualitas terbaik pada jaringan LAN/WAN jika jumlah *losses* paling kecil (Riadi & Wicaksono, 2011). *Packet loss* dianalisis berdasarkan berapa banyak paket yang hilang atau gagal mencapai tujuan pada waktu paket sedang berjalan. Kemudian perhitungan paket yang hilang dilakukan dengan persamaan 3 (Khalid, 2010) dalam satuan *byte* dan persamaan 4 dalam bentuk prosentase (Riadi & Wicaksono, 2011)

Packet loss = Jumlah paket dikirim – jumlah paket diterima (2.3)

Packet loss = (Pd/Ps) x 100% (2.4)

dimana :

Pd = Jumlah paket yang mengalami *drop* (paket)

Ps = Jumlah paket yang dikirim (paket)

Menurut *Telecommunications and Internet Protocol Harmonization Over Networks* (TIPHON), *paket loss* dapat dikategorikan menjadi empat. Kategori sangat bagus dengan nilai *packet loss* 0%, kategori bagus dengan nilai *packet loss* 3%, kategori sedang dengan nilai *packet loss* 15% dan kategori jelek dengan nilai *packet loss* 25%.

2.4.3 *Delay*

Delay atau *Latency* adalah apabila mengirimkan data sebesar 3 MB pada saat jaringan sepi waktunya 5 menit tetapi pada saat ramai sampai 15 menit, hal ini disebut *latency*. *Latency* pada saat jaringan sibuk berkisar 50-70 ms (Riadi & Wicaksono, 2011). *Latency* dianalisis berdasarkan berapa waktu tunda dari paket

yang diterima sampai tujuan dari masing-masing protokol yang dibandingkan. Perhitungan *delay* menggunakan Persamaan 5 (Khalid, 2010) berikut:

$$\textit{Delay}\ (t) = (Tr - Ts)\ \text{detik} \tag{2.5}$$

dimana:

Tr = Waktu penerimaan paket (detik)

Ts = Waktu pengiriman paket (detik)

Menurut *Telecommunications and Internet Protocol Harmonization Over Networks* (TIPHON), *delay* dapat dikategorikan menjadi empat. Kategori sangat bagus dengan nilai *delay* < 150 ms kategori bagus dengan nilai *delay* 150 s/d 300 ms, kategori sedang dengan nilai *delay* 300 s/d 450 ms dan kategori jelek dengan nilai *delay* > 450 ms.

2.5 *Fairness*

Pengukuran *fairness* digunakan pada jaringan komputer untuk menentukan apakah *users* atau aplikasi telah menerima sumber daya yang adil. Sebuah metric yang digunakan secara umum untuk menaksir *fairness* adalah Jain's *Fairness* Index (JFI) dengan persamaan 6 (Bhatti, 2008):

$$J(t) = \frac{(\sum_{n=1}^{N} rn(t))^2}{N \sum_{n=1}^{N} rn(t)^2} \tag{2.6}$$

Dimana $1/n \leq J \leq 1$, N adalah jumlah aliran, rn adalah nilai kelengkapan yang ditaksir aliran yaitu nilai *throughput* yang diukur. J = 1 berarti ada keseimbangan (*fairness*) pada semua aliran. J = 1/n menunjukkan tidak ada *fairness* (Bhatti, 2008).

2.6 *Network Simulator 2*

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 NS di dukung oleh DARPA (*Defense Advance 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 USC/ISI (*University of Southern California School of Engineering/Information Science Institute*). Tim gabungan ini membangun sebuah perangkat lunak simulasi jaringan internet untuk kepentingan riset interaksi antar protokol dalam konteks pengembangan protokol internet pada saat ini dan masa yang akan datang. (Indarto, 2004)

2.6.1 Konsep Dasar *Network Simulator 2*

Network Simulator dibangun dengan menggunakan dua bahasa pemrograman, yaitu C++ dan Tcl/OTcl. C++ digunakan untuk *library* yang berisi *event scheduler*, protokol dan *network component* yang diimplementasikan pada simulasi oleh *user*. Tcl/OTcl digunakan pada skrip simulasi yang ditulis oleh NS *user* dan pada *library* sebagai simulator objek.

Bahasa C++ digunakan pada *library* karena C++ mampu mendukung *runtime* simulasi yang cepat, meskipun simulasi melibatkan simulasi jumlah paket dan sumber data dalam jumlah besar.

Bahasa Tcl memberikan respon *runtime* yang lebih lambat daripada C++, namun jika terdapat kesalahan, respon Tcl terhadap kesalahan *syntax* dan perubahan *script* berlangsung dengan cepat dan interaktif. User dapat mengetahui

letak kesalahannya yang dijelaskan pada *console*, sehingga user dapat memperbaiki dengan cepat. Karena alasan itulah bahasa ini dipilih untuk digunakan pada skripsi simulasi.

2.6.2 Cara Membuat dan Menjalankan Skrip di NS

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

Contoh:

```
simulasi.tcl
```

Untuk menjalankan simulasi yang telah dibuat, *user* tinggal masuk ke folder tersebut dan mengaktifkan NS serta nama file tcl simulasi yang akan dijalankan.

Contoh:

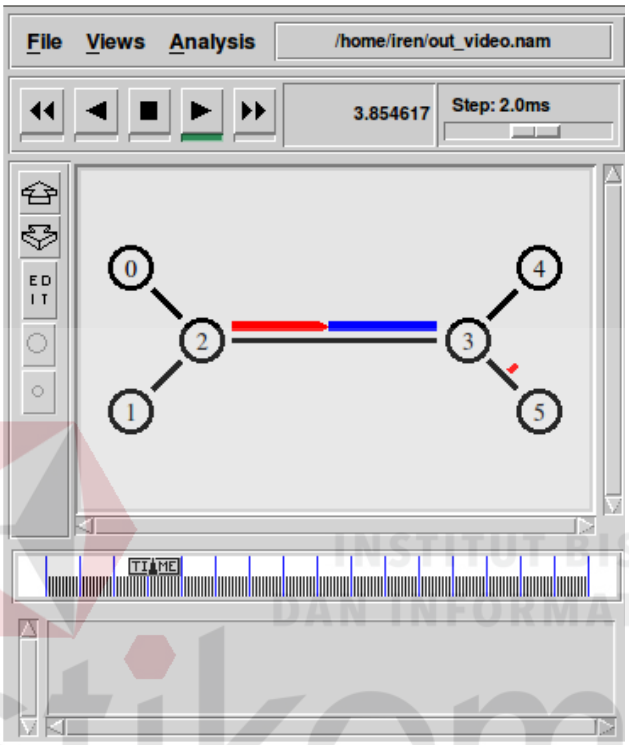
```
[root @ accessnet your_folder]# ns simulasi.tcl
```


2.6.3 *Output* NS-2

Pada saat satu simulasi berakhir, NS membuat satu atau lebih file *output text based* yang berisi detail simulasi jika dideklarasikan pada saat membangun simulasi. Ada dua jenis *output* NS, yaitu: *file trace* yang akan digunakan untuk analisis numerik dan simulasi yang disebut *network animator* (nam).

1. *Network Animator* (NAM)

NAM memiliki *interface* grafis yang mirip dengan CD *player* (play, *rewind*, *fast forward*, *pause*, dan lain-lain) dan juga memiliki kontroler kecepatan tampilan simulasi. Lihat Gambar 2.5.



Gambar 2.5. *Network Animator* (NAM)

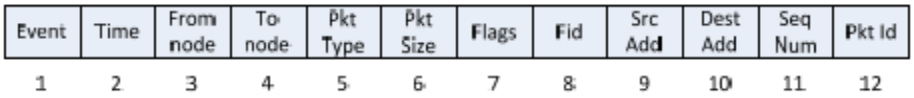
2. *Trace File*

Trace file merupakan isi dari out.tr (Gambar 2.6.) *Trace file* berisi data numerik yang akan digunakan untuk analisa simulasi.

```
+ 0.110419 1 2 tcp 1040 ------- 2 1.0 4.0 5 12
+ 0.110419 1 2 tcp 1040 ------- 2 1.0 4.0 6 13
- 0.110431 1 2 tcp 1040 ------- 2 1.0 4.0 5 12
- 0.110514 1 2 tcp 1040 ------- 2 1.0 4.0 6 13
r 0.11308 0 2 cbr 1000 ------- 1 0.0 3.0 2 8
```

Gambar 2.6. Contoh Isi dari *Trace File* (Teerawat Issariyakul, 2011)

Masing-masing baris pada *trace file* terdiri dari 12 kolom. Format umum masing-masing baris *trace file* ditunjukkan pada Gambar 2.7.



Gambar 2.7. Format Kolom Masing-Masing Baris *TraceFile* (Teerawat Issariyakul, 2011)

1. Kolom pertama adalah *event*. Kolom ini merupakan salah satu dari kemungkinan simbol yang berasosiasi dengan 4 macam *event*, yaitu:
   a. "r" merupakan simbol dari *receive* berarti bahwa paket telah sampai ke bagian output dari sebuah *link*.
   b. "+" merupakan simbol bahwa paket masuk ke proses antrian.
   c. "-" merupakan simbol bahwa paket keluar dari proses antrian.
   d. "d" merupakan simbol bahwa paket tersebut dibuang.
2. Kolom kedua menunjuk pada waktu saat sebuah *event* terjadi.
3. Kolom ketiga menunjuk pada *node input* dari sebuah *link* saat sebuah *event* terjadi.
4. Kolom keempat menunjuk pada *node output* dari sebuah *link* saat sebuah *event* terjadi.
5. Kolom kelima menunjuk pada tipe dari paket.
6. Kolom keenam menunjuk pada ukuran dari paket.
7. Kolom ketujuh menunjuk pada beberapa *flag*.
8. Kolom kedepalan adalah *flow id.*
9. Kolom kesembilan adalah alamat sumber dalam bentuk *node port*.
10. Kolom kesepuluh adalah alamat tujuan dalam bentuk *node port*.
11. Kolom kesebelas adalah *sequence number* dari paket data.
12. Kolom keduabelas adalah id unik dari paket.