

BAB 4 PERANCANGAN DAN IMPLEMENTASI

Dalam bab perancangan dan implementasi ini akan dijelaskan rancangan simulasi untuk mencapai tujuan dari penelitian. Setelah proses perancangan selesai dilakukan, dilanjutkan dengan implementasi dari penelitian ini.

4.1 Perancangan

Pada sub bab ini akan dijelaskan secara detail tentang perancangan pada penelitian ini. Tahap perancangan dimulai dari analisis kebutuhan, gambaran umum simulasi, alur kerja simulasi dilanjutkan dengan perancangan simulasi, perancangan topologi, perancangan TCP Vegas, perancangan TCP New Reno, Perancangan *Random Early Detection* dan Perancangan *Droptail*.

4.1.1 Analisis Kebutuhan

Dalam rangka meneliti perbandingan kinerja antara TCP Vegas dan TCP New Reno menggunakan manajemen antrian *Random Early Detection* dan *Droptail*, dibutuhkan kebutuhan yang dapat menunjang terciptanya lingkungan simulasi. Terdapat dua kebutuhan yaitu kebutuhan simulasi dan kebutuhan jaringan.

4.1.1.1 Kebutuhan Simulasi

Kebutuhan simulasi yaitu kebutuhan yang digunakan sebagai tempat untuk merancang dan mensimulasikan penelitian ini. Kebutuhan simulasi dibagi menjadi dua yaitu perangkat keras dan perangkat lunak.

1. Kebutuhan Perangkat Keras

Perangkat keras yang digunakan dalam penelitian ini, memiliki spesifikasi yang cukup menunjang dalam mengerjakan penelitian ini.

1. Satu laptop Lenovo G40-70 dengan spesifikasi sebagai berikut:
 - *Processor* : Intel I3-4010U, CPU @ 1.70GHz
 - *RAM* : 4 GB
 - *Harddisk* : 500 GB

2. Kebutuhan Perangkat Lunak

Perangkat lunak yang digunakan dalam penelitian ini, memiliki fungsi untuk merancang serta mensimulasikan penelitian yang akan dilakukan.

1. Sistem Operasi Ubuntu 16.04 64 bit
Merupakan salah satu sistem operasi dan bersifat *open source*.
2. *Network Simulator* 2.35
Merupakan salah satu perangkat lunak yang digunakan untuk mensimulasikan jaringan.

3. Network Animator (NAM)

Merupakan salah satu *tool* di *Network Simulator 2.35* yang digunakan untuk memvisualisasikan simulasi dari penelitian yang dibuat.

4.1.1.2 Kebutuhan Jaringan

1. Kebutuhan Router

Dalam penelitian ini *router* yang digunakan berjumlah 11. Dengan banyaknya *router*, semakin banyak *link* yang terhubung dan hal tersebut menyebabkan proses *routing* menjadi semakin kompleks dan juga ketika terjadi penumpukan data pada satu atau beberapa *router* akan terjadi *packet drop* yang bertujuan untuk menghindari kelumpuhan jaringan.

2. Kebutuhan Node Sumber

Dalam penelitian ini untuk menciptakan kondisi *congestion* dalam jaringan maka diterapkan 18 *node* sumber. Digunakannya 18 *node* sumber bertujuan untuk meningkatkan *traffic* data supaya terjadi penumpukan paket data yang menyebabkan *congestion* pada jaringan.

3. Kebutuhan Node Tujuan

Dalam penelitian ini terdapat 2 *node* tujuan yang dimaksudkan agar semua *node* sumber ketika mengirimkan data akan terjadi peningkatan *traffic* data yang terpusat pada antrian yang menuju *node* tujuan sehingga akan menyebabkan *congestion* pada jaringan.

4. Kebutuhan Link

Dalam penelitian ini *link* berperan sebagai kabel (*wired*) yang menghubungkan semua *node* yang ada dalam topologi jaringan. *Bandwidth* yang diterapkan pada *link* yaitu 10 Mbps dengan *delay* 2 ms.

5. Kebutuhan Buffer

Dalam penelitian ini, *buffer* digunakan untuk menampung paket data yang dikirimkan oleh *node* sumber. Banyaknya *buffer* dapat dihitung dengan melihat banyaknya aliran data dalam jaringan, besar kapasitas *bandwidth* dan nilai dari *propagation delay* (Appenzeller, 2005). Gunakan persamaan 4.1 untuk mencari *buffer*.

$$Buffer = \frac{2 \text{ way propagation delay} \times \text{kapasitas bandwidth}}{\sqrt{\text{jumlah aliran data (node sumber)}}} \quad (4.1)$$

Jadi,

$$Buffer = \frac{8 \times 10}{\sqrt{18}} = \frac{80}{\sqrt{4.24}} = 18.94, \text{ diasumsikan menjadi } 20 \text{ Mb paket}$$

Dari persamaan diatas, di dapat *buffer* yaitu 20 Mb paket.

4.1.2 Alur Kerja Simulasi

Alur kerja yang terjadi mulai awal menentukan varian TCP dan manajemen antrian lalu proses pengiriman data sampai terjadi *congestion* yaitu sebagai berikut:

1. Menentukan varian TCP, pada penelitian ini menggunakan TCP Vegas atau TCP New Reno. Kedua TCP tersebut akan digunakan secara bergantian setiap kali pengujian.
2. Menentukan manajemen antrian, pada penelitian ini menggunakan *Random Early Detection* dan *Droptail*. Kedua antrian tersebut akan digunakan secara bergantian setiap kali pengujian
3. Menentukan *traffic source* (FTP), alamat dari *node* tujuan dan besar paket data yang akan dikirimkan (1024 bytes).
4. Selanjutnya pengiriman data dilakukan, dimana setiap *node* sumber akan mengirimkan paket data secara bertahap dengan selang waktu 0.2 detik dengan lama waktu simulasi yaitu 300 detik.
5. Selanjutnya semua paket data dari setiap *node* sumber tersebut akan melewati *router* untuk diteruskan ke *node* tujuannya masing-masing.
6. Selanjutnya ketika semua *node* sumber sudah mengirimkan paket data secara bersamaan maka akan terjadi peningkatan *traffic* data sehingga terjadi penumpukan data pada *buffer* yang menyebabkan *congestion*.
7. Selanjutnya setelah terdeteksi *congestion* agar tidak terjadi penurunan kinerja maka mekanisme varian TCP dan mekanisme manajemen antrian dalam mengatasi *congestion* akan bekerja.

4.1.3 Perancangan Simulasi

Perancangan simulasi ini akan menjelaskan parameter yang akan digunakan. Parameter pada penelitian ini digunakan sebagai nilai yang berfungsi untuk melakukan proses komputasi saat simulasi berjalan. Nilai parameter yang ada didapat dari perhitungan serta referensi yang ada.

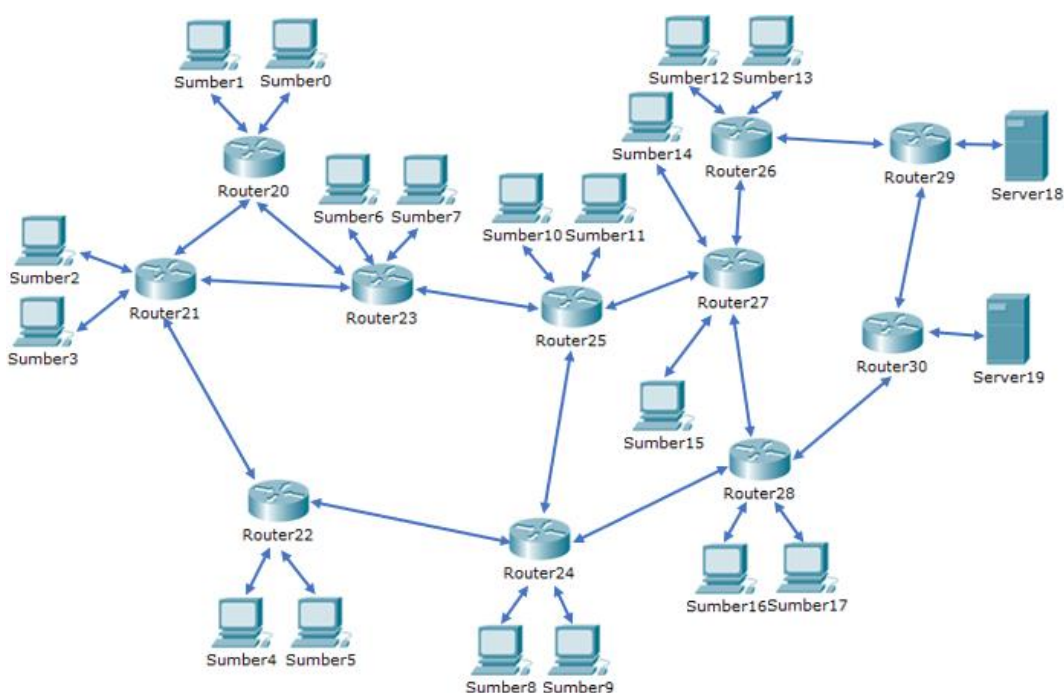
Tabel 4.1 Parameter Simulasi

Parameter Uji	Nilai			
Varian TCP	TCP Vegas	TCP Vegas	TCP New Reno	TCP New Reno
Antrian pada router	<i>Droptail</i>	<i>Random Early Detection</i>	<i>Droptail</i>	<i>Random Early Detection</i>
Buffer Size	20, 30, 40, 50 dan 60 paket	60 paket	20, 30, 40, 50 dan 60 paket	60 paket
Nilai min thresh & max thresh	-	1. <i>Min thresh</i> : 20, 25, 30, 35 dan 40 paket. <i>Max thresh</i> : 50 paket	-	1. <i>Min thresh</i> : 20, 25, 30, 35 dan 40 paket. <i>Max thresh</i> : 50 paket

		2. <i>Min thresh</i> : 20paket. <i>Max thresh</i> : 30, 35, 40, 45 dan 50 paket		2. <i>Min thresh</i> : 20paket. <i>Max thresh</i> : 30, 35, 40, 45 dan 50 paket
Jumlah node	31 node (18 node sumber, 11 router, 2 node tujuan)			
Waktu simulasi	300 detik			
Traffic source	FTP			
Ukuran Paket	1024 bytes			
Delay	2 ms			
Bandwidth	10 Mbps			

4.1.4 Perancangan Topologi





Rancangan topologi pada simulasi ini menggunakan model *Abilene*. Dimana akan terdapat 18 *node* sumber, 11 *router* dan 2 *node* tujuan. *Node-node* dalam topologi akan berkomunikasi secara kabel (*wired*) sehingga membentuk sebuah jaringan. Gambar 4.1 merupakan topologi *Abilene* yang nanti digunakan dalam penelitian ini.



Gambar 4.1 Topologi *Abilene* Pada NS-2

Dari Gambar 4.1 dapat dijelaskan bahwa sumber0 sampai sumber17 merupakan *node* sumber. Selanjutnya *server18* dan *server19* merupakan *node* tujuan. Terakhir *router20* sampai *router30* merupakan *router*. Keterangan dari setiap *node* pada gambar 4.1 dapat dilihat pada tabel 4.1.

Tabel 4.2 Keterangan Node

No	Gambar Node	Keterangan
1		Gambar no 1 mendefinisikan <i>router</i> . Di dalam <i>router</i> diterapkan antrian <i>Random Early Detection</i> dengan skema penambahan <i>min thresh</i> dan <i>max thresh</i> atau <i>Droptail</i> dengan skema penambahan kapasitas <i>buffer</i> .
2		Gambar no 2 mendefinisikan <i>node</i> sumber. Di dalam <i>node</i> sumber diterapkan varian TCP (Vegas atau New Reno), dengan <i>traffic source</i> FTP dan besar paket 1024 bytes.
3		Gambar no 3 mendefinisikan <i>node</i> tujuan. Di dalam <i>node</i> tujuan diterapkan TCP Sink yang bertujuan mengirimkan <i>acknowledgement</i> ketika ada paket yang diterima.
4		Gambar no 4 mendefinisikan <i>link</i> . <i>Link</i> tersebut bersifat dua arah atau <i>duplex link</i> yang menghubungkan <i>node</i> sumber, <i>router</i> dan <i>node</i> tujuan. Di dalam <i>link</i> tersebut diterapkan <i>bandwidth</i> 10 Mbps dengan <i>delay</i> 2 ms.

4.1.5 Perancangan TCP Vegas

Perancangan TCP Vegas akan menjelaskan bagaimana mekanisme proses pengiriman data dari sumber ke tujuan ketika dalam jaringan terjadi *congestion*. TCP Vegas ini akan diterapkan pada setiap *node* sumber dengan *traffic source* yang digunakan yaitu FTP dengan besar data 1024 bytes. Setelah proses pengiriman data dimulai, TCP Vegas akan mendeteksi *congestion* dari perubahan atau varian RTT yang dihasilkan oleh paket yang dikirimkan ke tujuan. Jika perubahan RTT kecil, maka jaringan dianggap normal sehingga pengiriman data kan di tambah. Jika perubahan RTT besar, maka jaringan dianggap mengalami *congestion* sehingga pengiriman data kan dikurangi.

4.1.6 Perancangan TCP New Reno

Perancangan TCP New Reno akan menjelaskan bagaimana mekanisme proses pengiriman data dari sumber ke tujuan ketika dalam jaringan terjadi *congestion*. TCP New Reno ini akan diterapkan pada setiap *node* sumber dengan *traffic source* yang digunakan yaitu FTP dengan besar data 1024 bytes. Setelah proses pengiriman data dimulai, TCP New Reno akan mendeteksi *congestion* dari *packet loss* yang terjadi selama proses pengiriman berlangsung. Jika terdapat *packet loss* maka *congestion* terdeteksi dan kecepatan pengiriman data akan dikurangi setengah.

4.1.7 Perancangan *Random Early Detection*

Perancangan *Random Early Detection* akan menjelaskan bagaimana mekanisme dalam melayani data yang masuk ke dalam antrian sampai terjadi

kondisi dimana antrian diharuskan melakukan *packet drop*. *Random Early Detection* akan diterapkan pada setiap antrian yang ada. Dimana *Random Early Detection* ini akan melakukan manajemen paket ketika terjadi penumpukan data pada antrian dengan menggunakan dua parameter yaitu *min thresh* dan *max thresh*. Jika data < *min thresh*, maka data akan dilayani lalu ditransmisikan. Jika *min thresh* < data < *max thresh*, maka data ditandai untuk di *drop* secara *random*, dan jika data > *max thresh*, maka data langsung di *drop*.

4.1.8 Perancangan *Droptail*

Perancangan *Droptail* akan menjelaskan bagaimana mekanisme dalam melayani data yang masuk ke dalam antrian sampai terjadi kondisi dimana antrian diharuskan melakukan *packet drop*. Dimana *Droptail* ini akan melakukan manajemen paket ketika terjadi penumpukan data pada antrian dengan menggunakan mekanisme *first in first out* (FIFO). Dimana data yang masuk kedalam antrian pertama kali akan dilayani dan di transmisikan pertama. Namun ketika antrian penuh data yang akan masuk akan di *drop*.

4.2 Implementasi

Pada sub bab ini akan dijelaskan secara detail tentang implementasi. Tahap implementasi dimulai dari instalasi NS-2.35 dan dilanjutkan dengan konfigurasi *script* simulasi yang bertujuan untuk membangun lingkungan simulasi.

4.2.1 Instalasi *Network Simulator 2.35*

Proses instalasi NS-2.35 berjalan lancar tergantung spesifikasi perangkat keras yang digunakan. Proses lengkapnya dapat dijelaskan sebagai berikut:

1. *Download file* NS-2.35 menggunakan terminal dengan perintah berikut.

```
wget http://sourceforge.net/projects/nsnam/files/allinone/nsallinone-2.35/ns-allinone-2.35.tar.gz
```

2. Ekstrak *file* NS-2.35 yang telah di *download* dengan mengetik perintah berikut.

```
tar zxvf ns-allinone-2.35.tar.gz
```

3. Melakukan pembaharuan sistem pada ubuntu, dengan mengetik perintah berikut.

```
sudo apt-get update
```

4. Melakukan instalasi komponen yang berhubungan dengan NS-2.35, dengan mengetik perintah berikut.

```
sudo apt-get build-essential autoconf automake libxmu-dev
```

5. Melakukan instalasi NS-2.35, dengan mengetik perintah berikut.

```
./install
```

6. Setelah proses instalasi selesai, buka *file* *.bashrc* dengan mengetik perintah berikut.

```
gedit ~/.bashrc
```

7. Pada file `.bashrc`, pada baris paling bawah tambahkan `PATH` seperti berikut.

```
# LD_LIBRARY_PATH
OTCL_LIB=/home/guntur/ns-allinone-2.35/otcl-1.13
NS2_LIB=/home/guntur/ns-allinone-2.35/lib
X11_LIB=/usr/X11R6/lib
USR_LOCAL_LIB=/usr/local/lib
Export
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$OTCL_LIB:$NS2_
LIB:$X11_LIB:$USR_LOCAL_LIB
#TCL_LIBRARY
TCL_LIB=/home/guntur/ns-allinone-2.35/tcl8.4.18/library
USR_LIB=/usr/lib
Export TCL_LIBRARY=$TCL_LIB:$USR_LIB
#PATH
XGRAPH=/home/guntur/ns-allinone-2.35/bin:/your/path/ns-allinone-
2.35/tcl8.4.18/unix:/home/guntur/ns-allinone-2.35/tk8.4.18/unix
NS=/home/guntur/ns-allinone-2.35/ns-2.35/
NAM=/home/guntur/ns-allinone-2.35/nam-1.14/
PATH=$PATH:$XGRAPH:$NS:$NAM
```

8. Menguji NS-2.35 apakah sudah terinstal dengan benar, dengan perintah berikut.

```
$ ns
```

9. Jika muncul tanda `%` pada terminal maka NS-2.35 siap digunakan.
10. Terakhir lakukan validasi dengan mengetik perintah berikut.

```
cd ns-2.35 kemudian ./validate.
```

Setelah proses instalasi NS-2.35 selesai, dilanjutkan dengan proses konfigurasi `script` untuk membangun lingkungan simulasi pada penelitian ini.

4.2.2 Konfigurasi *Script* Simulasi

Pada sub bab konfigurasi `script` simulasi ini, ada tahapan-tahapan yang perlu dilakukan agar simulasi dapat berjalan pada NS-2.35, yaitu sebagai berikut:

1. Mendefinisikan nama simulator dan memberi warna aliran data.

Mendefinisikan nama simulator dan pemberian warna untuk mempermudah dalam melihat aliran data saat simulasi berlangsung.

Tabel 4.3 Definisi Nama Simulator Dan Warna Aliran Data

No	Script
1	set ns [new Simulator]
2	\$ns color 1 Blue

Penjelasan :

Baris 1 : Mendefinisikan variabel sebagai nama dari kelas simulator.

Baris 2 : Memberikan warna pada aliran data.

2. Mendefinisikan *trace file* dan NAM

Mendefinisikan *file trace* dan *file nam* untuk menyimpan data hasil simulasi dan data hasil visualisasi.

Tabel 4.4 Definisi File Trace Dan File Nam

No	Script
1	set file1 [open newreno_buffer_15.tr w]
2	\$ns trace-all \$file1
3	set file2 [open newreno_buffer_15.nam w]
4	\$ns namtrace-all \$file2

Penjelasan :

Baris 1-2 : Mendefinisikan nama *file1* dan menyimpan data hasil simulasi pada *file newreno-dt-buf-20.tr*.

Baris 2-3 : Mendefinisikan nama *file2* dan menyimpan data hasil visualisasi pada *file newreno-dt-buf-20.nam*.

3. Pembuatan dan pemberian warna *node*

Pembuatan *node* digunakan untuk membentuk topologi yang telah dirancang. Sedangkan pemberian warna pada *node* bertujuan untuk membedakan antara *node* sumber, *router* dan *node* tujuan.

Tabel 4.5 Pembuatan Dan Pemberian Warna Node

No	Script
1	set n0 [\$ns node]
2	set n1 [\$ns node]
3	\$n0 color Blue
4	\$n1 color Red

Penjelasan :

Baris 1-2 : Mendefinisikan variabel *node*.

Baris 3-4 : Memberikan warna kepada setiap *node*.

4. Melakukan *setting* koneksi antar *node*

Setting koneksi antar *node* digunakan untuk menentukan arah aliran data dalam topologi, serta memberikan nilai *bandwidth* dan *delay* pada *link* dan juga menerapkan manajemen antrian pada *router*.

Tabel 4.6 Setting Koneksi Antar Node

No	Script
1	\$ns duplex-link \$n0 \$n1 10Mb 2ms DropTail/RED
2	\$ns queue-limit \$n2 \$n3 20

Penjelasan :

Baris 1 : Membuat koneksi untuk menghubungkan tiap *node*. Digunakan *duplex-link* untuk komunikasi dua arah. Juga mendefinikan *bandwidth* serta *delay* dalam *link*. Dan mendefinisikan antrian yang digunakan pada *router*.

Baris 2 : Membuat batas antrian pada *router*.

5. Melakukan *setting* antrian *Random Early detection*

Setting Random Early Detection berfungsi untuk menerapkan parameter yang ada, agar antrian dapat berjalan sesuai yang diharapkan.

Tabel 4.7 Setting *Random Early Detection*

No	Script
1	Queue /RED set bytes_ false
2	Queue /RED set queue_in_bytes_ false
3	Queue /RED set q_weight_ 0.002
4	Queue /RED set thresh_ 15
5	Queue /RED maxthresh_ 60

Penjelasan :

Baris 1- 2 : Menunjukkan perhitungan rata-rata ukuran *queue* dalam *bytes* dan *false* menunjukkan rata-rata ukuran *queue* akan dihitung dalam paket (bukan dalam *bytes*).

Baris 3 : Menunjukan rata-rata pergerakan eksponensial.

Baris 4 -5 : Mendefinisikan parameter *min* dan *max thresh*.

6. Pembentukan koneksi dan pembuatan aliran trafik

Pembentukan koneksi dan pembuatan aliran *traffic* data bertujuan untuk menyebabkan terjadinya pengiriman data dan penerimaan data dari *node* sumber ke *node* tujuan.

Tabel 4.8 Pembentukan Koneksi Dan Aliran Trafik

No	Script
1	set tcp [new Agent/TCP/(Vegas atau Newreno)]
2	\$ns attach-agent \$n0 \$tcp
3	set sink [new Agent/TCPSink]
4	\$ns attach-agent \$n4 \$sink
5	\$ns connect \$tcp \$sink
6	\$tcp set packetSize_ 1024
7	\$tcp set fid_ 1
8	set ftp [new Traffic source/FTP]
9	\$ftp attach-agent \$tcp
10	\$ftp set type_ FTP
11	\$ns at 0.2 "\$ftp start"
12	\$ns at 300.0 "\$ftp stop"

Penjelasan :

Baris 1 : Mendefinisikan TCP sumber (Vegas atau Newreno).

Baris 2 : Menggabungkan TCP dengan *node 0*.

Baris 3 : Mendefinisikan TCP tujuan (TCP sink).

Baris 4 : menggabungkan *node 1* dengan tcp sink.

Baris 5 : Membuat sumber dan tujuan tersambung.

Baris 6 : Mendefinisikan ukuran paket TCP (*bytes*)

Baris 7 : Memberikan identitas kepada aplikasi TCP.

Baris 8 : Mendefinisikan aplikasi yang digunakan (FTP)

Baris 9 : Menggabungkan FTP dengan TCP.

Baris 10 : Menentukan jenis aplikasi (FTP).

Baris 11-12: Mengatur jadwal *start & stop* untuk koneksi FTP.

7. Mengakhiri simulasi

Mengakhiri simulasi merupakan jadwal yang ditentukan. Pengaturan jadwal dilakukan dengan menetapkan waktu berhenti sebagai batas waktu simulasi.

Tabel 4.9 *Setting* Untuk Mengakhiri Simulasi

No	<i>Script</i>
1	proc finish {} {
2	global ns file1 file2
3	\$ns flush-trace
4	close \$file1
5	close \$file2
6	puts "running nam..."
7	exec nam newreno_buffer_15.nam &
8	exit 0
9	}
10	\$ns at 300.0 "finish"
11	\$ns run

Penjelasan :

Baris 1-2 : Mendeklarasikan prosedur *finish* pada simulasi.

Baris 3-5 : Menyimpan semua data hasil simulasi dalam *file1* dan *file2*.

Baris 6-7 : Melakukan eksekusi *file* nam.

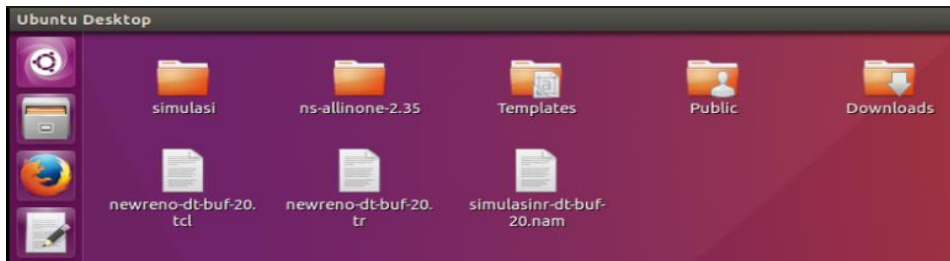
Baris 8-9 : Mengakhiri aplikasi dan mengembalikan status dengan angka 0 pada sistem.

Baris 10-11: Mendefinisikan waktu eksekusi dan menjalankan simulasi.

Untuk memastikan apakah lingkungan simulasi sudah dapat berjalan atau masih mengalami *error*, dengan mengetikkan perintah pada terminal sebagai berikut.

```
$ ns nama_file.tcl
```

Jika masih terjadi *error* maka harus diulangi lagi proses konfigurasi *script*. Jika berhasil, maka di dalam *folder home* akan muncul *file trace* dan *file nam*. Seperti di tunjukkan pada Gambar 4.3.



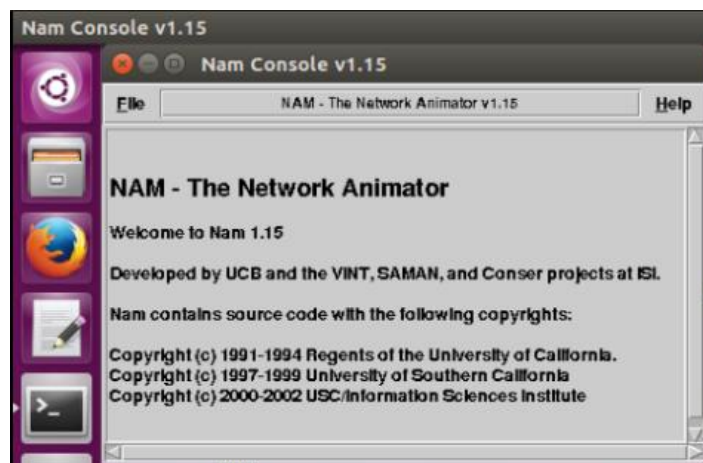
Gambar 4.2 File Trace dan File Nam

Gambar 4.3 memperlihatkan *file trace* (.tr) dan *file nam* (.nam) hasil dari menjalankan *script* simulasi. *File nam* yang berfungsi untuk memvisualisasikan topologi dan *file trace* yang mencatat seluruh hasil dari simulasi yang terjadi ketika topologi dijalankan.

Selanjutnya yaitu mencoba menjalankan *file nam*, dengan mengetik perintah berikut pada terminal.

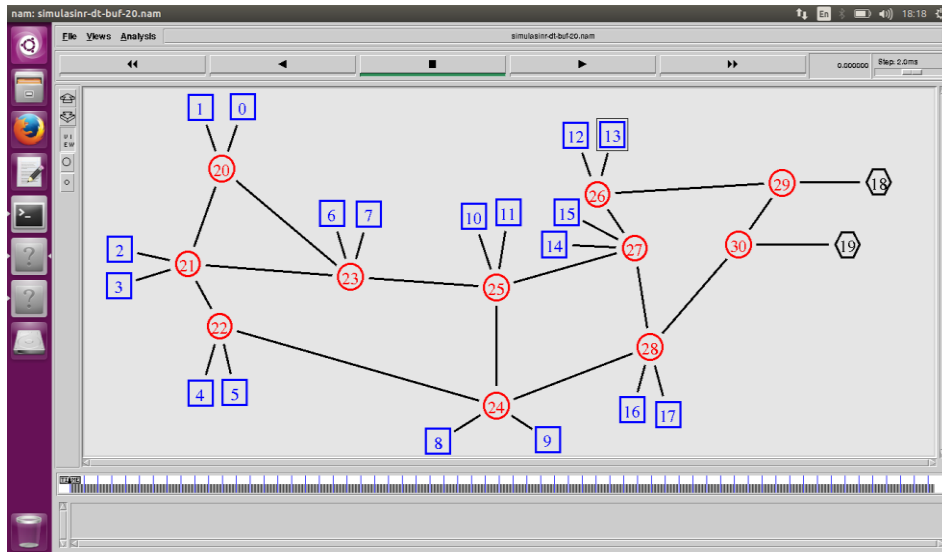
```
$ nam simulasinr-dt-buf-20.nam
```

Hasilnya yaitu akan muncul *Network Animator* (nam), seperti terlihat pada Gambar 4.3.



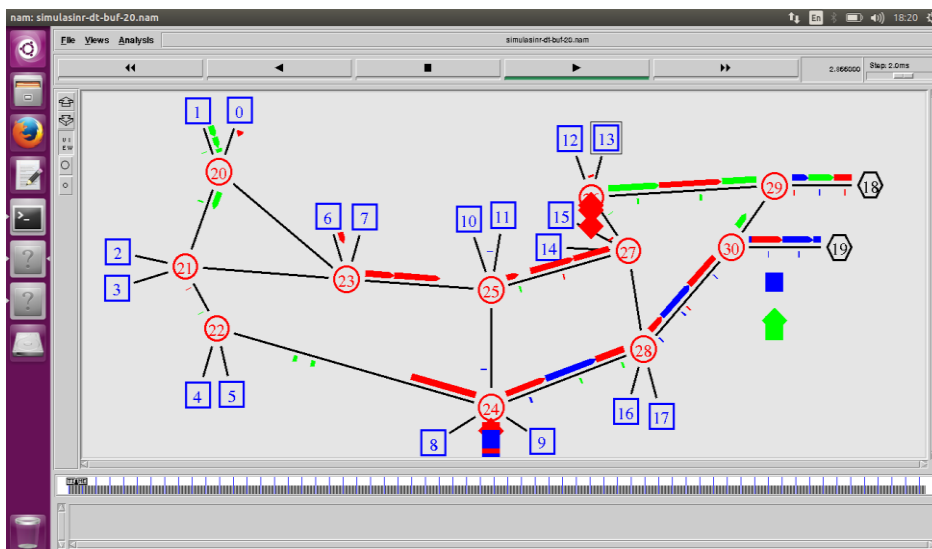
Gambar 4.3 Network Animator

Selain *Network Animator*, visualisasi dari topologi yang telah dibuat juga akan muncul, seperti terlihat pada Gambar 4.4.



Gambar 4.4 Tampilan Topologi Pada *Network Animator*

Terakhir, mencoba melakukan pengujian terhadap topologi yang telah dibuat dengan menekan tombol *start*, seperti yang ditunjukkan pada Gambar 4.5.



Gambar 4.5 Saat Topologi Dijalankan

Gambar 4.5 menjelaskan bahwa topologi, koneksi antar *node*, aliran paket data antar *node* dan manajemen antrian pada *router* untuk melakukan *packet drop* telah berhasil di implementasikan. Hal tersebut dapat dibuktikan dengan munculnya visualisasi topologi pada *Network Animator*, dilanjutkan dengan terlihatnya aliran data dari *node* sumber melalui *router* yang menuju *node* tujuan yang buktinya dapat dilihat pada Gambar 4.7 dan terjadinya *packet drop* pada beberapa *router* saat terjadinya pengiriman paket data. Bukti diatas menjelaskan bahwa konfigurasi *script* simulasi telah berhasil diimplementasikan dengan benar.

```

s vegas-red-max-45.tcl x pdr_pd.awk x delay.awk x
+ 0.2 0 20 tcp 1024 ----- 1 0.0 18.0 0 0
- 0.2 0 20 tcp 1024 ----- 1 0.0 18.0 0 0
r 0.202819 0 20 tcp 1024 ----- 1 0.0 18.0 0 0|
+ 0.202819 20 23 tcp 1024 ----- 1 0.0 18.0 0 0
- 0.202819 20 23 tcp 1024 ----- 1 0.0 18.0 0 0
r 0.205638 20 23 tcp 1024 ----- 1 0.0 18.0 0 0
+ 0.205638 23 25 tcp 1024 ----- 1 0.0 18.0 0 0
- 0.205638 23 25 tcp 1024 ----- 1 0.0 18.0 0 0
r 0.208458 23 25 tcp 1024 ----- 1 0.0 18.0 0 0
+ 0.208458 25 27 tcp 1024 ----- 1 0.0 18.0 0 0
- 0.208458 25 27 tcp 1024 ----- 1 0.0 18.0 0 0
r 0.211277 25 27 tcp 1024 ----- 1 0.0 18.0 0 0
+ 0.211277 27 26 tcp 1024 ----- 1 0.0 18.0 0 0
- 0.211277 27 26 tcp 1024 ----- 1 0.0 18.0 0 0
r 0.214096 27 26 tcp 1024 ----- 1 0.0 18.0 0 0
+ 0.214096 26 29 tcp 1024 ----- 1 0.0 18.0 0 0
- 0.214096 26 29 tcp 1024 ----- 1 0.0 18.0 0 0
r 0.216915 26 29 tcp 1024 ----- 1 0.0 18.0 0 0
+ 0.216915 29 18 tcp 1024 ----- 1 0.0 18.0 0 0
- 0.216915 29 18 tcp 1024 ----- 1 0.0 18.0 0 0
r 0.219734 29 18 tcp 1024 ----- 1 0.0 18.0 0 0

```

Gambar 4.6 Bukti aliran pengiriman data berhasil

Gambar 4.7 menjelaskan aliran data dari *node* 0 sebagai *node* sumber menuju ke *node* 18 sebagai *node* tujuan berhasil mengalir dengan benar dengan melewati *node* 20 dilanjutkan ke *node* 23 lalu ke *node* 25 lalu ke *node* 27 lalu ke *node* 26 lalu ke *node* 29 dan sampai ke *node* tujuan yaitu *node* 18.