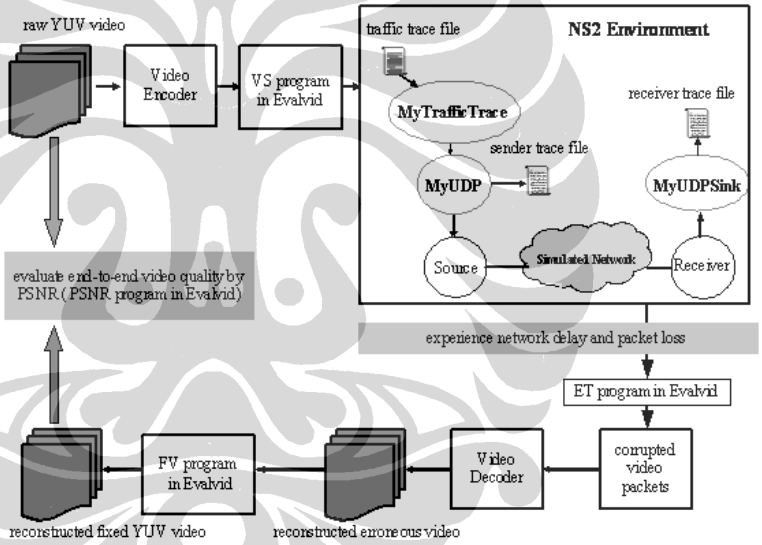
BAB III
PERANCANGAN SIMULASI JARINGAN

Proses implementasi penelitian terdiri dari encoder H.264, *video sender*, *network simulator* (NS-2), H.264 *decoder*, program *evaluate trace* (ET), program *peak signal to noise ratio* (PSNR), dan program *mean opinion score* (MOS). Hubungan antara komponen-komponen yang berbeda, file video *input*, dan file *output* yang di-*generate* dari tool-tool yang digunakan pada penelitian ini, ditunjukkan pada Gambar 3.1 [11]. Metodologi proses siklus implementasi penelitin ini dibangun di dalam *framework* EvalVid, dan diperluas dengan mengikutsertakan NS-2 untuk simulasi pada WiMAX.



Gambar 3.1 Arsitektur Sistem EvalVid [11]

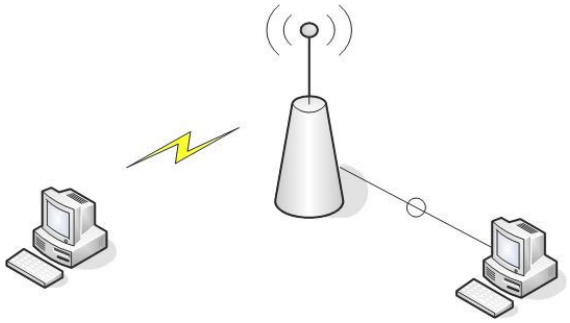
Secara singkat proses keseluruhan implementasi penelitian adalah sebagai berikut. *Raw video* yang umumnya disimpan dalam format YUV, diberikan pada sebuah encoder H.264 yang kemudian meng-*generate encoded video stream. Encoder 'lencode'* dan *decoder 'ldecode'* yang berbasis *open source* digunakan. *Encoded video stream* kemudian dibaca oleh *Video Sender (VS)* untuk meng-*generate* sebuah file *trace,* yang berisi informasi seperti tipe frame, ukuran frame, jumlah paket, dan waktu pengiriman untuk masing-masing frame video. File *trace* kemudian dimasukkan ke dalam *streaming server* yang dijalankan pada simulator

NS-2 untuk memproduksi v*ideo stream* pada WiMAX. Efek dari *streaming video* pada WiMAX di-*capture* pada sebuah *log* file *streaming client* yang di-*generate* oleh NS-2. File *log* terdiri dari informasi seperti *time stamp,* ukuran dan identitas pada masing-masing paket. Selain itu NS-2 juga meng-*generate* file *log* yang sama untuk *streaming server.* File *trace* dan file *log* digunakan oleh program *evaluate traces (ET)* untuk meng-*generate* kemungkinan file video yang *corrupt* dari hasil transmisi pada WiMAX. File video yang *corrupt* ini merupakan data yang penting untuk digunakan oleh *Peak Signal to Noise Ratio (PSNR)* dan *Mean Opinion Score (MOS)* dalam mengevaluasi kualitas video *end-to-end.*

3.1 Topologi Jaringan

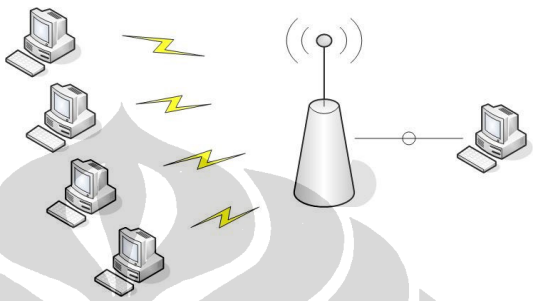
Penelitian ini menggunakan 3 kelas QoS yaitu, UGS, rtPS dan BE. *Subsciber Station* (SS) mengirimkan *video streaming* menggunakan WiMAX ke *base station* (BS), dan kemudian BS akan meneruskannya kepada *receiver* menggunakan *wired*. Simulasi ini menggunakan simulator NS-2 dengan tambahan modul *QoS-included WiMAX* untuk simulasi jaringan, dan EvalVid untuk evaluasi kualitas video.

Penelitian ini menerapkan 2 skenario yaitu pada skenario pertama *user* dari kelas QoS tertentu, mengirimkan *streaming video*. Pada skenario ini akan dilakukan 3 kali percobaan untuk mengetahui QoS mana yang terbaik. Dengan melihat hasil dari *frame loss, throughput, delay*, *jitter* PSNR dan MOS yang didapat dari simulasi ini. Topologi yang digunakan pada skenario ini seperti yang ditunjukkan pada Gambar 3.2



Gambar 3.2 Skenario 1

Pada Skenario kedua, akan mensimulasikan 4 SS, yaitu 1 SS video menggunakan QoS tertentu dan 3 SS data dari masing-masing kelas QoS UGS, rtPS, dan BE. Simulasi ini dilakukan dengan 3 kali percobaan. Topologi yang digunakan pada skenario ini seperti yang ditunjukkan pada Gambar 3.3.



Gambar 3.3 Skenario 2

3.1.1 *Traffic Generator*

Jenis aplikasi yang digunakan pada simulasi ini adalah layanan *video streaming.* Layanan ini menggunakan protokol utama UDP karena menyediakan penyampaian paket tepat pada waktunya. Namun demikian, UDP tidak menjamin sampainya paket dengan baik. Protokol *transport* RTP berjalan diatas UDP, yang melakukan pemaketan dan menyediakan penyampaian frame video berurut. RTCP digunakan oleh klien video untuk memberitahukan server video mengenai kualitas video yang diterima.

Implementasi aplikasi *video streaming* ini pada simulasi NS-2 membutuhkan pembangunan, konfigurasi *agent,* dan proses *attach* pada sebuah source data level aplikasi yang dinamakan *traffic generator. Traffic generator* akan dibangun sesuai dengan karakteristik layanan *video streaming.* Setelah itu, simulasi akan menjalankan *agent* dan *traffic generator.*

Tiga *agent* simulasi yang ditambahkan pada *traffic generator* adalah *MyTrafficTrace, MyUDP* dan *MyUDPSink. Agent-agent* ini akan didesain baik untik membaca file *trace* video atau untuk meng-*generate* data yang dibutuhkan untuk mengevaluasi kualitas video yang diinginkan. Cara kerja dan kegunaan masing-masing *agent* adalah sebagai berikut :

- ***Agent MyTrafficTrace*** meng-*generate* tipe frame dan ukuran frame dari file trace video yang di-*generate* oleh file *trace traffic*, memfragmentasi frame video pada segmen yang lebih kecil, dan mengirim segmen-segmen pada layer UDP yang lebih rendah pada waktu yang baik sesuai dengan konfigurasi user yang ditentukan pada file *script* simulasi.
- ***Agent MyUDP*** adalah *extension* dari *agent* UDP. *Agent* ini mengizinkan user untuk menentukan nama file *output* dari file *trace* pengirim dan merekam *timestamp* pada setiap paket yang ditransmisikan, *packetID,* dan *packet payload size.* Tugas dari *agent MyUDP* serupa dengan tugas beberapa tool seperti *tcp-dump* atau *win-dump* pada environment *real network.*
- ***Agent MyUDPSink*** merupakan *agent* penerima untuk frame video yang terfragmen yang dikirim oleh MyUDP. *Agent* ini juga merekam *timestamp, packet ID,* dan *payload size* dari masing-masing paket yang diterima pada file *trace* penerima user yang telah ditentukan. Setelah simulasi, berdasarkan file *trace* dan video asli yang di-*encode* ini, program ET memproduksi file video yang *corrupt*. Setelah itu, video *corrupt* di-*decode* dan *error* disembunyikan. Akhimya, video *fix* YUV yang terekonstruksi dapat dibandingkan dengan video *raw* YUV untuk mengevaluasi kualitas video *end-to-end* yang dikirimkan.

3.2 Pengolahan *Video Streaming*

Seperti terlihat pada Gambar 3.1 implementasi pengolahan *video streaming*, dimulai dari *raw video source*. Video ini akan diolah sedemikian rupa melalui proses *encoding* dan program *Video Sender (VS)* untuk kemudian dimasukkan ke dalam simulasi NS-2. Setelah itu, hasil yang diperoleh dari simulasi di-*decode* kembali dan dijalankan program *Fixed Video (FV)* sehingga didapat tampilan video pada sisi penerima. Selama pengolahan, data yang diproses pada arus transmisi video akan ditandai dan disimpan pada file-file yang beranekaragam

3.2.1 *Raw Video Source*

File *raw video source* umumnya disimpan di dalam format YUV, yang merupakan format input yang diterima pada banyak *video encoder*. File *raw video source* ini kemudian di-*encode* dan setelah itu file *trace* video diproduksi. File *trace* ini berisi seluruh informasi yang relevan pada modul EvalVid untuk memperoleh hasil yang diinginkan sesuai dengan parameter QoS. File *raw* video ini akan dibuat melalui kompresi *encode* pada 30 fps. Ukuran frame adalah 176xl44 pixel, yang juga dikenal sebagai *Quarter Common Intermediate Format* (QCIF).

Pada prinsipnya, *raw source video* adalah rangkaian dari *raw image* (pixel demi pixel). *Raw image* hanyalah *array* berupa *pixel* 2 dimensi. Masing-masing pixel diberikan 3 nilai warna, yaitu merah, hijau dan biru (RGB). Pada *video coding pixel* tidak diberikan oleh 3 warna dasar, tetapi lebih pada kombinasi dari 1 nilai *luminance* dan 2 nilai c*hrominance*. Representasi masing-masing dapat di-*convert* bolak-balik sehingga tepat sepadan.

Pada sebuah gambar mata manusia lebih sensitif ke komponen *luminance* daripada *chrominance*. Itulah mengapa pada *video coding*, komponen *luminance* dihitung pada tiap *pixel*, sedangkan 2 komponen *chrominance* seringkali rata-rata dihitung pada diatas 4 pixel. Ini membagi dua jumlah data yang ditransmisikan pada setiap *pixel* pada perbandingan dengan skema RGB. Terdapat beberapa kemungkinan yang dikenal sebagai coding YUV.

$$
\begin{aligned}
Y &= 0.299R + 0.587G + 0.114B \\
U &= 0.565(B - Y) \\
V &= 0.713(R - Y)
\end{aligned}
\qquad (3.1)
$$

$$
\begin{aligned}
R &= Y + 1.403V \\
G &= Y - 0.344U - 0.714V \\
B &= Y + 1.770U
\end{aligned}
$$

Proses *decode* pada banyak *decoder video* menghasilkan *raw video file* pada format YUV. *Decoder* MPEG-4 termasuk di dalamnya H.264 menulis file YUV pada format 4:2:0.

3.2.2 *Video Encoder* dan *Video Decoder*

Saat ini EvalVid hanya mendukung *single layer video coding*. EvalVid mampu mendukung berbagai jenis codec MPEG4, yaitu antara lain: National Chiao Twig University (NCTU) codec, ffmpeg, Xvid, dan H.264.

3.2.3 *Video Sender* (VS)

Untuk file video H.264, sebuah *parser* dikembangkan berdasarkan standar video H.264. Ini memungkinkan untuk membaca H.264 apapun yang diproduksi oleh *encoder* yang telah ditetapkan. Tujuan VS adalah untuk men-*generate* file trace dari file video yang telah di-*encode*. File output yang diproduksi oleh VS adalah 2 file *trace*, yaitu file *trace* pengirim dan file *trace* video.

Komponen VS membaca file video yang dikompresi dari output video *encoder*, memfragmentasi masing-masing frame video yang besar menjadi segmen-segmen yang kecil, dan kemudian mengirimkan segmen-segmen ini via paket UDP pada *real network* atau simulasi. Untuk *real-network* dapat digunakan *tcp-dump* atau *win-dump,* sedangkan untuk simulasi jaringan, dapat digunakan NS-2, Qualnet, atau OPNet. Komponen VS juga men-*generate* file trace video yang berisi informasi tentang tiap frame pada file video *real*.

Frame Number	Frame Type	Frame Size	Number of UDP-packets	Sender Time

Keterangan:

1. **Frame Number**
   Nomor urutan frame pada video yang sedang diteliti
2. **Frame Type**
   Tipe frame pada tiap nomor urutan frame, yaitu I-frame, P-frame, atau B-frame
3. **Frame Size**
   Ukuran frame
4. **Number of UDP-packets**
   Jumlah paket UDP
5. **Sender Time**
   Waktu paket dikirimkan

Gambar 3.4 Format File *Trace* Video

Dua file *trace* ini secara bersamaan merepresentasikan transmisi video lengkap (pada sisi pengirim) dan berisi seluruh informasi yang dibutuhkan untuk evaluasi lebih lanjut oleh EvalVid. Dengan menggunakan VS, file *trace*. dapat di-*generate* bersamaan untuk file video yang berbeda dan dengan ukuran paket yang

berbeda, yang dapat dimasukan ke dalam "*network black box*" seperti simulasi. Hal ini dilakukan dengan bantuan tool-tool yang disediakan oleh EvalVid. Jaringan kemudian menghasilkan *delay*, kemungkinan *loss*, dan re-*ordering* dari paket. Pada sisi penerima file *trace* penerima di-*generate* dengan bantuan dari *output* rutin EvalVid.

3.2.4 File *Trace* Simulasi

Setelah dilakukan simulasi jaringan berdasarkan file *trace* pengirim, NS-2 akan men-*generate* sebuah file *trace* utama, file *trace video sender*, file *trace video receiver*, dan file video. File *trace* utama merupakan pencatatan seluruh kejadian yang dialami oleh suatu simulasi paket pada simulasi yang dibangun. Format file *trace* simulasi NS-2 dapat dilihat pada Gambar 3.5.

Event	Time	From node	To node	Pkt type	Pkt size	Flags	Fid	Src addr	Dst addr	Seq num	Pkt id

Keterangan:

1. Event
Kejadian yang dicatat oleh NS yaitu:
r: receive (paket diterima oleh to node)
+: enqueue (paket masuk ke dalam antrian atau keluar dari *form node*)
-: dequeue (paket keluar dari antrian)
d: drop (paket drop dari antrian)

2. Time
Yaitu waktu terjadinya suatu kejadian dalam detik

3. From node dan To node
From node dan to node menyatakan keberadaan paket. Saat pencatatan kejadian, paket berada pada link diantara *from node* dan *to node*.

4. Pkt type
Adalah tipe paket yang dikirim seperti udp, tcp, ack, atau cbr.

5. Pkt Size
Adalah ukuran paket da/am byte.

6. Flag
Flag digunakan sebagai penanda. Padadata diatas flag tidak digunakan
Macam-macam Flag yang bisa digunakan yaitu:
· E untuk terjad congesti (Congstion Experienced/CE)
· N untuk indikasi ECT (ECN-Capable-transport) pada header IP
· C untuk ECN-Echo
· A untuk pengurangan window kogesti pada headerTCP
· P untuk prioritas
· F untuk TCP fast start

7. Fid
Adalah penomoran unik dari tiap aliran data

8. scr_addr
Adalah alamat asal paket
Format scr_addr adaiah : node.port, misalnya 3.0

9. dst_addr
Adalah alamat tujuan paket
Format dst_addr adalah node.port misalnya 0.0

10. sequence number
Adalah nomor urut tiap paket

11. paket Id
Adaiah penomoran unik dari tiap paket

Gambar 3.5 Format File *Trace* Simulasi NS-2

Selain file *trace* utama, NS-2 juga menghasilkan file *trace* pengirim dan file *trace* penerima. Kedua file *trace* ini memiliki format yang sarna seperti pada

Garnbar 3.6. Agar hasil simulasi dapat dianalisa dan ditampilkan dalam bentuk grafik, maka dapat dilakukan *record* atau parsing terhadap *trace* file untuk mengarnbil data yang benar-benar diperlukan. Parsing ini diimplementasikan dalam program *evaluate traces* (ET) yang akan dijabarkan kemudian.

Time stamp	Packet id	Payload size

Keterangan:

1. Time stamp

Pada file trace pengirim merupakan waktu pengiriman, yaitu waktu pada saat paket dikirimkan oleh pengirim. Pada file trace penerima adalah waktu yang diterima, artinya waktu pada saat paket diterima oleh penerima.

2. Packet Id

Merupakan urutan ID, yaitu nomor urutan dari tiap paket.

3. Payload size

Merupakan ukuran paket, yang berarti ukuran paket yang berisi packet header

Gambar 3.6 Format File *Trace* Pengirim dan Penerima

3.2.5 *Fix Video* (FV)

Pengujian kualitas video digital dilakukan frame demi frame. Ini berarti dibutuhkan jumlah frame yang sarna antara sisi penerima dengan sisi pengirim. Ini menimbulkan pertanyaan bagaimana seharusnya *frame loss* diperlakukan jika *decoder* tidak men-*generate* frame "kosong" untuk frame yang hilang. Tool FV hanya dibutuhkan jika *codec* yang digunakan tidak .menyediakan frame yang hilang.

Oleh karena ini merupakan masalah r*eordering*, frame yang sudah di-*code* tidak cocok pada frame *decodec* (YUV) dengan jumlah yang sarna. FV memperbaiki masalah, dengan mencocokkan frame tampilan (YUV) pada frame transmisi (*coded*). Terdapat skema *coding* yang lebih mungkin (seperti skema tanpa *B-frame*, dengan salah satu *B-frame* diantara 2 *I-frame* atau *P-frame*), tetapi pada prinsipnya *reordering* tetaplah sama.

Masalah lain diperbaiki oleh FV adalah kemungkinan ketidakcocokan dari jumlah frame pada *decoded* pada jumlah frame pada video asli yang disebabkan oleh *loss*. Ketidakcocokan dapat menyebabkan pengujian kualitas menjadi tidak valid. *Decoder* yang layak dapat men-*decode* tiap frame, dimana diterima sebagian. Namun beberapa *decoder* menolak untuk men-*decode* lagi frame atau

decode B-frame, dimana 1 frame hilang dari video aslinya. Untuk menangani frame hilang atau *corrupt* yang disebabkan oleh *decoder*, FV dapat dikonfigurasi dengan memasukkan frame yang hilang. Terdapat dua kemungkinan melakukan hal tersebut. Pertama dengan memasukkan frame "kosong" untuk tiap frame yang tidak di-*decode* untuk alasan apapun. Frame kosong adalah frame yang tidak berisi informasi. Frame kosong akan, menyebabkan beberapa *decoder* menampilkan gambar hitam atau putih. Hal ini merupakan penanganan yang kurang baik, karena biasanya perbedaan rendah antara dua frame video yang berurutan. Oleh karena itu, FV menggunakan kemungkinan kedua, dimana pemasukkan dari frame *decoded* terakhir pada kasus frame *loss* pada *decoder*. Penanganan ini memiliki keuntungan lebih besar dibandingkan mencocokkan perilaku dari video *player* sesungguhnya.

3.3 *Evaluasi Trace*

Pusat evaluasi *framework* adalah program bemama *evaluate traces* (ET). Evaluasi berlangsung pada sisi pengirim ketika transmisi video berakhir. Disinilah letak perhitungan aktual dari *paket loss, frame loss, delay* dan *jitter*. Untuk perhitungan data tersebut dibutuhkan 3 file *trace*. Perhitungan *loss* sungguh mudah, mengingat ketersediaan *packet id* yang unik. Dengan bantuan file *trace* video, tiap paket ditetapkan sebuah tipe. Tiap paket pada tipe ini yang tidak termasuk pada *trace* penerima dihitung *loss.* Frame *loss* dihitung dengan melihat pada frame manakah salah satu segmen (paket) hilang. Jika segmen pertama dari frame adalah diantara segmen yang hilang, frame dihitung *loss.* lni dikarenakan video *decoder* tidak bisa men-*decode* frame, dimana bagian awal hilang.

ET dapat juga mempertimbangkan kemungkinan dari adanya beberapa loncatan waktu. Jika terdapat *buffer* tidak terpakai diimplementasi pada entitas jaringan yang diterima, *buffer* akan berjalan kosong. Jika tidak ada frame yang tiba untuk beberapa saat, maksimum ukuran *buffer* tidak terpakai. Metrik kualitas video obyektif seperti PSNR tidak dapat mempertimbangkan *delay* atau *jitter.* Bagaimanapun, *buffer* yang tidak terpakai kosong (atau penuh) secara efektif menyebabkan *loss* (tidak ada frame untuk ditampilkan). Maksimum ukuran *buffer* yang tidak terpakai dapat digunakan untuk mengubah *delay* menjadi *loss*. ET

dapat melakukan ini dengan menyediakan maksimum *play-out buffer* sebagai parameter.

Tugas ET lain adalah men-*generate* dari file video yang *corrupt* (karena *loss*). File *corrupt* ini nanti dibutuhkan untuk melakukan pengujian kualitas video *end-to-end*. Kemudian file lain dibutuhkan sebagai *input* ET, bernama *encoded* video file asli. Pada dasarnya, *generate* dari video yang *corrupt* dilakukan dengan meng-*copy* video asli paket demi paket dimana *packet loss* diabaikan. ET harus memberi perhatian pada kemampuan penanganan *error* yang sesungguhnya pada video *decoder* yang sedang digunakan. Jika mungkin, *decoder* mengharapkan penandaan khusus pada kasus data yang hilang, seperti karakter khusus atau *buffer* kosong (diisi dengan 0) daripada paket yang hilang.

ET men-*generate* file *trace* akhir, yaitu file *trace delay*. Format file *trace* ini dapat dilihat masing-masing pada Gambar 3.7. Selain itu, ET juga menghasilkan tampilan video yang akan diterima, dimana nantinya dapat dibandingkan dengan *raw video source* untuk memperoleh nilai kualitas video, yaitu PSNR dan MOS.

Frame id	loss flag	end-to- end delay	Inter-frame gap sender	Inter-frame gap receiver	Cumulative jitter

Keterangan:

1. Frame id
Nomor urut frame

2. Loss flag
Flag yang hilang

3. End-to-end delay
Delay yang terjadi pada transmisi video

4. Inter-frame gap sender
Intel-frame gap pada sisi pengirim

5. Inter-frame gap receiver
Inter-frame gap pada sisi penerima

6. Cummulative jitter
Jitter kumulatif

Gambar 3.7 Format File *Trace Delay*

3.4 Spesifikasi Perangkat

Adapun beberapa spesifikasi perangkat yang dibutuhkan untuk membangun simulasi jaringan ini antara lain:

1. *Interface Hardware*

   Hardware yang digunakan adalah 1 buah PC (*laptop*) dengan spesifikasi sebagai berikut:

   - Axioo Centaur
   - *Processor*: Intel Centrino Duo @ 1.60GHz

- *Memory*: 1.5 GB DDR2
- HDD: 80 GB

2. *Interface Software*

   Simulasi ini berjalan baik pada spesifikasi *software* sebagai berikut:

   - Sistem Operasi: Windows XP
   - Aplikasi Simulasi: *Cygwin*, NS-2.29, modul *QoS-included WiMAX* dan EvalVid ver 1.2
   - Aplikasi Grafik: Microsot Excel 2003
   - Penyunting Teks: Ultra Edit

3.5 Instalasi dan Implementasi

3.5.1 Instalasi Software NS-2

Pembangunan simulasi ini dibuat diatas operating system Windows XP yang diinstal *cygwin*. *Cygwin* diperoleh dari http://140.116.72.80/~smallko/ns2/cygwin.rar dimana *cygwin* tersebut merupakan hasil *repack* untuk menjalankan EvalVid, sehingga dapat dijalankan tanpa perlu menghabiskan ruang harddisk untuk paket yang tidak diperlukan. Installasi *cygwin* cukup mudah, karena sudah menggunakan GUI.

NS yang digunakan sendiri yaitu NS-2.29. Untuk menginstalnya, cukup dengan men-*download* file ns-allinone-2.29.tar.gz dari website http://www/isi.edu/nsnam/dist/. File ini sudah mencakup beberapa paket *dependancy* yang dibutuhkan oleh NS-2, yaitu *xlibs-dev, tel, tk, otcl, nam*, dan *xgraph.*

```
# tar xvzf ns-allinone-2.29.tar.gz
# cd ns-alllinone-2.29
# ./install
```

Proses instalasi akan memakan waktu kurang lebih 20 menit, namun tergantung dari komputer yang digunakan. Setelah instalasi selesai, *update environment* variabel pada *'/home/Delax/.bashrc'* sebagai berikut:

```
export NS_HOME=`pwd`/ns-allinone-2.29

export
PATH=$NS_HOME/tcl8.4.11/unix:$NS_HOME/tk8.4.11/unix:$NS_HOME/bin:$
NS_HOME/ns-2.29:$NS_HOME/ns-2.29/wimax:$NS_HOME/ns-
2.29/diffusion3/filter_core:$NS_HOME/xgraph-12.1:$NS_HOME/nam-
1.11:/usr/X11R6/bin:$NS_HOME/trace-analyzer/bin:$PATH

export
LD_LIBRARY_PATH=$NS_HOME/tcl8.4.11/unix:$NS_HOME/tk8.4.11/unix:$NS
_HOME/otcl-1.11:$NS_HOME/lib:$LD_LIBRARY_PATH

export TCL_LIBRARY=$NS_HOME/tcl8.4.11/library

export DISPLAY=:0.0
```

3.5.2 Instalasi Modul WiMAX

Secara *default* NS-2 tidak memiliki modul untuk WiMAX, sehingga diperlukan penambahan modul. Dikarenakan kebutuhan akan QoS dalam WiMAX untuk pengujian, maka modul yang digunakan haruslah QoS *Enabled*. Modul diinstalasi menggunakan *patch* yang diambil dari website http://perso.telecom-bretagne.eu/aymenbelghith/tools/ file yang di-*download patch-QoS-WiMAX_prerelease-09-04-2008. Patch* tersebut dibuat oleh Aymen Belghith yang memodifikasi modul WiMAX NIST dengan penambahan fitur QoS dan *scheduling*. Setelah *patch* di-*download* ke direktori ns-allinone-2.29 dilakukan *patch* dengan metode berikut:

```
patch -p0 < patch-QoS-WiMAX_prerelease-09-04-2008
./configure
make clean
make
```

3.5.3 Instalasi Modul EvalVid

Seperti halnya WiMAX, EvalVid bukan merupakan fitur *default* dari WiMAX sehingga dibutuhkan penambahan modul EvalVid yang diperoleh dari http://140.116.72.80/~smallko/ns2/Evalvid_in_NS2.rar, modul EvalVid ditambahkan dengan modifikasi *packet, agent,* dan *traffic agent*. Modifikasi dilakukan dengan langkah sebagai berikut[4].

EvalVid berupa file kompresi 'evalvid-2.4.tar.bz2' yang dapat diperoleh dari website http://www.tkn.tu-berlin.de/research/evalvid/ serta penambahan modul EvalVid pada NS-2 melalui modifikasi *packet, agent* dan *traffic agent*. Modifikasi ini dilakukan dengan beberapa langkah manual sebagai berikut:

```
Modifikasi packe:t.h
Modifik~si agent.h
Modifikasi agent.cc
Penambahan file myudp.cc, myudp.h, myudpsink2.cc, myudpsink2.h,
mytraffictrace2.cc
Modifikasi ns-default.tcl
Modifikasi makefile
./configure ; make clean ; make
```

3.5.4 Codec File Video

Pada awalnya sebuah sumber video dibutuhkan, sebuah video *raw* (belum di *encode*) biasanya disimpan dalam format YUV, karena YUV meripakan format yang didukung oleh banyak video *encoder*.

File YUV di-*encode* menjadi mp4 melalui dua tahap, yaitu menggunakan *tool* ffmpeg.exe dan MP4 Box sebagai berikut :

```
ffmpeg.exe -s qcif -vcodec mpeg4 -r 30 -g 9 -bf 2 -i
foreman_qcif.yuv foreman_qcif.m4v
```

Pada langkah ini *raw* video *.yuv yang akan disimulasikan diubah menjadi *coded* video *.m4v, dimana kedua file berformat *qcif*, file tujuan berformat mpeg4, frame *rate* 30fps, ukuran *group of pictures 9.*

```
./MP4Box.exe -hint -mtu 1024 -fps 30 -add foreman_qcif.m4v
foreman_qcif.mp4
```

Dari file *.m4v selanjutnya dikonversi menjadi file *.mp4 agar dapat ditransmisikan dengan *Maximum transmission Unit* (MTU) 1024bit dan *frame rate* 30fps.

Salah satu bagian dari *tool* evaluasi EvalVid ini adalah menghasilkan *trace* dari sebuah video agar dapat dapat dianalisa menggunakan tool EvalVid dan NS file video diterjemahkan menjadi file *trace* yang berisi nomor frame, jenis frame (B, I, P), besar paket dan waktu. File ini kemudian akan diproses pada saat menjalankan file *.tcl NS-2.

```
./mp4trace.exe -f -s 127.0.0.1 8080 foreman_qcif.mp4 > foreman_qcif.st
```

3.5.5 Menjalankan *Script* Tcl

Pada tahapan ini dijalankan *script* Tcl yang merepresentasikan jaringan WiMAX yang digabungkan dengan jaringan *wired*. Bermodalkan file *trace* pengirim foreman_qcif.st maka simulasi dapat dijalankan. *Script* tcl disimpan dengan nama file 'qoswimax.tcl'

Script Tcl dibuat sedemikian rupa sehingga pada *command prompt* dapat ditentukan *node* apa yang akan dijalankan video, berapa buah *node-node* lain untuk masing-masing kelas QoS yaitu, UGS, rtPS dan BE serta berapa besar *bandwidth* yang akan diberikan untuk setiap *node*. *Script* dijalankan dengan cara :

ns qoswimax.tcl <A> <B> <C> <D> <E>

A : Jenis QoS pada SS dengan video

B : Jumlah SS tambahan UGS

C : Jumlah SS tambahan rtPS

D : Jumlah SS tambahan BE

E : *Bandwidth* untuk setiap SS (bit)

Dalam penulisan ini akan dibahas dua macam simulasi untuk ketiga QoS, yaitu simulasi 1 SS, dan simulasi 4 SS dimana 1 SS video akan menjalankan trafik bersamaan dengan 1 SS UGS, 1 SS rtPS dan 1 SS BE.

Berikut contoh *command* yang dijalankan

```
ns qoswimax.tcl ugs 0 0 0 3000
```

Pada *command* di atas, SS yang membangkitkan video menggunakan UGS dan tidak ada SS lain, *bandwidth* untuk setiap SS 3000bit

NS-2 akan menghasilkan sebuah file *output trace* berekstensi *.tr, sesuai *event-even*t yang akan dilakukan oleh simulasi. Di samping itu beberapa file parameter umum seperti *delay, cumulative jitter* dan *interframe gap,* juga di*generate* secara langsung melalui aplikasi NS-2.

3.5.6 Evaluasi File *Trace*

Bagian utama dari *framework* evaluasi ini adalah *Evaluation Trace* (ET). Disinilah penghitungan aktual dari *losses* dan *delay* paket serta frame dilakukan. Untuk penghitungan yang dibutuhkan hanya tiga file *trace*, yaitu *sender* dan *trace dump* serta file *trace* video.Dimana setiap paket yang dikirim namun tidak ada di file *trace* penerima maka dihitung sebagai frame/paket yang hilang. Menampilkan analisa frame dan paket yang dikirim dan diterima *loss* atau tidak, untuk masing-masing tipe frame (B, I, P).

Langkah pertama dalam proses evaluasi adalah menghitung video PSNR referensi. PSNR referensi diambil dari video yang di-*code* dan di-*decode* (*codec*) tanpa adanya *error* atupun *loss* pada transmisi ulang mengacu pada *raw* video *source* yang tidak mengalami proses *codec*. Kemudian dilakukan perbandingan antara video referensi dengan video hasil *output* (kemungkinan *corrupt*).

```
./psnr 176 144 420 foreman_qcif.yuv foreman_ref.yuv > ref_psnr.txt
```

Untuk keperluan analisa lanjutan dibutuhkan file video format YUV hasil rekonstruksi jaringan agar dapat dibandingkan kualitasnya dengan file YUV di awal, langkah pertama yang dilakukan adalah dengan menggunakan etmp4.exe

```
./etmp4.exe sd_qos_wimax_ugs000 rd_ qos_wimax_ugs000
foreman_qcif.st foreman_qcif.mp4 ugs000
```

Command tersebut akan mengenerate file video yang mungkin *corrupt* dalam bentuk file MP4 yang berisi video track yang *damaged* (ugs000). Kemudian file-file ini di-*decode* menggunakan ffmpeg.exe untuk memproduksi file YUV seperti terlihat di sisi penerima sebagai berikut:

```
./ffmpeg –i foreman_qcife.m4v foreman_qcife.yuv
```

Selain menghasilkan file MP4 etmp4.exe juga menghasilkan beberapa file sebagai berikut (xxxx : nama file output).

loss_xxxx.txt	Berisi *frame loss* pada I, P, B dan overall frame loss in %)
delay_xxxx.txt	berisi *frame-number., lost-flag, end-to-end delay, inter-frame gap sender, inter-frame gap receiver*, dan *cumulative jitter in seconds*
rate-s_xxxx.txt	berisi waktu, *bytes per second* (*current time interval*), dan *bytes per second* (*cumulative*) yang dihitung di sisi pengirim
rate_r_xxxx.txt	Berisi waktu, *bytes per second* (*current time interval*), dan *bytes per second* (*cumulative*) yang dihitung di sisi penerima

Dari file-file di atas yang akan digunakan untuk analisa adalah loss_xxxx.txt untuk analisa *loss* dan delay_xxxx.txt untuk analisa *delay* dan *jitter*.

3.5.7 Menjalankan Script awk

Satu-satunya parameter QoS yang tidak dihasilkan langsung oleh tool-tool EvalVid pada NS-2, yaitu *throughput* dilakukan dengan menggunakan *script Awk* yang dibuat oleh Marco Fiore yang sedikit dimodifikasi sehingga hanya *throughput* video yang tampil. Perintahnya sebagai berikut:

```
Awk –f videoThroughput ugs000.tr > throughputugs000.txt
```

3.5.8 Kualitas Video Akhir

Output file YUV yang benar adalah yang memiliki jumlah frame yang tepat sebanyak file YUV aslinya. Namun, banyak *codec* tidak dapat men-*decode* file video yang *corrupt* dengan baik. Seperti, ffmpeg biasanya memproduksi frame yang lebih sedikit atau bahkan *crash.* Kadangkala frame terakhir tidak di-*decode*,

namun itu bukanlah masalah. Seluruh hasil *output* PSNR disimpan dalam sebuah direktori, misalnya '/psnr'. PSNR video penerima dihitung dengan :

```
./psnr 176 144 420 foreman_qcif.yuv ugs000.yuv > psnr/psnr_ugs000.txt
```

Karena nilai PSNR masih kurang mudah untuk digunakan dalam menilai kualitas video, maka digunakanalah *quality metric* yang dapat menghitung perbedaan kualitas antara *encoded* video dan video yang telah diterima (kemungkinan *corrupt*). Setelah dilakukan pengukuran PSNR untuk seluruh video, dimana keseluruhan file *output* PSNR diletakkan pada direktori '/psnr', dijalankan perintah berikut untuk mengukur nilai MOS:

```
mos psnr ref_psnr.txt > mos_all.txt
```

3.5.9 Grafik dan Analisa

Untuk mempermudah proses analisa, data dan menghitungan akhir yang diperoleh yaitu *throughput*, *frame loss*, *delay* dan *jitter*, divisualisasikan ke dalam bentuk grafik.