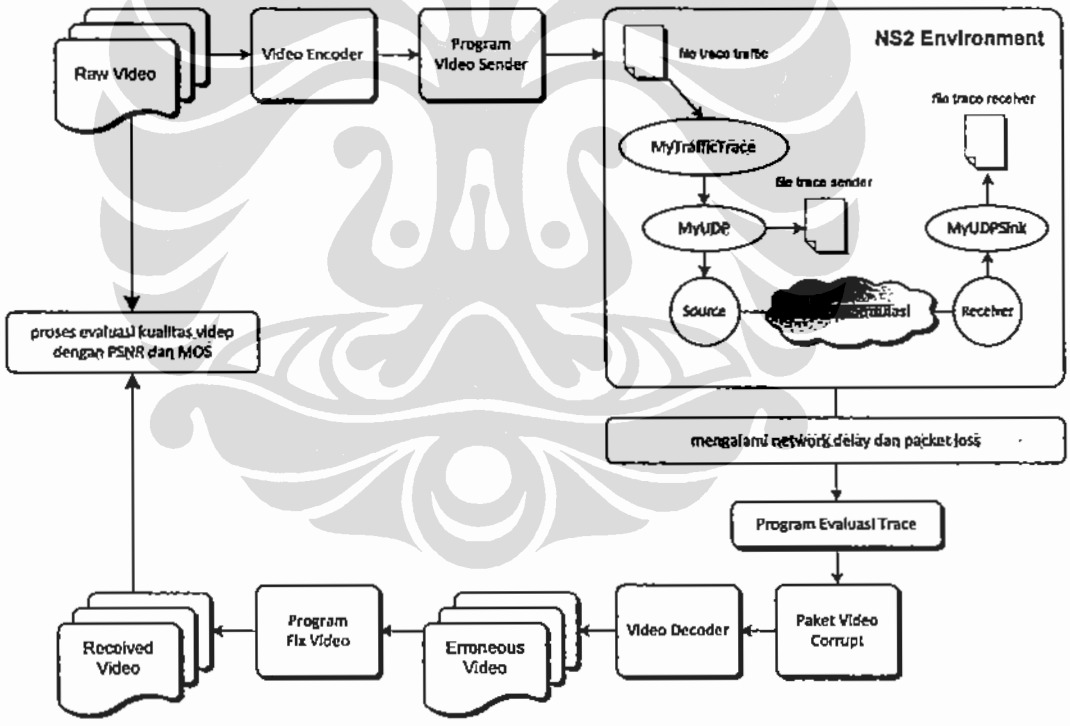
BAB III
PERANCANGAN SIMULASI JARINGAN

Proses siklus implementasi penelitian terdiri dari encoder H.264, video sender, Network Simulator (NS-2), H.264 decoder, program *Evaluate Trace (ET)*, program PSNR, dan program MOS. Gambar 3.1 [9] menggambarkan hubungan antara komponen-komponen yang berbeda, file video input, dan file output yang di-*generate* dari tool-tool yang digunakan pada penelitian ini. Metodologi proses siklus implementasi penelitian ini dibangun di dalam *framework* EvalVid, dan diperluas dengan mengikutsertakan NS-2 dan EURANE untuk simulasinya pada HSDPA.



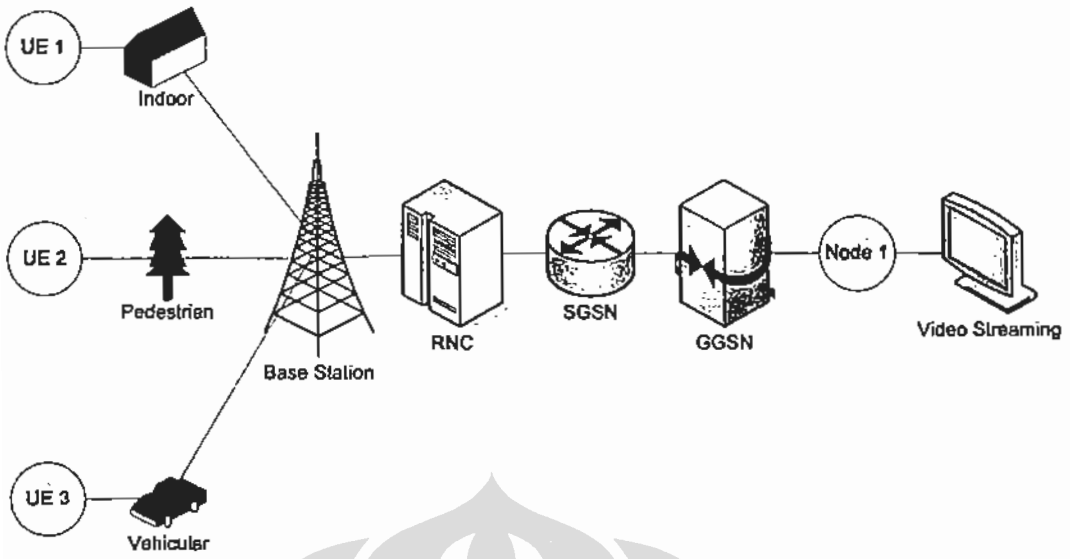
Gambar 3.1 Proses Keseluruhan Implementasi Penelitian

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 pengirim untuk masing-masing frame video. File trace kemudian dimasukkan ke dalam *streaming server* yang dijalankan pada simulator NS-2 untuk memproduksi *video stream* pada HSDPA. Efek dari streaming video pada HSDPA 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 untuk 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 ET (*evaluate traces*) untuk meng-*generate* kemungkinan file video yang *corrupt* dari hasil transmisi pada HSDPA. File video yang *corrupt* ini merupakan data yang penting untuk digunakan oleh *Peak Signal to Noise Ration (PSNR)* dan *Mean Opinion Score (MOS)* dalam mengevaluasi kualitas video *end-to-end*.

3.1 Topologi Jaringan

Penelitian ini melakukan simulasi *user* yang berada di beberapa *environment* yang berbeda untuk mendapatkan layanan *video streaming* di dalam konteks Enhanced-UMTS. Simulasi ini terdiri dari 3 *user*, dimana masing-masing berada pada *environment* yang berbeda. *User* pada setiap environment yang berbeda akan mengakses layanan *video streaming*. Topologi yang akan digunakan secara umum adalah seperti yang ditunjukkan pada Gambar 3.2. Simulasi ini menggunakan network simulator NS-2 dengan tambahan modul EURANE untuk simulasi jaringan dan EvalVid untuk evaluasi kualitas layanan video.



Gambar 3.2 Simulasi Performa Layanan *Video Streaming* pada Jaringan UMTS

3.1.1 Environment yang Disimulasikan

Pada simulasi ini environment dibatasi pada 3 environment yang berbeda, yaitu pada *Indoor*, *Pedestrian*, dan *Vehicular Environment* [4]. Karakteristik pada ketiga environment ini didasari oleh Recommendation ITU-R M.1225 [5] yang dibangun dengan menggunakan metode empiris. Beberapa parameter penting yang digunakan untuk mengidentifikasi model referensi adalah meliputi environment propagasi, kondisi trafik, *user information rate* untuk prototipe layanan *voice* dan data, serta kriteria performa objektif pada masing-masing environment yang beroperasi. Model referensi digunakan untuk memperkirakan aspek kritis, seperti spektrum, *coverage*, dan efisiensi *power*. Estimasi ini didasarkan pada perhitungan *system-level* dan simulasi software *link-level* menggunakan propagasi dan model trafik.

Adapun propagasi *multi-path* menyebabkan *fading* dan penyebaran waktu *channel*. Karakteristik *fading* beranekaragam dengan environment propagasi dan pengaruhnya pada kualitas komunikasi (seperti bentuk *error bit*) yang lebih bergantung pada kecepatan mobile station relatif terhadap base station yang melayani. Karakteristik propagasi radio harus ditentukan untuk mengembangkan sistem desain untuk IMT-2000, yang meliputi maksimum jangkauan transmisi,

model prediksi *overall path loss*, *multipath delay spread*, statistik *slow fading*, statistik *fast fading*, dan maksimum pergerakan *Doppler*.

Untuk masing-masing environment, digunakan model respon *channel impulse* yang didasari oleh model *tapped-delay line* yang diberikan. Model ini dicirikan oleh jumlah tap, time delay relatif pada tap pertama, rata-rata power relatif pada tap terkuat, dan spektrum Doppler pada masing-masing tap. Tabel 3.1 [5] menjelaskan parameter-parameter *tapped-delay-line* pada environment *Indoor A*, *Pedestrian A*, dan *Vehicular A*. Variasi kecil, ± 3%, pada *time delay* relatif diperbolehkan sehingga *channel sampling rate* dapat dibuat untuk menyesuaikan beberapa *sample rate* multi simulasi hubungan. Dalam implementasinya pada simulasi di NS-2, *user environment* ini di-generate melalui aplikasi MatLAB R2008a.

Tabel 3.1 Parameter tapped-delay-line pada masing-masing environment

Tap	Indoor A		Pedestrian A		Vehicular A	
	Relative delay (ns)	Average power (dB)	Relative delay (ns)	Average power (dB)	Relative delay (ns)	Average power (dB)
1	0	0	0	0	0	0
2	50	−3.0	110	−9.7	310	−1.0
3	110	−10.0	190	−19.2	710	−9.0
4	170	−18.0	410	−22.8	1090	−10.0
5	290	−26.0	–	–	1730	−15.0
6	310	−32.0	–	–	2510	−20.0

3.1.1.1 Indoor

Indoor merupakan *environment* dimana *user* dan *base station* berada di dalam ruangan. Berdasarkan perspektif mobilitas, *indoor environment* dibentuk oleh kecepatan yang lambat dimana mobilitasnya relatif ditentukan oleh topologi arsitektur dan bentuk aktivitas. Beberapa *indoor environment* partikular memperlihatkan fiturnya masing-masing, seperti rumah, *office environment*, bandara dan stasiun kereta, zona komersial, teater atau hiburan umum, dan area parkir. Karakteristik utama dari *environment* ini adalah *cell* yang kecil dan *transmit power* yang rendah. Karena lokasinya yang statis, *indoor* memiliki prioritas besar untuk mendapatkan traffic yang tinggi.

Path loss rule bervariasi dikarenakan penghamburan dan pelemahan oleh dinding, lantai, dan struktur metalik seperti partisi dan *filling cabinet*. Objek-objek ini juga memproduksi efek *shadowing*. Standar deviasi *shadow fading* pada log-normal diharapkan sebesar 12 dB. Jangkauan *fading* dari Rician ke Rayleigh, dengan frekuensi Doppler offset diatur oleh kecepatan berjalan.

Beberapa parameter environment *indoor* yang ditetapkan pada simulasi ini adalah sebagai berikut:

- Kecepatan: 3 km/jam
- Jarak *node – User Equipment*: 500m
- Panjang *trace*: 200s
- Jumlah *user*: 1

3.1.1.2 Pedestrian

Sesuai dengan namanya, pedestrian diasosiasikan dengan *environment* urban yaitu kondisi user berjalan dimana *base station* diletakkan di luar (*outdoor*) tetapi juga dapat meng-*cover* area bangunan internal. Karakteristik dari *environment* pedestrian adalah *cell* yang kecil, area yang kecil dengan bangunan-bangunan tinggi, kepadatan user yang tinggi, *transmit power* yang rendah dan mobilitas yang rendah. Pada umumnya pedestrian mendapatkan trafik sedang dari total *bandwidth*.

Path loss rule geometris dari R^{-4} layak, tetapi jangkauan lebih luas dapat dipertimbangkan. Jika *path* adalah *line of sight* pada jalan seperti tebing, *path loss rule* mengikuti R^{-2} dimana terdapat *Fresnel zone clearance*. Untuk daerah dimana tidak terdapat *Fresnel zone clearance*, *path loss rule* R^{-4} layak tetapi jangkauan hingga R^{-6} mungkin teradi dikarenakan pepohonan dan halangan lain sepanjang *path*. *Shadow fading* pada log-normal dengan standar deviasi 10 dB layak untuk outdoor dan 12 dB untuk indoor. *Loss* penetrasi gedung rata-rata 12 dB dengan standar deviasi 8 dB. *Fading rate Rayleigh* dan/atau *Rician* umumnya diatur oleh kecepatan berjalan, tetapi *fading* lebih cepat dikarenakan refleksi dari kendaraan bergerak sesekali terlihat.

Untuk mereprentasikan *environment* ini pada umumnya digunakan model Manhattan, seperti terlihat pada Gambar 3.2 [4]. Bangunan bujur sangkar yang

homogen dari sisi jalan 200 m dan lebar jalan 30 m membentuk karakter *environment* ini. Pada model ini, *mobile* bergerak sepanjang jalan dan mungkin berputar haluan dengan probabilitas yang diberikan. Posisi *mobile* di-*update* setiap 5 meter dan kecepatan dapat berubah pada masing-masing *update* posisi sesuai dengan probabilitas yang diberikan.

Gambar 3.3 Model *pedestrian environment*

Beberapa parameter environment *pedestrian* yang diterapkan pada simulasi ini adalah sebagai berikut:

- Kecepatan: 10 km/jam
- Jarak *node – User Equipment*: 500m
- Panjang *trace*: 200s
- Jumlah *user*: 1

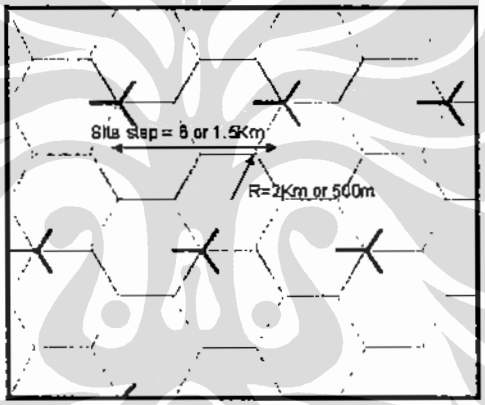
3.1.1.3 Vehicular

Vehicular merupakan *environment* untuk kondisi *mobile environment* dengan mobilitas yang sangat tinggi. *Environment vehicular* diaplikasikan pada skenario urban dan suburban dimana bangunan memiliki tinggi yang serupa dan memiliki karakteristik *cell* yang lebih besar dan *transmit power* yang lebih tinggi. Pada umumnya *vehicular environment* memperoleh trafik yang rendah.

Path loss rule geometris dari R^{-4} dan *shadow fading* pada *log-normal* dengan standar deviasi 10 dB adalah layak pada area urban dan suburban. *Path loss* pada area rural dengan daerah datar lebih rendah daripada area urban dan suburban. Pada area pegunungan, jika halangan *path* dihindari oleh lokasi base station yang dipilih, *path loss rule* mendekati R^{-2} adalah layak. *Rate fading*

Rayleigh diatur pada kecepatan kendaraan. *Rate fading* lebih rendah layak pada aplikasi yang menggunakan terminal tak bergerak.

Model referensi mobilitas *vehicular* menggunakan model mobilitas *pseudorandom* dengan lintasan *semi-direct*, posisi mobile di-*update* sesuai dengan panjang dan arah dekorelasi yang dapat berubah pada masing-masing *update* posisi sesuai dengan probabilitas yang diberikan di dalam sebuah sektor. *Mobile* terdistribusi secara seragam pada peta dan arahnya dipilih secara acak. Radius *cell* adalah 2000 m untuk layanan hingga 144 kbps dan 500 m untuk layanan diatas 144 kbps. Tinggi antena *Base Station* harus lebih dari rata-rata tinggi atap 15 m. Skema penggelaran diasumsikan memiliki layout *cell* hexagonal dengan jarak antara *Base Station* adalah 6 km. Gambar 3.3 menunjukkan penggunaan antena di setiap 3 sektor *cell* [4].



Gambar 3.4 Model *vehicular environment*

Beberapa parameter environment *vehicular* yang diterapkan pada simulasi ini adalah sebagai berikut:

- Kecepatan: 50km/h
- Jarak *node - User Equipment*: 500m
- Panjang *trace*: 200s
- Jumlah *user*: 1

3.1.2 Traffic Generator

Jenis aplikasi yang digunakan pada simulasi ini adalah layanan *video streaming*, yaitu teknologi yang dapat mengirimkan file audio dan video digital secara *real time*. 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 paketisasi dan menyediakan penyampaian frame video berurut. RTCP digunakan oleh video klien untuk memberitahukan video server 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 juga dinamakan *traffic generator. Traffic generator* akan dibangun sesuai dengan karakteristik dari 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 didesain baik untuk membaca file trace video atau untuk men-*generate* data yang dibutuhkan untuk mengevaluasi kualitas video yang dikirimkan. Cara kerja dan kegunaan masing-masing *agent* adalah sebagai berikut:

- ***Agent* MyTrafficTrace** meng-*extract* 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 ini 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 file name output dari file trace pengirim dan merekam *timestamp* pada setiap paket yang ditransmisikan, *packet ID*, 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** adalah *agent* penerima untuk paket frame video yang terfragmen yang dikirimkan 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. Akhirnya, video fix YUV yang terekonstruksi dapat dibandingkan dengan video raw YUV untuk mengevaluasi kualitas video *end-to-end* yang dikirimkan.

3.1.3 Scheduling

Dalam modul EURANE ada tiga macam scheduling yang dapat digunakan yaitu *Round Robin, Maksimum C/I* dan *Fair Channel Dependent Scheduling (FCDS)* [2]. *Scheduling* yang digunakan pada simulasi ini adalah FCDS. FCDS memiliki keunggulan dibandingkan dibanding kedua mekanisme lainnya yaitu dari sisi *fairness* terhadap user dan juga tetap mempergunakan *power* dengan efisien. FCDS merupakan mekanisme scheduling hasil *trade-off* dari scheduling berdasarkan C/I yang optimal dari segi power dan juga Round Robin yang berbasis *fair sheduling* [2].

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 176x144 pixel, yang juga dikenal sebagai *Quarter Common Intermediate Format*

(QCIF). QCIF digunakan karena merupakan format umum untuk *video streaming* pada jaringan *mobile*.

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 *chrominance*. 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* setringkali 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.

$$Y = 0{,}299R + 0.587G + 0.114B \quad \text{..... (3.1)}$$

$$U = 0{,}565\,(B - Y)$$

$$V = 0{,}713\,(R - Y)$$

$$R = Y + 1{,}403V$$

$$G = Y - 0{,}344U - 0{,}714V$$

$$B = Y + 1{,}770U$$

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

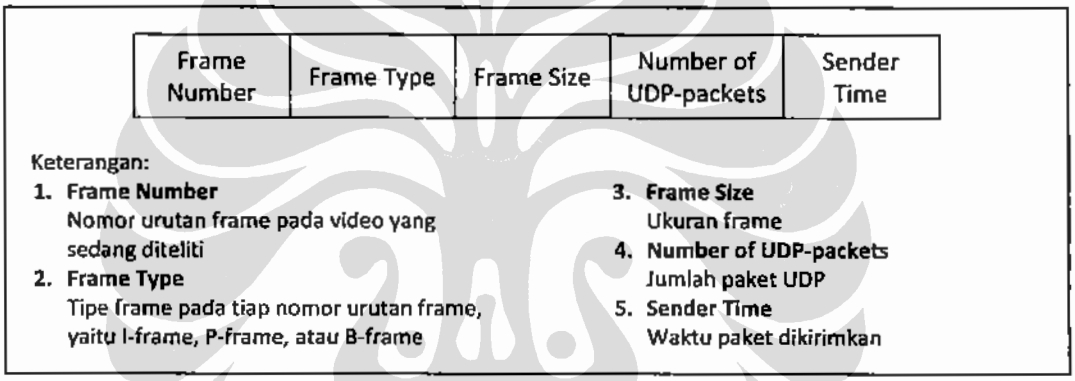
Saat ini EvalVid hanya mendukung *single layer video coding*. EvalVid mampu mendukung berbagai jenis codec MPEG4, yaitu antara lain: National Chiao Twig University (NCTU) codec, ffmpeg, Xvid, dan H.264.

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 tiap paket UDP yang ditransmisikan, VS merekam *timestamp*, *packet ID*, dan *packet payload size* pada file trace pengirim dengan bantuan *third-party tool*. 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.



Gambar 3.5 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*, baik dengan bantuan dari output rutin EvalVid, atau pada kasus ini transmisi sebenarnya, dengan mudah oleh *TCP-dump*, yang menghasilkan file trace yang sesuai dengan EvalVid. Adapun format file trace tersebut dapat dilihat pada Gambar 3.5.

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 *event* (kejadian) yang dialami oleh suatu simulasi paket pada simulasi yang dibangun. File *trace* utama ini mengalami penambahan 1 field 'Pkt Id' yang merupakan hasil output khusus pada modul EURANE. Gambar 3.6 [2] mendeskripsikan format file *trace* utama yang disimulasikan oleh NS-2.

Event	time	From node	To node	Pkt type	Pkt Size	Flags	Fid	Src addr	Dst Addr	Seq Num	Pkt Id

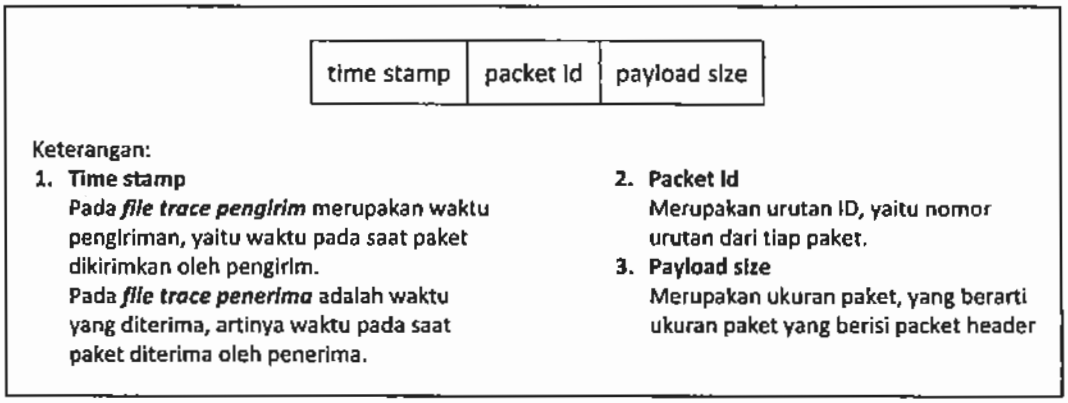
Keterangan:

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 dalam byte.
6. **Flag**
   Flag digunakan sebagai penanda. Pada data 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 header TCP**
   - **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 adalah : *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**
    Adalah penomoran unik dari tiap paket

Gambar 3.6 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 sama seperti pada Gambar 3.7. Agar hasil simulasi dapat dianalisa dan ditampilkan dalam bentuk grafik, maka dapat dilakukan *record* atau parsing terhadap *trace file* untuk mengambil data yang benar-benar diperlukan. *Parsing* ini diimplementasikan dalam program ET (*evaluate traces*) yang akan dijabarkan kemudian.



Gambar 3.7 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 sama 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 *reordering*, frame yang sudah di-*code* tidak cocok pada frame *decodec* (YUV) dengan jumlah yang sama. 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-* 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 pendekatan tidak cerdas, 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 bernama *ET (evaluate traces)*. 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*. Ini dikarenakan video decoder tidak bisa men-*decode* frame, dimana bagian awal hilang.

ET juga dapat digunakan untuk penghitungan waktu *inter-packet*. Namun pada kasus *packet loss*, persamaan ini tidak dapat diaplikasikan begitu saja. Hal ini dikarenakan pada kasus *packet loss* tidak ada *time-stamp* yang tersedia pada file trace penerima untuk paket yang hilang. Hal ini menimbulkan pertanyaan bagaimana waktu *inter-packet* dihitung, jika setidaknya 1 atau 2 paket berurutan hilang? Satu kemungkinan akan mengatur waktu *inter-packet* pada kasus *packet loss* pada sebuah nilai "error" seperti 0. Jika kemudian paket aktual diterima, akan mencari ke belakang hingga nilai yang tepat ditemukan. Waktu *inter-packet* pada kasus ini menjadi $t_n - t_{\text{last-received packet}}$. Namun kerugiannya tidak mendapatkan nilai pada tiap paket dan waktu *inter-packet* akan berkembang besar tanpa alasan. Itulah mengapa pendekatan menggunakan ET sedikit berbeda. Jika setidaknya satu paket hilang, tidak akan men-*generate* nilai yang salah, melainkan nilai yang "ditebak". Hal ini dilakukan dengan menghitung perkiraan waktu kedatangan dari

paket yang hilang. Untuk *packet loss* nilai pengharapan waktu *inter-packet* pengirim digunakan. Di sisi lain, jika terdapat *loss rate* yang sangat besar, direkomendasikan kemungkinan lain: menghitung hanya paket yang diterima secara pasangan dan menghitung *packet loss* secara terpisah. Dengan demikian, file trace penerima memiliki *time-stamp* yang valid pada tiap paket, *inter-packet* (termasuk *inter-frame*) delay dapat dihitung berdasarkan persamaan 3.2 berikut.

Arrival time (*packet loss*) $tR_n = tR_{n-1} + (tS_n - tS_{n-1})$ ..... (3.2)

dimana: tS_n : *time-stamp* dari paket ke-n yang dikirim

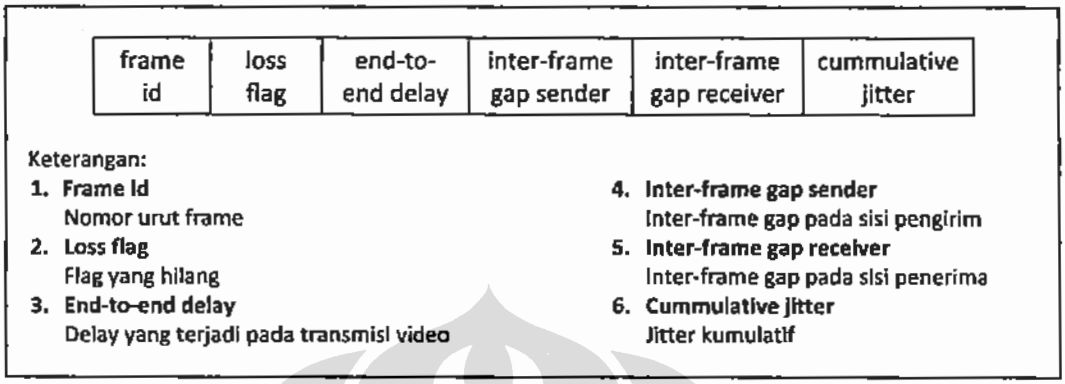
tR_n : *time-stamp* dari paket ke-n yang diterima atau tidak

ET dapat juga dipertimbangkan 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. *Metric* 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.

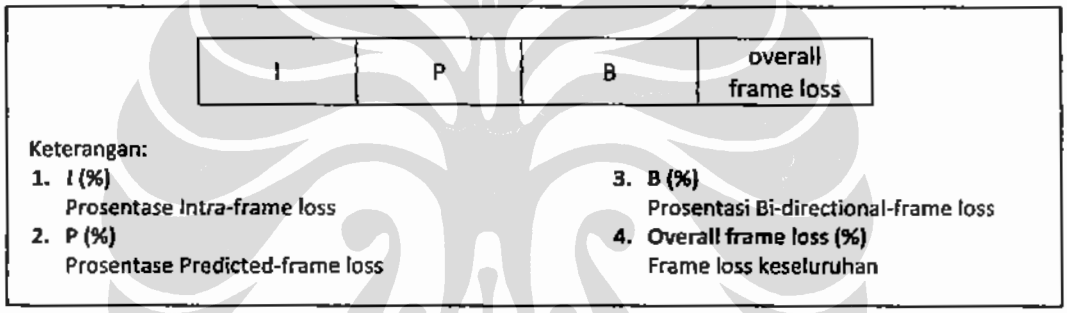
Tugas ET lain adalah men-*generate* dari file video yang *corrupt* (karena *loss*). File *corrupt* ini nanti dibutuhkan untuk melakukan pengujian kualitas video *end-to-end*. Kemudian file lain dibutuhkan sebagai input ET, bernama *encoded video file* asli. Pada dasarnya, *generate* dari video yang *corrupt* 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* beberapa file trace akhir, yaitu file trace *delay* dan file trace *packet loss*. Format kedua file trace ini dapat dilihat masing-masing pada

Gambar 3.8 dan Gambar 3.9. 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.



Gambar 3.8 Format File *Trace* delay



Gambar 3.9 Format File *Trace* packet loss

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:

   - Acer TravelMate 270
   - *Processor*: Intel Pentium IV CPU 1.4 MHz
   - *Memory*: 512 MB SDRAM
   - *HDD*: 100 GB

2. **Interface Software**

Simulasi ini harus berjalan baik pada spesifikasi software sebagai berikut:

- Sistem Operasi: Fedora Core 7
- Aplikasi Simulasi: NS-2.30, EURANE, dan EvalVid
- Aplikasi *Environment Generator*: MatLAB R2008a
- Aplikasi Grafik: Microsot Excel 2007
- Penyunting Teks: Gnome Editor, Vi, dan Notepad++

3.5 Tahap Implementasi

Sebagaimana diilustrasikan pada Gambar 3.10 tahap implementasi penelitian dibagi menjadi 5 tahap, yaitu tahap pemodelan sistem, tahap pembangunan *user environment*, tahap *coding* dan *deployment*, tahap simulasi data dan grafik, dan tahap analisa hasil. Berikut adalah penjabaran lebih lanjut mengenai tahap-tahap implementasi:

Gambar 3.10 Tahapan Kerja Implementasi Penelitian

1. **Pemodelan Sistem**

Tahap awal implementasi yaitu memodelkan topologi teknologi HSDPA pada jaringan UMTS. Topologi jaringan ini juga meliputi jumlah *user* yang terdapat pada 1 *node*, besarnya *bandwidth* dan *delay*, tipe antrian yang digunakan, batasan *packet data*, *transport agent* yang digunakan, trafik yang akan dibangkitkan, jenis *codec* yang digunakan, dan alat pengukuran yang selama ini digunakan. Dengan adanya pemodelan sistem ini akan mempermudah proses pembangunan simulasi jaringan secara keseluruhan.

2. **Pembangunan *User Environment***

Tahap kedua yaitu pembangunan *user environment* sesuai dengan data umum masing-masing environment. Data tersebut dibagi sesuai dengan 3

environment yang digunakan pada simulasi ini, yaitu *Indoor*, *Pedestrian*, dan *Vehicular*. Data-data ini kemudian akan di-*generate* dengan menggunakan MatLab R2008a menjadi sebuah file data *matrix*, untuk selanjutnya dimasukkan ke dalam aplikasi NS-2.

3. ***Coding* dan *Deployment***

   Tahap ketiga adalah tahap inti dari implementasi ini, yaitu *coding* dan pemrograman bahasa Tcl pada aplikasi NS-2. Deskripsi dari pemrograman ini yaitu layanan *video streaming* akan diletakkan pada ujung *node* yang berada pada node *fixed* (tidak bergerak/bukan *node* UMTS) di jaringan IP, dimana masing-masing *user* yang berada pada *environment* yang berbeda akan mencoba mengakses layanan yang disediakan di *node* sumber.

4. **Simulasi Data dan Grafik**

   Data video akan diambil dalam format YUV, untuk selanjutnya mengalami proses encode dan decode. Data ini akan dijalankan pada simulasi NS-2 yang sudah ditambahkan dengan modul EURANE dan EvalVid. Hasil simulasi adalah berupa *file trace* 'output.tr', 'st' 'sd_a0x.txt', 'rd_a0x.txt', dan 'videox.dat'. File-file ini berisi data statistik yang kemudian akan disimulasikan ke dalam bentuk grafik oleh aplikasi Microsoft Excel 2007.

5. **Analisa Hasil**

   Tahap terakhir adalah melakukan analisa hasil berdasarkan data dan grafik yang diperoleh. Analisa ini meliputi nilai dari parameter-parameter yang ingin dicari yaitu *packet loss*, *frame loss*, *delay*, *jitter*, *PSNR*, dan *MOS*. Semuanya ini akan disimpan ke dalam file-file bernama 'x_delay.txt', 'x_rate-r', 'x_rate-s', 'psnr_ ouputx.txt' dan 'mos_outputx.txt'. File-file tersebut juga berisi data statistik yang kemudian akan disimulasikan ke dalam bentuk grafik oleh aplikasi Microsoft Excel 2007.

3.6 Instalasi dan Implementasi

3.6.1 Instalasi Software NS-2

Pembangunan simulasi ini dibuat diatas *operating system* Fedora 7 Core yang berbasis kernel 2.6.21 mengingat Fedora merupakan OS yang ampuh dan

stabil dalam *software development*. NS yang digunakan sendiri yaitu NS-2.30. Untuk menginstallnya, cukup dengan mendownload file ns-allinone-2.30.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, tcl, tk, otcl, nam, dan xgraph.

```
# tar xvzf ns-allinone-2.30.tar.gz
# cd ns-alllinone-2.30
# ./install
```

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

```
# LD_LIBRARY_PATH
OTCL_LIB=/usr/local/ns/otcl-1.12
NS2_LIB=/usr/local/ns/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=/usr/local/ns/tcl8.4.13/library
USR_LIB=/usr/lib
export TCL_LIBRARY=$TCL_LIB:$USR_LIB

# PATH
XGRAPH=/usr/local/ns/bin:/usr/local/ns/tcl8.4.13/unix:/usr/local/ns/tk8.4.13/unix
NS=/usr/local/ns/ns-2.30/
NAM=/usr/local/ns/nam-1.12/
PATH=$PATH:$XGRAPH:$NS:$NAM
```

3.6.2 Instalasi Modul EURANE

Secara default NS-2 tidak memiliki modul untuk UMTS, sehingga diperlukan patch khusus yaitu *Enhanced UMTS Radio Access Network Extensions for ns-2 (EURANE)*. *Patch* EURANE menambahkan tiga node pada aplikasi NS-2, yaitu *Radio Network Controller (RNC)*, *Basestation (BS)*, dan *User Equipment (UE)*, dengan fungsionalitas mendukung *transport channel* berikut: FACH, RACH, DCH, dan HS-DSCH. Untuk menginstalnya, mula-mula download file 'ns-2.30_eurane-1.12.diff.gz' yang dapat diambil dari website http://eurane.ti-

wmc.nl/eurane/ yang kemudian dilanjutkan dengan *patching*. Untuk melakukan *patch* dapat dilakukan metode berikut [11]:

```
patch -p1 < ns-2.30_eurane-1.12.diff.gz
./configure
make clean
make
```

3.6.3 Instalasi Modul EvalVid

Sama halnya dengan EURANE, NS-2 secara default tidak memiliki modul untuk agen traffic generator codec H.264. Oleh karena itu dibutuhkan tool khusus 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 [12]:

```
Modifikasi packet.h
Modifikasi agent.h
Modifikasi agent.cc
Penambahan file myudp.cc, myudp.h, myudpsink3.cc, myudpsink3.h, mytraffictrace3.cc
Modifikasi ns-default.tcl
Modifikasi Makefile
./configure && make clean && make
```

3.6.4 Codec File Video

Langkah pertama proses evaluasi adalah mengkompresi *raw video* dengan *encode* pada 30 fps, panjang GOP 30 frame tanpa adanya B-frame. Ukuran frame adalah 176x144 pixel, yang juga dikenal sebagai *Quarter Common Intermediate Format (QCIF)*. QCIF digunakan karena merupakan format umum untuk *video streaming* pada jaringan *mobile*. Untuk itu dibutuhkan file 'a01.cfg' yang berfungsi sebagai konfigurasi khusus *codec* H.264. Setelah itu digunakan perintah MP4Box untuk membuat file ISO MP4 yang berisi sample video (dalam frame) dan menandai (*hint*) track video yang menjelaskan bagaimana mempaketisasi frame dalam proses transportasinya dengan RTP.

```
./lencod –d a01.cfg
./MP4Box -hint -mtu 1024 -fps 30 -add a01.m4v a01.mp4
```

Untuk beberapa metode pengujian kualitas video dibutuhkan video referensi. Hal ini baik file YUV asli sebelum *encoding* atau file YUV yang dibuat oleh decoding video yang dicode. Dibutuhkan atau tidaknya YUV yang di-*decode* bergantung pada apa yang ingin diperoleh. Jika ingin menguji kualitas video pada sistem jaringan dan tidak hanya kualitas *encoder* harus dibuat file YUV *decoded*. Untuk memproduksi file-file YUV ini harus digunakan *video decoder* open source yang cocok, seperti xvid_decraw, ffmpeg, ldec, dan lain-lain.

```
ffmpeg –i a01.mp4 a01_ref.yuv
```

Tool *mp4trace* dari Evalvid digunakan untuk mengirimkan *hint* file mp4 per RTP/UDP untuk menentukan host tujuan. Oleh karena output *mp4trace* ini akan dibutuhkan kemudian, hasilnya disimpan ke dalam file video trace pengirim 'st_a00'.

```
./mp4trace -f -s 192.168.0.2 12346 a01.mp4 > st_a00
```

Secara *default*, program *mp4trace* menandai 'I' frame dengan 'H'. Oleh karena itu untuk mencapai hasil yang optimal setelah simulasi, secara manual edit file output 'st_a00' dan ubah karakter 'H' menjadi 'I'.

3.6.5 Menjalankan Script Tcl

Setelah itu, kemudian menjalankan script Tcl yang merepresentasikan jaringan UMTS dilengkapi dengan teknologi HSDPA. Bermodalkan file trace video pengirim 'st_a00' dan *user environment* yang diproduksi oleh MatLAB, maka simulasi dapat dijalankan. Script Tcl yang dibangun disimpan dengan nama file 'tsel.tcl', dan berikut adalah command untuk menjalankannya.

```
ns tsel.tcl
```

NS-2 akan menghasilkan output trace umum yaitu 'tsel.tr' sesuai dengan *event-event* yang dilakukan oleh simulasi. Di samping itu, beberapa file parameter umum QoS seperti *delay*, *packet loss*, *cumulative jitter*, dan *inter-frame gap*, turut di-*generate* secara langsung melalui aplikasi NS-2 ini berupa file-file 'delay_a00e-a02e', 'loss_a00e-a02e', dan 'ratc_r_a00e.txt-a02e.txt. File-file ini

kemudian di-*compile* dan dianalisa lebih lanjut untuk memperoleh hasil akhir parameter QoS yang diinginkan.

3.6.6 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 [10]. Untuk perhitungan *throughput* ini dilakukan dengan perintah sebagai berikut:

```
awk –f InstantThroughput.awk tic=0.05 src=1 dst=2 flow=0 pkt=40 tsel.tr > analyze/tput1-2.out
awk –f instantThroughput.awk tic=0.05 src=1 dst=3 flow=1 pkt=40 tsel.tr > analyze/tput1-3.out
awk –f instantThroughput.awk tic=0.05 src=1 dst=4 flow=2 pkt=40 tsel.tr > analyze/tput1-4.out
```

3.6.7 Evaluasi File Trace

Langkah pertama dalam proses evaluasi adalah menghitung video PSNR referensi. PSNR referensi ini diambil dari video yang di-*code* dan di-*decode* (*codec*) tanpa adanya error atau *loss* pada transmisi yang 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 akiyo_qcif.yuv a01_ref.yuv > ref_psnr.txt
```

Langkah selanjutnya adalah merekonstruksi video yang ditransmisikan seperti yang terlihat oleh penerima. Untuk ini, file video dan file trace diproses oleh *etmp4*. Frame yang hilang diganti dengan angka 0 (jika –f tidak bekerja, gunakan -p).

```
./etmp4 -p 0 sd_a00 rd_a00 st_a00 a01.mp4 a00e
./etmp4 -p 0 sd_a01 rd_a01 st_a01 a01.mp4 a01e
./etmp4 -p 0 sd_a02 rd_a02 st_a02 a01.mp4 a02e
```

Ini akan meng-*generate* file video yang mungkin *corrupt*, dimana seluruh frame yang hilang atau *corrupt* akan dihapus dari track video asli. Sebenarnya, kedua file disimpan, file MP4 yang berisi video track yang *damaged* (a01c.mp4) dan file raw video yang berisi hanya frame yang *undamaged*. File-file ini di-

decode untuk memproduksi file YUV seperti terlihat di sisi penerima. Jika *decoder* yang layak dapat memproduksi YUV dengan banyak frame seperti aslinya, dapat mencoba:

```
./ffmpeg –i a00e.mp4 a00e.yuv
./ffmpeg –i a01e.mp4 a01e.yuv
./ffmpeg –i a02e.mp4 a02e.yuv
```

3.6.8 Kualitas Video Akhir

Output file YUV seharusnya berisi jumlah frame 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 di dalam sebuah directory, misalnya '/work'. PSNR pada video penerima dihitung dengan:

```
./psnr 176 144 420 akiyo_qcif.yuv a00e.yuv > work/psnr_01.txt
./psnr 176 144 420 akiyo_qcif.yuv a01e.yuv > work/psnr_02.txt
./psnr 176 144 420 akiyo_qcif.yuv a02e.yuv > work/psnr_03.txt
```

Karena hanya PSNR tidak banyak berarti, digunakanlah quality metric, yang dapat menghitung perbedaan antara kualitas *encoded video* dan video yang diterima (kemungkinan *corrupt*). Setelah dua pengukuran dan perhitungan telah dibuat seperti 'ref.psnr.txt', 'psnr_00.txt', 'psnr_01.txt', dan 'psnr_02.txt' letakkan file-file ini pada directory '/work', perintahnya adalah:

```
mos work ref_psnr.txt 25 > mos.txt
```

Hal ini menghitung rata-rata MOS dari setiap file psnr (kolom terakhir dari mos.txt) dan prosentase dari frame dengan MOS lebih buruk dari file referensi pada interval *sliding* 25 frame. Prosentase ini disimpan dalam file 'miv_00.txt', 'miv_01.txt', dan 'miv_02.txt'. Akhirnya perintah MIV menghitung prosentase maksimum dengan MOS yang lebih buruk dari aslinya:

```
miv work > miv.txt
```

3.6.9 Grafik dan Analisa

Untuk mempermudah proses analisa, data perhitungan akhir yang diperoleh yaitu *throughput*, *packet loss*, *delay*, dan *jitter* divisualisasikan ke dalam bentuk grafik. Proses pengolahan data ke dalam bentuk grafik ini dengan cepat dibuat dengan menggunakan Microsoft Excel 2007.