# Rancang Bangun RTP Packet-Chunk *De-encapsulator* Data AV Stream Format RTP Sebagai *Terminal Access Multi-Source Streaming Server*

Ahmad Budi Setiyawan[1], A.Subhan KH, ST[2],
[1]Mahasiswa Politeknik Elektronika Negeri Surabaya, Jurusan Teknik Telekomunikasi
[2] Dosen Politeknik Elektronika Negeri Surabaya Institut Teknologi Sepuluh Nopember
Kampus ITS, Surabaya 60111
e-mail : ahbuset@student.eepis-its.edu e-mail : subhankh@lecturer.eepis-its.edu,

**Abstrak** - Teknologi internet berbasis IP berkembang sangat pesat dan telah menjadi standar untuk sistem komunikasi data secara global. IP sangat baik dalam skalabilitas sehingga membuat teknologi internet cukup murah dan bisa menangani kebutuhan internet yang semakin meningkat. Namun, masalah mungkin terjadi ketika *streaming server* yang ada saat ini hanya dapat mengirimkan paket RTP dari satu sumber video saja. Sehingga *streaming server* yang sudah ada saat ini belum bisa dijadikan dasar pengembangan IPTV.

*Multi-Source Streaming Server* (MSS) merupakan konsep sistem streaming video yang memiliki beberapa sumber. Konsep *Multi-Source Streaming Server* diharapkan bisa sebagai acuan untuk mengembangkan IPTV.

Kata kunci : *Streaming server, Multi-Source Streaming Server*, IPTV

## 1. Pendahuluan

Pada saat ini sudah ada berbagai macam *streaming server* yang dibangun untuk mentransmisikan data *stream*. *Streaming server* mentransmisikan data secara real time dan bekerja menggunakan protokol RTP. Namun dibalik keunggulannya *streaming server* yang sudah ada, kebanyakan hanya bisa mentransmisikan data dari satu *source* saja. Dan tidak tertutup kemungkinan bahwa *streaming server* bisa dikembangkan, sehingga memiliki lebih dari satu *sourc*e. Konsep sistem *streaming server* yang memiliki banyak *source* ini disebut *Multi-Source Streaming Server* (MSS). Konsep MSS ini diharapkan bisa sebagai acuan untuk mengembangkan IPTV.

Maka pada tugas akhir ini dibuatlah rancang bangun RTP *packet-chunk* de-enkapsulator data AV stream format RTP sebagai *terminal access multi-source streaming server*. Tujuan dari proyek akhir ini adalah untuk merancang RTP *packet-chunk* de-enkapsulator data AV *stream* format RTP sebagai *terminal access multi-source streaming server*. Sehingga pada akhir perancangan ini diharapkan dapat diimplementasikan pada multi-source streaming server dalam arsitektur IPTV *peer-to-peer* (P2P).

## 2. Teori Penunjang

### Aplikasi Multimedia Networking

Aplikasi *multimedia networking* mengirimkan paket data yang memuat data audio atau video. Aplikasi *multimedia networking* juga disebut *continues-media applications*, karena pengirimannya yang secara real-time dan terus menerus. Aplikasi *multimedia networking* sangatlah sensitif terhadap delay, dimana delay harus tidak melebihi 400 milidetik.

Contoh aplikasi *multimedia networking* :

1. *Streaming stored audio and video* : Pada aplikasi jenis ini client melakukan *request on-demand* terhadap file audio atau video terkompresi yang telah tersimpan pada server.
2. *Streaming live audio/video*: Pada aplikasi jenis ini mirip dengan penyiaran radio dan televisi, kecuali transmisi melalui jaringan internet. Pada aplikasi ini *user* diijinkan untuk menerima transmisi radio dan televisi yang dipancarkan dari penjuru bumi. Biasanya, ada banyak user yang secara bersamaan menerima audio/video secara real-time.
3. *Real-time interactive audio/video* : Pada aplikasi jenis ini mengijinkan client

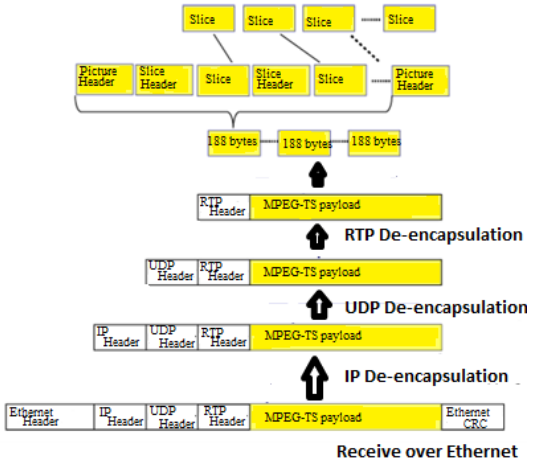
menggunakan audio/video untuk berkomunikasi dengan setiap client yang lain secara real-time. Dengan video interaktif, juga disebut *video conferencing*, setiap client dapat berkomunikasi visual maupun secara lisan.

***De-encapsulasi***

*De-encapsulasi*, terjadi saat client tujuan menerima paket data. Data mengalir ke atas dari layer yang lebih rendah ke layer yang lebih tinggi dari TCP/IP protocol. Setiap layer membuka header yang cocok dan menggunakan informasi yang dikandungnya untuk disampaikan ke layer di atasnya sampai pada layer tertinggi atau layer aplikasi mendapatkan data dalam jaringan sesungguhnya.

Proses kerja de-enkapsulasi :

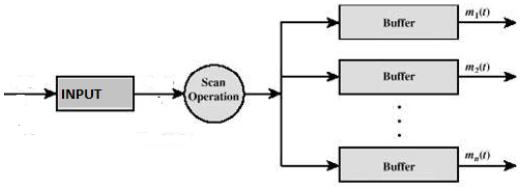
1. Pada tujuan, bit stream ini kemudian diubah menjadi FRAME.
2. FRAME-header kemudian dilepas dan dikirim ke layer-3 sebagai PAKET.
3. Paket selanjutnya melepas Header dan mengirim data tersebut ke layer-4 sebagai SEGMENT.
4. SEGMENT kemudian melepas layer-4 header dan memberikan data ke layer-5,6,7 yang akhirnya diterima oleh user sebagai data.
5. Proses pelepasan header dari layer ke layer disebut sebagai De-enkapsulasi.




**Gambar 1.** Blok Diagram RTP *De-encapsulation*

***Demultiplexing***

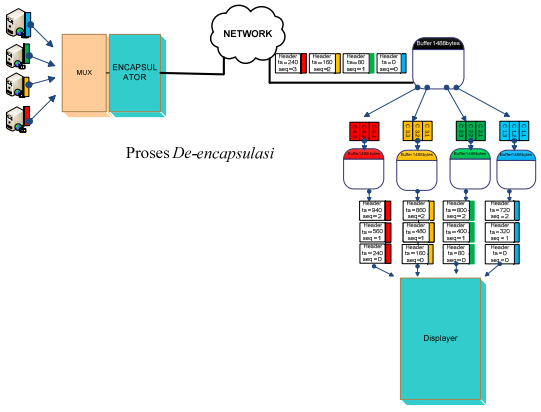
*Demultiplexer* mengambil data dari sebuah input kanal dan membagi data tersebut menjadi beberapa output, dimana pada setiap output yang akan ditransferkan ditentukan oleh informasi yang dikandung oleh paket tersebut. Data yang dikeluarkan di output sama seperti urutan yang diterima oleh input, terlepas dari saluran, paket, frame atau sinyal yang lain.




**Gambar 2.** *Demultiplexer*

## 3. Perancangan Sistem

Secara umum proyek akhir ini dibuat sesuai dengan Gambar 3. Setelah data diterima oleh Ethernet, proses *de-encapsulasi* dilakukan oleh library oRTP. Proses *demultiplexing* dilakukan pada RTP *de-encapsulasi*. *Demultiplexing* dilakukan untuk memisahkan bagian dari satu video ke video yang lain dengan cara melihat timestamp yang telah ditambahkan oleh library oRTP dibagian *multiplexer*-nya.

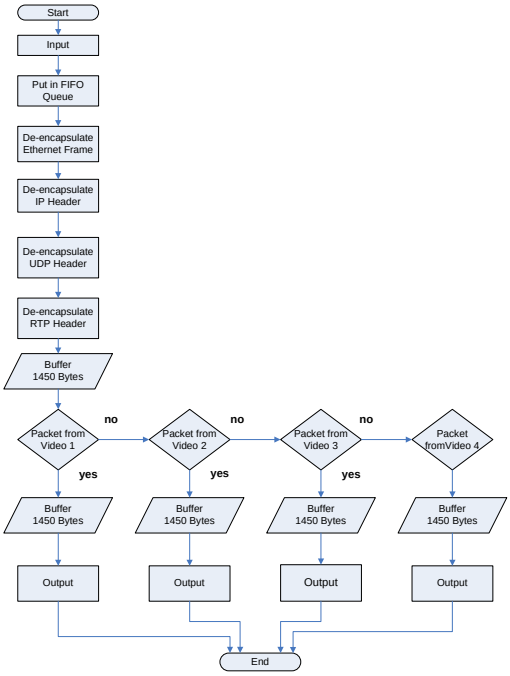



**Gambar 3.** Blok Diagram *De-encapsulator*

**Tahap Perencanaan Flowchart**

Pada pembuatan proyek akhir ini dimulai dari pemrograman untuk mendesain algoritma yang diimplementasikan untuk proses de-enkapsulasi paket RTP dan

pemisahan *RAW* video. Flowchart sistem ditunjukan pada gambar 4.



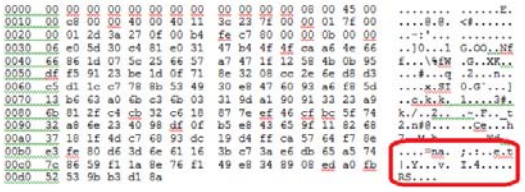

**Gambar 4.** Flowchart Sistem

## 4. Pengujian dan Analisa

- **Pengujian Demultiplexing Pengiriman Packet**

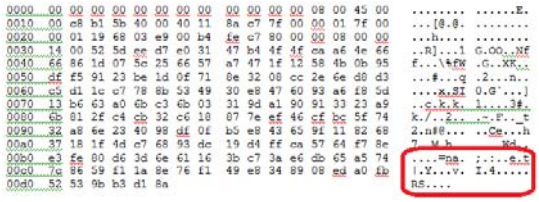
  Pada pengujian *demultiplexing* ini, *multiplexer* mengirimkan 4 paket RTP secara berurutan yang berasal dari video yang berbeda-beda. Setelah keempat video diterima oleh demultiplexer maka keempat paket tersebut dipisah dan dikirimkan lagi ke *terminal access* dengan nomor port berbeda-beda. Analisa dalam pengujian ini menggunakan tool wireshark untuk melihat apakah paket yang diterima dengan paket yang dikirim memiliki ujung-ujung yang sama. Berikut adalah hasil *capture* oleh wireshark :

1. Paket pertama yang diterma oleh demultiplexer :



**Gambar 5.** Paket Sebelum Dikirim

2. Paket yang dikirimkan kembali ke terminal access



**Gambar 6.** Paket Setelah Diterima

Dengan melihat hasil *capture* diatas maka *demultiplexer* paket bisa berjalan dengan baik, karena *demultiplexer* dapat memisahkan ujung-ujung dari payload tanpa mengurangi isi payload itu sendiri. Hal ini dapat dibuktikan dengan melihat ujung akhir dari payload, disini terlihat bahwa ujung akhir dari setiap payload saat diterima maupun dikirim selalu sama. Sedangkan untuk ujung depan terlihat berbeda karena oleh *demultiplexer* header paket yang ditambahkan oleh *multipxer* dihapus dan diganti dengan yang baru (diisi dengan *timestamp* dan *sequence number* yang berbeda).

- **Analisa MTU dan Penambahan RTP Header**

Pengujian ini bertujuan untuk mengetahui berapa besar maksimum data/payload yang dapat dikirimkan, supaya diterima dengan baik di tujuan. *Maximum Transmission Unit* (MTU) untuk paket RTP adalah data/payload ditambah 12 bytes untuk RTP header. Analisa MTU bisa dilakukan dengan mengubah-ubah ukuran buffer, karena dengan mengubah ukuran buffer maka paket yang dikirimkan juga ikut berubah.

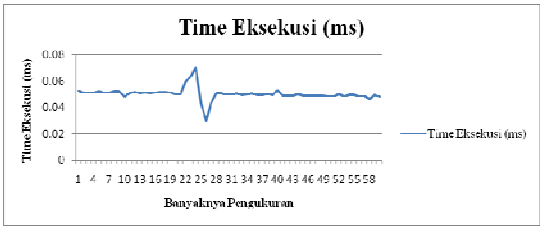
**Tabel 1.** Pengaruh Ukuran Data, RTP header, MTU Terhadap Kondisi Video

| Ukuran Buffer | Ukuran Data (Bytes) | RTP header (Bytes) | MTU (Bytes) | Kondisi Video |
|---|---|---|---|---|
| 160 | 160 | 12 | 172 | Baik |
| 320 | 320 | 12 | 332 | Baik |
| 480 | 480 | 12 | 492 | Baik |
| 640 | 640 | 12 | 652 | Baik |
| 800 | 800 | 12 | 812 | Baik |
| 960 | 960 | 12 | 972 | Baik |
| 1120 | 1120 | 12 | 1132 | Baik |
| 1280 | 1280 | 12 | 1292 | Baik |
| 1440 | 1440 | 12 | 1452 | Baik |
| 1488 | 1488 | 12 | 1500 | Baik |
| 1489 | 1489 | 12 | 1501 | Rusak |
| 1600 | 1600 | 12 | 1612 | Rusak |

Dari Tabel 1 menunjukkan bahwa pada jaringan berbasis teknologi ethernet, ukuran MTU maksimum adalah 1500 byte. Dari 1500 byte tersebut terdiri dari 1488 byte payload serta penambahan 12 byte untuk RTP header. Jika paket yang dikirimkan melebihi MTU maka data yang diterima oleh client akan mengalami kerusakan meskipun itu lebih 1 byte saja.

- **Pengukuran Time Eksekusi *Demultiplexer***

Dari grafik dibawah diperoleh informasi bahwa maksimal waktu yang dibutuhkan untuk melakukan *demultiplexing* adalah 0.070736 milidetik, sedangkan waktu minimal yang dibutuhkan adalah 0.030242 milidetik. Sedangkan rata-rata *time eksekusi* dalam interval 1 menit adalah 0.050401567 milidetik. Karena *time eksekusi* masih di bawah 150 milidetik yang merupakan *delay* maksimal untuk QoS audio/video streaming, maka algoritma yang digunakan sudah memenuhi untuk QoS Audio/Video Streaming.



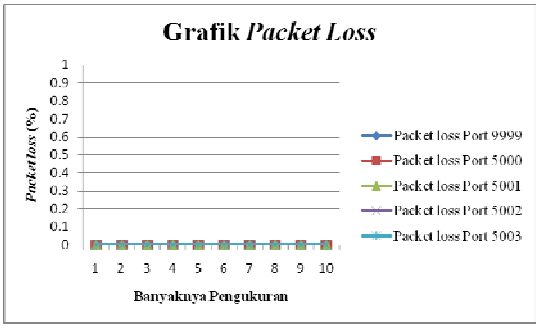

**Gambar 7.** Grafik Nilai Time Eksekusi pada *Demultiplexer*

- **Pengukuran Audio/Video Streaming**

Pengukuran pada pengiriman paket RTP jaringan *multi-source* ini merupakan penjabaran dari nilai parameter QoS hasil pengukuran dari paket yang diterima dari *multiplexer* maupun paket yang dikirimkan *demultiplexer* ke *displayer*. Parameter-parameter QoS yang diukur meliputi nilai *packet loss*, *delay*, *jitter* dan *throughput.*

1. *Packet Loss*

Pengukuran dilakukan dengan mengambil data pada sesi komunikasi antara *demultiplexer* dengan *multiplexer* pada port 9999 dan komunikasi antara *demultiplexer* dengan *displayer* pada port 5000, port 5001, port 5002 dan port 5003. Pada komunikasi tersebut data yang diperoleh adalah sebagai berikut :



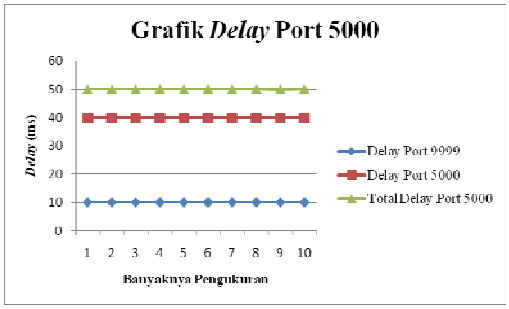

**Gambar 8.** Grafik Nilai *PacketLloss* pada *Demultiplexer*

Berdasarkan grafik Gambar 8, *packet loss* dalam komunikasi tersebut dapat dikategorikan komunikasi yang sangat bagus. Dapat dikategorikan sangat bagus apabila *packet loss* yang terjadi 0%. Hal ini terlihat pada data diatas, pada masing–

masing port terjadi packet loss 0%. Packet loss 0% bisa terjadi karena jalur yang digunakan dalam komunikasi dalam status tidak sibuk atau dengan kata lain tidak ada pembebanan paket data yang tinggi.

2. *Delay*

Komunikasi pada port 9999 merupakan pengiriman paket RTP yang dilakukan oleh *multiplexer* dan kemudian diterima oleh *demultiplexer*. Paket yang dikirimkan oleh *multiplexer* ini merupakan paket RTP yang telah di-*multiplexing*. Sedangkan komunikasi pada port 5000 merupakan pengiriman paket RTP ke *displayer* oleh d*emultiplexer* setelah paket dilakukan *demultiplexing*.




**Gambar 9.** Grafik Nilai *Delay* pada *Demultiplexer*

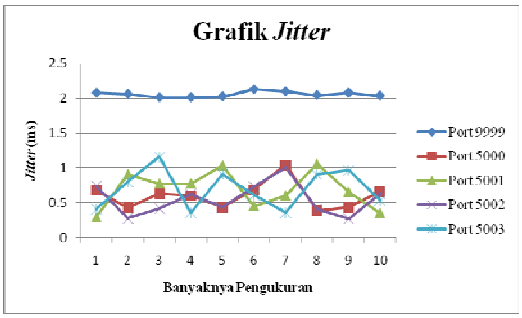
**Port 5000 :**

X = Total Delay + Time Eksekusi
= 49.9858614 + 0.050401567
= 50.036263 milidetik

Standarisasi *delay* streaming video *one way* (satu arah) menurut ITU-T G.1010 adalah lebih kecil 150 milidetik. Pada perhitungan diatas, terlihat jelas bahwa pada sistem ini rata-rata *delay* yang terjadi sebesar 50 milidetik. Karena rata-rata *delay* yang terjadi sebesar 50 milidetik, maka pada sistem ini dapat dikatakan masih layak.

3. *Jitter*

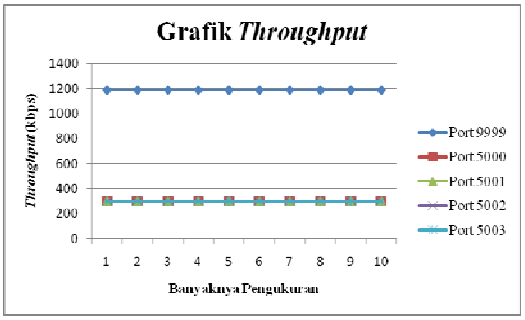
Berdasarkan grafik Gambar 10 dapat diketahui bahwa nilai jitter pada port 9999 lebih besar dibandingkan ke 4 port yang lain. Hal ini disebabkan karena pada port 9999 memiliki pembebanan paket RTP 4 kali lebih besar dibanding dengan 4 port yang lain. Pada port 9999 mengirimkan paket RTP yang yang berasal dari 4 sumber video, sedangkan port yang lain hanya mengirimkan paket RTP dari 1 sumber video saja.




**Gambar 10.** Grafik Nilai *Jitter* pada *Demultiplexer*

4. *Throughput*

Data dari grafik Gambar 11, terlihat bahwa besar *throughput* untuk paket video berada pada range 1187.53 kbps pada port 9999, range 302.75 pada port 5000, range 302.73 pada port 5001, range 302.79 pada port 5002 dan 302.72 pada port 5003. Perbedaan mencolok *throughput* pada port 9999 dengan port yang lain disebabkan karena pada port 9999 tidak mengalami pembagian bandwidth dalam satu kanal. Sebaliknya dengan port 5000, port 5001, port 5002 dan port 5003 yang mengalami pembagian bandwidth dalam satu kanal. Karena ada pembagian kanal untuk 4 port mengakibatkan *throughput* yang diperoleh 4 kali lebih kecil.




**Gambar 11.** Grafik Nilai *Throughput* pada *Demultiplexer*

## 5. Kesimpulan

Dari hasil pengujian dan analisa diperoleh beberapa kesimpulan sebagai berikut :

1. Proses *demultiplexer* bisa dilakukan dengan cara pemisahan setiap blok paket data dengan ukuran buffer yang sama antara pengirim dan penerima.
2. Paket yang dikirmkan melebihi MTU sebesar 1500 byte akan mengalami kerusakan saat diterima *receiver*.
3. Rata-rata *packet loss* yang terjadi pada jaringan *peer to peer* adalah 0% yang sudah memenuhi standar ITU-T Recommendation G.1010 yaitu dibawah 1%.
4. Rata-rata nilai *delay* yang terukur pada jaringan *peer to peer* adalah 50 milidetik, sehingga sudah memenuhi standar ITU-T Recommendation G.1010 yaitu dibawah 150 milidetik.
5. Rata–rata nilai *jitter* pada jaringan *peer to peer* untuk komunikasi antara *multiplexer* dan *demultiplexer* sebesar 2 milidetik sedangkan komunikasi antara *demultiplexer* dan *displayer* lebih kecil, *yaitu* sebesar 0.7 milidetik.
6. Rata–rata nilai *throughput* pada jaringan *peer to peer* untuk komunikasi antara *multiplexer* dan *demultiplexer* sebesar 1187 kbps sedangkan komunikasi antara *demultiplexer* dan *displayer* lebih kecil, *yaitu* sebesar 302 kbps.

**Daftar Pustaka :**


[1] F.Kurose, James dan W. Ross, Keith , “Computer Networking”, Addison-Wesley , 1999-2000.

[2] Taweewit Pensawat, “ Real-Time Ethernet Networks Simulation Model”, Journal Halmstad University, December 21, 2006.

[3] http://www.grogy.com/local_doc/share/doc/ortp/ortpapi.html

[4] http://www.cs.arizona.edu/projects/scout/Papers/mosberger/doc023.html

[5] http://www.networksorcery.com/enp/protocol/rtp.htm

[6] Firmansyah, Rizal Aulia, ”Aplikasi Video Switch Berbasis Web pada EEPIS I-Studio”, Proyek Akhir PENS-ITS, Juli 2008.

[7] Bilton, Fin Tho’at, ” Rancang Bangun IPTV Set Top Box pada EEPIS I-Studio”, Proyek Akhir PENS-ITS, Juli 2008.