RANCANG BANGUN TERMINAL ACCESS MULTI-SOURCE STREAMING SERVER DENGAN JAVA MEDIA FRAMEWORK

Heru Surya Pratama[1], A. Subhan KH, ST[2],
[1]Mahasiswa Politeknik Elektronika Negeri Surabaya, Jurusan Teknik Telekomunikasi
[2]Dosen Politeknik Elektronika Neheri Surabaya, Jurusan Teknik Telekomunikasi
Kampus ITS, Keputih Sukolilo, Surabaya 60111
e-mail : heru06@student.eepis-its.edu e-mail : subhankh@eepis-its.edu

Abstrak - Pada saat ini kebanyakan *streaming server* dibangun dengan hanya memiliki satu *source*. Tidak tertutup kemungkinan bahwa *streaming server* bisa memiliki lebih dari 1 *sourc*e. Konsep sistem *streaming server* yang memiliki banyak *source* ini disebut *Multi-source Streaming Server* (MSS).

Untuk dapat menikmati video *streaming* dari sistem MSS ini maka dibutuhkan sebuah *terminal access*. *Terminal access* ini memiliki beberapa bagian yaitu blok *de-encapsulator*, demultiplexer, dan *displayer*. Konsep MSS yang memiliki beberapa *streaming server* sebelum mentransmisikan data, maka akan dimultiplexing agar AV *stream* dari beberapa *streaming server* akan dapat ditransmisikan ke dalam jalur yang sedikit. Pada saat multiplexing satu paket AV *stream* memiliki beberapa AV *stream* dari beberapa streaming server. Tugas demultiplexer adalah agar paket yang ditransmisikan tadi dapat menjadi AV *stream* sesuai dengan aslinya yaitu dari streaming server mana . Seperti aplikasi pemutar *video streaming* pada umumnya yang memiliki fasilitas *play*, *pause*, dan *stop* maka aplikasi ini nantinya akan memiliki fasilitas yang sama. Karena sistem MSS memiliki banyak *streaming server* maka aplikasi pemutar ini memiliki fasilitas tambahan untuk memilih video *streaming* dari *streaming server* mana yang ingin ditampilkan.

Hasil yang diharapkan adalah berupa aplikasi pemutar video *streaming* dimana *user* dapat menentukan pilihan video yang ingin dilihat dari *streaming server* yang dipilih, dan *user* yang melihat video mendapatkan kualitas video yang baik.

Kata kunci : *streaming server*, AV *stream*, MSS, JMF, *terminal access*

1. PENDAHULUAN

Pada saat ini banyak layanan multimedia berkembang di internet. Ini disebabkan karena pengembangan *broadband access* untuk pengguna akses internet di rumah. Salah satu layanan multimrdia itu adalah video *streaming*. Dengan video *streaming*, *user* dapat melihat video tanpa harus men-*download*-nya terlebih dahulu.

Sekarang kebanyakan *Streaming Server* (SS) dibangun dengan hanya memiliki satu *source*. Akibatnya adalah jika ingin membangun SS dengan multi layanan sangatlah sulit. Tapi tidak tertutup kemungkinan bahwa SS bisa memiliki lebih dari satu *sourc*e. Konsep sistem *streaming server* yang memiliki banyak *source* ini disebut *Multi-source Streaming Server* (MSS). Sehingga nantinya dengan MSS dapat diterapkan sistem dengan multi layanan.

Untuk menikmati layanan MSS maka dibutuhkan sebuah *terminal access* untuk menampilkan video *streaming*. Dimana *terminal access* pada sistem MSS ini memiliki beberapa bagian yaitu *de-encapsulator*, demultiplexer, dan *displayer*. Akibat dari sistem MSS yang memiliki beberapa *streaming server* dan jalur kepada *user* yang tersedia hanya satu maka sebelum AV *stream* ditransmisikan, data *stream* tersebut akan mengalami proses multiplexing. Tugas dari demultiplexer ini adalah untuk membuat bagaimana data *stream* yang diterima menjadi AV *stream* dari bererapa *streaming server*. Sedangkan fungsi *displayer* adalah untuk menampilkan video *streaming* dan nantinya *user* yang menggunakan *displayer* dapat memilih video dari *streaming server* mana yang ingin dilihat.

2. TEORI

2.1 Pemrograman Java

Java adalah bahasa pemrogra-man yang dapat dijalankan di berbagai komputer termasuk telepon genggam. Dikembangkan oleh Sun Microsystems dan

diterbitkan tahun 1995. Java tidak boleh disalahpahami sebagai JavaScript. JavaScript adalah bahasa scripting yang digunakan oleh web browser.

Bahasa pemrograman Java pertama lahir dari The Green Project, yang berjalan selama 18 bulan, dari awal tahun 1991 hingga musim panas1992. Proyek tersebut belum menggunakan versi yang dinamakan Oak. Proyek ini dimotori oleh Patrick Naughton, Mike Sheridan, James Gosling dan Bill Joy, beserta sembilan pemrogram lainnya dari Sun Microsystems. Salah satu hasil proyek ini adalah maskot *Duke* yang dibuat oleh Joe Palrang.

Bahasa pemrograman Java memilik beberapa kelebihan dibandingkan dengan bahasa pemrograman lainnya, yaitu sebagai berikut:

1) *Multiplatform*.
2) OOP *(Object Oriented Programming*)
3) Perpustakaan Kelas Yang Lengkap,.
4) Bergaya C++.
5) Pengumpulan sampah otomatis.

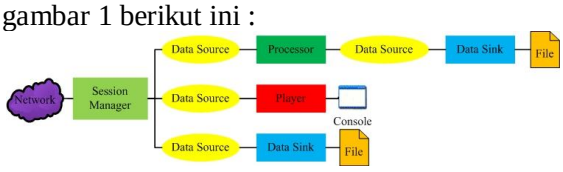
2.2 Java Media Framework

Java Media Framework (JMF) API merupakan arsitektur yang menggabungkan protokol dan pemrograman *interface* untuk merekam, mentransmisi, dan mem-*playback* media.

Beberapa dari fungsi JMF, yaitu :

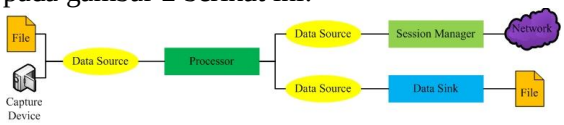
1) Dapat digunakan untuk berbagai file multimedia pada Java Applet atau aplikasi..
2) *Play* media *streaming* dari internet.
3) *Capture* audio dan video dengan mikropon dan kamera video kemudian menyimpan data tersebut kedalam format yang mendukungnya.
4) Mengirimkan audio dan video secara *realtime* ke dalam jaringan internet atau intranet.
5) Dapat digunakan untuk pemrograman penyiaran radio atau televisi secara langsung.

Mekanisme RTP Receive dapat dilihat pada gambar 1 berikut ini :



Gambar 1 RTP Reception

Pada RTP Reception ini, RTP digunakan untuk mengirimkan hasil capture atau menyimpan media stream dalam network. Mekanisme RTP Transmision dapat dilihat pada gambar 2 berikut ini:



Gambar 2 RTP Transmision

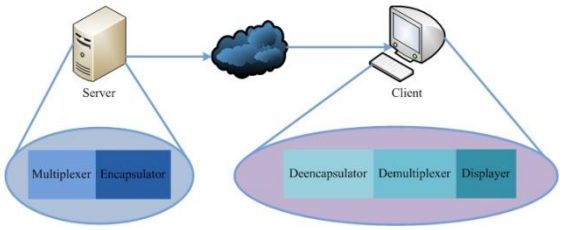
Kita dapat melakukan pengiriman paket stream dengan menggunakan RTP session yang terpisah dari macam-macam tipe media (capture device) [1].

JMF 2.1.1 API merupakan rilis terbaru dari Sun's. *Interface* dan *Class* untuk koneksi *high-level* API dalam pemrograman JMF adalah sebagai berikut:

1) DataSource
2) Capture Device.
3) Player
4) Processor
5) DataSink
6) Format
7) Manager

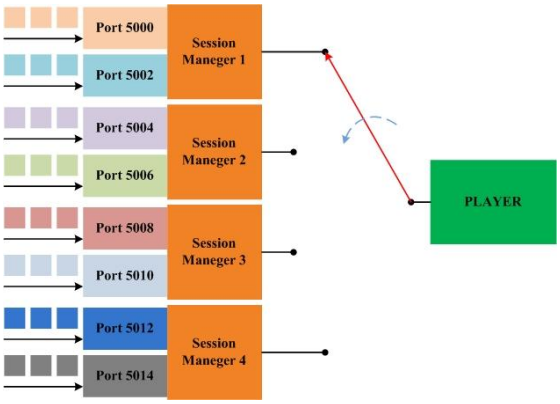
3. Perancangan Sistem

Sistim utama pada proyek akhir ini adalah membuat dan mengimplementasikan Multi-source Streaming Server. Secara umum blok diagram sistem ditunjukan seperti pada Gambar 4:



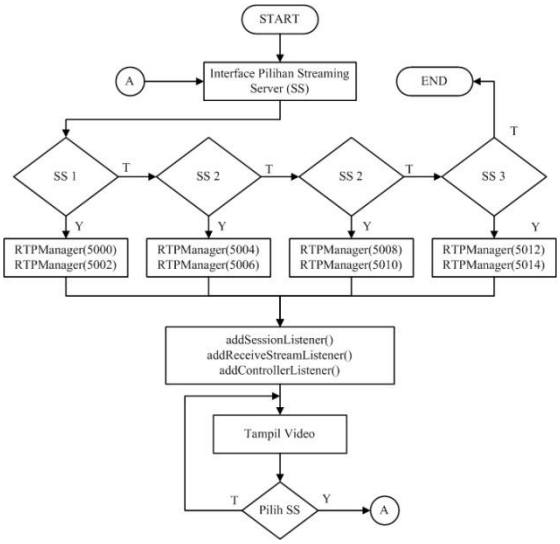
Gambar 3 Blok Diagram Sistem MSS

Pada proyek akhir ini blok sistem yang dikerjakan adalah blok displayer yang dapat menampilkan video streaming dari sistem Multi-source Streaming Server. Gambar berikut adalah blok sistem displayer:



Gambar 4 Blok Sistem *Dispalyer*

Untuk melihat video streaming dari multi-source streaming server dibuat sebuah *displayer* yang dibuat dengan *java media framework* . Displayer dapat menangani permintaan *user* untuk melihat video dari streaming server mana yang diinginkan. Prinsip kerjanya adalah jika user memilih untuk melihat video dari streaming server 1 maka *displayer* akan mengambil data AV stream dari sistem session manager streaming server 1 dan kemudian menampilkannya. Jika user memilih untuk melihat video dari streaming server 2 maka langkahnya sama yaitu *displayer* akan mengambil data AV stream dari sistem session manager streaming server 2 dan menampilkannya. Begitu seterusnya.



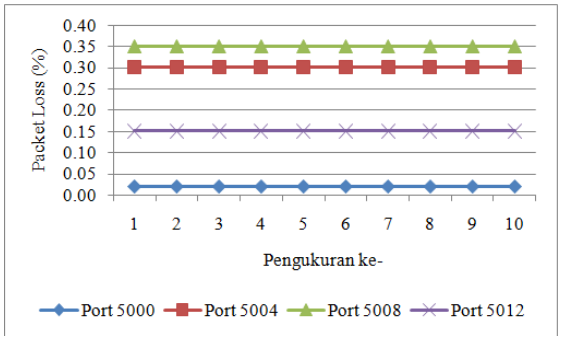
Gambar 5 *Flowchart* Sistem

4. Pengujian dan Analisa

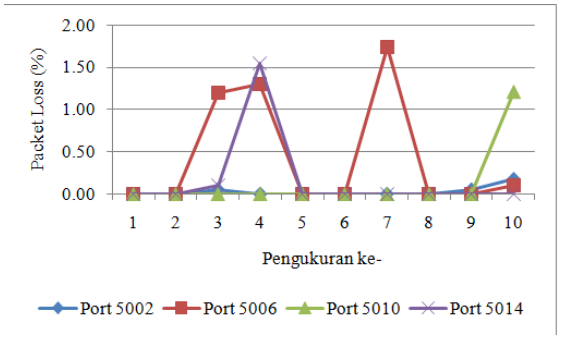
4.1 Pengujian Kualitas Layanan

Pengujian ini dilakukan agar dapat mengetahui apakah sistem dapat memberi layanan dengan baik. Parameter-parameter yang akan diuji adalah: *packet loss*, *delay*, *jitter* dan *throughput*. Parameter-parameter tersebut diukur dengan menggunakan aplikasi wireshark dengan derasi peng-*capture*-an selama 2 menit dan dilakukan sebanyak 10 kali.

- *Packet Loss*



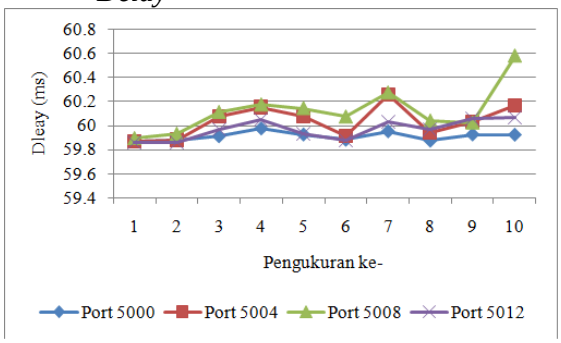
Gambar 6 Grafik *Packet Loss* Sesi Audio



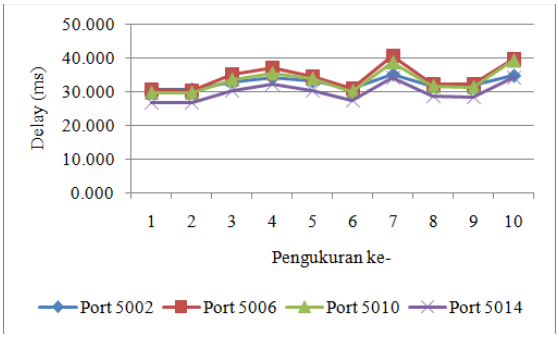
Gambar 7 Grafik *Packet Loss* Sesi Video

Data hasil pengukuran ini dapat dikatakan telah memenuhi standar karena berdasarkan standar ITU-T Recommendation G.1010 nilai *packet loss* adalah dibawah 1%. Ini bisa terjadi karena sistem ini memakai protokol UDP sebagai *transport protocol*-nya yang melakukan sesi komunikasi satu arah sehingga jarang kemungkinan terjadi *collision* dan *congestion*.

- *Delay*



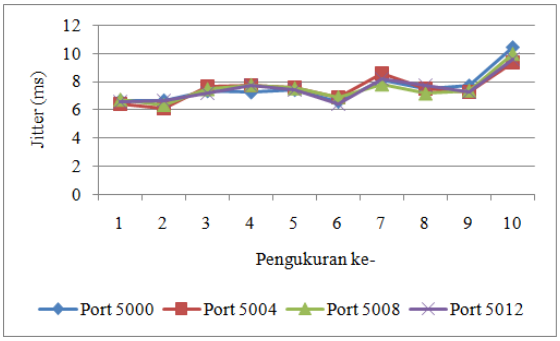
Gambar 8 Grafik *Delay* Sesi Audio



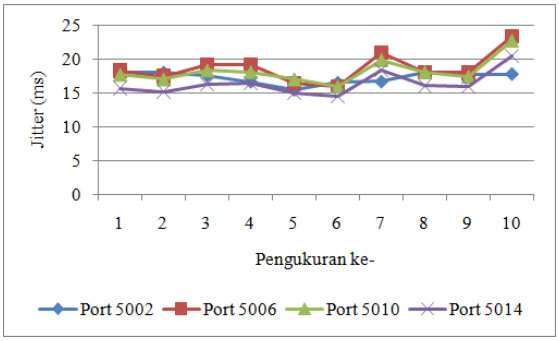
Gambar 9 Grafik *Delay* Sesi Video

Data hasil pengukuran ini dapat dikatakan telah memenuhi standar karena berdasarkan standar ITU-T Recommendation G.1010 nilai *delay* adalah dibawah 10 detik. Terlihat bahwa *delay* sesi audio lebih besar dari pada sesi video. Ini disebabkan karena paket stream audio dihasilkan lebih sedikit dari paket stream video pada waktu yang bersamaan. Dengan kata lain server sudah mengirim satu paket stream audio sedangkan paket stream video sudah terkirim lebih dari satu. Akibatnya waktu pengiriman stream audio lebih lama dibandingkan stream video.

- *Jitter*



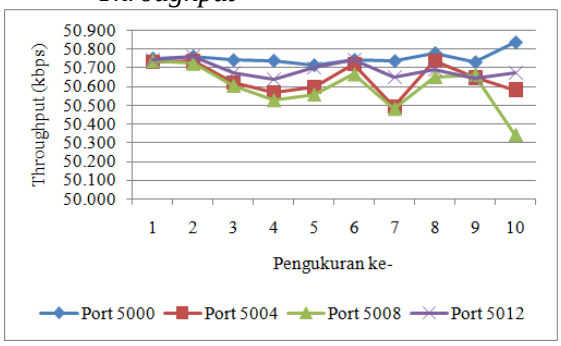
Gambar 10 Grafik *Jitter* Sesi Audio



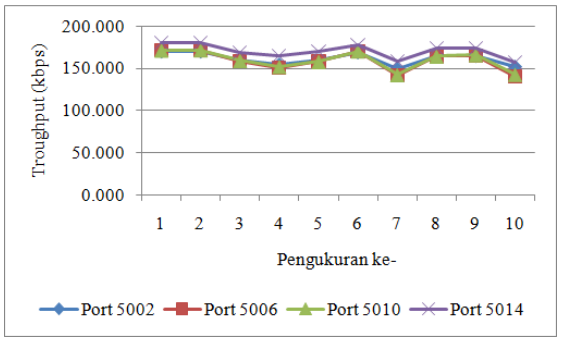
Gambar 11 Grafik *Jitter* Sesi Video

Data hasil pengukuran ini tidak bisa langsung dikatakan telah memenuhi standar karena sesuai standar ITU-T Recommendation G.1010 nilai *jitter* tidak ada. Server menghasilkan stream audio lebih sedikit dibandingkan dengan stream video. Akibatnya data yang masuk ke *port* sesi video akan mengalami antrian yang cukup panjang. Sehingga nilai *jitter* pada sesi video akan lebih besar dibandingkan dengan nilai *jitter* sesi audio, ini sesuai dengan data yang didapatkan.

- *Throughput*



Gambar 12 Grafik *Throughput* Sesi Audio

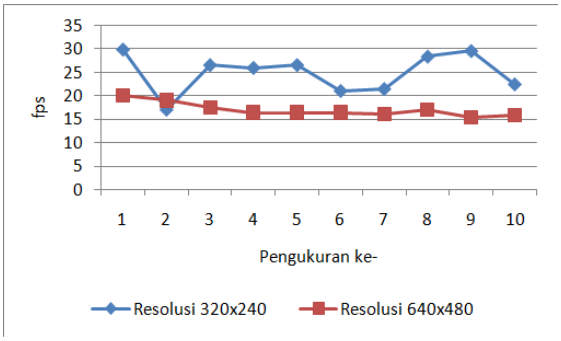


Gambar 13 Grafik *Throughput* Sesi Video

Dari hasil pengukuran ini dapat dilihat bahwa nilai *throughput* sesi audio berada pada kisaran yang sama yaitu ~50 kbps, sedangkan *throughput* sesi video bervariasi dari 140-180 kbps. Nilai *throughput* sesi video lebih besar dibandingkan sedi audio, penyebabnya adalah karena paket stream sesi video yang datang lebih banyak dibandingkan paket stream sesi audio. Selain itu panjang stream sesi audio yang diterima untuk kasus ini selalu sebesar 716 *byte* sedangkan panjang stream sesi video yang diterima bervariasi dari 400-1040 *byte*.

4.2 Pengukuran Frame Per Second

Pengujian ini dilakukan untuk mengetahui berapa frame *per second* (fps) yang diterima oleh sistem. Untuk pengukuran pertama sistem menerima paket stream video resolusi 320x240 dan untuk pengukuran kedua resolusinya diganti menjadi 640x480. Berikut grafik hasil pengukuran.

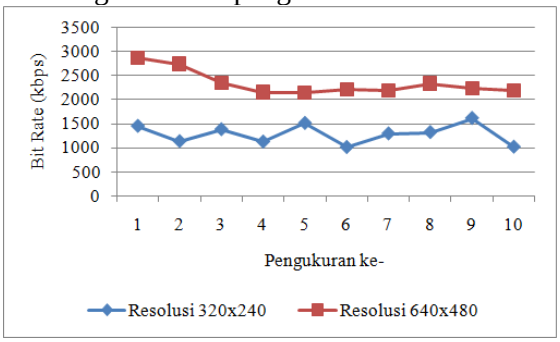


Gambar 14 Grafik *Frame Per Second*

Berdasarkan grafik dapat dikatakan hasil fps dari stream video resolusi 320x240 lebih baik dari stream video resolusi 640x480. Ini disebabkan karena total paket stream video resolusi 320x240 lebih kecil dibandingkan dengan yang resolusi 640x480. Akibatnya kualitas video dengan resolusi yang kecil menghasilkan kualitas yang lebih baik dari video dengan resolusi yang lebih besar.

4.3 Pengukuran Bit Rate

Sama dengan pengujian fps, pengukuran *bit rate* ini akan mengukur stream video resolusi 320x240 dan video resolusi 620x480. Berikut grafik hasil pengukuran.



Gambar 15 *Grafik Bit Rate*

Berdasarkan grafik dapat diamati bahwa *bit rate* video resolusi 320x240 lebih kecil dari video resolusi 640x480. Semakin besar *bit rate* akan menghasilkan kualitas video yang semakin baik dan datanya juga akan semakin besar. Jika data semakin besar maka paket yang dikirimkan juga semakin besar. Akibatnya jaringan akan terbebani dengan banyaknya data, sehingga kemungkinan paket hilang akan semakin besar. Dengan *bit rate* yang besar memang akan menghasilkan kualitas video yang baik, tetapi jika di-streaming-kan maka video yang tampil seperti video yang mengalami efek *slow motion*.

4.4 Pengujian Interface

Pengujian ini dilakukan untuk melihat tampilan video dari streaming server, berikut ini tampilan utamanya:

Gambar 16 Interface Pemilihan Streaming Server

Dengan memanfaatkan interface diatas kita dapat memilih video dari streaming server yang kita inginkan. Jika mengklik tombol Streaming Server Satu, maka video dari streaming server satu akan tampil seperti berikut ini:

Gambar 17 Video Streaming Server Satu

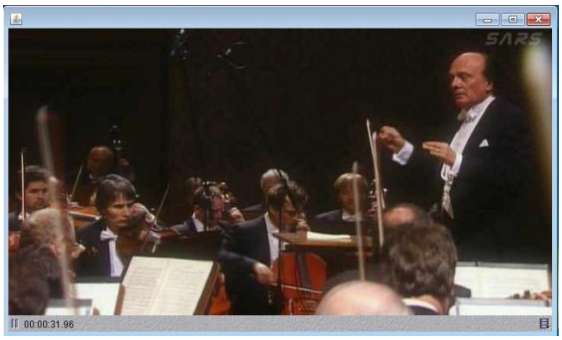
Jika mengklik tombol Streaming Server Dua maka video dari streaming server dua akan tampil dan hasilnya seperti ini:

Gambar 18 Video Streaming Server Dua

Jika mengklik tombol Streaming Server Tiga maka video dari streaming server tiga akan tampil dan hasilnya seperti ini:

Gambar 4.19 Video Streaming Server Tiga

Jika mengklik tombol Streaming Server Empat maka video dari streaming server empat akan tampil dan hasilnya seperti ini:



Gambar 4.20 Video Streaming Server Empat

Waktu perpindahan saat mengganti video tidak membutuhkan waktu yang lama yaitu sekitar 2 detik, ini sesuai dengan waktu *buffering*-nya. Video tidak akan langsung tampil tapi akan menunggu selama 2 detik, ini membuat kita dapat melihat video dengan waktu yang hampir bersamaan dengan yang di-*play*-kan oleh server. Atau bisa dikatakan apa yang kita lihat di server, maka client akan melihat hal yang sama.

5. Kesimpulan

Dari hasil pengujian dan analisa diperoleh beberapa kesimpulan sebagai berikut:

1. Rata-rata *packet loss* yang terjadi pada kedua sesi adalah dibawah 0.5%, yang sudah memenuhi standar dari ITU-T Recommendation G.1010 yaitu dibawah 1%.
2. Rata-rata nilai *delay* yang terukur pada sesi audio adalah dibawah 60.5 ms, sedangkan pada sesi video adalah dibawah 34.5 ms dan kedua sesi sudah memenuhi standar dari ITU-T Recommendation G.1010 yaitu dibawah 10 detik.
3. Rata-rata nilai *jitter* yang terukur pada sesi audio adalah dibawah 7.6 ms, sedangkan pada sesi video adalah dibawah 19 ms.
4. Rata-rata nilai *throughput* yang terukur pada sesi audio adalah mendekati nilai 51 kbps, sedangkan pada sesi video adalah bervariasi mulai dari 158 – 170 kbps.
5. Rata-rata nilai fps yang terukur untuk resolusi 320x240 adalah 24.83 fps dan untuk resolusi 640x480 adalah 16.99 fps.
6. Rata-rata nilai *bit rate* yang terukur untuk resolusi 320x240 adalah 1278.834 kbps dan untuk resolusi 640x480 adalah 2339.352 kbps.
7. Waktu perpindahan video dan *buffering* tidak membutuhkan waktu yang lama karena nilainya berkisar 2 detik.

6. Daftar Pustaka

[1] http://journal.uii.ac.id/index.php/Snati/article/viewFile/835/766
[2] http://java.sun.com/products/java-media/jmf2.1.1/guide/