ANALISIS PERBANDINGAN UNJUK KERJA

PROTOKOL TCP, UDP, DAN SCTP MENGGUNAKAN SIMULASI

LALU LINTAS DATA MULTIMEDIA

TUGAS AKHIR

Nama : **Rinda Tri Yuniar Anggraeni**

NIM : **09. 41020. 0038**

Program : **S1 (Strata Satu)**

Jurusan : **Sistem Komputer**

SEKOLAH TINGGI

MANAJEMENINFORMATIKA &TEKNIK KOMPUTER

SURABAYA

2013

Tugas Akhir

ANALISIS PERBANDINGAN UNJUK KERJA
PROTOKOL TCP, UDP, DAN SCTP MENGGUNAKAN SIMULASI
LALU LINTAS DATA MULTIMEDIA

dipersiapkan dan disusun oleh

Rinda Tri Yuniar Anggraeni
NIM : 09.41020.0038

Telah diperiksa, diuji dan disetujui oleh Dewan Penguji
pada : Maret 2013

Susunan Dewan Penguji

Pembimbing

I. **Dr. Jusak** _____

II. **Anjik Sukmaaji, S.Kom., M.Eng** _____

Penguji

I. **Tutut Wurijanto, M.Kom** _____

II. **Yuwono Marta Dinata, S.T., M.Eng** _____

Tugas Akhir ini telah diterima sebagai salah satu persyaratan
untuk memperoleh gelar Sarjana

Pantjawati Sudarmaningtyas, S.Kom, M.Eng, OCA
Pembantu Ketua Bidang Akademik

SEKOLAH TINGGI MANAJEMEN INFORMATIKA DAN TEKNIK KOMPUTER SURABAYA

ABSTRAK

Pemanfaatan layanan multimedia saat ini telah digunakan secara meluas dalam berbagai tujuan. Karena perkembangannya yang pesat, maka diperlukan suatu aturan yang mampu meningkatkan kualitas dari layanan ini. Suatu Protokol adalah sebuah aturan yang mendefinisikan beberapa fungsi yang ada dalam sebuah jaringan komputer, misalnya mengirim pesan, data, informasi, dan fungsi lain yang harus dipenuhi oleh pengirim (*transmitter*) dan penerima (*receiver*) agar komunikasi dapat berlangsung dengan benar.

Makalah ini menggambarkan hasil perbandingan protokol *User Datagram Protocol* (UDP), *Transmission Control Protocol* (TCP) dan *Streaming Control Transmission Protocol* (SCTP) terhadap pengiriman data multimedia *Voice Over IP* (VOIP) dan *Internet Protocol television* (IPTV). Simulasi dilakukan pada NS-2 *software* untuk menghasilkan kualitas parameter layanan yang meliputi *latency*, *jitter packet loss* dan *queue.*

Berdasarkan hasil simulasi, dapat disimpulkan bahwa protokol UDP memiliki *latency* terendah dibandingkan dengan TCP dan SCTP, selain itu juga memiliki variasi *delay (jiter)* terendah. Namun, dalam hal *packet loss*, UDP memiliki kecenderungan lebih besar untuk kehilangan paket dibandingkan dengan TCP dan SCTP. Hal ini berkaitan dengan besarnya penggunaan *queue* yang digunakan. Semakin besar penggunaan *queue* yang digunakan oleh suatu jaringan akan memperbesar kemungkinan paket yang hilang.

Kata Kunci : NS-2, VOIP, IPTV, Perbandingan TCP-UDP-SCTP

DAFTAR ISI

BAB I

PENDAHULUAN

1.1 Latar Belakang

Perkembangan teknologi multimedia berbasis video *streaming* dan *voice* saat ini semakin banyak digunakan sebagai aplikasi komunikasi pada internet. Pemanfaatan penggunaan teknologi tersebut juga sangat beragam dalam berbagai kegiatan misalnya pendidikan jarak jauh ataupun sebagai sarana *monitoring*, sehingga pada data multimedia dibutuhkan suatu unjuk kerja protokol yang handal dan cepat dalam proses pengirimannya. Di dalam dunia komunikasi data komputer, protokol berfungsi dalam mengatur satu komputer dengan komputer yang lainnya untuk dapat saling berkomunikasi. Semakin banyak paket data yang harus dikirim ke jaringan maka semakin besar pula *bandwidth* yang harus disediakan oleh jaringan. Jika panjangnya paket data pada jaringan yg ada melebihi kapasitas *bandwidth* yang disediakan maka dapat menimbulkan suatu kongesti (gangguan/tabrakan). Semakin tinggi tingkat kongesti suatu jaringan maka dapat menyebabkan hilangnya sejumlah paket di jaringan. (Sariningrum, 2008). Terjadinya kongesti dapat ditangani oleh administrator jaringan dengan menggunakan teknologi Qos *(Quality Of Service)*. (Yonathan, et al., n.d). Qos dapat diukur untuk mengetahui kualitas suatu layanan dalam pengiriman data dengan menggunakan parameter yang diantaranya meliputi *delay, packet loss, jitter, throuhput, queue*, dll.

Pada proses pengiriman data yang menggunakan *layer transport* dalam OSI model, terdapat beberapa macam protokol di didalamnya, yang dikelompokan

dalam dua kategori yaitu sebagai protokol yang memiliki *reliabilitas* dan protokol yang tidak memiliki *reliabilitas*.

Protokol yang tidak memiliki *reliabilitas* seperti pada UDP ini ditujukan untuk kecepatan pengiriman data tanpa memperhatikan adanya kontrol kongesti dan koreksi kesalahan di dalam suatu jaringan. Akibatnya kecepatan pengiriman data tidak dapat dikendalikan. Sehingga protokol UDP akan menggunakan seluruh *bandwidth* yang ada di dalam jaringan. Sedangkan protokol yang memiliki *reliabilitas* dapat ditunjukan pada protokol TCP yang memiliki karakteristik berbeda dari UDP. TCP merupakan protokol berorientasi *byte*, yaitu menerima sebuah pesan dari suatu proses, pengiriman pesan itu sebagai aliran *byte*, dan mengirimkannya dalam segmen. Tidak ada penjagaan batas-batas pesan. Namun, TCP merupakan protokol yang handal. penduplikatan segmen terdeteksi, segmen yang hilang akan dikirim ulang, dan *byte* yang dikirim ke akhir proses secara berurutan. TCP juga memiliki mekanisme kontrol kongesti dan *flow control.* (Alwi & Syawie, n.d.). Namun seiring berkembangnya layanan internet seperti multimedia, TCP dirasakan masih kurang dalam memenuhi layanan yang ada. Dalam TCP ketika jaringan padat yang otomatis berdampak pada kongesti sangat tinggi menyebabkan *time-out* dan akan mengirimkan *retransmisi* karena sifatnya yang *conection oriented.* (Telstra & Huston, n.d). Hal ini akan menyebabkan *delay* yang tinggi dan berakibat turunnya *throughput.* Maka mulailah dikembangkan protokol lain di *layer transport* seperti *Stream Control Transmission Protocol* (SCTP) yang lebih dapat diandalkan. *Stream Control Transmision Protocol* (SCTP) adalah protokol baru yang dapat diandalkan dan memiliki bayak kelebihan dibanding *Transmission Control Protocol* (TCP). SCTP

bersifat *message-oriented* yang handal, SCTP juga menyimpan batas-batas pesan dan disaat yang sama mampu mendeteksi kehilangan data, duplikasi data, dan *out of order* data. (Sapura & Pramartha, n.d.). Di SCTP terdapat *multistream* sehingga bisa mengirim data lebih banyak. Dengan fitur SCTP yang *multihoming* sehingga dapat meminimalisasi paket *retransmisi* yang akan mengatasi *delay*. Dikarenakan SCTP mempunyai *multistream* maka SCTP memakan *bandwidth* yang lebih banyak dari TCP sehingga untuk jaringan yang mempunyai sumber daya terbatas justru SCTP akan menyebabkan sering terjadinya *error*. (Budiraharjo, 2009).

Dari perbandingan karakteristik protokol tersebut, maka penulis melakukan penelitian mengenai perbandingan dari masing-masing protokol *transport layer* (dalam hal ini yang digunakan adalah protokol UDP, TCP dan SCTP) dari data multimedia (*Voice Over IP* (VOIP) dan *Internet Protocol Television* (IPTV)). Penelitian dalam tugas akhir ini merujuk pada hasil jurnal yang berjudul *Simulation of TCP, UDP and SCTP with constant traffic for VOIP service*. (Gangurde, et al., 2012). Jurnal tersebut menganalisis protokol TCP, UDP dan SCTP dengan mengamati parameter *delay*, *packet loss*, dan rata-rata *throughput* dengan memberikan masukan ukuran paket 1000 *byte* dan *channel capacity* 0.2 Mb pada masing-masing protokol. Perbandingan antara ketiga protokol terhadap hilangnya paket memberikan hasil seperti pada Tabel 1.1.

Tabel 1.1 Hasil Perbandingan Protokol TCP, UDP dan SCTP

Protocol	***Sent***	***Receiver***	***Packet Loss***	***Avg. Delay***	***Variance of delay***	***% Throughput***
TCP	712	693	19	0.787	0.01028	97.331
UDP	1599	723	846	1.931	0.1428	46.080
SCTP	750	687	63	1.681	0.3154	91.6

Sumber : (gangurde, Waware, & Sarwade, 2012)

Dari Tabel 1.1 dapat disimpulkan bahwa dengan memberikan ukuran paket 1000 *byte* dan *channel capacity* 0.2 Mb, protokol TCP memberikan hasil yang lebih baik dibanding ketiganya, namun pada implementasinya SCTP menggunakan sistem *best effort* dengan fitur *multihomming* dan *multistreaming* yang dimilikinya.

Penelitian yang akan dilakukan menggunakan skenario dari tiga parameter eksternal yaitu *bandwidth, delay propagation,* dan jumlah kanal untuk menganalisis *latency, packet loss, jitter* dan *queue*. Keempat parameter eksternal tersebut berguna dalam proses analisis sehingga mampu di implementasikan pada keadaan *real-time.* Tujuan dari analisis keempat parameter ini yang pertama untuk mengetahui jumlah *packet loss* yang dihasilkan pada jalur *bottleneck* masing-masing protokol jika diberikan nilai inputan yang sama dengan berbagai macam kondisi. Kedua untuk mengetahui besar *latency* dan *jitter* yang dihasilkan pada pengiriman paket data dari node sumber sampai ke node tujuan. Dan yang ketiga yaitu untuk mengetahui kapasitas antrian yang digunakan saat pengiriman data di jalur *bottleneck*.

Dari hasil analisis yang dibuat, maka diharapkan dapat dijadikan sebagai bahan pertimbangan dalam penentuan protokol *transport layer* ke depan untuk aplikasi IPTV dan VOIP.

1.2 Rumusan Masalah

1. Bagaimana membangun simulasi jaringan data multimedia dengan memanfaatkan protokol lapisan *transport* TCP, UDP dan SCTP pada *Network Simulator*-2?

2. Bagaimana melakukan analisis perbandingan unjuk kerja protokol TCP, UDP, dan SCTP menggunakan *Network Simulator*-2 dalam hal *Latency, packet loss, jitter* dan *queue* untuk lalu lintas data multimedia?

1.3 Batasan Masalah

Batasan masalah dari pembahasan tugas akhir ini adalah :

1. Simulasi analisis menggunakan software *Network Simulator*-2.
2. Model topologi yang digunakan yaitu menggunakan *dumb-bell topology*
3. Bahasa pemrograman menggunakan Tcl/OTcl pada NS-2.
4. Protokol yang dibandingkan adalah TCP, UDP, dan SCTP.
5. Hasil analisis membandingkan *Latency, packet loss, jitter* dan *queue* dari masing-masing protokol.
6. Input data yang digunakan adalah trafik CBR untuk merepresentasikan data VOIP dan IPTV yang akan disimulasikan bersama protokol TCP, UDP dan SCTP.

1.4 Tujuan Masalah

Dalam menganalisis perbandingan dari ke tiga protokol yaitu TCP, UDP dan SCTP ini, bertujuan untuk :

1. Menghasilkan simulasi jaringan dengan data multimedia dengan protokol lapisan *transport* TCP, UDP dan SCTP pada *Network Simulator* 2.
2. Menghasilkan perbandingan unjuk kerja protokol TCP, UDP, dan SCTP menggunakan *Network Simulator*-2 dalam menganalisis parameter *Latency, packet loss, jitter* dan *queue.*

1.5 Kontribusi Penelitian

Kontribusi yang di dapat dari hasil tugas akhir ini adalah :

1. Dari hasil perbandingan antara protokol TCP, UDP dan SCTP, diharapkan dapat memberikan kontribusi untuk perkembangan protokol lapisan *transport* pada layanan data multimedia ke depan.
2. Memberikan analisis perbandingan ke-3 protokol pada data multimedia, dalam hal ini akan dibahas data *Voice Over IP* (VOIP) dan *Internet Protocol Television* (IPTV).

1.6 Sistematika Penulisan

Pembahasan Tugas Akhir ini secara Garis besar tersusun dari 5 (lima) bab, yaitu diuraikan sebagai berikut:

BAB I. PENDAHULUAN

Pada Bab ini akan dibahas mengenai latar belakang masalah, batasan masalah, tujuan penulisan, dan sistematika penulisan.

BAB II. LANDASAN TEORI

Pada Bab ini akan dibahas teori penunjang dari permasalahan, yaitu mengenai konsep dasar internet, lapisan OSI model, protokol TCP, UDP dan SCTP, *quality of service*, data multimedia dan *software Network Simulator* - 2.

BAB III. METODE PENELITIAN DAN PERANCANGAN SISTEM

Pada Bab ini akan dibahas tentang blog diagram sistem serta metode yang dilakukan dalam perancangan simulasi sistem, meliputi cara dalam pembuatan skrip ns2, melakukan parsing file, serta ploting file

untuk menghasilkan data dalam bentuk grafik maupun tabel yang dibutuhkan dalam analisis.

BAB IV. HASIL DAN PEMBAHASAN

Pada Bab ini akan dibahas mengenai hasil yang diperoleh dari proses simulasi, yang akan dibandingkan berdasarkan kondisi dan parameter-parameter untuk digunakan dalam proses analisis yang meliputi analisis *latency, jitter, packet loss* dan *queue* dari data VOIP dan IPTV.

BAB V. KESIMPULAN DAN SARAN

Berisi kesimpulan yang didapat dari hasil penelitian berdasarkan rumusan masalah serta saran untuk perkembangan penelitian selanjutnya.

BAB II

LANDASAN TEORI

2.1 *Transport Layer Protocol*

Transport layer atau lapisan transpor merupakan lapisan keempat dari model referensi OSI dan jantung dari hirarki protokol secara keseluruhan. Tugas *layer* ini menyediakan data transpor yang bisa dihandalkan dan efektif biayanya dari komputer sumber ke komputer tujuan., yang tidak bergantung pada jaringan fisik atau jaringan-jaringan yang digunakan. Tanpa *transport layer*, seluruh konsep protokol yang menggunakan *layer* tidak akan ada gunanya. (Kristanto, 2003).

Lapisan transpor bertanggung jawab atas keutuhan dari *transmisi* data. Lapisan ini sangat penting karena bertugas memisahkan lapisan tingkat atas dengan tingkat bawahnya. Pada lapisan ini data diubah menjadi segmen atau data *stream*. Ada dua jenis hubungan pada lapisan transpor, yaitu :

a. *Connection-Oriented*: hubungan ini disebut *connection-oriented* karena ditunjang oleh *Transmission Control protocol* (TCP). Hubungan ini *reliable* karena setiap *session* digaransi. *Connection-oriented* memiliki tiga langkah untuk melakukan pengiriman sebagai berikut:

1. Mengadakan hubungan - dimana jalur antara pengirim dan penerima dibangun.
2. Pengiriman paket-paket – data dikirim lewat jalur yang telah dibangun.
3. Pemutusan hubungan – hubungan jalur yang tak dipakai lagi akan diputuskan.

Ciri-ciri *connection oriented* adalah sebagai berikut :

1. Semua paket mendapat tanda terima (*ackowledge*) dari penerima.

2. Paket yang tidak diterima dikirim ulang.

3. Paket-paket diurut kembali (*sequence*), misalnya berdasarkan asal waktu pengirimannya.

b. *Connectionless-Oriented* : hubungan *connestionless-oriented* ditunjang oleh *User Datagram Protocol* (UDP). Penerima tidak mengirimkan tanda terima, dan paket-paket tidak diurut kembali seperti asalnya. Namun dibandingkan hubungan *connection-oriented*, hubungan *connectionless-oriented* ini mempunyai keunggulan yaitu penggunaan *bandwidth* nya efektif karena semua jalur yang tersedia dapat digunakan oleh pemakai-pemakai lainnya. Oleh karena jalur yang digunakan tergantung dari paket per paket, maka jika terjadi kemacetan di jalur satu, paket dapat disalurkan ke jalur lain. (Forouzan, 2007).

2.2 *Transmission Control Protocol (TCP)*

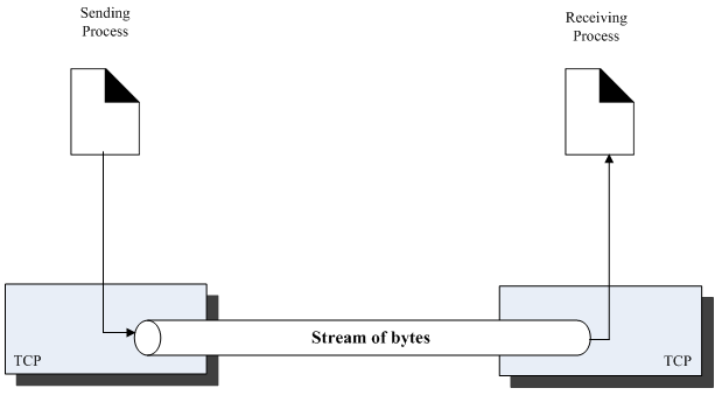
TCP adalah protokol yang berorientasi koneksi yang menciptakan suatu koneksi virtual antara dua TCP untuk mengirim data. Di samping itu, TCP menggunakan aliran dan mekanisme *error control* pada level trans*port*asi.

Secara ringkas, TCP disebut *connection oriented*, protokol transpor yang handal. TCP menambah fitur orientasi koneksi dan kehandalan untuk layanan pada IP.

a. Aliran Layanan Pengiriman

TCP memungkinkan proses pengiriman untuk mengirimkan data sebagai aliran *byte* dan memungkinkan proses penerimaan untuk memperoleh data sebagai aliran *byte*. TCP menciptakan suatu keadaan di mana dua proses dihubungkan oleh sebuah "*tub*" *imajiner* yang membawa data mereka melewati internet. Keadaan *imajiner* digambarkan pada Gambar 2.1. Proses pengiriman

menghasilkan (menulis) aliran *byte*, dan proses penerimaan yang menggunakannya (membaca).

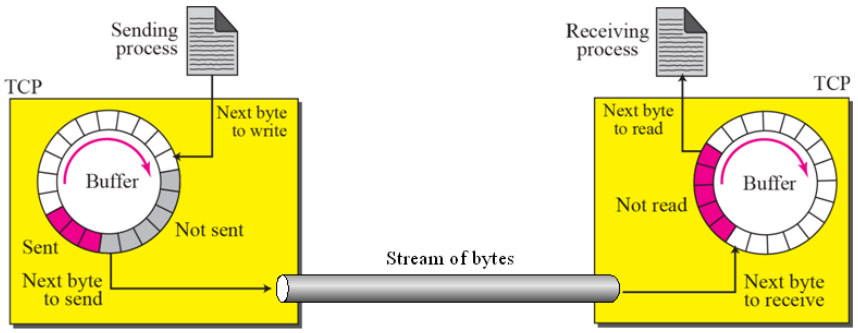


Gambar 2.1 Pengiriman *Stream*

Sumber: (Forouzan, 2007)

b. Pengiriman dan Penerimaan *Buffer*

Karena proses pengiriman dan penerimaan tidak bisa menulis atau membaca data dengan kecepatan yang sama, TCP membutuhkan *Buffer* untuk penyimpanan. Terdapat dua *buffer* yaitu *buffer* pengiriman dan *buffer* penerima, setiap arah mempunyai salah satu *buffer* tersebut. Salah satu cara untuk mengimplementasikan *buffer* adalah dengan menggunakan array melingkar dari 1-*byte* lokasi seperti yang ditunjukkan pada Gambar 2.2.



Gambar 2.2 Pengiriman dan Penerimaan *Buffer*

Sumber: (Forouzan, 2007)

Gambar 2.2 menunjukan perpindahan data pada satu arah. Pada arah pengiriman, memiliki tiga jenis ruang. Bagian putih mengandung ruang kosong yang dapat diisi oleh proses pengiriman (penghasil). Wilayah abu-abu menyimpan *byte* yang telah dikirim tetapi belum diakui. TCP menjaga *byte* ini dalam *buffer* sampai ia menerima pengakuan (ACK). Daerah yang berwarna berisi *byte* yang akan dikirim oleh tcp pengirim. Namun, TCP mungkin hanya dapat mengirim sebagian dari bagian yang berwarna. Hal ini dapat disebabkan oleh lambatnya proses penerimaan atau mungkin kemacetan dalam jaringan. Perhatikan juga bahwa setelah *byte* dalam ruang abu-abu diakui, ruang tersebut akan di pakai kembali dan tersedia untuk digunakan pada proses pengiriman. Hal inilah yang menyebabkan buffer melingkar

Pengolahan pada *buffer* di situs penerima lebih sederhana. *Buffer* yang melingkar dibagi menjadi dua daerah (ditunjukan pada warna putih dan yang berwarna). Daerah putih berisi ruang kosong untuk diisi oleh *byte* yang diterima dari jaringan, sedangkan bagian yang berwarna berisi *byte* yang diterima yang dapat dibaca oleh proses penerimaan,

c. Komunikasi *Full Duplex*

TCP menawarkan layanan *full duplex,* di mana data dapat mengalir dalam dua arah pada waktu yang sama. Setiap TCP mempunyai *buffer* pengiriman dan penerimaan, dan segmen yang bergerak di kedua arah.

d. Layanan Orientasi Koneksi

Ketika proses di situs A ingin mengirim dan menerima data dari proses lain di situs B, berikut ini yang akan terjadi:

1. Dua TCP membentuk koneksi diantara mereka.
2. Data dipertukarkan di kedua arah.
3. Koneksi diakhiri.

Perhatikan bahwa ini adalah koneksi *virtual*, bukan koneksi fisik. Segmen TCP dienkapsulasi dalam datagram IP dan dapat dikirim keluar dari urutan, atau hilang, atau rusak, lalu kemudian akan dikirim kembali. Masing-masing dapat menggunakan jalur yang berbeda untuk mencapai tujuan. Disana tidak ada koneksi fisik. TCP membuat orientasi aliran suatu keadaan di mana ia menerima tanggung jawab mengirimkan *byte* untuk sisi yang lainnya. Situasi ini mirip dengan pembuatan sebuah jembatan yang mencakup beberapa kepulauan dan melewatkan semua *byte* dari satu pulau ke pulau lain dalam satu koneksi tunggal.

e. Layanan Kehandalan

TCP adalah protokol trans*port* yang dapat dihandalkan. Menggunakan sebuah mekanisme pengakuan untuk memeriksa keamanan dan tanda kedatangan data. (Forouzan, 2007).

2.2.1 Segmen TCP

Bit Offset	0-3	4-7	8-15							16-31
0	Source Port									Destination Port
32	Sequence Number									
64A	Acknowledgement Number									
96	Data Offset	Reserved	CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Windows Size
128	Cheksum									Urgent Pointer
160	Option (Optional)									
160/192+	Data									

Gambar 2.3 Segmen TCP

Sumber : (Sofana, 2009)

Pada *Gambar* 2.3 menggambarkan segmen dari TCP yang dapat di jelaskan sebagai berikut :

1. *Source Port* (16 *bits*) : Berisi informasi *port* pengirim
2. *Destination Port* (16 *bits*) : Berisi informasi *port* penerima
3. *Sequence Number* (32 *bits*) : Berupa *sequence number* yang terdiri atas dua kondisi berikut:
   a. Jika *flag* SYN di-set (yang ada di bagian *field Flags*), maka *field* ini berisi awal (inisial) dari *sequence number*.
   b. Jika *flag* SYN tidak di-set, maka nilai pada *field* ini merupakan *sequence number*.
4. *Acknowledgement* atau ACK (32 *bits*) : jika *flag* ACK di-set, maka nilai pada *field* ini adalah nilai *sequence number* berikutnya yang di-"harapkan" oleh penerima.

5. *Data offset* (4 *bits*) : Menunjukan ukuran TCP *header*. Total *header* sepanjang 32-*bit* words. Ukuran minimum *header* adalah 5 word. Data *offset* juga merupakan awal dari data.
6. *Reserved* (4 *bits*) : Untuk keperluan tertentu di masa akan datang. Nilai pada *field* ini semestinya adalah *zero* (nol).
7. *Flags* (8 *bit*) : *field* untuk kontrol *bit* (masing-masing 1 *bit*), yaitu :
   a. CWR (*Congestion Windows Reduced*)
   b. ECE (ECN-*Echo*)
   c. URG (*URGent*)
   d. ACK (*ACKnowledgement*)
   e. PSH (*Push function*)
   f. RST (*Reset*)
   g. SYN (*Synchronize*)
   h. FIN (*Finish*)
8. *Window* (16 *bits*) : menunjukan ukuran *window* penerima (*receive window*). Agar data dapat diterima dengan baik maka diperlukan pengaturan ukuran jumlah *byte* optimal yang ditentukan oleh field ini.
9. *Checksum* (16 *bits*) : digunakan untuk *error-checking* dari *header* dan data
10. *Urgent pointer* (16 *bits*) : digunakan untuk *sequence number* yang menandakan *urgent data byte* terakhir.
11. *Option (Variable bits)* : Berisi berbagai opsi berupa angka sebagai berikut :
    a. 0-*End of option list*

b. 1-*No operation list*

c. 2-*Maximum segment size*

d. 3-*Window scale*

e. 4-*Selective Acknowledgement*

f. 5,6,7

g. 8-*Timestamp*

12. Data : Berisi data yang dikirim.

2.2.2 Koneksi TCP

TCP merupakan *connection-oriented*. TCP membentuk sebuah jalur virtual antara sumber dan tujuan. Semua segmen yang dimiliki sebuah pesan kemudian dikirim diatas jalur *virtual* ini. Menggunakan sebuah jalur *virtual* tunggal untuk seluruh pesan memudahkan dalam proses pengakuan serta pengiriman ulang frame yang rusak atau hilang. TCP beroperasi pada tingkat yang lebih tinggi. TCP menggunakan Jasa IP untuk mengirimkan masing-masing segmen ke penerima, tetapi itu mengontrol koneksinya sendiri. Jika segmen hilang atau rusak, maka akan dikirimkan ulang.

Pada TCP, yang merupakan pengiriman berorientasi koneksi membutuhkan tiga tahap :

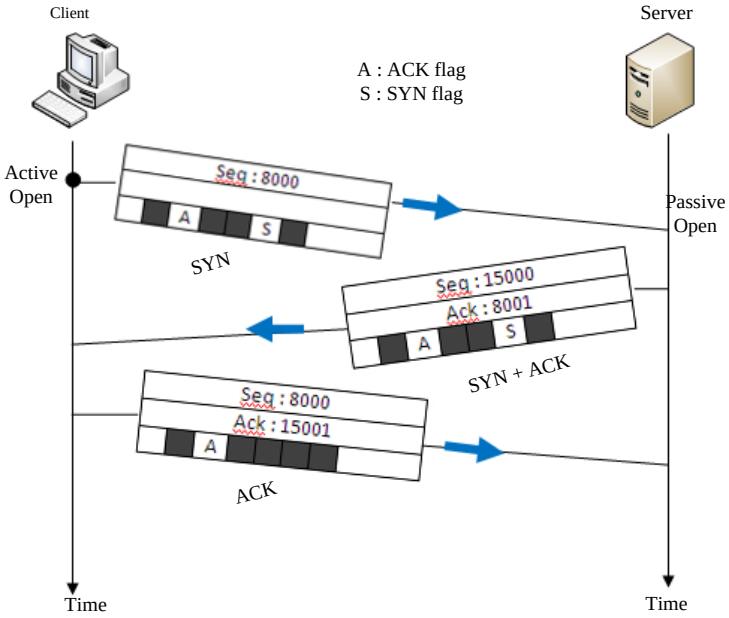
1. Pembentukan Koneksi

TCP mengirimkan data pada mode *full-duplex*. Ketika dua TCP pada dua mesin terhubung, mereka dapat mengirimkan segmen satu sama lain secara simultan. Menunjukkan bahwa setiap pihak harus menginisialisasi komunikasi dan mendapatkan persetujuan dari pihak lain sebelum data dikirim.

Pembentukan koneksi TCP disebut *Three Way Handshake*. Dalam contoh, sebuah program aplikasi, yang disebut klien, ingin membuat koneksi dengan program aplikasi lain, yang disebut *server*, menggunakan TCP sebagai protokol *transport-layer*.

Proses dimulai dengan *server*. Program *server* memberitahu TCP nya yang siap untuk menerima koneksi. Hal ini disebut permintaan *passive open.* Meskipun *server* TCP siap menerima koneksi dari seluruh komputer di dunia, dia tidak bisa membuat koneksi sendiri.

Permasalan permintaan *active open* ada pada program klien. Klien ingin terhubung ke suatu *open server*, dengan mengatakan bahwa TCP ini perlu terhubung ke *server* tertentu. TCP sekarang dapat memulai proses *three way handshaking* yang ditunjukkan/diperlihatkan pada Gambar 2.4. Untuk menunjukkan proses tersebut, kita menggunakan dua garis waktu: satu waktu pada setiap sisi. Masing-masing segmen memiliki nilai untuk semua *header fields* dan mungkin juga untuk beberapa opsi *fields*.



Gambar 2.4 Pembentukan Koneksi *Three Way*

Sumber : (Forouzan, 2007)

Pada Gambar 2.4 ditunjukkan *sequence number*, *acknowledgement number*, *control flag* (jika sudah diatur), dan *windows size*, jika tidak kosong. Tiga langkah dalam tahap ini adalah sebagai berikut:

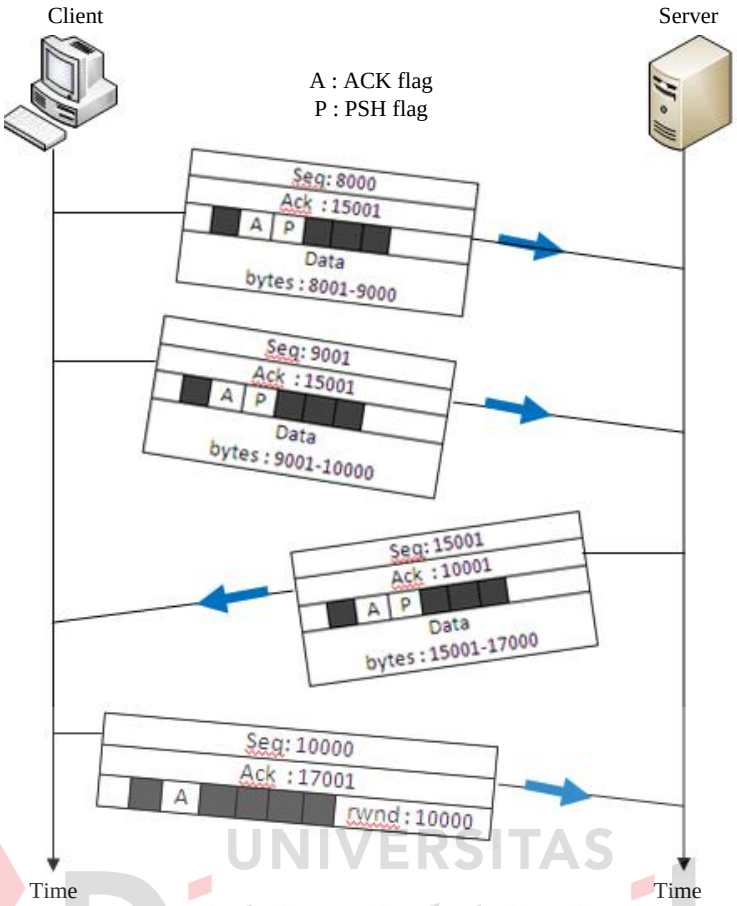
a. *Client* mengirimkan segmen pertama, segmen SYN, di mana hanya *flag* SYN yang diatur. segmen ini adalah untuk mensinkronisasi nomor urut. Itu mengkonsumsi satu nomor urut. Ketika *transfer* data dimulai, nomor urut bertambah sebesar 1. Dapat dikatakan bahwa segmen SYN tidak membawa data yang sebenarnya, tetapi bisa dianggap itu berisi 1 *byte* imajiner.

b. *Server* mengirim segmen kedua, segment SYN + ACK , dengan 2 *flag bit* yang diatur: SYN dan ACK. Segmen ini memiliki tujuan ganda. Yaitu segmen SYN untuk komunikasi pada arah lain dan berfungsi sebagai pengakuan (*acknowledgement*) untuk segmen SYN. Segmen ini mengkonsumsi satu nomor urut.

c. Klien mengirimkan segmen ketiga. Ini hanyalah segmen ACK. Yang merupakan pengakuan penerimaan segmen kedua dengan *flag* ACK dan pengakuan nomor *field*. Perhatikan bahwa urutan angka dalam segmen ini adalah sama dengan yang ada di segmen SYN; segmen ACK tidak menggunakan nomor urutan.

2. Pengiriman Data

Setelah koneksi ditetapkan, pengiriman data dua arah dapat berlangsung. Klien dan *server* dapat saling mengirim data dan *acknowledgement*. Data yang berjalan searah dengan *acnowledgment* yang dibawa oleh segmen yang sama**.** Gambar 2.5 menunjukkan sebuah contoh. Dalam contoh ini, setelah koneksi didirikan (tidak ditampilkan dalam gambar), *client* mengirimkan 2000 *byte* data dalam dua segmen. *server* kemudian mengirimkan 2000 *byte* dalam satu segmen. *Client* mengirimkan satu segmen lagi. Tiga segmen pertama membawa kedua data dan ACK, namun segmen terakhir hanya membawa ACK karena disana tidak ada data lagi yang bisa dikirim. Segmen data yang dikirim oleh klien memiliki *flag* PSH (*push*) yang sudah ditetapkan sehingga *server* TCP mengetahui pengiriman data ke proses *server* segera setelah mereka menerimanya. Segmen dari *server*, sebaliknya, tidak mengatur *flag push*. Pada penerapannya sebagian besar TCP memiliki pilihan untuk menetapkan atau tidak menetapkan *flag* ini.



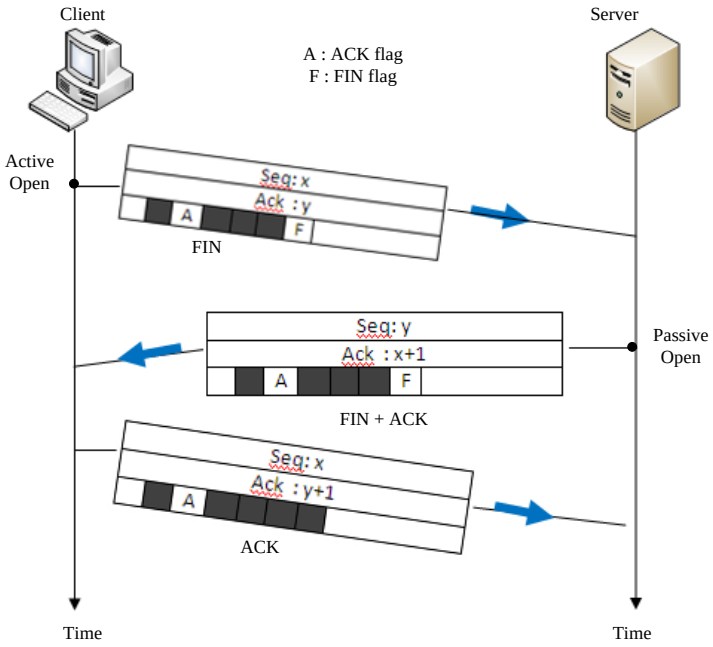
Gambar 2.5 Pengiriman Data

Sumber : (Forouzan, 2007)

3. Pemutusan Koneksi

Salah satu dari dua pihak yang terlibat dalam pertukaran data (*client* atau *server*) dapat mengahiri koneksi, walaupun biasanya dimulai oleh klien. Sebagian besar implementasinya saat ini memungkinkan dua pilihan untuk pemutusan koneksi: *three way hanshake* dan *four way hanshake* dengan pilihan *half-close*.

1. *Three Way Handshaking* : saat ini implementasi yang paling memungkinkan yaitu *three way hanshake* untuk pemutusan hubungan seperti yang ditunjukan pada Gambar 2.6.



Gambar 2.6 Pemutusan Koneksi menggunakan *Three Way Handshake*

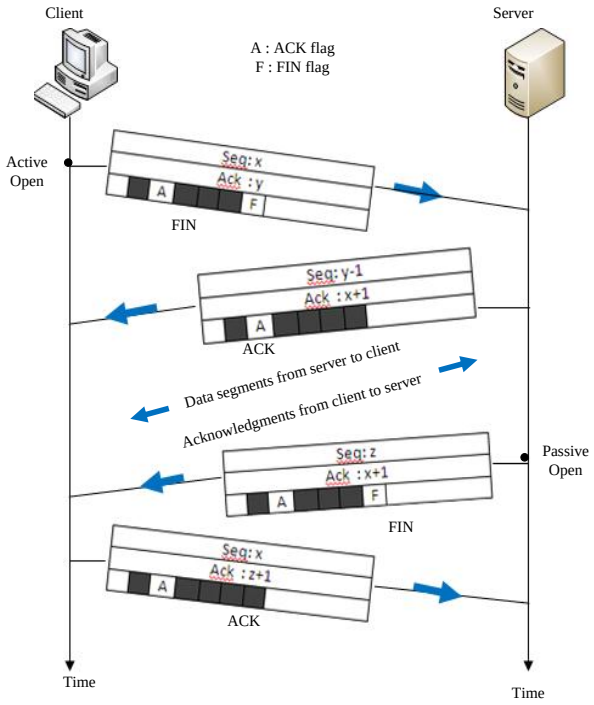
Sumber : (Forouzan, 2007)

a. Dalam situasi normal, TCP *client*, setelah menerima sebuah perintah proses penutupan dari *client*, lalu mengirimkan segmen pertama, sebuah segmen FIN di mana flag FIN diatur. Segmen FIN dapat menyertakan potongan terakhir dari data yang dikirim oleh klien, atau bisa juga hanya segmen kontrol. Jika segment kontrol, maka hanya akan menggunakan satu nomor urut.

b. Pada TCP *server*, setelah menerima segmen FIN, lalu akan menginformasikan proses tersebut dan mengirimkan segmen kedua, segmen FIN + ACK, untuk mengkonfirmasi penerimaan segmen FIN dari klien dan pada saat yang sama mengumumkan penutupan koneksi ke arah sebaliknya. Segmen ini dapat juga berisi potongan terakhir data dari *server*. Jika tidak membawa data, maka hanya menggunakan satu nomor urutan.

c. TCP klien mengirimkan segmen terakhir, segmen ACK, untuk mengkonfirmasi penerimaan segmen FIN dari TCP *server*. Segmen ini berisi nomor ACK, yang merupakan 1 ditambah nomor urut yang diterima di segmen FIN dari *server*. Segmen ini tidak bisa membawa data dan tidak menggunakan nomor urutan.

2. *Half-close* : Pada TCP, salah satu sisi dapat menghentikan pengiriman data pada saat masih menerima data. Ini disebut pemutusan sebagian. Meskipun keduanya dapat melakukan pemutusan sebagian, biasanya dimulai oleh klien. Hal itu dapat terjadi ketika membutuhkan semua data sebelum proses dimulai. sebuah contoh yang baik adalah pengurutan. ketika klien mengirimkan data ke *server* untuk diurutkan, *server* perlu menerima semua data sebelum pengurutan dapat mulai. Hal iini berarti *klien*, setelah mengirim semua data, bisa menutup sambungan ke arah keluar. Namun, arah masuknya data harus tetap terbuka untuk menerima urutan data. *Server*, setelah menerima semua data masih membutuhkan waktu untuk pengurutan ; arah keluar harus tetap terbuka.



Gambar 2.7 *Half-close*

Sumber : (Forouzan, 2007)

Gambar 2.7 menunjukkan contoh pemutusan sebagian. klien memutuskan sebagian sambungan dengan mengirimkan segmen FIN. *Server* menerima *half-close* dengan mengirimkan segmen ACK. Pengiriman data dari klien ke *server* berhenti. Namun *server* masih dapat mengirim data. Ketika *server* telah mengirim semua data olahan, maka ia akan mengirimkan segmen FIN, yang diakui oleh ACK dari klien.

Setelah sebagian koneksi tertutup, data dapat berjalan dari *server* ke klien dan ACK dapat berjalan dari klien ke *server*. Klien tidak dapat mengirim lebih banyak data ke *server*. Perhatikan urutan nomor yang telah digunakan. Segmen kedua (ACK) tidak menggunakan urutan nomor. Meskipun klien telah menerima urutan nomor y-1 dan mengharapkan y, urutan nomor *server* masih y-1. ketika koneksi akhirnya menutup, urutan jumlah ACK segmen ini masih x, karena tidak

ada urutan nomor yang digunakan selama *transfer data* pada arah tersebut. (Forouzan, 2007).

2.3 *User Datagram Protocol* (UDP)

User Datagram Protocol (UDP) disebut *conectionless*, protokol transpor yang tidak dapat dihandalkan. Disana tidak menambahkan sesuatu ke layanan IP kecuali untuk menyediakan komunikasi *process-to-process* bukan komunikasi *host-to-host.* Dan juga, dalam melakukan pengecekan error sangat terbatas.

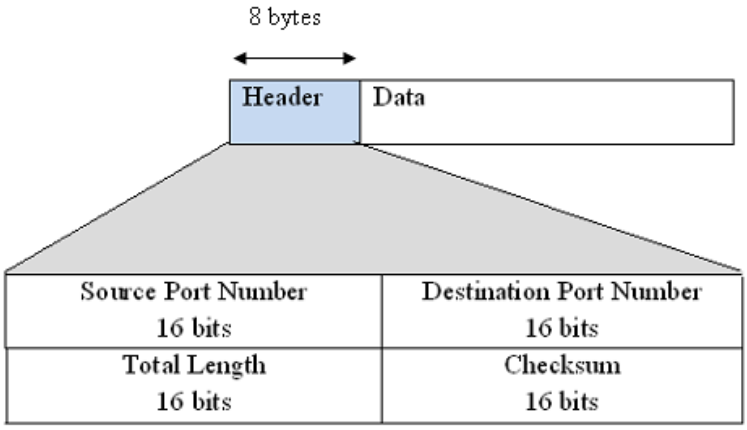
Dengan kelemahan pada UDP memberikan beberapa keuntungan. UDP adalah protokol yang sangat sederhana menggunakan minimum *overhead*. Jika sebuah proses ingin mengirim pesan yang kecil dan tidak peduli tentang kehandalannya, maka dapat menggunakan UDP. Mengirim pesan kecil menggunakan UDP membuat lebih sedikit interaksi antara pengirim dan penerima daripada menggunakan TCP atau SCTP.

2.3.1 *User Datagram*

Paket-paket UDP, yang disebut *user datagram*, memiliki ukuran *header* yang tetap yaitu 8 *byte*. Gambar 2.8 menunjukkan format *user datagram*. Bagian-bagiannya adalah sebagai berikut:

1. *Source port number* : ini adalah nomor *port* yang digunakan oleh proses yang berjalan pada *host* sumber. Panjangnya adalah 16 *bit*, yang berarti bahwa nomor *port* dapat berkisar dari 0 sampai 65,535. Jika sumber *host* adalah *client* (klien mengirimkan permintaan), nomor *port*, dalam banyak kasus, adalah permintaan *port* sementara yang diproses dan di pilih oleh software UDP yang

berjalan pada *host* sumber. Jika sumber *host* adalah *server* (*server* mengirim tanggapan), nomor *port*, dalam banyak kasus, adalah nomor *port* yang dikenal.



Gambar 2.8 *User Datagram Format*

Sumber : (Forouzan, 2007)

2. *Destination port number* : ini adalah nomor *port* yang digunakan pada proses yang berjalan di *host* tujuan. Panjangnya juga 16 *bit*. jika *host* tujuan adalah *server* (klien mengirimkan suatu permintaan), nomor *port*, dalam banyak kasus, adalah nomor *port* yang dikenal. Jika *host* tujuan adalah *client* (*server* mengirim respon), nomor *port*, dalam banyak kasus, adalah nomor pot sementara. dalam hal ini, *server* menyalin nomor *port* sementara itu yang telah diterima pada paket *request.*
3. *Length* : ini adalah *field* 16 *bit* yang mendefinisikan panjang keseluruhan dari *user datagram*, *header* ditambah data. 16 *bit* dapat menentukan panjang keseluruhan 0-65,535 *byte*. Namun, total panjang harus lebih sedikit karena *user datagram* pada UDP disimpan dalam sebuah datagram IP dengan panjang total 65.535 *byte*. Panjang *field* pada *user datagram* UDP ini benar-benar diperlukan. *User datagram* dirumuskan dalam datagram IP. Disana sebuah

field pada datagram IP mendefinisikan panjang keseluruhan. *Field* lainnya pada datagram IP mendefinisikan panjang dari *header*. Jadi pengurangan pada nilai dari *field* yang kedua di kurangi yang pertama, akan didapatkan panjang datagram UDP yang dirumuskan dalam datagram IP. Namun, perancang protokol UDP merasa bahwa itu lebih efisien untuk tujuan UDP dalam menghitung panjang data dari informasi yang diberikan *user datagram* UDP daripada meminta IP perangkat lunak untuk memberikan informasi ini. Ketika IP perangkat lunak memberikan *user datagram* UDP ke lapisan UDP, itu telah menurunkan *header* IP.

4. *Checksum* : *field* ini digunakan untuk mendeteksi kesalahan di datagram seluruh pengguna (*header* ditambah data).

2.3.2 *Operation* UDP

UDP menggunakan konsep-konsep umum untuk lapisan trans*port*asi.

1. Layanan *Connectionless*

Seperti disebutkan sebelumnya, UDP menyediakan layanan *connectionless*. *Connectionles* berarti bahwa setiap *user datagram* dikirim melalui UDP datagram independen. Tidak ada hubungan antara *user datagram* yang berbeda bahkan jika mereka berasal dari proses sumber yang sama ke suatu program tujuan yang sama. *User datagram* tidak diperhitungkan. Tidak ada pembentukan koneksi dan pemutusan koneksi, seperti halnya pada TCP. Hal ini berarti bahwa setiap *user datagram* dapat melakukan perjalanan di jalur yang berbeda.

Salah satu konsekuensi *connectionless* adalah bahwa proses yang menggunakan UDP tidak dapat mengirim aliran data ke UDP dan mengharapkan

UDP untuk memotongnya kedalam *user datagram* yang berbeda. Sebaliknya, setiap permintaan harus lebih kecil untuk dimasukan ke dalam satu *user datagram*. hanya proses pengiriman pesan singkat yang sebaiknya menggunakan UDP.

2. Kontrol Aliran dan *Error*

UDP adalah protokol *transportasi* yang sangat sederhana, tidak dapat dihandalkan. Tidak ada kontrol aliran dan karenanya tidak ada mekanisme *windows.* Maka penerima akan sangat banyak menerima pesan masuk.

Disana tidak ada mekanisme kontrol kesalahan pada UDP kecuali *checksum*. Hal ini berarti bahwa pengirim tidak tahu jika pesan telah hilang atau diduplikasi. Ketika Penerima mendeteksi *error* melalui *checksum, datagram* pengguna diam-diam akan dibuang. Kurang adanya kontrol aliran dan kontrol kesalahan berarti bahwa proses yang menggunakan UDP harus menyediakan mekanisme ini.

3. Enkapsulasi dan Dekapsulasi

Untuk mengirim pesan dari satu proses ke proses yang lain, protokol UDP mengenkapsulasi dan mendekapsulasi pesan dalam *datagram* IP. (Forouzan, 2007).

2.4 *Stream Control Transmission Protocol* (SCTP)

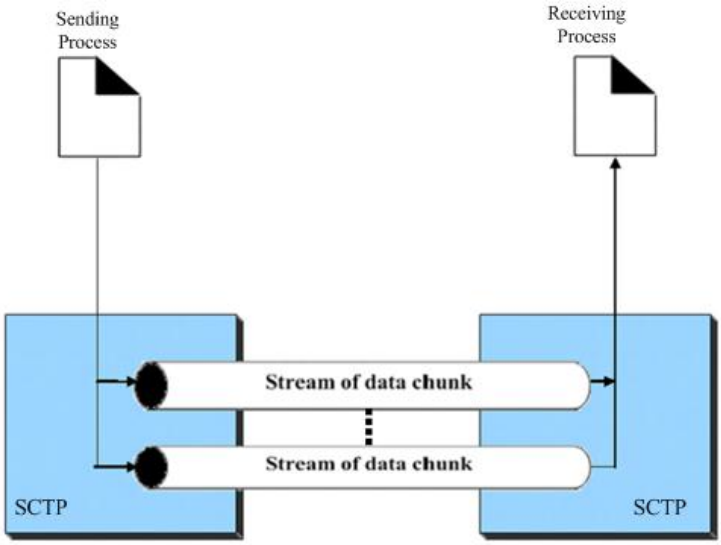
SCTP adalah suatu kehandalan terbaru, protokol lapisan transpor yang berorientasi pesan. Namun, SCTP kebanyakan di rancang untuk aplikasi internet yang yang baru-baru ini telah diperkenalkan. Aplikasi baru ini, seperti IUA (*ISDN over IP*), M2UA dan M3UA (*telephony signaling*), H.248 (*media gateway*

control), H.323 (*IP telephony*), and SIP (*IP telephony*), membutuhkan pelayanan yang lebih canggih daripada yang disediakan oleh TCP.

SCTP menggabungkan fitur terbaik dari UDP dan TCP. SCTP adalah protokol *message-oriented* yang handal. SCTP menyimpan batas-batas pesan dan pada saat yang sama mendeteksi kehilangan data, duplikasi data, dan *out-of-order* data. SCTP juga memiliki kontrol kongesti dan mekanisme kontrol aliran. Layanan yang ditawarkan SCTP untuk proses lapisan aplikasi yaitu :

1. *Multiple Streams*

Di TCP setiap koneksi antara TCP klien dan TCP *server* melibatkan satu aliran. Masalahnya dengan pendekatan ini adalah kerugian pada setiap titik di blok *stream* pengiriman sisa data. Hal ini bisa diterima ketika kita sedang memindahkan teks; bukan seperti pada saat kita mengirim data *real-time* seperti audio atau video. SCTP memungkinkan layanan *multistream* di setiap koneksi yang disebut asosiasi dalam *terminologi* SCTP. Jika salah satu aliran diblokir, maka aliran yang lain masih bisa mengirim datanya . Hal ini di ibaratkan seperti beberapa jalur di jalan raya. Setiap jalur dapat digunakan untuk berbagai jenis lintasan. Misalnya, satu jalur bisa digunakan untuk lintasan biasa, satu lagi untuk lintasan kusus mobil. Jika lintasan yang diblokir untuk kendaraan biasa, maka lintasan khusus kendaraan mobil masih bisa mencapai tujuan mereka. Konsep dari mekanisme *multistream* terlihat pada Gambar 2.9.

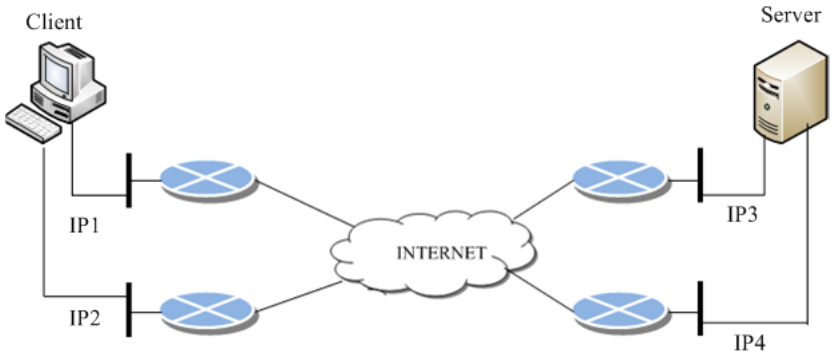


Gambar 2.9 Konsep *Multiple-Stream*

Sumber : (Forouzan, 2007)

2. *Multihoming*

Sebuah koneksi TCP melibatkan salah satu sumber dan satu tujuan alamat IP. Hal ini berarti jika pengirim atau penerima merupakan *host multihomed* (terhubung ke lebih dari satu alamat fisik dengan beberapa alamat IP), hanya satu dari alamat-alamat IP untuk setiap akhir dapat digunakan selama masih terhubung. Sebuah asosiasi SCTP, di sisi lain, mendukung *multihoming* layanan. *Host* mengirim dan menerima dapat menentukan beberapa alamat IP pada setiap akhir untuk asosiasi. Dalam pendekatan *fault-tolerant*, ketika salah satu jalur gagal, antarmuka lain dapat digunakan untuk pengiriman data tanpa gangguan. Fitur *fault-tolerant* ini sangat membantu ketika kita mengirim dan menerima muatan *real-time* seperti *Internet telephony*.



Gambar 2.10 Konsep *Multihoming*

Sumber : (Forouzan, 2007)

Dalam Gambar 2.10 *Multihoming* konsep, klien terhubung ke dua jaringan lokal dengan dua alamat IP. *Server* ini juga terhubung ke dua jaringan dengan dua alamat IP. Klien dan *server* dapat membuat asosiasi, menggunakan empat pasang alamat IP yang berbeda. Namun, perlu diketahui bahwa dalam implementasi saat ini SCTP hanya sepasang alamat IP dapat dipilih untuk komunikasi normal; alternatif digunakan jika pilihan utama gagal. Dengan kata lain, pada saat ini, SCTP tidak mengijinkan berbagi beban antara jalur yang berbeda.

3. Komunikasi *Full-Duplex*

Seperti TCP, SCTP menawarkan layanan *full-duplex*, di mana data dapat mengalir dalam dua arah pada saat yang sama. Setiap SCTP kemudian memiliki *buffer* mengirim dan menerima, dan paket dikirim di kedua arah.

4. Layanan *Connection-Oriented*

Seperti TCP, SCTP merupakan protokol berorientasi koneksi. Namun dalam SCTP, koneksi ini disebut asosiasi. Proses ketika suatu proses pada situs A ingin mengirim dan menerima data dari proses lain di situs B:

a. Kedua SCTP membangun hubungan antara satu sama lain.

b. Data dipertukarkan dalam kedua arah.

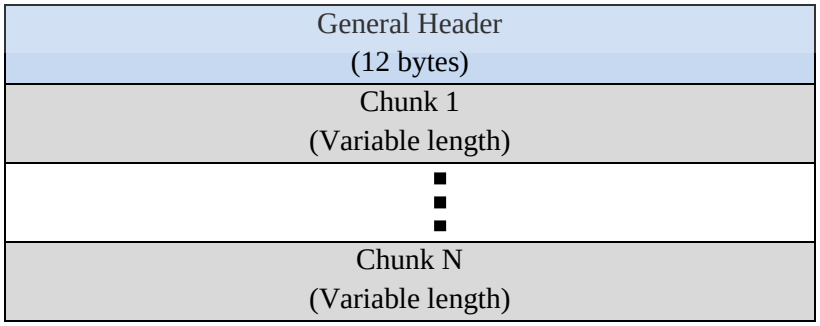
c. Asosiasi dihentikan

5. Reliable Service

Seperti TCP, SCTP merupakan protokol *transport* yang handal. SCTP menggunakan mekanisme pengakuan untuk memeriksa data suara itu aman.

2.4.1 Format paket

Sebuah paket SCTP memiliki *header* umum yang wajib dan satu set blok yang disebut potongan. Ada dua jenis potongan: potongan kontrol dan potongan data. Sebuah potongan kontrol mengendalikan dan memelihara asosiasi; kumpulan data membawa data pengguna. Dalam sebuah paket, potongan kontrol datang sebelum potongan data. Gambar 2.11 menunjukkan format umum dari sebuah paket SCTP.



Gambar 2.11 Format Paket SCTP

Sumber : (Forouzan, 2007)

1. *Header* Umum

Header umum (paket *header*) mendefinisikan mendefinisikan titik akhir dari setiap asosiasi yang termasuk paket, menjamin bahwa paket adalah milik asosiasi

tertentu, dan menjaga integritas isi dari paket termasuk *header* itu sendiri. Format dari *header* umum ditunjukan oleh Gambar 2.12.

Source port address *16 bits*	*Destination port address* *16 bits*
Verification Tag *32 bits*	
Checksum *32 bits*	

Gambar 2.12 *Header* Umum

Sumber : (Forouzan, 2007)

Ada empat bidang di *header* umum:

a. Sumber alamat *port*. Ini adalah bidang 16-*bit* yang mendefinisikan nomor *port* dari proses mengirim paket.

b. alamat *port* tujuan. Ini adalah bidang 16-*bit* yang mendefinisikan nomor *port* dari proses penerimaan paket.

c. Verifikasi *tag* adalah angka yang cocok dengan paket ke asosiasi. Verifikasi *tag* mencegah paket dari sebuah *asosiasi* sebelumnya dari yang salah sebagai paket dalam asosiasi. Hal tersebut berfungsi sebagai pengidentifikasi untuk asosiasi, hal ini diulang dalam setiap paket selama asosiasi. Ada verifikasi terpisah digunakan untuk setiap arah dalam asosiasi.

d. *Checksum. Field* ini berisi 32-*bit* CRC-32 *checksum*. Perhatikan bahwa ukuran *checksum* meningkat dari 16 (di UDP, TCP, dan IP) untuk 32 *bit* untuk memungkinkan penggunaan CRC *checksum*-32.

2. *Chunk* (Potongan Data)

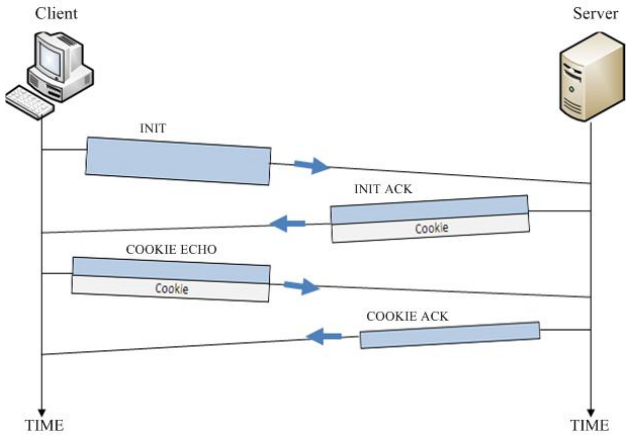
Control information atau pengguna data dibawa dalam potongan. Tiga bidang pertama umum untuk semua potongan, bidang informasi tergantung pada jenis potongan. Poin penting untuk diingat adalah bahwa SCTP memerlukan bagian informasi untuk kelipatan dari 4 *byte*, jika tidak, lapisan *byte* (delapan angka 0) ditambahkan di akhir bagian.

2.4.2 Asosiasi SCTP

Seperti TCP, SCTP merupakan protokol *connection-oriented*. Namun, sambungan dalam SCTP disebut asosiasi untuk menekankan *multihoming*.

1. Pembentukan Asosiasi

Pembentukan asosiasi dalam SCTP membutuhkan empat arah lintasan yang saling terhubung. Dalam prosedur ini, suatu proses, biasanya klien, ingin mendirikan asosiasi dengan proses lain, biasanya *server* menggunakan SCTP sebagai protokol lapisan *transport*. Serupa dengan TCP, SCTP *server* perlu dipersiapkan untuk menerima asosiasi (terbuka pasif). Asosiasi pendirian dibangun oleh klien (terbuka aktif).



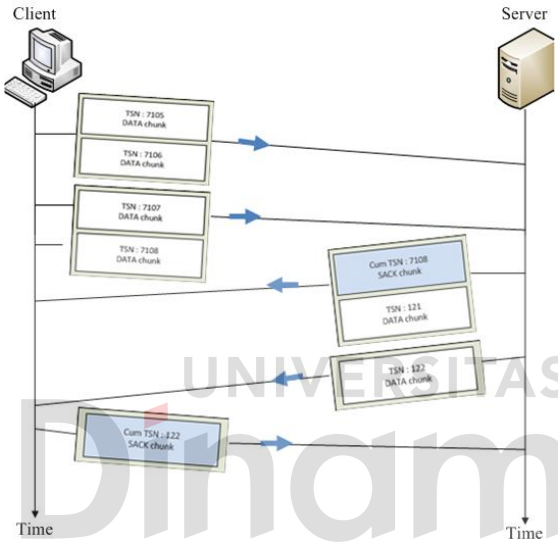
Gambar 2.13 *Four-Way Handshaking*

Sumber : (Forouzan, 2007)

Gambar 2.13 merupakan gambaran *four way handshake* untuk proses pembentukan asosiasi pada protokol SCTP.

2. Pengiriman Data

Tujuan utama asosiasi adalah untuk mengirim data antara kedua ujung. Setelah asosiasi ini dibangun, maka dua arah *transfer* data dapat terjadi. Klien dan server dapat saling mengirim data.



Gambar 2.14 *Simple Data Transfer*

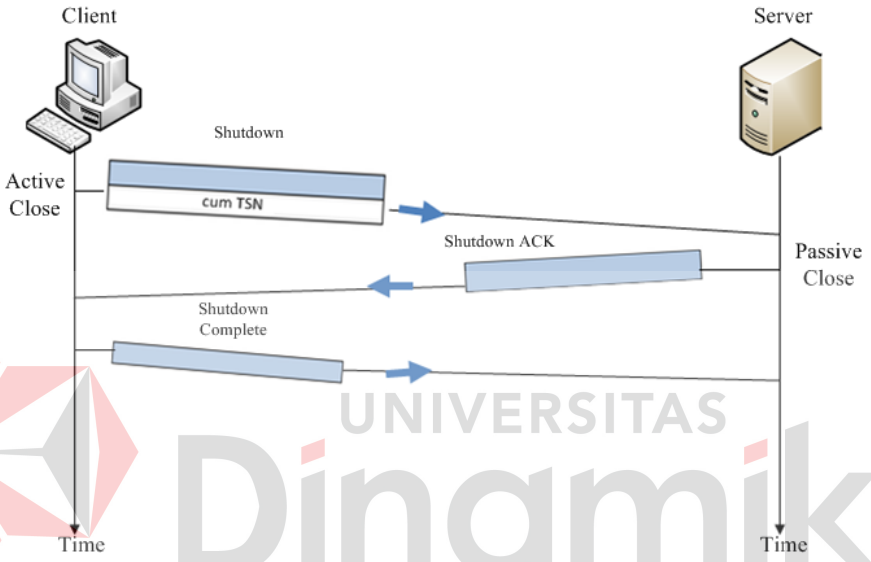
Sumber : (Forouzan, 2007)

Gambar 2.14 merupakan gambaran *simple Data Transfer* untuk proses pengiriman asosiasi pada protokol SCTP.

3. Pemutusan Asosiasi

Dalam SCTP, seperti TCP, salah satu dari dua pihak yang terlibat dalam pertukaran data (*client* atau *server*) dapat menutup koneksi.Namun, tidak seperti TCP, SCTP tidak mengijinkan situasi setengah menutup. Jika salah satu ujung menutup asosiasi, ujung yang lain harus menghentikan pengiriman data baru. Jika

ada data yang tersisa dalam antrian penerima permintaan penghentian, mereka akan dikirim dan asosiasi ditutup. Asosiasi pemutusan menggunakan tiga paket, seperti yang ditunjukkan pada Gambar 2.15. Perhatikan bahwa meskipun angka tersebut menunjukkan kasus di mana pemutusan dimulai oleh klien, juga dapat dimulai oleh server. Perhatikan bahwa tidak bisa beberapa skenario pemutusan hubungan asosiasi. (Forouzan, 2007)



Gambar 2.15 *Asosiation Termination*

Sumber : (Forouzan, 2007)

Gambar 2.13 merupakan gambaran *Asosiation Termination* untuk proses terminasi asosiasi pada protokol SCTP.

2.5 *Streaming* Multimedia

Streaming multimedia adalah suatu teknologi yang mampu mengirimkan file audio dan video digital secara real time pada jaringan komputer. Karakteristik Multimedia Data dapat dilihat sebagai berikut :

1. *Voluminous*: Membutuhkan *data rate* tinggi dan berukuran besar.

2. *Real-time and Interactive*:

   a. Membutuhkan *low delay*.

   b. Membutuhkan sinkronisasi dan interaktif. (Distribusi Multimedia, n.d).

2.6 *Internet Protocol Television* (IPTV)

IPTV (*Internet Protocol Television*) yaitu layanan multimedia dalam bentuk televisi, *video, audio, text, graphic,* data yang disalurkan kepada pelanggan melalui jaringan IP (*Internet Protocol*). IPTV juga memiliki fitur-fitur tambahan salah satunya adalah *codec* (*compression – decompression*) yaitu perangkat keras atau perangkat lunak yang digunakan untuk melakukan kompresi dan dekompresi terhadap data video dan audio.

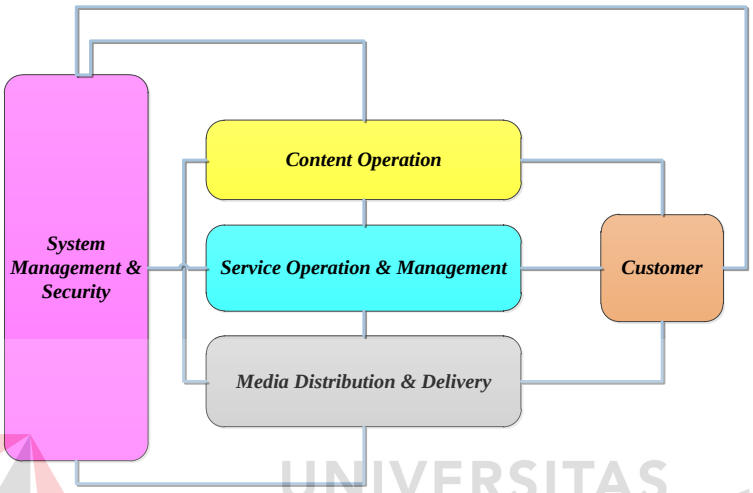
IPTV adalah suatu pengembangan baru dalam *software* komunikasi *client-server* yang mem-*broadcast* video yang berkualitas tinggi (setara *real time full motion video* secara simultan) ke *user window* melalui jaringan data yang ada sekarang. Beberapa *feature* yang dimiliki oleh IPTV ini adalah :

1. *Live atau prerecorded digital*, video program-program pendidikan, komersial, dan sebagainya, serta dapat melakukan *capturing* dan *transmission program* dari berbagai *source*.

2. *Scheduling***,** penjadwalan program sesuai dengan kebutuhan antara pemilik informasi dan *audience*. *Viewer* dapat memilih program dari suatu *listing* yang akan dilihatnya.

3. Mendukung format standard MPEG (*Motion Picturre Experts Group*), untuk memberikan *high quality, full motion video. Feature* ini merupakan tambahan terhadap standar *CODEC* (*compression/decompression*) untuk menjamin kualitas gambar yang optimal. (Rosalin, n.d).

Layanan IPTV menyajikan program-program TV interaktif dengan gambar berkualitas melalui jaringan Internet pitalebar (*broadband*) yang terkelola dengan baik. Sistem layanan IPTV terdiri dari 5 kelompok fungsi, yaitu : *Content Operation, System Management & Security, Service Operation & Management, Media Distribution & Delivery, Customer*, seperti terlihat pada Gambar 2.16.



Gambar 2.16 Kelompok Fungsi IPTV

Sumber : (Kartika, n.d)

a. *Content Operation Function Set*: Menyediakan program-program TV dan konten multimedia lainnya.

b. *System Management and Security Function Set*: Bertanggung jawab dalam pengawasan dan perlindungan sistem, menyediakan pengawasan kualitas layanan, pemeriksaan kegagalan, dan perlindungan layanan.

c. *Service Operation and Management Function Set*: Bertanggungjawab dalam pengendalian dan pengaturan khusus layanan IPTV.

d. *Media Distribution and Delivery Function Set*: *Stream* konten layanan IPTV dikirim ke *subscriber* disertai dengan fungsi-fungsi pengendalian, distribusi, penyimpanan dan *Streaming*. Sistem pengiriman dan distribusi media seharusnya

diterapkan berdasarkan pada topologi yang handal untuk mengimbangi permintaan efisiensi dan ketersediaan yang tinggi dengan harga yang tetap rendah.
e. *Customer Function Set*: Sekumpulan fungsi eksekusi layanan sistem IPTV pada sisi pelanggan. (Kartika, n.d).

2.7 *Voice Over IP* (VOIP)

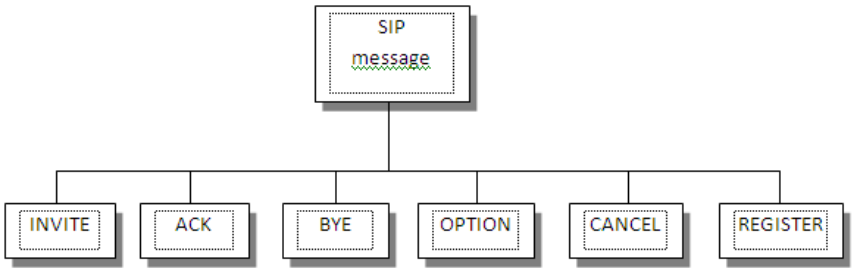
Salah satu aplikasi audio/video interaktif secara *real time* adalah *voice over IP* (VOIP), atau *Internet Telephony*. Idenya adalah untuk menggunakan internet sebagai suatu jaringan telepon dengan beberapa kemampuan tambahan. Aplikasi ini tidak berkomunikasi melalui jaringan *circuit-switched*, tetapi memungkinkan melakukan komunikasi antara dua pihak melalui internet *packet-switched*. Terdapat dua protokol yang telah dirancang untuk menangani komunikasi jenis ini yaitu SIP dan H.323.

2.7.1 SIP

Session initiation protocol (SIP) telah dirancang oleh *Internet Engineering Task Force* (IETF). SIP merupakan protokol yang berada pada *layer application* yang mendefinisikan proses pembentukan (*establishes*), pengaturan (*manages*), dan pengakhiran / pemutusan (*terminates*) suatu sesi komunikasi multimedia. Dapat digunakan untuk komunikasi dua pihak, beberapa pihak atau *multicast*. SIP dirancang untuk menjadi aplikasi yang independen yang mendasari *transport layer*; dan dapat berjalan pada UDP, TCP, atau SCTP.

a. *Messages*/pesan

SIP merupakan suatu protokol *text-based* (berbasis teks), HTTP. SIP, seperti HTTP, yang memanfaatkan pesan (*messages*). Terdapat enam pesan yang didefinisikan pada Gambar 2.17.



Gambar 2.17 SIP *Message*

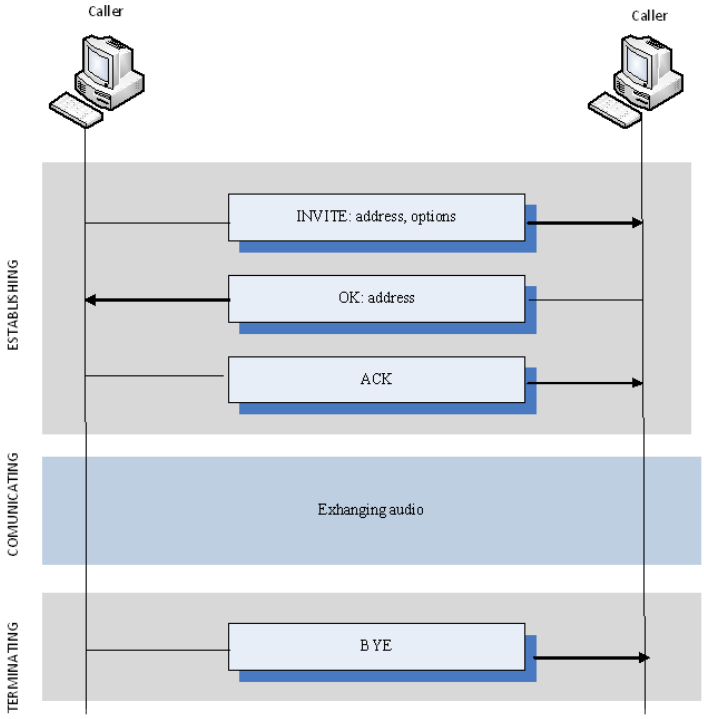
Sumber : (Forouzan, 2007)

Setiap pesan memiliki suatu *header* dan *body*. *header* terdiri dari beberapa baris yang mendeskripsikan struktur pesan, kemampuan pengirim (*caller*), jenis media dan sebagainya.

Pengirim menginisialisai suatu sesi dengan pesan *INVITE*. Setelah penerima menjawab panggilan, pengirim mengirimkan suatu pesan ACK untuk konfirmasi. Pesan *BYE* akan mengakhiri/pemutusan suatu sesi. Pesan *OPTIONS* akan mempertanyakan kemampuan/kapabilitas suatu mesin/*server*. Pesan *CANCEL* membatalkan proses inisialisasi yang telah dimulai. Pesan *REGISTER* membuat koneksi ketika penerima belum ada.

b. *Simple Session*/Sesi Sederhana

Suatu sesi menggunakan SIP terdiri dari 3 modul: pembentukan (*establishing*), komunikasi (*communicating*), dan pemutusan (*terminating*). Gambar 2.18 menunjukkan suatu sesi menggunakan SIP.



Gambar 2.18 *Simple Session* SIP

Sumber : (Forouzan, 2007)

1. *Establishing a Session*/Membentuk suatu sesi

Proses awal suatu sesi dalam SIP memerlukan *three-way handshake*. pengirim mengirim pesan INVITE, bisa menggunakan UDP, TCP, atau SCTP untuk memulai komunikasi. Jika penerima bersedia memulai sesi, penerima akan mengirimkan pesan balasan. Untuk mengkonfirmasi bahwa balasan tersebut sudah diterima, *caller* mengirimkan suatu pesan ACK.

2. Komunikasi

Setelah proses awal sesi sudah dibangun, pengirim dan penerima dapat berkomunikasi menggunakan dua *port* sementara.

3. Mengakhiri sesi

Sesi dapat diakhiri dengan suatu pesan *BYE* yang dikirimkan oleh pihak lain. (Forouzan, 2007).

2.8 *Quality of Service* (QoS)

Quality of Service (QoS) adalah sebuah rangkaian teknik untuk mengatur sumber daya jaringan dalam rangka untuk memungkinkan jaringan tersebut membedakan dan mengatasi lalu lintas jaringan berdasarkan kebijakan yang ada. Hal ini berarti menyediakan pengantaran data yang konsisten dan dapat diprediksikan kepada *user* atau aplikasi yang mendukung didalam sebuah jaringan. QoS dapat tercapai dengan mengatur parameter-parameter seperti *delay, delay variation, bandwidth dan packet loss* pada sebuah jaringan agar menjadi sebuah kunci sukses dalam sebuah solusi bisnis *end- to-end*. Kualitas suara dipengaruhi oleh dimensi QoS berikut:

1. *Throughput*

Adalah jumlah dari lalu lintas atau *bandwidth* yang disampaikan melalu sebuah periode waktu tertentu. Biasanya dalam lingkungan LAN semakin tinggi *throughput* semakin baik jaringan tersebut.

2. *Delay / Lantency*

Adalah waktu antara inisiasi suatu transaksi oleh suatu pengirim sampai ada respon pertama yang diterima oleh pengirim tersebut. Juga, waktu yang diperlukan untuk memindahkan paket dari sumber ke tujuan melalui jalur yang sudah ditentukan.

Percakapan interaktif menjadi terganggu ketika *delay* melebih 100-150 ms, ketika melebihi 200 ms, pemakai akan menemui gangguan dan menyatakan bahwa kualitas suara itu sangat lemah. Untuk menyediakan kualitas suara yang tinggi, jaringan *IP Telephony* harus mampu menjamin *latency* yang rendah. ITU-T G.114 merekomendasikan batas maksimum yang diterima dalam

perjalanan paket adalah 300 ms antara kedua gateway *IP Telephony* (150 ms *delay* per jalur).

Ada banyak kompenen dari *delay* dalam sebuah jaringan yang perlu dipahami, termasuk didalamnya adalah *packetization delay*, *queuing delay*, dan *propagation delay*.

a. *Packetization delay* adalah jumlah waktu yang diperlukan untuk mengambil *codec* yang digunakan untuk melengkapi konversi dari analog ke digital. *IP Telephony* selalu menciptakan beberapa ukuran *delay*, ketika algoritma yang menspesifikasikan "*listen"* atau contoh dari suara yang dispesifikasikan dalam periode waktu tertentu, diikuti oleh *packetization*.

b. *Propagation delay* adalah waktu yang diperlukan untuk mengambil informasi melewati sebuah kabel *copper*, *fiber* atau jalur *wireless*. *Delay* ini juga merupakan sebuah fungsi dari kecepatan cahaya, konstanta yang universal, dan kecepatan persinyalan dari media fisik. Sebagai contoh, jika sebuah panggilan sudah melalui sebuah *node transit*, maka *delay* diperkenalkan.

c. *Queuing delay* dibebankan kepada suatu paket pada titik *congestion* (lalu lintas pada jaringan melebihi kapasitas) ketika menantikan putarannya untuk diproses sementara paket yang lain yang dikirim melalui sebuah *switch* atau kabel. Sebagai contoh, ATM yang dikurangi *queuing delay*-nya dengan membagi paket ke dalam bagian kecil, membungkus mereka kedalam sebuah sel, lalu meletakkan mereka kedalam sebuah prioritas antrian yang absolut. Karena sel-sel

tersebut kecil, maka prioritas *queue* yang terbesar dapat dilayani lebih sering, mengurangi waktu tunggu untuk paket dalam sebuah antrian kedalam level yang *deterministik*. Bagaimanapun juga, pada kecepatan giga*bit*, waktu tunggu untuk lalu lintas yang mempunyai prioritas tinggi sangat kecil sekali bahkan di berbagai kondisi yang terburuk, dalam kaitannya dengan kecepatan link dan daya proses yang tersedia.

Delay = $(Tr - Ts)$ detik ..... $0 \leq t \leq T$ 2.1

Sumber : Darmawan, Alif, & Basuki, 2009

Keterangan :

Tr = Waktu penerimaan paket (detik)

Ts = Waktu pengiriman paket (detik)

t = Waktu Simulasi (detik)

T = Waktu pengambilan *sample* (detik)

3. *Delay Variation* (*Jitter* dan *Wander*)

Delay variation adalah perbedaan pada penundaan yang diperlihatkan oleh paket berbeda yang menjadi bagian dari arus lalu lintas yang sama. Frekuensi tinggi pada *delay variation* dikenal sebagai *jitter*, sedangkan frekuensi rendah pada *delay variation* disebut *wander*. *Jitter* disebabkan terutama oleh perbedaan dalam antrian waktu menunggu untuk paket yang berurutan didalam suatu arus. Jenis lalu lintas tertentu terutama *real-time* lalu lintas seperti suara adalah sangat tidak toleran dalam *jitter*. Perbedaan waktu tiba paket menyebabkan *choppiness* di dalam suara.

Jitter berlebihan dapat diatasi dengan penyangga, tetapi dapat menyebabkan permasalahan lain, seperti peristiwa *"walkie-talkie"* yang

disebabkan ketika dua sisi suatu percakapan mempunyai *latency*. *Jitter* harus kurang dari 60ms (60ms= rata-rata mutu, 20ms= mutu bea).

Jitter (i) = $(R_{i+1} - S_{i+1}) - (R_i - S_i)$ 2.2

Sumber : Darmawan, Alif, & Basuki, 2009

Keterangan :

R = Receive Time

S = Sent Time

4. *Packet loss*

Kerugian akibat kesalahan *bit* maupun paket yang hilang, mempunyai suatu dampak lebih besar atas jasa *IP telephony* atau VoIP dibanding pada data. Selama suatu transmisi suara, hilangnya berbagai *bit* atau paket suatu arus boleh menyebabkan suatu letusan dapat didengar yang akan menjadi mengganggu kepada pemakai. Didalam suatu transmisi data, hilangnya *bit* tunggal atau berbagai paket informasi hampir tidak pernah dicatat oleh para pemakai. Bagaimanapun, jika paket hilang sangat banyak, akan menurunkan kualitas pengiriman. Tingkat kerugian paket harus kurang dari 5% untuk mutu minimum dan kurang dari 1% untuk mutu bea. (Andrew, Susanto, & Aprilianto, 2006).

Packet Loss = $\left(\frac{Pd}{Ps}\right)$ x 100 % ..... $0 \leq t \leq T$ 2.3

Sumber : Darmawan, Alif, & Basuki, 2009

Keterangan :

Pd = Paket yang mengalami *drop* (paket)

Ps = Paket yang dikirimkan (paket)

T = Waktu Simulasi (detik)

t = waktu pengambilan *sampel* (detik)

Kendala-kendala yang dapat terjadi dalam melakukan *streaming* multimedia:

1. *Bandwidth* : *Bandwidth* sangat berpengaruh terhadap kualitas presentasi suatu data *stream*. Di samping kondisi jaringan juga mempengaruhi *bandwidth*, hal yang perlu diperhatikan adalah ukuran data *stream* harus sesuai dengan kapasitas *bandwidth* jaringan. Untuk mengatasinya digunakan kompresi data dan penggunaan *buffer*.
2. *Sinkronisasi* dan *delay* : Agar media yang berbeda sampai dan dipresentasikan pada *user* seperti aslinya, maka media tersebut harus tersinkronisasikan sesuai dengan *timeline* presentasi tersebut dan *delay* seminimal mungkin. Adanya kerugian *sinkronisasi* dan *delay* dapat disebabkan oleh kondisi jaringan yang buruk, sehingga mengakibatkan *timeline* presentasi menjadi kacau. (Distribusi Multimedia, n.d).

2.9 *Network simulator*-2

Network simulator (NS) pertama kali di bangun sebagai varian dari REAL Network Simulator pada tahun 1989 di UCB (*University of California Berkeley*). Pada tahun 1995 pembangunan NS di sukung oleh DARPA (*Defense Advance Research Project Agency*) melalui VINT (*Virtual Internet Testbed*) Project, yaitu sebuah tim riset gabungan yang beranggotakan tenaga ahli dari LBNL (*Lawrence Berkeley of National Laboratory*), *Xerox* PARC, UCB dan USC/ISI (*University of Southern California School of Engineering/Information Science Institute*). Tim gabungan ini membangun sebuah perangkat lunak simulasi

jaringan internet untuk kepentingan riset interaksi antar protokol dalam konteks pengembangan protokol internet pada saat ini dan masa yang akan datang.

Beberapa keuntungan menggunakan NS sebagai perangkat lunak simulasi pembantu analisis dalam riset, diantaranya:

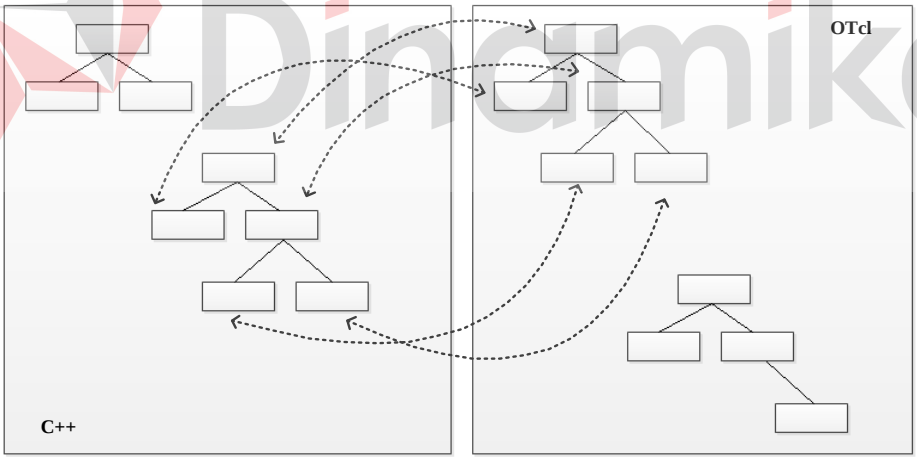
1. NS dilengkapi dengan *tool* validasi. *Tool* validasi digunakan untuk menguji validitas pemodelan yang ada pada NS. Secara default, semua pemodelan pada NS akan dapat melewati proses validasi ini. Jika ingin melakukan validasi terhadap pemodelan protokol yang ada pada *library* NS , dapat melakukannya dengan mengetikan ./*validate* pada *consol* saat berada pada *directory* NS2.
2. NS bersifat *open source* di bawah GPL (*GNU Public Licence*), sehingga dapat langsung di *download* dan di gunakan secara gratis. Sifat *open souce* juga mengakibatkan pengembangan NS menjadi lebih dinamis. Pemodelan media, protokol, *network* komponen dan perilaku trafik cukup lengkap bila dibandingkan dengan *software* sejenis lain. Ini disebabkan oleh pengembangan NS yang dilakukan oleh banyak periset dunia.

Nework Simulator (NS) mensimulasikan jaringan berbasis TCP/IP dengan berbagai macam medianya. Dapat juga digunakan untuk mensimulasikan protokol jaringan (TCPs/UDP/RTP), *traffic behaviour* (FTP, *Telnet*, CBR, dan lain-lain), *Queue manajemen* (RED, FIFO, CBQ) *algoritma routing unicast (Distance Vector, Link State) dan multicast*, (PIM SM, PIM DM, DVMRP, *Shared Tree* dan Bi *directional Shared Tree*), aplikasi multimedia yang berupa *layered video*, *Quality of Service* video-audio dan *transcoding.* NS juga mengimplementasikan beberapa MAC (IEEE 802.3, 802.11), di berbagai media, misalnya jaringan *wired*

(seperti LAN, WAN, *point to point*), *wireless* (seperti mobile IP, wireless LAN), bahkan simulasi hubungan antar node jaringan yang menggunakan media satelit.

2.9.1 Dasar Simulasi NS

Network simulator dibangun dengan dua bahasa pemrograman, yaitu C++ dan Tcl/OTcl. Hubungan antara Tcl/OTcl dengan C++ dapat diamati pada Gambar 2.19. C++ digunakan untuk *library* yang berisi *event scheduler*, protokol dan *network* komponen yag diimplementasikan pada simulasi oleh *user*. Tcl/OTcl digunakan pada *script* simulasi yang digunakan yang ditulis pada NS *user* dan pada *library* sebagai *simulator* objek. OTcl juga nantinya akan berperan sebagai *intrepeter*. Hubungan antar bahasa pemrograman dapat digambarkan sebagai berikut.



Gambar 2.19 Hubungan Tcl/OTcl Dengan C++

Sumber : (Wirawan & Indarto, 2004)

Bahasa C++ digunakan pada library karena C++ mampu mendukung *runtime* simulasi yang cepat, meskipun simulasi melibatkan simulasi jumlah paket dan sumber data dalam jumlah besar.

Bahasa Tcl memberikan *respon runtime* yang lebih lambat daripada C++, namun jika terdapat kesalahan, respon Tcl terhadap kesalahan *syntax* dan perubahan skrip berlangsung dengan cepat dan interaktif. *User* dapat mengetahui letak kesalahannya yang dijelaskan pada *console*, sehingga *user* dapat memperbaiki dengan cepat. Karena alasan itulah bahasa ini dipilih untuk digunakan pada skrip simulasi. (Wirawan & Indarto, 2004)

2.9.2 Cara Membuat dan Menjalankan Skrip NS

Dalam membuat skrip simulasi yaitu dengan menggunakan program teks *editor* yang terdapat pada linux,dan disimpan dalam sebuah folder dengan ekstensi .tcl. Contoh:

```
Simulasitcp.tcl
```

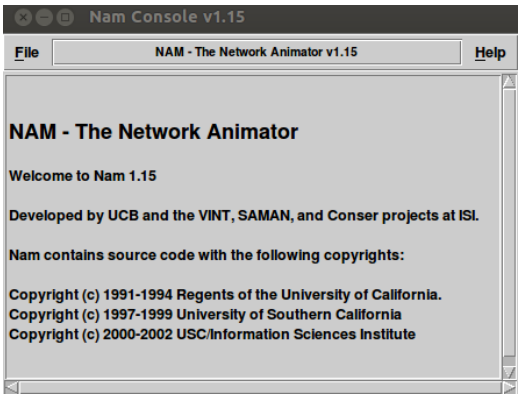
Untuk menjalankan simulasi yang telah dibuat, dapat dilakukan dengan masuk pada folder tersebut dan mengetikan NS serta nama file tcl simulasi yang ingin dijalankan. (Wirawan & Indarto, 2004). Contoh:

```
[root @accessnet your_folder]#ns
Simulasitcp.tcl
```

2.9.3 Output Simulasi NS

Pada saat satu simulasi berakhir, NS membuat satu atau lebih *file output text-based* yang berisi detail simulasi jika dideklarasikan pada saat membangun simulasi. Ada dua jenis output NS, yaitu : *file trace* yang akan digunakan untuk analisa numerik dan *file namtrace* yang digunakan sebagai input tampilan grafis simulasi yang disebut *network animator* (nam) yang ditampilkan pada Gambar 2.20. Jika ingin menampilkan tampilan grafis simulasi, harus menjalankan *system*

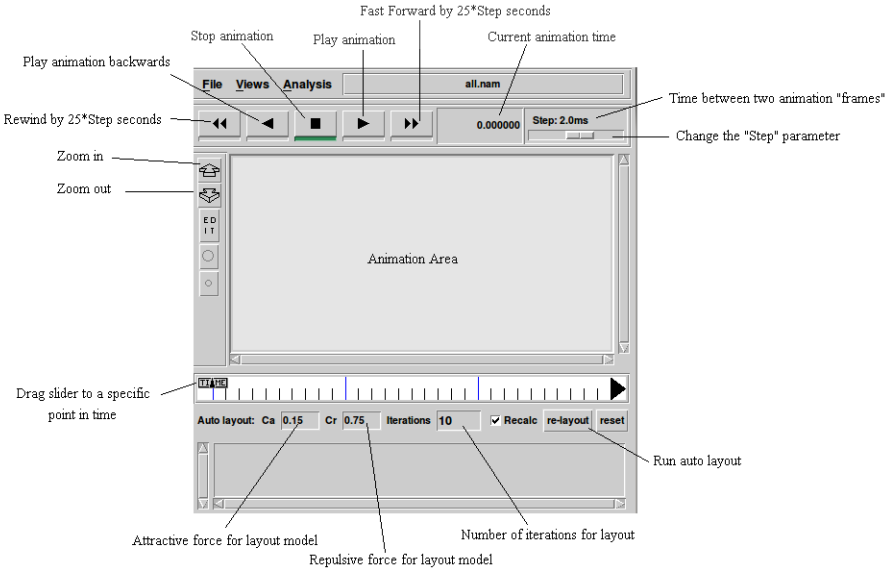
X *Windows* pada linux, yang bisa berupa KDE atau Gnome. (Wirawan & Indarto, 2004)



Gambar 2.20 Nam Konsole

Sumber : (Wirawan & Indarto, 2004)

Nam memiliki *interface grafis* yang mirip dengan CD *player* (*play*, *rewind*, *fast forward*, *pause*, dan lain-lain) dan juga memiliki kontroler kecepatan tampilan simulasi, seperti ditampilkan pada Gambar 2.21 berikut.



Gambar 2.21 *Network Animator* (Nam)

Sumber : (Wirawan & Indarto, 2004)

2.9.4 Dasar Bahasa TCL dan OTCL

Tcl (*Tool Command Language*) adalah *string-based command language*. Bahasa Tcl diciptakan oleh John Ousterhout pada akhir tahun 1980-an sebagai *command language* dengan *tool interaktif*. Tcl didesain untuk menjadi semacam "perekat" yang membangun *software building block* menjadi suatu aplikasi.

Otcl (*Object Oriented*) adalah ekstensi tambahan pada tcl yang memungkinkan fungsi *object oriented* (OO) pada tcl. Ini memungkinkan pendefinisian dan penggunaan *class* Otcl. NS telah menyediakan objek simulasi yang berupa *class* pemrograman dalam Otcl. (Wirawan & Indarto, 2004).

2.9.5 *Transport Agent* Pada NS

Pada jaringan internet, dikenal istilah *layer* komunikasi. Ada 4 *layer* komunikasi TCP/IP yaitu : *layer* aplikasi, *transport*, IP dan *network access*. Lapisan transpor merupakan *layer* komunikasi yang mengatur komunikasi data yang akan digunakan oleh lapisan aplikasi di atasnya. NS mensimulasikan lapisan transpor dengan objek simulasi yang bernama *transport agent*. Pada NS, *transport agent* diletakan diatas node.

Pada simulasi pengiriman data, *transport agent* tidak dapat berdiri sendiri. *Transport agent* membutuhkan lapisan aplikasi di atasnya yang berfungsi sebagai trafik generator. Protokol lapisan tranpor data yang didukung *network simulator* diantaranya :

1. TCP

Network simulator mendukung 2 jenis TCP agent, yaitu *one way TCP agent* dan *two way TCP agent*. Perbedaan kedua jenis TCP agent adalah :

a. *Two way TCP agent mensupport* proses *handshaking* pada saat *connection setup*, sehingga koneksi dapat dibangun, atau *drop* tergantung pada kondisi jaringannya. *One way TCP* tidak mendukung proses *handshaking*. Pertukaran data menggunakan *agent* ini di asumsikan telah melewati proses *handshaking.*

b. *Two way TCP agent* mendukung data *transfer* dua arah.

c. Penomoran pada jumlah *byte* yang di *transfer*, bukan pada jumlah paket

Simulasi koneksi pada *one way TCP* dilakukan dengan menggunakan 2 *agent* yang berpasangan, yaitu TCP *sender* dan TCP *Sink. Network Simulator* 2 mendukung beberapa jenis TCP *sender agent* yaitu :

a. TCP *Sender base* (Tahoe TCP)

Agent/TCP

b. Reno TCP

Agent/TCP/Reno

c. New Reno TCP

Agent/TCP/NewReno

d. Vegas TCP

Agent/TCP/Vegas

e. SACK (*Selective* ACK) TCP

Agent/TCP/Sack1

f. FACK (*Forward* ACK)TCP

Agent/TCP/Fack

Masing-masing *sender agent* pada NS memiliki perilaku yang sama dengan implementasi masing-masing jenis TCP di dunia nyata. TCP Sink

bertugas mengirimkan ACK per paket yang diterima pada TCP sender pasangannya. Beberapa macam TCP Sink yang di *support* NS yaitu :

a. *Base* TCP Sink

Agent/TCPSink

b. *Delayed* ACK

Agent/TCPSink/DelAck

c. *Sack* TCP Sink

Agent/TCPSink/Sack

d. *Delayed* ACK dengan *Sack*

Agent/TCPSink/Sack1/DelAck

Pada *Two way TCP agent*, NS hanya mensimulasikan satu jenis *two way TCP agent* yaitu:

Agent/TCP/FullTCP

Pada saat ini *two way TCP* hanya disimulasikan dengan Reno *congestion control*. Pada *two way* TCP, pembuatan TCP sink sama dengan *source* TCP, namun sink ditempatkan pada kondisi *listen.*

2. UDP

Koneksi dengan menggunakan UDP pada NS dilakukan dengan menggunakan *agent* UDP sebagai pengirim dan *agent* Null sebagai penerima. UDP *sender agent* merupakan *agent* pengirim, ditetapkan pada NS sebagai :

Agent/UDP

Agent UDP pada NS memiliki penomoran dan *time stamp* seperti yang dimiliki RTP. Meskipun paket UDP yang sebenarnya tidak mengandung penomoran atau *time stamp*, penomoran dan *time stamping* yang ada sangat

berguna untuk analisis data hasil simulasi. Penomoran dan *time stamp* ini tidak mempengaruhi *overhead agent* UDP. *Agent null* merupakan pasangan UDP sebagai tujuan trafik. (Wirawan & Indarto, 2004).

Agent/Null

2.9.6 Level Aplikasi Pada NS

NS mensimulasikan level aplikasi dalam bentuk *simulated application* dan generator trafik. Pada sistem dunia nyata, aplikasi terhubung dengan lapisan transpor yang ada dibawahnya melalui sebuah *Application Program Interface* (API). Jenis API yang umum digunakan yaitu *sockets*. Pada NS, yang bekerja untuk menghubungkan *transport agents* dengan aplikasi adalah API berupa *function*. Ada dua tipe dasar aplikasi yang disimulasikan pada NS, yaitu :

1. *Simulated Application*

Pada saat ini terdapat dua aplikasi yang disimulasikan oleh NS, yaitu :

a. FTP

Application/FTP

FTP dibangun untuk mensimulasikan *bulk data transfer*.

b. Telnet

Application/Telnet

Dibangun untuk mensimulasikan transfer data dengan ukuran kecil.

2. Generator Trafik

Generator trafik membangkitkan trafik dengan metode *on/off*. Selama periode *on* paket membangkitkan data dengan kecepatan tetap, dan selama periode *off* tidak satupun trafik yang dibangkitkan. *Inter arrival paket* dibangkitkan sesuai

dengan fungsi trafik generator yang di gunakan dengan rataan yang sesuai dengan konfigurasi. Objek generator trafik terbagi atas 4 tipe, yaitu :

a. Exponensial

Application/Traffic/Exponential

Generator trafik ini membangkitkan trafik dengan *inter arrival time* antarpaket sesuai dengan fungsi *exponensial*.

b. Pareto

Application/Traffic/Pareto

Generator traffic ini membangkitkan trafik dengan *inter arrival time* antarpaket sesuai dengan fungsi *pareto.*

c. *Constant Bit rate* (CBR)

Application/Traffic/CBR

Fungsi ini membangkitkan data secara kontinu dengan *bit rate* yang konstan.

d. Trafik Trace

Application/Traffic/Trace

Generator ini membangkitkan trafik dari sebuah *trace file*. (Wirawan & Indarto, 2004).

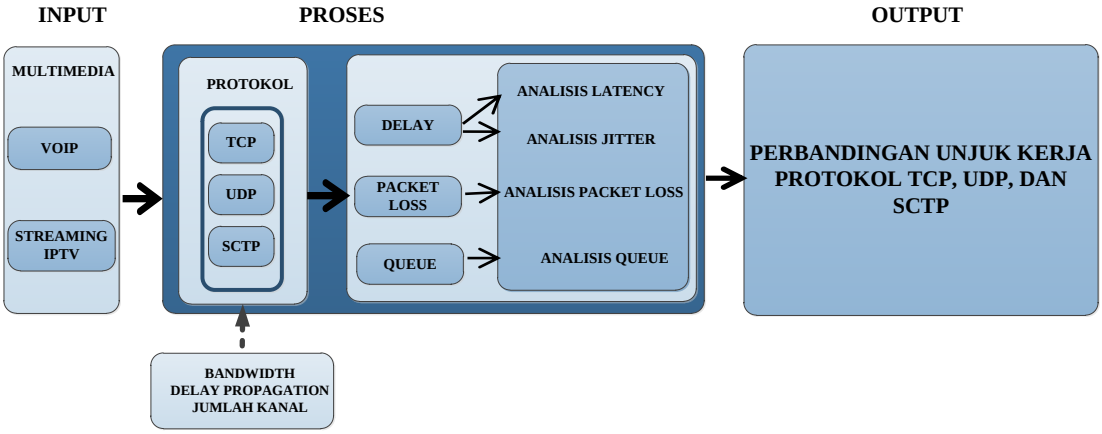
BAB III

METODE PENELITIAN DAN PERANCANGAN SISTEM

3.1 Metode Penelitian

Metode penelitian yang digunakan pada tugas akhir ini dilakukan dengan mencari informasi tentang data yang dibutuhkan dalam melakukan tugas akhir ini. Data-data tersebut meliputi karakteristik protokol yang digunakan (TCP, UDP dan SCTP), data CBR sebagai input data yang merepresentasikan data multimedia VOIP dan IPTV, serta informasi tentang parameter-parameter yang digunakan dalam perhitungan. Setelah didapatkan informasi mengenai hal-hal yang dibutuhkan maka langkah selanjutnya adalah melakukan perancangan simulasi menggunakan *software* NS-2 untuk menghasilkan informasi data yang nantinya akan digunakan dalam proses perhitungan dan analisis.

Gambar 3.1 merupakan gambar blok diagram sistem yang merupakan penjelasan singkat dari perancangan simulasi yang dibuat pada judul tugas akhir "Analisis Perbandingan Unjuk Kerja Protokol TCP, UDP dan SCTP Menggunakan Simulasi Lalu Lintas Data Multimedia" yang menggunakan *software* Network Simulator 2.35.



Gambar 3.1 Blok Diagram Sistem

Dari blok diagram sistem Gambar 3.1, terbagi menjadi 3 kelompok bagian, yaitu bagian input, proses dan output.

3.1.1 *Input data*

Bagian *input* adalah proses di mana data mulai berjalan yaitu berupa *input* trafik data. Pada proses simulasi, data VOIP yang digunakan dengan *codec* G.711 di gambarkan menggunakan trafik CBR dengan *packet size* 160 *byte*. (Wang, et al., 2008). Sedangkan data *streaming* IPTV yang digunakan menggunakan trafik CBR dengan *packet size* 1300 *byte*. (CTTL, 2008).

Data VOIP dan IPTV ini dibawa oleh protokol TCP, UDP dan SCTP dengan waktu yang tidak bersamaan karena proses simulasi dilakukan secara terpisah dengan topologi yang berbeda.

3.1.2 Bagian Proses

Pada bagian proses ini akan dibangun simulasi jaringan menggunakan NS-2. Disini penulis mengkonfigurasi jaringan, mulai membuat desain topologi yang

digunakan, mengatur skenario simulasi dan parameter eksternal hingga menjalankan konfigurasi *.tcl.

Bagian proses terdiri dari tiga protokol *transport layer* yang dibandingkan yaitu TCP, UDP dan SCTP. Simulasi yang dilakukan menggunakan topologi *dumb-bell.* Pada proses simulasi ini terdapat parameter eksternal yang digunakan untuk memberikan efek dalam membandingkan ke tiga protokol berdasarkan kondisi-kondisi yang telah ditentukan. Parameter eksternal meliputi *bandwidth*, *delay propagation,* dan jumlah kanal. Besarnya nilai dari parameter-parameter eksternal ini akan diijelaskan pada subbab berikutnya.

Setelah desain topologi dan konfigurasi dari beberapa skenario yang dibentuk selesai dibuat, kemudian dilakukan proses pengambilan data *delay* yang berguna untuk menghasilkan nilai *latency* yang merupakan rata-rata dari *delay* dan menghasilkan nilai *jitter* yang merupakan variasi *delay*. Data *delay* didapat dari simulasi *.tcl yang menghasilkan out.tr yang berisi segala macam informasi mengenai proses pengiriman data mulai dari node-node sumber hingga node-node tujuan. Setelah data *trace file* diperoleh, data ini kemudian di *parsing* dengan perl untuk mendapatkan data *delay*. Parameter yang dihasilkan selanjutnya yaitu *packet loss* dan *queue. Packet loss* dan *queue* didapat dari simulasi *.tcl yang mengahasilkan output qm.out. Dari informasi qm.out didapatkan beragam informasi yang berada pada jalur *bottleneck*. Di jalur ini digunakan sebagai jalur antrian data untuk di proses pada *router* dengan *bandwidth* dan *delay propagation* yang telah ditetapkan yang akan di kirimkan menuju node tujuan. *Packet loss* sengaja diambil dari jalur ini disebabkan karena karakteristik *retransmisi* dari TCP dan SCTP yang menyebabkan data akan dikirimkan ulang saat paket data

hilang saat pengiriman, sehingga akan lebih efektif jika di lakukan pengambilan informasi di jalur *bottleneck*. Data *queue* diambil dari qm.out untuk mengetahui kapasitas antrian yang digunakan masing-masing protokol untuk proses pengiriman. Hasil dari ke pengambilan data ini berupa analisis.

1. Analisis perbandingan *latency*: yaitu rata-rata dari *interval* waktu yang dibutuhkan oleh suatu paket data saat data mulai dikirim dan keluar dari proses antrian dari titik sumber awal hingga mencapai titik tujuan.
2. Analisis perbandingan *packet loss*: yaitu besarnya kehilangan paket data saat berada pada jalur *bottleneck*. Semakin kecil jumlah paket yang hilang, maka semakin bagus suatu protokol tersebut dalam proses pengiriman.
3. Analisis perbandingan *jitter : Jitter* merupakan variasi dari *delay*. Didapat dari hasil *trace file* yang di *parsing* dengan pemrograman bahasa perl untuk mencari *delay*. Setelah didapatkan hasilnya maka dapat ditentukan variasi dari *delay* dengan mengurangkan *delay* kedua dengan *delay* pertama, *delay* ketiga dengan *delay* kedua, dst. Semakin besar *jitter*, maka semakin tidak stabil waktu pengiriman suatu protokol tersebut.
4. Analisis perbandingan *queue : Queue* merupakan besarnya kapasitas antrian yang digunakan oleh masing-masing protokol. Setiap protokol berbeda dalam penggunaan *queue* bergantung dari karakteristik protokol itu. *Queue* didapat dari jalur *bottleneck* menggunakan skrip *queue* dari file *output* qm.out.

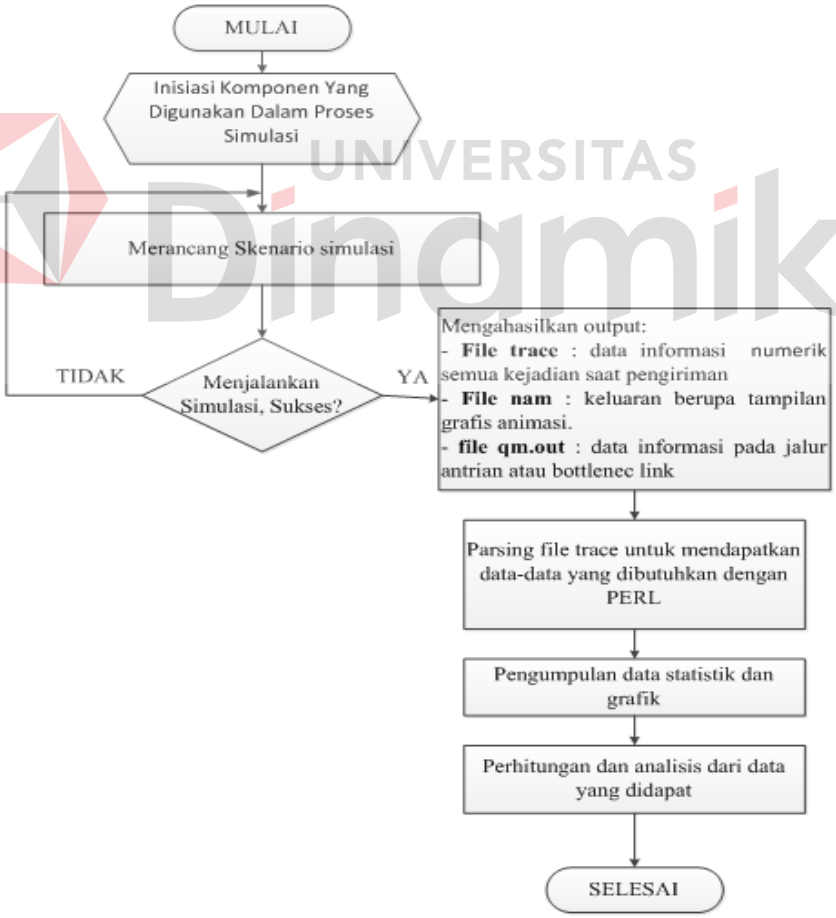
3.1.3 Bagian *output*

Setelah data didapat dan dilakukan analisis terhadap kualitas layanan dari masing-masing data multimedia, maka akan di hasilkan suatu protokol dari ketiga protokol yang dibandingkan, apakah TCP, UDP atau SCTP dengan kondisi

parameter yang telah ditentukan yang karakteristiknya sesuai untuk memenuhi kriteria layanan dalam membawa data multimedia.

3.2 Arsitektur Sistem Jaringan

NS2 tidak hanya dapat mensimulasikan kejadian yang sedang berlangsung di jaringan, tapi juga dapat mengkalkulasikan berbagai macam perhitungan seperti perhitungan parameter-parameter QoS serta membuat grafik dari hasil simulasi. Berbagai kelengkapan yang dimiliki ns2 tersebut membuat simulator jaringan ini banyak digunakan sebagai *tester* pengembangan jaringan. (Darmawan, Alif, & Basuki, 2009). Gambar 3.2 merupakan alur pengambilan data simulasi pada NS2



Gambar 3.2 Diagram Alur Proses Simulasi NS2

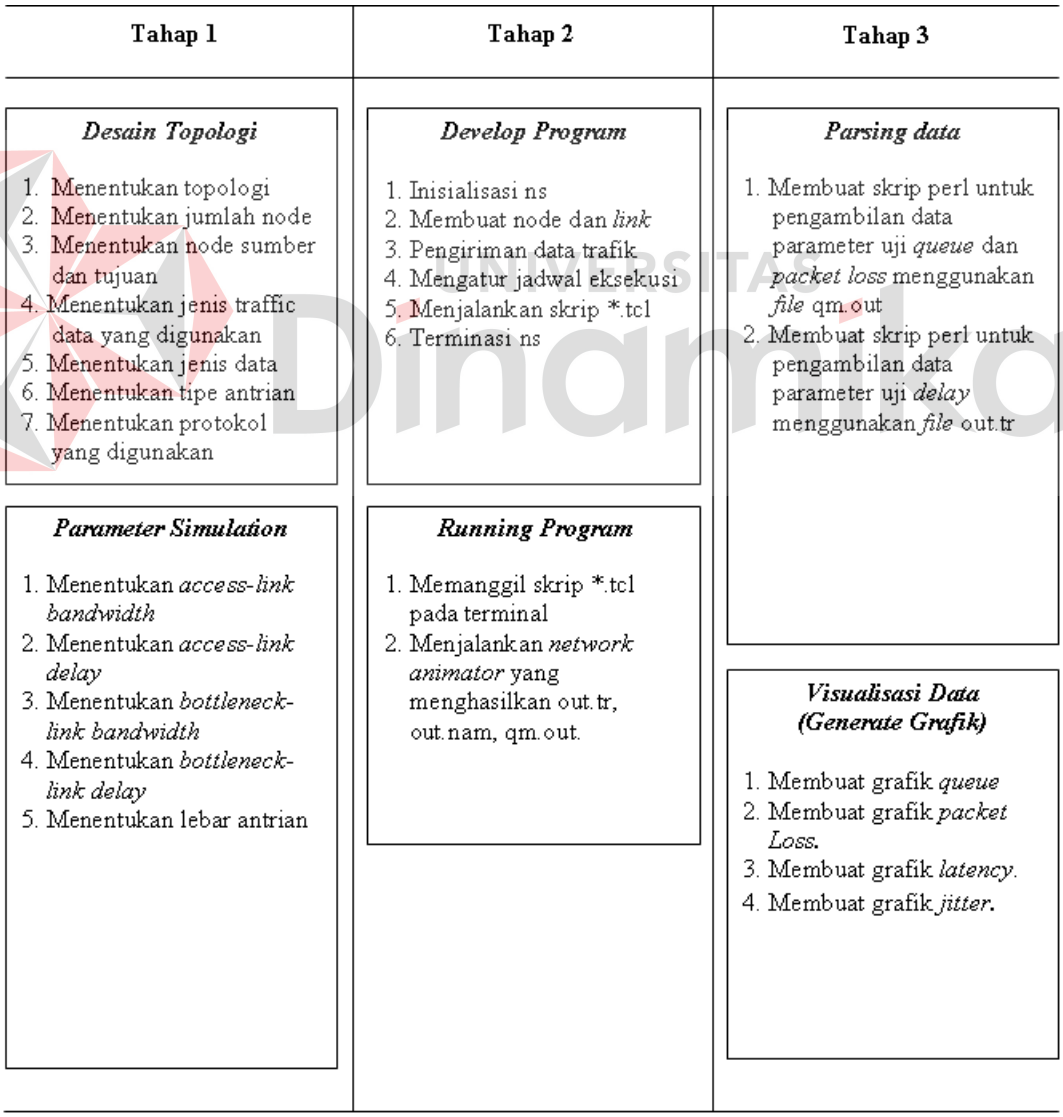
Simulasi dijalankan berdasarkan tcl *script* dan terbentuk file NAM yang menggambarkan topologi jaringan. Simulasi akan berjalan sesuai dengan skenario yang dirancang, setelah simulasi berjalan akan dihasilkan *output* data statistik berupa out.tr dan qm.out. out.tr tersebut menggambarkan segala macam kejadian yang terjadi selama simulasi berlangsung, sedangkan qm.out menggambarkan segala macam kejadian yang terjadi pada jalur *bottleneck. File trace* dari out.tr dan file *queue monitoring* dari qm.out tesebut digunakan sebagai acuan dalam pengukuran QoS (berdasarkan parameter QoS) dengan bantuan file PERL yang berisi rumus perhitungan parameter QoS. File PERL akan menampilkan persentasi dan hasil pengukuran berdasarkan parameter-parameter yang digunakan. Dalam menjalankan hasil dari *.tcl dan perl dilakukan pada terminal. Untuk memperoleh data *delay*, file perl dijalankan bersama dengan out.tr, sedangkan untuk mendapatkan nilai *packet loss* dan *queue*, file perl yang dibuat dijalankan bersama dengan qm.out. setelah semua data berhasil di dapatkan, maka langkah selanjutnya adalah membuat grafik masing-masing percobaan dengan gnuplot. Dari gnuplot ini akan dihasilkan tampilan data grafis sesuai dengan data masing-masing yang telah dihasilkan, sedangkan untuk menggabungkan data yang sama dari ketiga protokol untuk mempermudah proses analisis menggunakan grafik pada *microsoft excel* 2007.

3.3 Simulasi Sistem

Pada perancangan simulasi yang dilakukan, dapat dilihat berdasarkan tahap setiap proses simulasi yang akan di jalankan pada bagan Gambar 3.3.

Gambar 3.3 merupakan Bagan tahap proses simulasi NS2 yang akan dilakukan. Pada bagan ini proses simulasi dapat dibagi ke dalam tiga kelompok.

Tahap pertama yang dilakukan adalah membuat desain topologi simulasi dan menentukan parameter yang akan digunakan dalam pembuatan skrip *.tcl. Tahap kedua yaitu membuat skrip *.tcl pada NS2 dan menjalankan skrip sesuai dengan pengaturan dan parameter yang ditentukan. Tahap yang ketiga adalah melakukan *parsing* data atau pengambilan data yang diinginkan yaitu nilai *latency*, *jitter*, *packet loss*, dan *queue*. Dan proses terakhir yaitu membuat grafik menggunakan gnuplot. Penjelasan perancangan lebih jelasnya dijelaskan pada sub bab berikutnya.



Gambar 3.3 Bagan Tahap Proses Simulasi NS2

3.3.1 Desain Topologi

Pada simulasi ini digunakan model topologi *dumb-bell*. Model topologi *dumb-bell* ini digunakan untuk mempelajari tentang efek jalur *bottleneck* oleh banyak node sumber. (Computer Science Department, 2007).

1. Simulasi Pertama dengan menggunakan satu jalur VOIP maupun IPTV berdasarkan Gambar 3.4.

Gambar 3.4 Simulasi Jalur 1 Kanal

Pada Simulasi dengan menggunakan satu jalur data VOIP maupun satu jalur data IPTV ini diharapkan proses pengiriman data berjalan tanpa adanya kongesti (jalur sibuk). Saat pengiriman data VOIP dijalankan, maka jalur pada data IPTV akan dihentikan, dan sebaliknya.

a. Jumlah node yang digunakan pada simulasi ini berjumlah 6 node

b. Node 0 mengirimkan data VOIP yaitu trafik CBR 160 *byte* dengan tujuan akhir node 4. Dengan menggunakan jalur *duplex* dan tipe antrian *DropTail*.

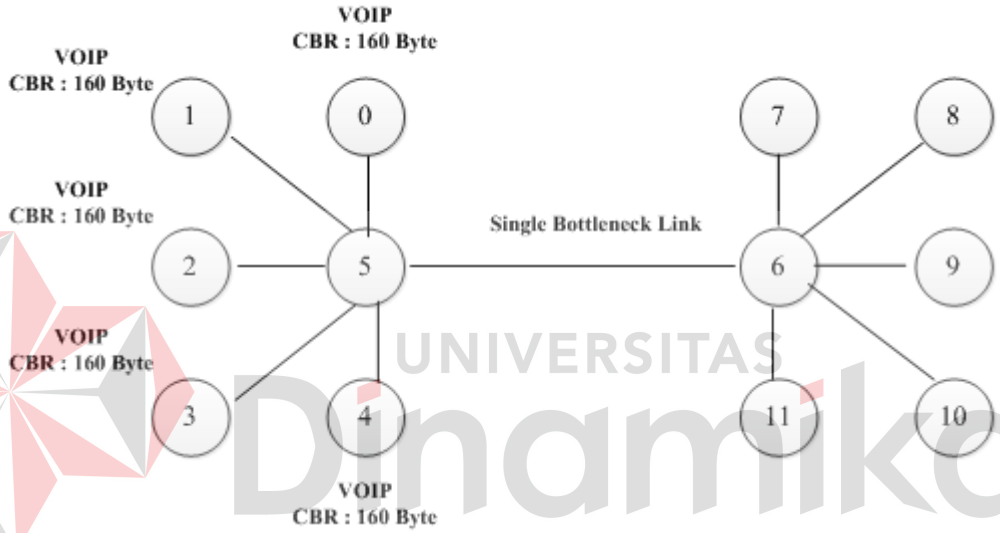
c. Node 1 mengirimkan data IPTV yaitu trafik CBR 1300 *byte* dengan tujuan akhir node 5. Dengan menggunakan jalur *duplex* dan tipe antrian *DropTail.*

d. Pada jalur *single bottleneck* yaitu node 2 ke node 3 dan sebaliknya menggunakan jalur *simplex*

e. Proses pengiriman data dimulai pada detik ke 0.1 dan berakhir pada detik ke 10.0 sedang proses simulasi berakhir pada detik ke 10.5.

f. Simulasi dilakukan dengan menggunakan protokol TCP, UDP dan SCTP untuk masing-masing topologi ini.

2. Simulasi Kedua dengan menggunakan 5 jalur VOIP berdasarkan Gambar 3.5.



Gambar 3.5 Simulasi Jalur 5 Kanal VOIP

Pada simulasi yang kedua yaitu data VOIP menggunakan 5 jalur data dan satu *bottleneck link*. Simulasi ini dimaksudkan untuk memberikan kongesti pada jalur *bottleneck link* sehingga unjuk kerja protokol akan dapat diketahui saat jalur mengalami kongesti.

a. Node 0 mengirimkan data VOIP yaitu trafik CBR 160 *byte* dengan tujuan akhir node 7 dengan menggunakan jalur *duplex* dan jenis antrian *DropTail.*

b. Node 1 mengirimkan data VOIP yaitu trafik CBR 160 *byte* dengan tujuan akhir node 8 dengan menggunakan jalur *duplex* dan jenis antrian *DropTail.*

c. Node 2 mengirimkan data VOIP yaitu trafik CBR 160 *byte* dengan tujuan akhir node 9 dengan menggunakan jalur *duplex* dan jenis antrian *DropTail.*

d. Node 3 mengirimkan data VOIP yaitu trafik CBR 160 *byte* dengan tujuan akhir node 10 dengan menggunakan jalur *duplex* dan jenis antrian *DropTail.*

e. Node 4 mengirimkan data VOIP yaitu trafik CBR 160 *byte* dengan tujuan akhir node 11 dengan menggunakan jalur *duplex* dan jenis antrian *DropTail.*

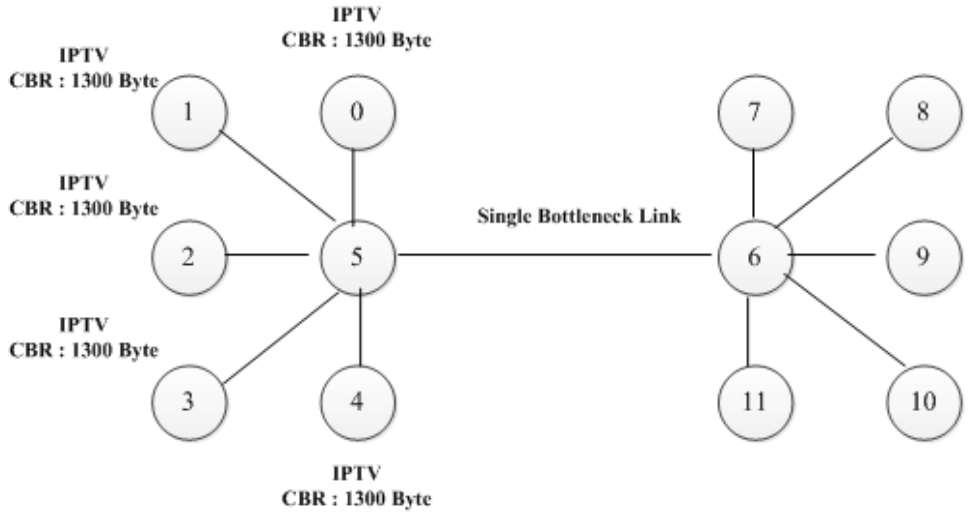
f. Pada jalur single *bottleneck* yaitu antara node 5 ke node 6 dan sebaliknya menggunaka jalur *simplex.*

g. Proses pengiriman data dimulai pada detik ke 0.1 dan berakhir pada detik ke 10.0 sedang proses simulasi berakhir pada detik ke 10.5.

h. Simulasi dilakukan dengan menggunakan protokol TCP, UDP dan SCTP untuk masing-masing topologi ini.

i. Pengamatan dilakukan pada node 0 ke node 7 sedangkan pengiriman pada node yang lain hanya dimaksudkan untuk membuat kongesti dari jalur *bottleneck.*

3. Simulasi Ketiga dengan menggunakan 5 jalur IPTV berdasarkan Gambar 3.6.



Gambar 3.6 Simulasi Jalur 5 Kanal IPTV

Pada simulasi yang ketiga yaitu data IPTV menggunakan 5 jalur data dan satu *bottleneck link.* Simulasi ini dimaksudkan untuk memberikan kongesti pada jalur *bottleneck link* sehingga unjuk kerja protokol akan dapat diketahui saat jalur mengalami kongesti.

a. Node 0 mengirimkan data IPTV yaitu trafik CBR 13000 *byte* dengan tujuan akhir node 7 dengan menggunakan jalur *duplex* dan jenis antrian *DropTail*.

b. Node 1 mengirimkan data IPTV yaitu trafik CBR 13000 *byte* dengan tujuan akhir node 8 dengan menggunakan jalur *duplex* dan jenis antrian *DropTail*.

c. Node 2 mengirimkan data IPTV yaitu trafik CBR 13000 *byte* dengan tujuan akhir node 9 dengan menggunakan jalur *duplex* dan jenis antrian *DropTail*.

d. Node 3 mengirimkan data IPTV yaitu trafik CBR 13000 *byte* dengan tujuan akhir node 10 dengan menggunakan jalur *duplex* dan jenis antrian *DropTail*.

e. Node 4 mengirimkan data IPTV yaitu trafik CBR 13000 *byte* dengan tujuan akhir node 11 dengan menggunakan jalur *duplex* dan jenis antrian *DropTail*.

f. Pada jalur single *bottleneck* yaitu antara node 5 ke node 6 dan sebaliknya menggunaka jalur *simplex.*

g. Proses pengiriman data dimulai pada detik ke 0.1 dan berakhir pada detik ke 10.0 sedang proses simulasi berakhir pada detik ke 10.5.

h. Simulasi dilakukan dengan menggunakan protokol TCP, UDP dan SCTP untuk masing-masing topologi ini.

3.3.2 Parameter Simulasi

Parameter simulasi yang digunakan dalam tugas akhir ini yaitu node-node yang berada pada *access link* diberikan *bandwidth* dan *delay propagation* sebesar 1 Mb dan 15 ms dengan lebar antrian untuk jalur *bottleneck* sebesar 20. Nilai ini diambil dari penelitian yang dilakukan oleh Budiardjo dan Thiotrisno (2003). Besar nilai *bandwidth* dan *delay propagation* pada jalur *bottleneck link* dapat dijelaskan menggunakan Tabel 3.1 nilai parameter eksternal berikut.

Tabel 3.1 Nilai Parameter Jalur *Bottleneck*

VOIP *Packet Size* 160 *Byte*			IPTV *Packet Size* 1300 *Byte*		
Protokol	*Delay propagation*	*Bandwidth-Bottleneck Link*	Protokol	*Delay propagation*	*Bandwidth-Bottleneck Link*
TCP	100 ms	128 Kb	TCP	100 ms	512 Kb
		384 Kb			1 Mb
		512 Kb			3 Mb
	300 ms	128 Kb		300 ms	512 Kb
		384 Kb			1 Mb
		512 Kb			3 Mb

UDP	100 ms	128 Kb	UDP	100 ms	512 Kb
		384 Kb			1 Mb
		512 Kb			3 Mb
	300 ms	128 Kb		300 ms	512 Kb
		384 Kb			1 Mb
		512 Kb			3 Mb
SCTP	100 ms	128 Kb	SCTP	100 ms	512 Kb
		384 Kb			1 Mb
		512 Kb			3 Mb
	300 ms	128 Kb		300 ms	512 Kb
		384 Kb			1 Mb
		512 Kb			3 Mb

Nilai *delay propagation* yang diberikan berdasarkan pada standard Qos menurut jurnal Chandra & Iskandar : 2012 pada Tabel 3.2:

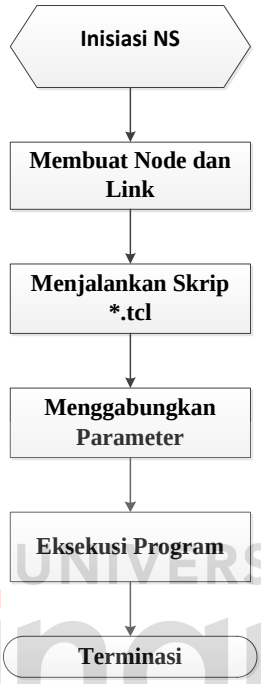
Tabel 3.2 Kategori *delay*

Delay	Kategori
0-150 ms	Dapat diterima untuk hampir seluruh aplikasi
150 -400 ms	Secara umum dapat di terima
> 400 ms	Tidak dapat diterima

Dari referensi tersebut, simulasi tugas akhir ini mengambil nilai *delay* 100 ms, yaitu nilai *delay* yang dapat diterima untuk hampir seluruh aplikasi dan 300 ms, yaitu nilai *delay* yang secara umum dapat diterima untuk layanan aplikasi. Sedangkan nilai *bandwidth* dari VOIP diambil dari nilai *range bandwidth* terendah yaitu 128 Kb, menengah yaitu 384 Kb (Deziel, 2013) dan *bandwidth* yang berukuran tinggi dalam kasus menggunakan data VOIP yaitu 512 Kb. (Unuth, 2013). Nilai *bandwidth* untuk IPTV diambil dari hasil penelitian yang didapat sebelumnya yang menggunakan nilai *bandwidth* 512 Kb, 1 Mb dan 3 Mb. (LAKSONO, 2010). Jumlah kanal pada proses simulasi yang menggunakan data VOIP dan IPTV akan dibentuk dalam dua jenis jumlah saluran data untuk masing-masing protokol TCP, UDP dan SCTP. Jenis pertama yaitu menggunakan satu

kanal dan yang kedua menggunakan lima kanal. Parameter-parameter eksternal ini akan di gabungkan dalam satu topologi simulasi untuk mengahasilkan data yang dibutuhkan.

3.3.3 Membuat skrip *.tcl



Gambar 3.7 Alur Pembuatan Skrip *.tcl

Untuk melakukan proses simulasi terlebih dahulu harus melakukan *setting* pada skrip NS agar mendapat hasil sesuai dengan yang dibutuhkan. Berikut ini penjabaran dari Gambar 3.7 proses pembuaan dari skrip *.tcl.

1. Inisiasi Simulasi

Simulasi NS dimulai dengan menuliskan *script* Tcl berikut ini. *Script* ini merupakan inisialisasi simulasi dan harus ada dalam setiap simulasi yang dibuat.

```
set ns [new Simulator]
#memanggil simulator object
set nf [open out.nam w]
$ns namtrace-all $nf
#open file handle untuk simulator nam trace data
```

```
set tr [open out.tr w]
$ns trace-all $tr
#open file handle untuk simulator file trace data
```

Skrip di atas akan membuat file `out.tr` yang akan digunakan untuk menyimpan data hasil simulasi dan file `out.nam` untuk menyimpan data hasil visualisasi. Deklarasi '`w`' pada bagian akhir dari perintah `open` adalah perintah tulis.

2. Pembuatan *Node* dan *Link*

Sebuah objek *node* pada NS didefinisikan dengan command `$ns node`. Perintah pembuatan *node* pada NS sebagai berikut :

```
Set node [$ns node]
```

Untuk menggunakan *node* n0 dilakukan dengan cara memanggil variabel `$n0`. Demikian pula *node-node* yang lain dapat dibuat dengan cara yang sama sesuai dengan kebutuhan dalam simulasi, seperti:

```
set node0 [$ns node]
set node1 [$ns node]
set node2 [$ns node]
set node3 [$ns node]
set node4 [$ns node]
set node5 [$ns node]
```

Setelah *node* terbuat, maka langkah selanjutnya adalah membuat *link* yang akan membuat hubungan antar *node*. Ada dua jenis *link* yang bisa digunakan pada NS, yaitu *simplex link* dan *duplex link*. Berikut ini contoh skrip pembuatan *link* antar *node* yang digunakan dalam program simulasi:

```
$ns duplex-link $node0 $node2 1Mb 15ms DropTail
$ns duplex-link $node1 $node2 1Mb 15ms DropTail

$ns simplex-link $node2 $node3 1Mb 100ms DropTail
$ns simplex-link $node3 $node2 1Mb 100ms DropTail

$ns duplex-link $node3 $node4 1Mb 15ms DropTail
$ns duplex-link $node3 $node5 1Mb 15ms DropTail
```

Perhatikan bahwa penentuan jarak *link* pada NS dilakukan dengan menggunakan parameter *delay propagasi link*. *Delay propagasi* adalah *delay* yang dipengaruhi oleh kecepatan gelombang elektromagnetik pada media *transmisi*. *Delay propagasi* bernilai tetap skitar 6 microsecond (μs) per kilometer. Misalnya jika ingin membuat simulasi *link* dengan jarak antar-node 1000km, parameter *delay propagasi* diatur sebesar 6 ms.

Karena *link* dianggap memiliki memori, maka kita perlu mendefinisikan kapasitas antrian dari *link* sebagai berikut:

```
$ns queue-limit $n2 $n3 20
```

3. Mengirimkan Data Trafik

Proses pengiriman data pada NS dilakukan dengan membuat trans*port* agent dan aplikasi di atasnya. *Transport agent* dibuat berpasangan, satu berfungsi sebagai sumber data dan pasangannya berfungsi sebagai tujuan.Contoh skrip pengiriman data trafik CBR dengan protokol UDP dapat dituliskan sebagai berikut.

```
set udp0 [new Agent/UDP]
$ns attach-agent $node0 $udp0
$udp0 set fid_ 1
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 160
$cbr0 attach-agent $udp0
set sink0 [new Agent/Null]
$ns attach-agent $node4 $sink0
$ns connect $udp0 $sink0
```

Dari skrip di atas bahwa protokol UDP dibangkitkan dengan trafik generator CBR yang mempunyai ukuran paket sebesar 160 (data VOIP). `Agent/UDP` merupakan agen pengiriman dan `Agent/Null` merupakan pasangan UDP sebagai tujuan trafik. Trafik CBR dikirim dari *node* 0 ke *node* 4 dengan pengaturan atribut link yang pertama ditandai dengan perintah `$udp0 set fid_1`

4. Mengatur Jadwal Eksekusi

Utuk menjalankan program simulasi pengiriman data trafik CBR, perintah yang dilakukan pada skrip yaitu:

```
$ns at 0.1 "$cbr0 start"
$ns at 10.0 "$cbr0 stop"
```

Skrip tersebut artinya data akan dikirimkan pada detik ke 0.1 dan akan berakhir atau dihentikan pada detik ke 10.0.

5. Menjalankan skrip *.tcl

```
$ns run
```

Skrip tersebut diatas merupakan skrip untuk melakukan *starting* program simulasi yang telah di buat. Perintah *run* diletakan pada bagian akhir dari program. Ini harus dilakukan karena NS mengeksekusi perintah baris per baris secara berurutan, sehingga jika objek simulasi diletakan setelah `$ns run`, objek simulasi yang telah dibuat tidak akan dieksekusi.

6. Terminasi ns

Untuk menghentikan program simulasi dilakukan dengan perintah pada skrip :

```
Proc finish {} {
#prosedure finish berguna untuk menyelesaikan simulasi,
#menutup file dan memulai nam (network animator)
  global ns nf tr
  $ns flush-trace
  close $nf
  close $tr
  exec nam out.nam &
  exit 0
  }
```

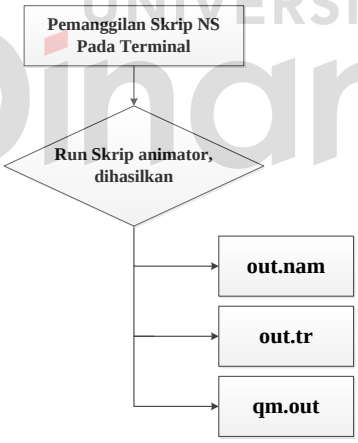
Perhatikan bahwa prosedur tersebut menggunakan variabel global `ns`, `nf` dan `tr`. Perintah `flush-trace` digunakan untuk menyimpan semua data

hasil simulasi ke dalam file `namfile` dan `tracefile`. Perintah `exit` akan mengakhiri aplikasi dan mengembalikan status dengan angka 0 kepada sistem. Perintah `exit 0` adalah perintah default untuk membersihkan memori dari sistem, nilai yang lain dapat gunakan untuk misalnya untuk memberikan status gagal.

```
$ns at 5.0 "finish"
# mengeksekusi prosedure finish pada saat detik ke 5.0
$ns run
#menjalankan simulasi
```

Prosedur `finish` harus dipanggil dengan indikasi waktu (dalam detik) terminasi dari program. Dalam skrip ini prosedure *finish* akan dilakukan pada detik ke 5.0 saat program berjalan.

3.3.4 Menjalankan Skrip *.tcl



Gambar 3.8 Proses Menjalankan Simulasi

Dalam menjalankan program simulasi *.tcl, dapat dilakukan dengan mengetikan perintah-perintah pada layar terminal dalam mengeksekusinya.

Setelah proses pembuatan program selesai dilakukan dan program disimpan dengan nama udp_topologi.tcl maka pemanggilan skrip simulasi program pada *terminal* dilakukan dengan perintah

```
ns udp_topologi.tcl
```

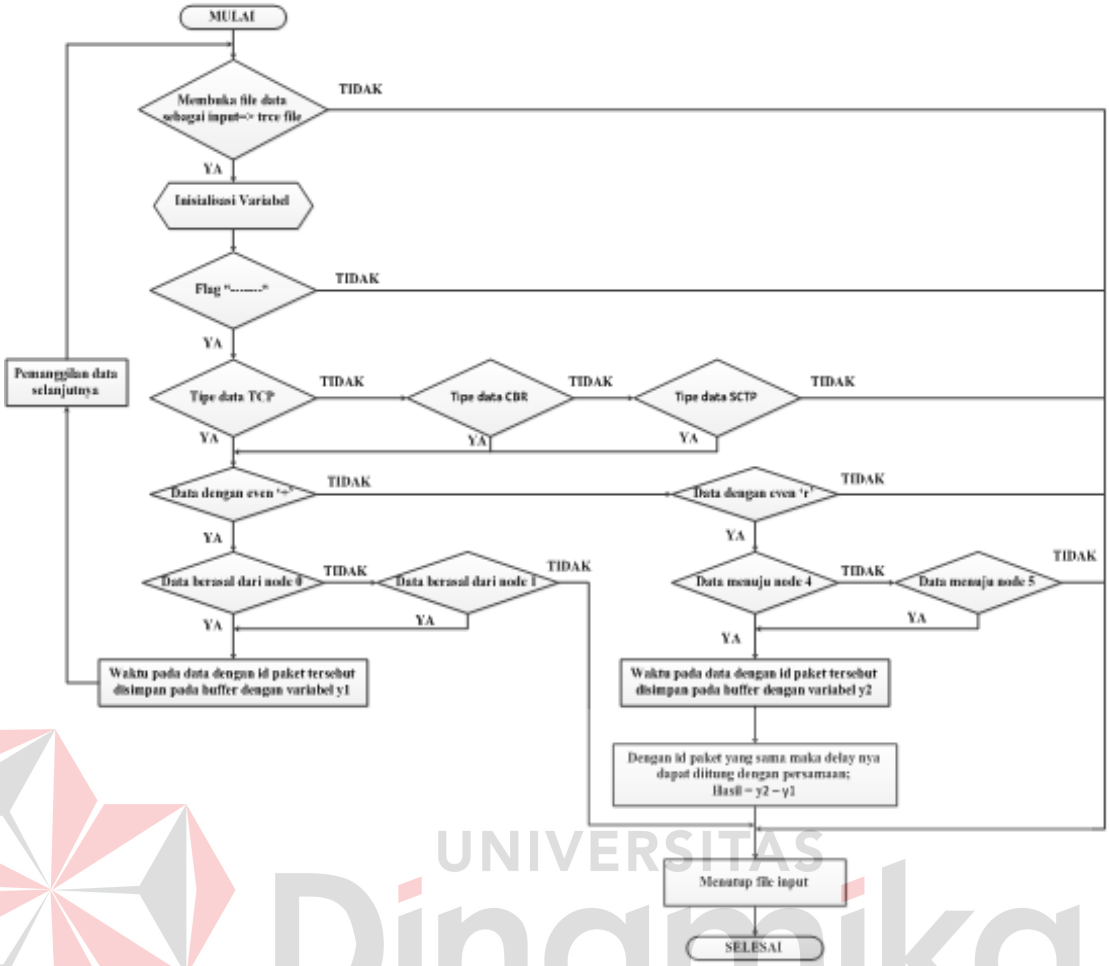
Saat perintah tersebut di jalankan akan muncul jendela nam yang merupakan hasil visualisasi topologi dari program yang telah dibuat.

Dari nam tersebut setelah dijalankan akan menghasilkan file out.tr dan out.nam. perintah tambahan pada skrip .tcl akan menghasilkan file qm out yang berguna sebagai informasi setiap kejadian dalam bentuk numerik statistika di antrian pada jalur *bottleneck.*

3.3.5 Proses *Parsing* Data

1. Perl untuk mendapatkan ukuran *delay*

Gambar 3.9 merupakan bentuk *flowchart* dari perl *delay*.



Gambar 3.9 *Flowchart Delay*

Dari *flowchart* tersebut langkah awal yaitu dengan membuka *file trace* yang dihasilkan pada simulasi program dengan exention *.tcl sebagai data masukan untuk di proses. Jika data mempunyai *flag* "-------" maka data akan di teruskan dengan penentuan protokol yang digunakan. Langkah selanjutnya yaitu dengan mengindikasi adanya *even*:

a. Jika *even* "+" pada data yang masuk ke tahap ini maka *time* dari data yang masuk yang berasal dari node0 atau node1, disimpan menurut *id packet* dan asal *node* nya dan disimbolkan dengan variabel y1.

b. Jika *even* "r" pada data yang masuk ke tahap ini maka time dari data yang masuk yang menuju *node* 4 atau *node* 5, disimpan menurut *id packet* dan asal *node* nya dan disimbolkan dengan variabel y2. Langkah selanjutnya yaitu dengan mengurangkan antara y2 – y1.

File perl *delay* ini dapat disimpan dengan *extention* `delay.pl`. untuk menghasilkan file *delay* dari data trace file, pemanggilan skrip perl *delay* ini dilakukan pada terminal dengan mengetikan peintah berikut:

```
Perl delay.pl out.tr > delay
```

Dari data *delay* yang dihasilkan maka dapat dilakukan perhitungan *latency* dan *jitter* dengan menggunakan *microsoft excel* 2007. *Latency* diperoleh dengan menentukan rata-rata dari *delay*. Didapatkan dari rumus 1 yang terdapat pada bab 2 mengenai perhitungan delay pada masing-masing data dibagi dengan jumlah data delay yang dihasilkan.

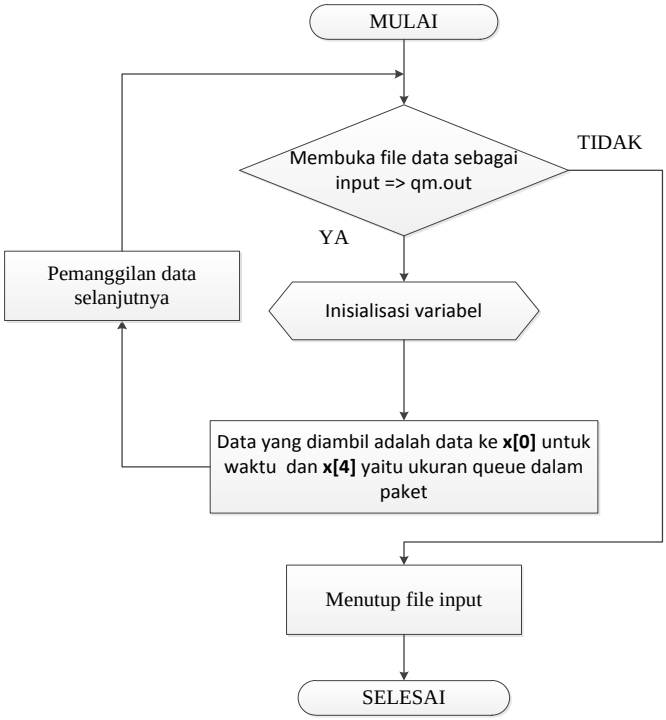
$$Latency = \frac{Delay}{N} \text{ detik} \quad \text{..... 3.1}$$

Dimana : N = jumlah data yang dihasilkan *delay*

Sedangkan *jitter* diperoleh dari variansi *delay* yang telah dihasilkan, dapat dihitung dengan menggunakan rumus 2.2 yang telah di bahas pada bab 2

2. Perl untuk mendapatkan ukuran *queue*

Gambar 3.10 merupakan bentuk *flowchart* dari *queue*.



Gambar 3.10 *Flowchart Queue*

Perl *queue* ini dibutuhkan untuk mendapatkan ukuran antrian dalam satuan paket yang didapatkan dari hasil simulasi per satuan waktu. Langkah awal dilakukan yaitu dengan membuka file qm.out yang didapat dari hasil simulasi yang digunakan sebagai *input data*. *Parsing* dilakukan dengan mengambil data pada qm.out pada urutan ke 0 menunjukan waktu saat kejadian dan urutan ke 4 yang merupakan ukuran dari *queue* dalam satuan paket.

File perl *queue* disimpan dengan *extention* `queue.pl`. untuk menghasilkan file *queue* dari data `qm.out`, pemanggilan skrip perl queue ini dilakukan pada *terminal* dengan mengetikan peintah berikut:

```
Perl queue.pl qm.out >queue
```

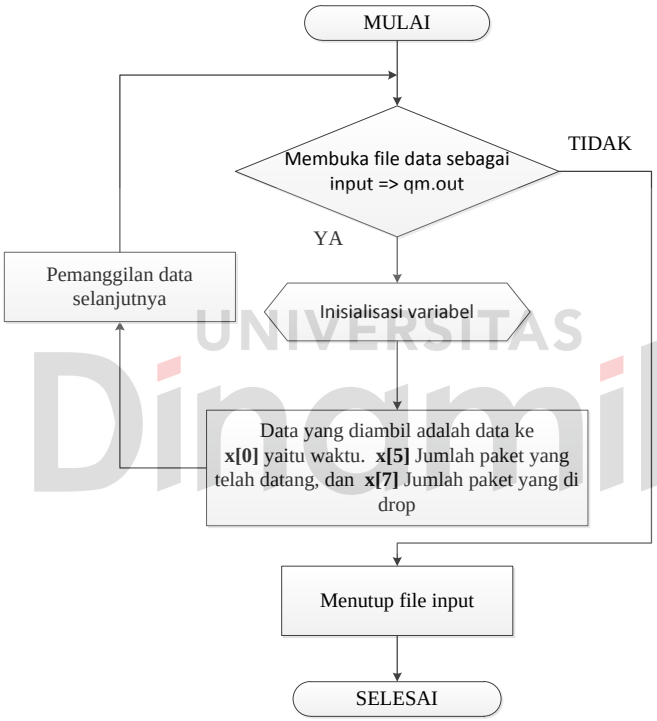
Data yang digunakan sebagai inputan dalam perintah perl *queue* ini adalah data yang dihasilkan oleh qm.out saat menjalankan simulasi *.tcl. Penambahan skrip `qm.out` pada *.tcl adalah sebagai berikut:

```
set qmon [$ns monitor-queue $node2 $node3 [open qm.out w]
0.1];
[$ns link $node2 $node3] queue-sample-timeout;
```

Pada skrip tersebut *queue monitoring* diambil pada *bottleneck link* yaitu *node* 2 dan *node* 3 pada simulasi pertama, dan *node* 5 dan *node* 6 pada simulasi dengan 5 kanal.

3. Perl untuk perhitungan *packet loss*

Gambar 3.11 merupakan bentuk *flowchart* dari *packet loss*.



Gambar 3.11 *Flowchart Packet Loss*

Perl *packet loss* juga diambil dari data *queue monitoring* karna pada TCP dan SCTP yang mempunyai karakteristik *retransmisi data loss* akan lebih efektif saat pengambilan informasi data dari jalur *bottleneck* atau jalur antrian. Langkah awal dilakukan yaitu dengan membuka file qm.out yang didapat dari hasil simulasi yang digunakan sebagai *input data*. *Parsing* dilakukan dengan mengambil data pada qm.out pada urutan ke 0 menunjukan waktu saat kejadian

dan urutan ke 5 yang merupakan jumlah paket yang tiba di antrian dan urutan ke 7 yang menunjukan jumlah paket yang hilang di jalur antrian ini.

File perl *packet loss* ini dapat disimpan dengan *extention* `loss.pl`. untuk menghasilkan file *packet loss* dari `qm.out`, perintah untuk memanggil perl *packet loss* yaitu dengan skrip :

```
perl paketloss.pl qm.out > loss
```

3.3.6 *Plotting Data*

Setelah dilakukan parsing data menggunakan perl dan didapatkan file hasil. Maka langkah selanjutnya adalah menyajikan data yang ada. Proses penyajian data ini dimaksudkan untuk memberi kemudahan dalam proses analisis yang di buat. Dua cara yang dilakukan dalam menampilkan data statistika yaitu dalam bentuk tabel dan grafik.

Tampilan data dalam bentuk tabel dimaksutkan untuk mengetahui perbandingan dari hasil yang didapat pada masing-masing protokol dalam bentuk data statistika. Sedangkan dalam bentuk grafik akan menampilkan hasil data secara grafis. Grafik ini dibuat dengan dua cara, yaitu menggambarkan keadaan data pada masing-masing skenario yang dibuat dengan jenis protokol yang dibandingkan dan yang kedua menggabungkan ketiga protokol yang dibandingkan per satu skenario yang di buat.

BAB IV

HASIL DAN PEMBAHASAN

4.1 Kebutuhan *Hardware* dan *Software*

Untuk dapat menjalankan sistem simulasi yang dibuat, diperlukan perangkat keras dan perangkat lunak dengan spesifikasi tertentu.

4.1.1 Kebutuhan Perangkat Keras (*Hardware)*

Penulis membutuhkan perangkat keras pendukung komputer/laptop dengan spesifikasi sebagai berikut:

1. Prosesor : *Intel Pentium Dual Core*
2. Tipe : P6100 2GHZ
3. *Chipset* : Intel HM55
4. Memori : 1GB SDRAM PC-8500
5. Sistem Operasi: Linux Ubuntu 10.04 dan Windows XP

4.1.2 Kebutuhan Perangkat Lunak (*Software)*

Perangkat lunak yang digunakan dalam penelitian ini antara lain:

a. *Network Simulator 2* (NS-2) versi 2.35. Aplikasi ini merupakan aplikasi utama yang digunakan untuk menjalankan proses simulasi.
b. *Perl*. Aplikasi ini digunakan untuk mengolah file *.tr yang merupakan data output dari simulasi dengan menggunakan NS-2.
c. *Microsoft Excel 2007*. Aplikasi ini digunakan untuk mengolah data dan membuat grafik dari data hasil simulasi.

4.2 Menjalankan Simulasi

Dalam Menjalankan simulasi ini akan digunakan beberapa parameter yang di set untuk menadapatkan hasil yang berbeda sesuai dengan kondisi sebenarnya. Berikut Tabel 4.1 merupakan parameter yang digunakan dalam menjalanan program.

Tabel 4.1 Tabel Parameter Simulasi

Percobaan	Jenis Data	*Access bandwidth/ link delay*	Lebar Queue	*Delay propagation*	*Bandwidth-Bottleneck Link*	Model Queue
1	VOIP	1Mb / 15ms	20	100 ms	128 Kb	Drop Tail
2		1Mb / 15ms	20		384 Kb	Drop Tail
3		1Mb / 15ms	20		512 Kb	Drop Tail
4		1Mb / 15ms	20	300 ms	128 Kb	Drop Tail
5		1Mb / 15ms	20		384 Kb	Drop Tail
6		1Mb / 15ms	20		512 Kb	Drop Tail
7	IPTV	1Mb / 15ms	20	100 ms	512 Kb	Drop Tail
8		1Mb / 15ms	20		1 Mb	Drop Tail
9		1Mb / 15ms	20		3 Mb	Drop Tail
10		1Mb / 15ms	20	300 ms	512 Kb	Drop Tail
11		1Mb / 15ms	20		1 Mb	Drop Tail
12		1Mb / 15ms	20		3 Mb	Drop Tail

4.2.1 *Transmission Control Protokol* (TCP)

Dari Tabel 4.1 dilakukan dalam 2 kali percobaan, yaitu menggunakan 1 kanal, dan 5 kanal. Jumlah kanal digunakan untuk menunjukan kondisi jaringan saat data di jalankan tanpa adanya kongesti dengan menggunakan 1 kanal, dan dengan 5 kanal akan menunjukan kinerja dari jaringan saat jalur *bottleneck* harus berbagi data dengan yang lain. Percobaan dilakukan dengan menjalankan skrip *.tcl pada NS, yang hasilnya adalah *nam file*, *trace file* dan **.out*, kemudian

dengan menambahkan beberapa skrip perl untuk mendapatkan data yang diinginkan seperti *delay, paket loss* dan *queue*.

Di bawah ini merupakan skrip percobaan 1 yaitu tcp_topologi.tcl dengan besar variasi ukuran *delay propagation/bandwidth*=100ms/128Kb menggunakan layanan data VOIP dan IPTV. Kedua data ini pada ns2 direpresentasikan oleh trafik cbr dengan ukuran paket yang berbeda. Ukuran paket yang digunakan untuk data VOIP sebesar 160 *byte*, sedangkan untuk data IPTV yaitu1300 *byte*. Percobaan menggunakan 1 kanal dan 5 kanal. Untuk percobaan menggunakan 1 kanal, data VOIP direpresentasi oleh variabel cbr0 sehingga ketika akan mensimulasikannya waktu untuk proses simulasi cbr0 diaktifkan dan waktu proses simulasi untuk cbr1 yang merupakan variabel representasi dari IPTV di nonaktifkan dengan memberikan komentar pada awal baris dan sebaliknya. Untuk percobaan menggunakan 5 kanal, data VOIP dan IPTV direpresentasikan dengan variabel cbr0, cbr1, cbr2, cbr3, dan cbr4 dengan topologi yang terpisah, namun yang digunakan dalam proses pengambilan data hanya variabel cbr0, sedangkan variabel yang lain merupakan trafik yang digunakan untuk menghasilkan kongesti.

Hasil grafik sesuai dengan percobaan yang dilakukan yaitu sebagai berikut.

1. Skrip percobaan dengan 1 kanal :

```
set ns [new Simulator]

set tr [open tcp_topologi.tr w]
set f [open tcp_topologi.nam w]
$ns trace-all $tr
$ns namtrace-all $f

proc finish {} {
 global ns f tr
 $ns flush-trace
```

```
 close $f
 close $tr
 exec nam tcp_topologi.nam &
 exit 0
 }
set node0 [$ns node]
set node1 [$ns node]
set node2 [$ns node]
set node3 [$ns node]
set node4 [$ns node]
set node5 [$ns node]

$ns duplex-link $node0 $node2 1Mb 15ms DropTail
$ns duplex-link $node1 $node2 1Mb 15ms DropTail
$ns simplex-link $node2 $node3 128Kb 100ms DropTail
$ns simplex-link $node3 $node2 128Kb 100ms DropTail
$ns duplex-link $node3 $node4 1Mb 15ms DropTail
$ns duplex-link $node3 $node5 1Mb 15ms DropTail

$ns duplex-link-op $node0 $node2 orient right-down
$ns duplex-link-op $node1 $node2 orient right-up
$ns simplex-link-op $node2 $node3 orient right
$ns simplex-link-op $node3 $node2 orient left
$ns duplex-link-op $node3 $node4 orient right-up
$ns duplex-link-op $node3 $node5 orient right-down

$ns queue-limit $node2 $node3 20

$ns color 1 red
$ns color 2 blue

#pengiriman trafik data VOIP
set tcp0 [new Agent/TCP]
$ns attach-agent $node0 $tcp0
$tcp0 set fid_ 1
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 160
$cbr0 attach-agent $tcp0
set sink0 [new Agent/TCPSink]
$ns attach-agent $node4 $sink0
$ns connect $tcp0 $sink0

#pengiriman trafik data IPTV
set tcp1 [new Agent/TCP]
$ns attach-agent $node1 $tcp1
$tcp1 set fid_ 2
set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 1300
$cbr1 attach-agent $tcp1
set sink1 [new Agent/TCPSink]
$ns attach-agent $node5 $sink1
$ns connect $tcp1 $sink1

set qmon [$ns monitor-queue $node2 $node3 [open
tcp_topologi.out w] 0.1];
[$ns link $node2 $node3] queue-sample-timeout;

$ns at 0.1 "$cbr0 start"
$ns at 10.0 "$cbr0 stop"
```

```
#diaktifkan bila mensimulasikan data IPTV
$ns at 0.1 "$cb\
       r1 start"
$ns at 10.0 "$cbr1 stop"

$ns at 10.5 "finish"
$ns run
```

2. Skrip percobaan dengan 5 kanal :

```
set ns [new Simulator]

set tr [open tcp_topologi.tr w]
set f [open tcp_topologi.nam w]
$ns trace-all $tr
$ns namtrace-all $f

proc finish {} {
 global ns f tr
 $ns flush-trace
 close $f
 close $tr
 exec nam tcp_topologi.nam &
 exit 0
 }

set node0 [$ns node]
set node1 [$ns node]
set node2 [$ns node]
set node3 [$ns node]
set node4 [$ns node]
set node5 [$ns node]
set node6 [$ns node]
set node7 [$ns node]
set node8 [$ns node]
set node9 [$ns node]
set node10 [$ns node]
set node11 [$ns node]

$ns duplex-link $node0 $node5 1Mb 15ms DropTail
$ns duplex-link $node1 $node5 1Mb 15ms DropTail
$ns duplex-link $node2 $node5 1Mb 15ms DropTail
$ns duplex-link $node3 $node5 1Mb 15ms DropTail
$ns duplex-link $node4 $node5 1Mb 15ms DropTail
$ns simplex-link $node5 $node6 128Kb 100ms DropTail
$ns simplex-link $node6 $node5 128Kb 100ms DropTail
$ns duplex-link $node6 $node7 1Mb 15ms DropTail
$ns duplex-link $node6 $node8 1Mb 15ms DropTail
$ns duplex-link $node6 $node9 1Mb 15ms DropTail
$ns duplex-link $node6 $node10 1Mb 15ms DropTail
$ns duplex-link $node6 $node11 1Mb 15ms DropTail

$ns duplex-link-op $node0 $node5 orient down
$ns duplex-link-op $node1 $node5 orient right-down
$ns duplex-link-op $node2 $node5 orient right
$ns duplex-link-op $node3 $node5 orient right-up
$ns duplex-link-op $node4 $node5 orient up
$ns simplex-link-op $node5 $node6 orient right
```

```
$ns simplex-link-op $node6 $node5 orient left
$ns duplex-link-op $node6 $node7 orient up
$ns duplex-link-op $node6 $node8 orient right-up
$ns duplex-link-op $node6 $node9 orient right
$ns duplex-link-op $node6 $node10 orient right-down
$ns duplex-link-op $node6 $node11 orient down

$ns queue-limit $node5 $node6 20

$ns color 1 red
$ns color 2 blue
$ns color 3 green
$ns color 4 pink
$ns color 5 black

set tcp0 [new Agent/TCP]
$ns attach-agent $node0 $tcp0
$tcp0 set fid_ 1
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 160
$cbr0 attach-agent $tcp0
set sink0 [new Agent/TCPSink]
$ns attach-agent $node7 $sink0
$ns connect $tcp0 $sink0
set tcp1 [new Agent/TCP]
$ns attach-agent $node1 $tcp1
$tcp1 set fid_ 2
set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 160
$cbr1 attach-agent $tcp1
set sink1 [new Agent/TCPSink]
$ns attach-agent $node8 $sink1
$ns connect $tcp1 $sink1

set tcp2 [new Agent/TCP]
$ns attach-agent $node2 $tcp2
$tcp2 set fid_ 3
set cbr2 [new Application/Traffic/CBR]
$cbr2 set packetSize_ 160
$cbr2 attach-agent $tcp2
set sink2 [new Agent/TCPSink]
$ns attach-agent $node9 $sink2
$ns connect $tcp2 $sink2

set tcp3 [new Agent/TCP]
$ns attach-agent $node3 $tcp3
$tcp3 set fid_ 4
set cbr3 [new Application/Traffic/CBR]
$cbr3 set packetSize_ 160
$cbr3 attach-agent $tcp3
set sink3 [new Agent/TCPSink]
$ns attach-agent $node10 $sink3
$ns connect $tcp3 $sink3

set tcp4 [new Agent/TCP]
$ns attach-agent $node4 $tcp4
$tcp4 set fid_ 5
set cbr4 [new Application/Traffic/CBR]
$cbr4 set packetSize_ 160
```

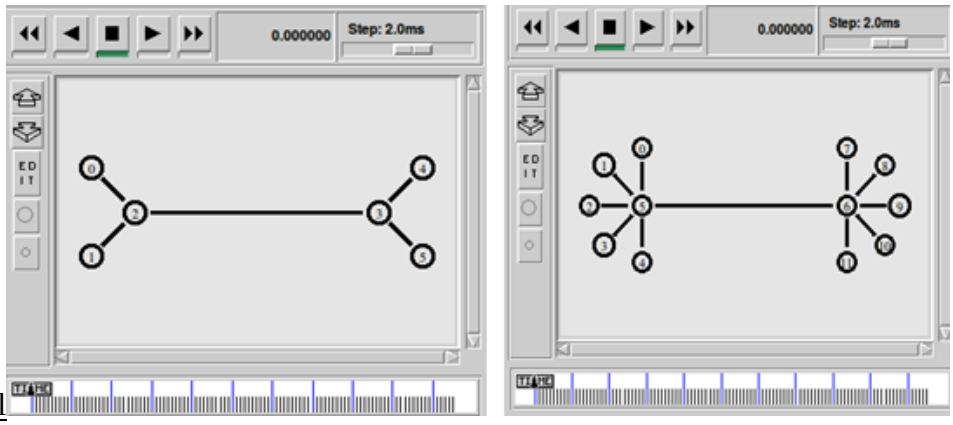
```
$cbr4 attach-agent $tcp4
set sink4 [new Agent/TCPSink]
$ns attach-agent $node11 $sink4
$ns connect $tcp4 $sink4

set qmon [$ns monitor-queue $node5 $node6 [open
tcp_topologi.out w] 0.1];
[$ns link $node5 $node6] queue-sample-timeout;

$ns at 0.1 "$cbr0 start"
$ns at 10.0 "$cbr0 stop"
$ns at 0.1 "$cbr1 start"
$ns at 10.0 "$cbr1 stop"
$ns at 0.1 "$cbr2 start"
$ns at 10.0 "$cbr2 stop"
$ns at 0.1 "$cbr3 start"
$ns at 10.0 "$cbr3 stop"
$ns at 0.1 "$cbr4 start"
$ns at 10.0 "$cbr4 stop"

$ns at 10.5 "finish"
$ns run
```

Teks yang tercetak tebal menandakan parameter yang harus diubah-ubah. Besar perubahan *bandwidth* dan *delay propagation* bergantung dari nilai yang telah ditentukan dalam tabel. Sedangkan besar *packet size* bergantung jenis data yang digunakan, 160 *byte* untuk VOIP dan 1300 *byte* untuk IPTV. Simulasi dijalankan dengan mengetikan perintah berikut pada jendela terminal.

```
rinda@rinda:~$ ns tcp_topologi.tcl
```

Setelah program simulasi dijalankan maka akan didapatkan file nam seperti pada Gambar 4.1.

Dengan 1 kanal Dengan 5 Kanal



Gambar 4.1 Tampilan Nam

Dari hasil nam diatas dapat dilihat topologi dari skrip yang telah di buat. Dengan topologi menggunakan 1 kanal, *node-node* yang dibutuhkan oleh sistem dalam melakukan simulasi yaitu sebanyak 6 *node* dengan bentuk topologi *dumb-bell* standar seperti pada Gambar 4.1. aliran dari *node* 0 akan membawa data VOIP sedang aliran data pada *node* 1 akan membawa data IPTV. Saat menjalankan program dengan data VOIP, maka aliran data IPTV dihentikan, dan sebaliknya.

Pada hasil simulasi menggunakan 5 kanal, *node-node* yang dibutuhkan dalam melakukan proses simulasi menggunakan *node* berjumlah 12 *node*. *Node* 0,1,2,3,4 mengalirkan data cbr dengan besarnya ukuran paket bergantung dari jenis data yang di gunakan.

Simulasi berjalan saat tombol *play* di jalankan dan menghasilkan out.tr yang merekam seluruh kejadian dari pengiriman node sumber ke node tujuan, selain itu simulasi juga menghasilkan *.out yang merupakan informasi kejadian pada jalur *bottleneck*. Selanjutnya dilakukan *parsing file* dengan menggunakan skrip perl *delay*, *queue* dan *packet loss* berikut.

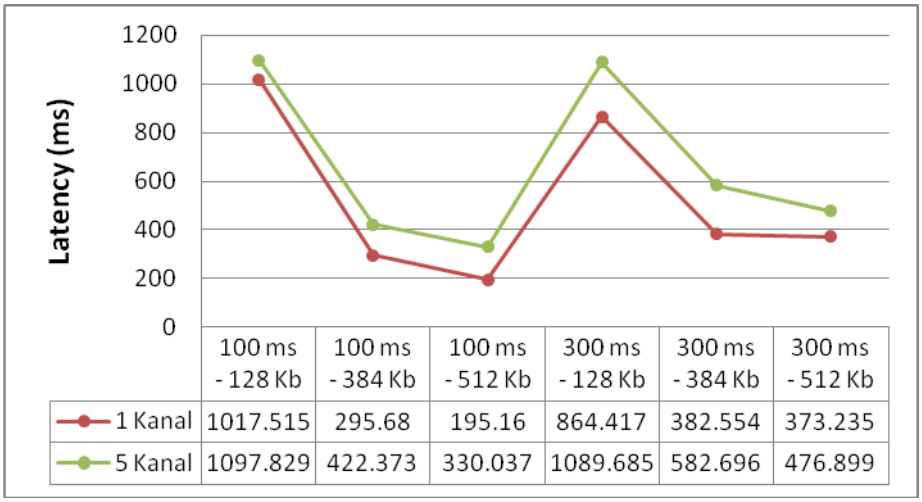
3. Skrip perl untuk *delay* sesuai dengan *flowchart* pada Gambar 3.9.

```
# type: perl delay.pl <trace file> > output file
$infile=$ARGV[0];
open (DATA,"<$infile") || die "Can't open $infile $!";
while (<DATA>)
{
 @x = split(' ');
 @TCP0;
 @TCP4;
 @hsl;
 if ($x[6] eq '-------')
 {
     if ($x[4] eq 'TCP')
     {
           if ($x[0] eq '+')
           {
                 if ($x[2] eq '0')
                 {
                    $TCP0[$x[11]]=$x[1];
                 }
           }
           if ($x[0] eq 'r')
           {
                 if ($x[3] eq '4')
                 {
                    $TCP4[$x[11]]=$x[1];
                $hsl[$x[11]]=$TCP4[$x[11]]-$TCP0[$x[11]];
                    print STDOUT "$x[10] $hsl[$x[11]] \n";
                 }
           }
     }
  }
}
close DATA;
exit(0);
```

Dari skrip perl tersebut dapat dipanggil dengan perintah berikut pada terminal:

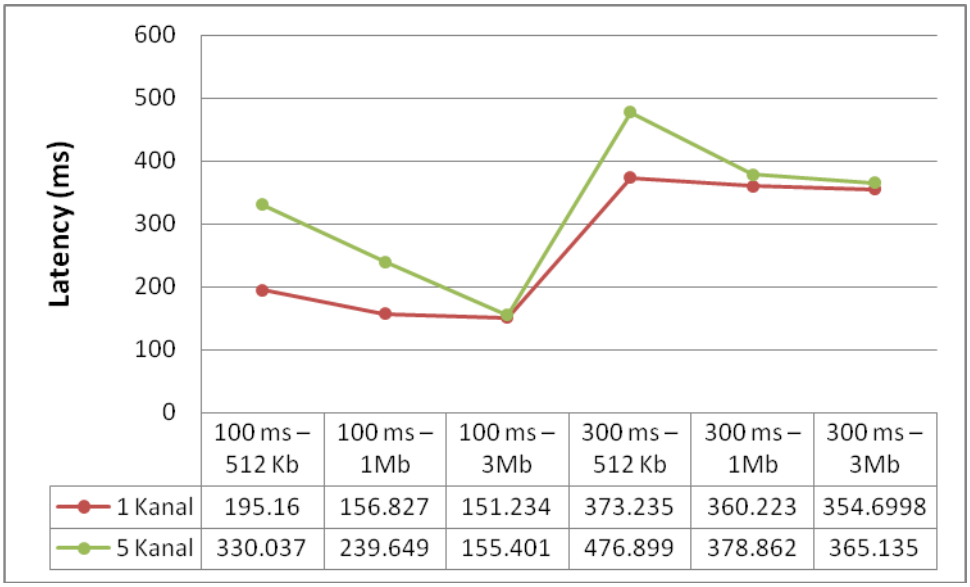
```
rinda@rinda:~$ perl delay.pl tcp_topologi.tr > delay
```

Dari data *delay* yang diperoleh dengan melakukan *parsing* menggunakan perl, data tersebut diolah kembali untuk menghasilkan nilai *latency* dan *jitter* dengan menggunakan rumus yang telah di jelaskan pada bab sebelumnya menggunakan *microsoft excel* 2007. Data ini kemudian ditampilkan menggunakan grafik untuk lebih memperjelas dalam melakukan analisis.



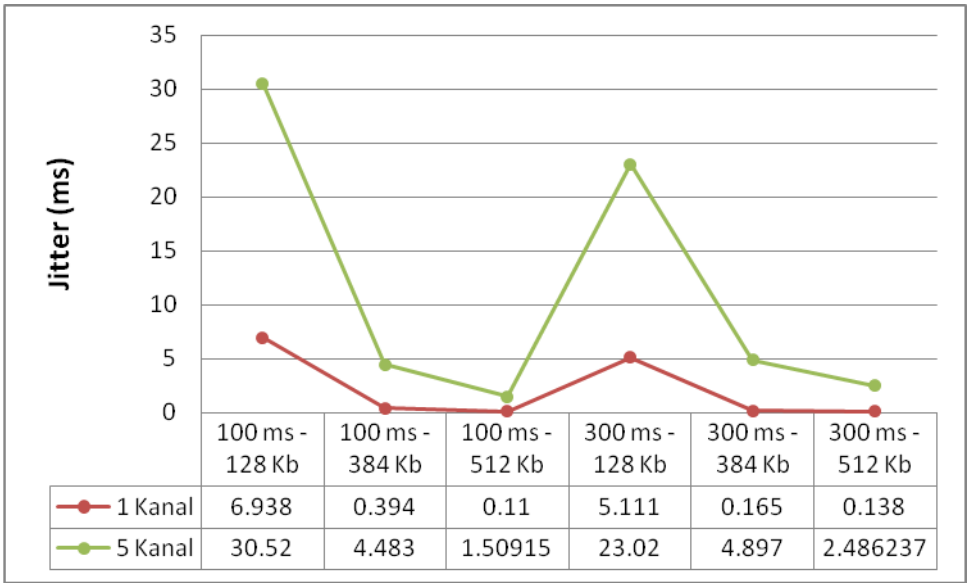
Gambar 4.2 Besar *Latency* Data VOIP

Gambar 4.2 menunjukan besar *latency* dari protokol TCP dengan menggunakan layanan dari data VOIP. Dari grafik tersebut dapat dilihat bahwa dengan menggunakan 1 kanal dan 5 kanal, *latency* TCP dengan *bandwidth* 128 Kb dan *delay propagation* 100 ms merupakan *latency* yang paling besar nilainya dibanding dengan yang lain. Sedangkan *latency* terendah terjadi saat penggunaan *bandwidth* 512 Kb dan *delay propagation* 100 ms untuk kedua kondisi kanal. Ini disebabkan karena dengan *bandwidth* yang tinggi dan *delay propagation* rendah akan mengurangi kongesti dan berakibat berkurangnya pula nilai *delay* yang dihasilkan.



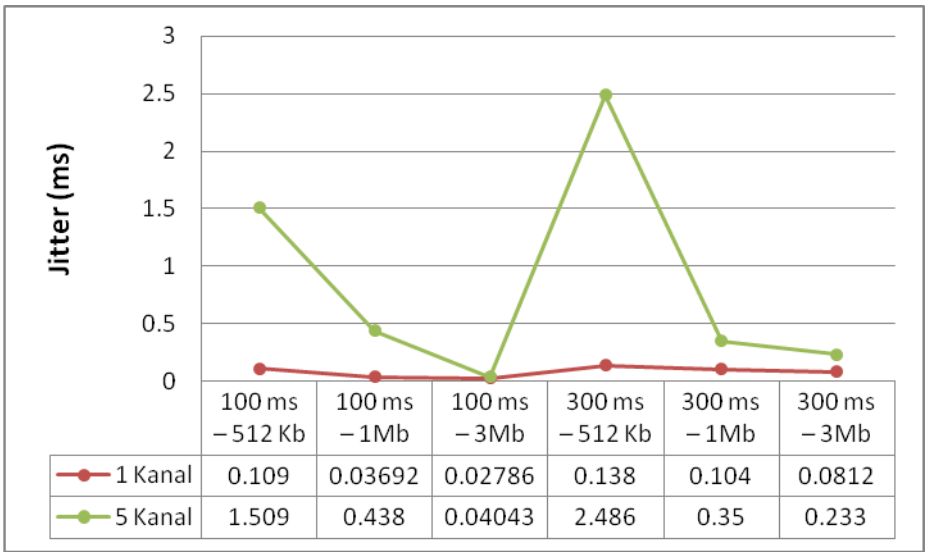
Gambar 4.3 Besar *Latency* Data IPTV

Gambar 4.3 menunjukan besar *latency* dari protokol TCP dengan menggunakan layanan dari data IPTV. Dari grafik tersebut dapat dilihat bahwa dengan menggunakan 1 kanal dan 5 kanal latency TCP dengan bandwidth 512Kb dan *delay propagation* 300 ms merupakan *latency* yang paling besar nilainya dibanding dengan *latency* pada *bandwidth* dan *delay propagation* yang lain. Sedangkan *latency* terendah terjadi saat penggunaan *bandwidth* 3 Mb dan *delay propagation* 100 ms. Ini disebabkan karena dengan *bandwidth* yang tinggi dan *delay propagation* rendah akan mengurangi kongesti dan berakibat berkurangnya pula nilai *delay* yang dihasilkan.



Gambar 4.4 Besar *Jitter* Data VOIP

Pada Gambar 4.4 didapatkan bahwa besar nilai variasi *delay* atau *jitter* dari TCP yang menggunakan layanan data VOIP dengan 1 kanal maupun 5 kanal terbesar berada pada *bandwidth* 128 Kb dan *delay propagation* 100 ms. Sedangkan *jitter* terendah terjadi saat penggunaan *bandwidth* 512 Kb dan *delay propagation* 100 ms.



Gambar 4.5 Besar *Jitter* Data IPTV

Pada Gambar 4.5 didapatkan bahwa besar nilai variasi *delay* atau *jitter* dari TCP yang menggunakan layanan data IPTV dengan 1 kanal maupun 5 kanal terbesar berada pada *bandwidth* 512 Kb dan *delay propagation* 300 ms. Sedangkan *jitter* terendah terjadi saat penggunaan *bandwidth* 3 Mb dan *delay propagation* 100 ms.

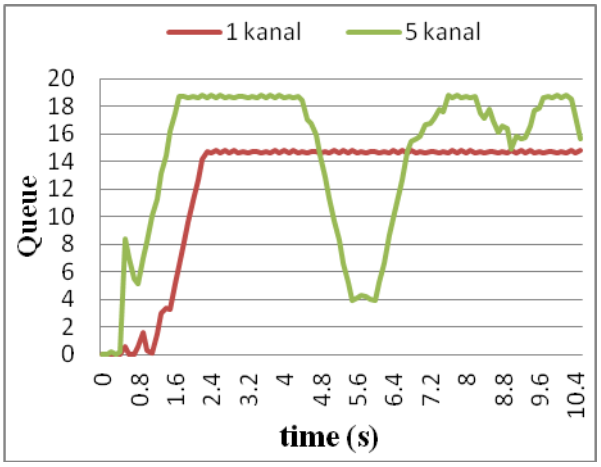
4. Skrip perl untuk *queue* sesuai dengan *flowchart* pada Gambar 3.10:

```
# perl queue.pl  <qm.out>  > queue
$infile=$ARGV[0];
open (DATA,"<$infile") || die "Can't open $infile $!";
while (<DATA>)
{
 @x = split(' ');
 print STDOUT "$x[0]  $x[4]\n";

}
close DATA;
exit(0);
```

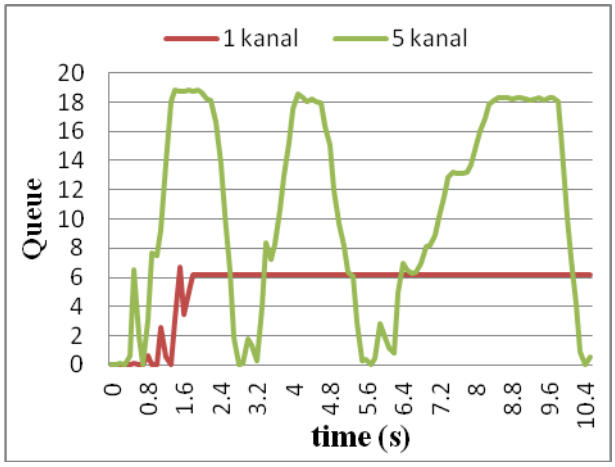
Dari skrip perl tersebut dapat dipanggil dengan perintah berikut pada terminal:

```
rinda@rinda:~$ perl queue.pl tcp_topologi.tr > queue
```

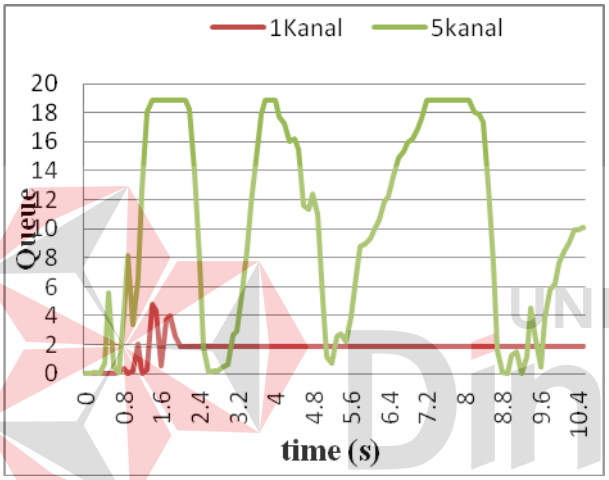
Data hasil *queue* yang telah di dapatkan kemudian digambarkan dalam bentuk grafik menggunakan *microsoft excel* 2007. Grafik ini di buat untuk lebih mempermudah dalam melakukan analisis.



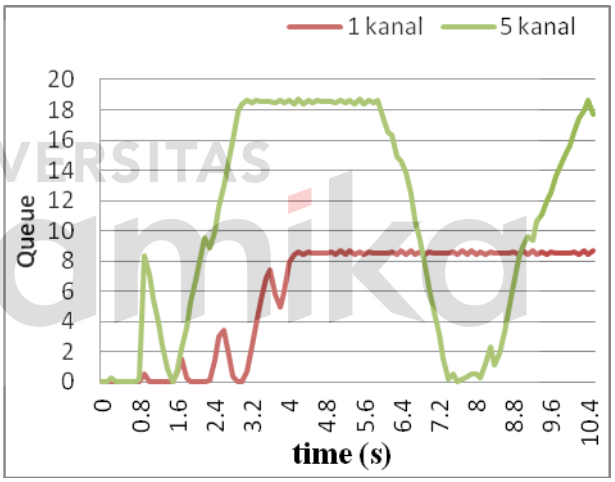
a. *Queue* Data VOIP 100ms-128Kb



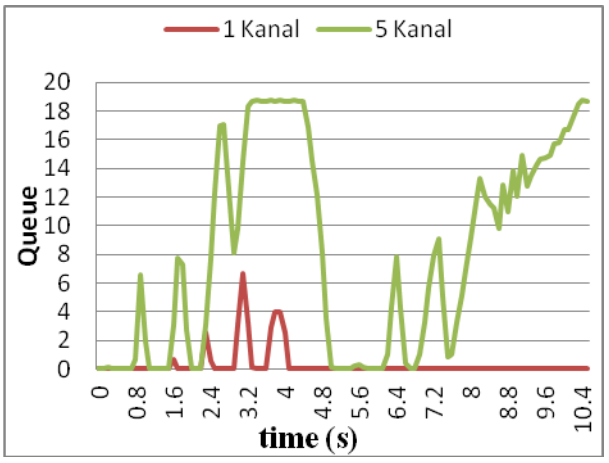
b. *Queue* Data VOIP 100ms-384Kb

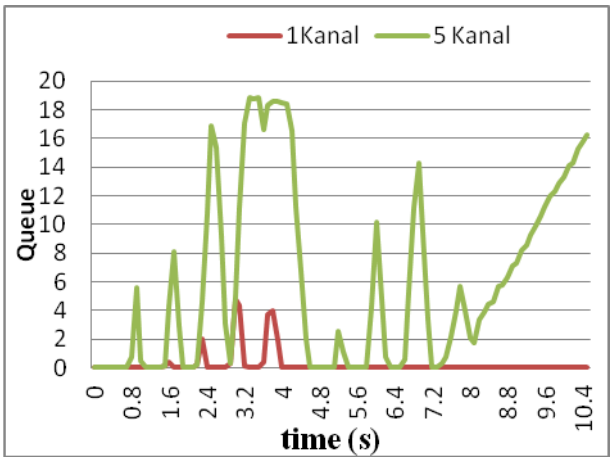


c. *Queue* Data VOIP 100ms-512Kb



d. *Queue* Data VOIP 300ms-128Kb

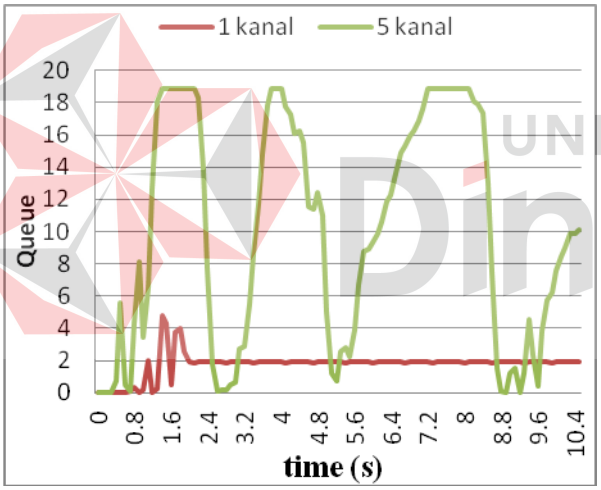


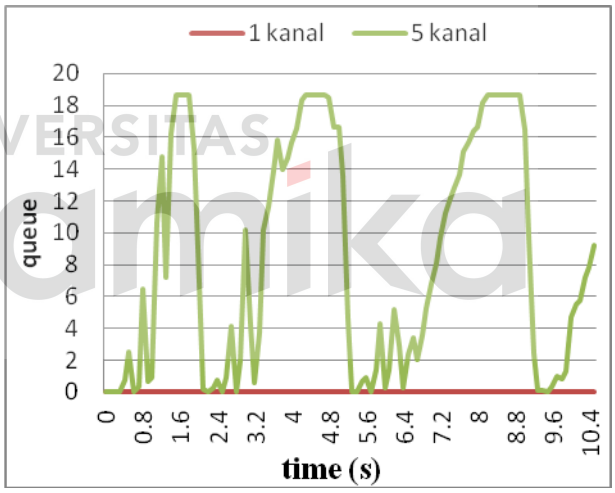


e. *Queue* Data VOIP 300ms-384Kb f. *Queue* Data VOIP 300ms-512Kb

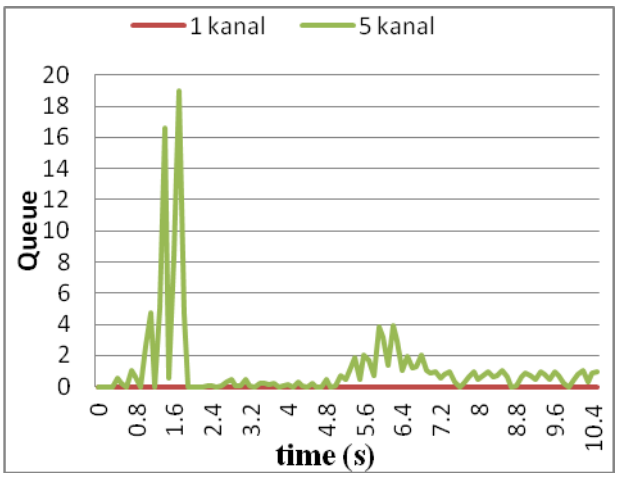
Gambar 4.6 *Queue* Data VOIP

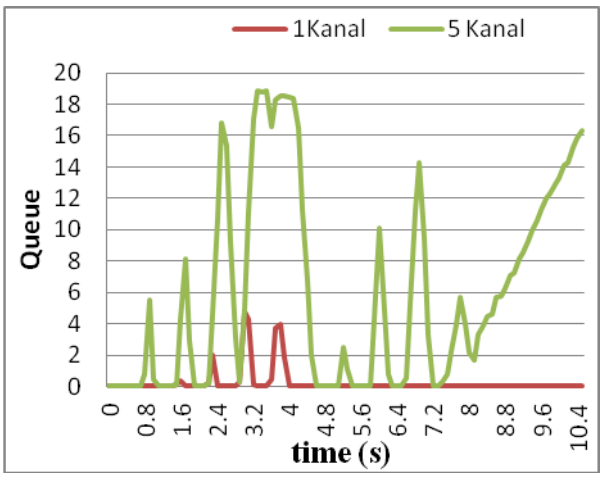
Pada Gambar 4.6 menunjukan nilai *queue* yang di dapat terhadap waktu (*time)* dari protokol TCP dengan data VOIP. Grafik yang naik menunjukan bahwa kapasitas antrian yang terpakai oleh sistem pada jalur *bottleneck* mengalami kenaikan untuk rentang waktu tersebut. *Queue* maksimum yang telah di seting adalah sebesar 20. Dengan menggunakan 5 Kanal, *queue* pada TCP menjadi lebih besar karena dengan jumlah kanal yang besar maka trafik data yang masuk ke antrian juga semakin besar.



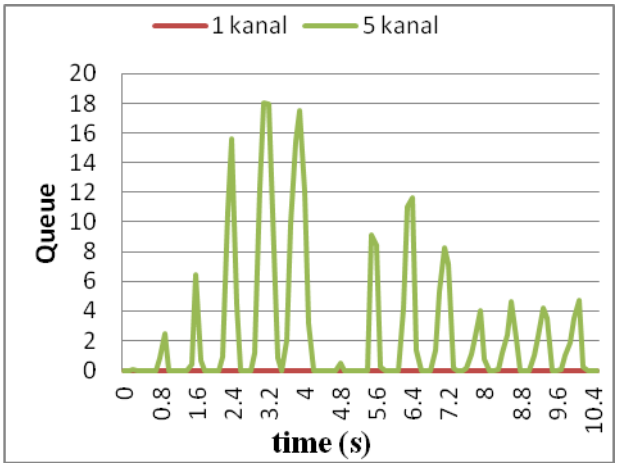


a. Queue Data IPTV 100ms-512Kb b. *Queue* Data IPTV 100ms-1 Mb

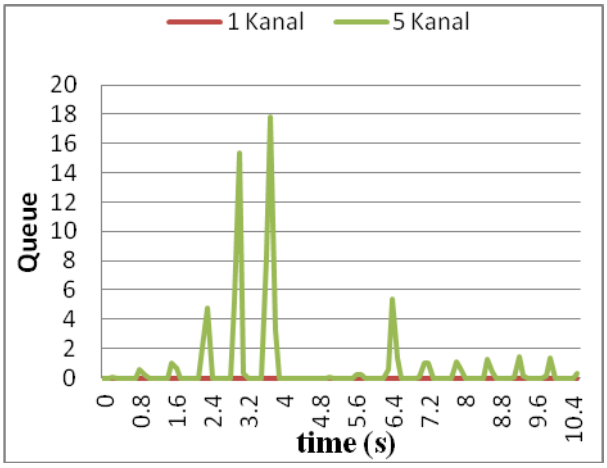




c. *Queue* Data IPTV 100ms-3 Mb



d. *Queue* Data IPTV 300ms-512 Kb



e. *Queue* Data IPTV 300ms-1 Mb

f. *Queue* Data IPTV 300ms-3 Mb

Gambar 4.7 *Queue* Data IPTV

Pada Gambar 4.7 menunjukan nilai *queue* yang di dapat terhadap waktu (*time*) dari protokol TCP dengan data IPTV. Grafik yang naik menunjukan bahwa kapasitas antrian yang terpakai oleh sistem pada jalur *bottleneck* mengalami kenaikan untuk rentang waktu tersebut. *Queue* maksimum yang telah di seting adalah sebesar 20, namun pada pemakaian *queue* pada TCP penggunaan kapasitas *queue* tidak mencapai batas maksimal.

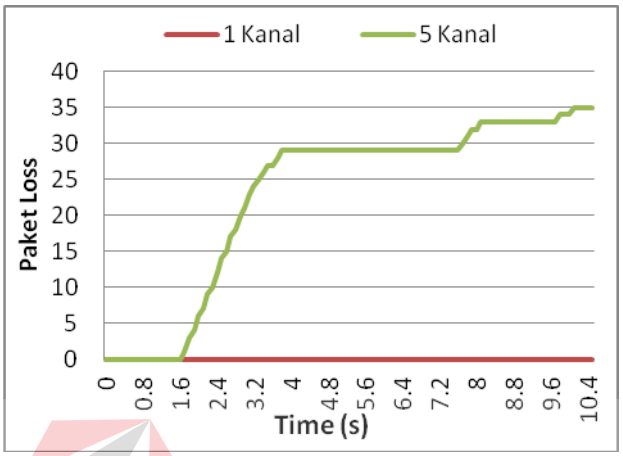
5. Skrip perl untuk *packet loss* sesuai dengan *flowchart* pada Gambar 3.11:

```
# perl packetloss.pl <tcp_topologi.out> > loss
$infile=$ARGV[0];
open (DATA,"<$infile") || die "Can't open $infile $!";
while (<DATA>)
{
 @x = split(' ');
 print STDOUT "$x[0] $x[5] $x[7]\n";
}
close DATA;
exit(0);
```

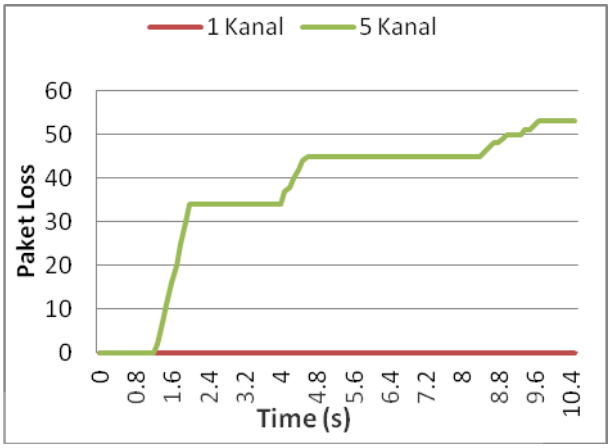
Dari skrip perl tersebut dapat dipanggil dengan perintah berikut pada terminal:

```
rinda@rinda:~$ perl packetloss.pl tcp_topologi.out > loss
```

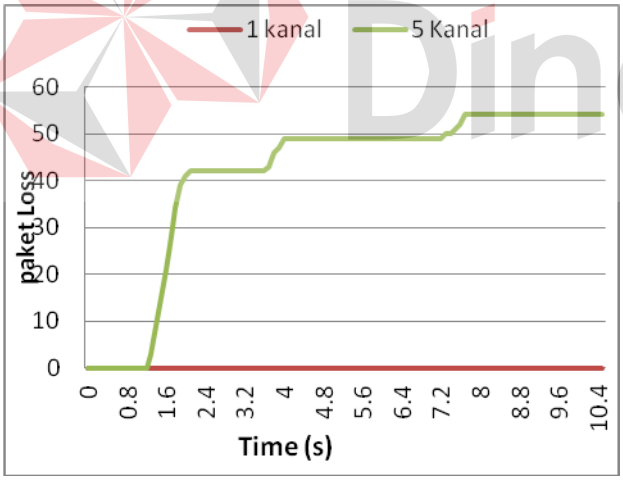
Data hasil *packet loss* yang telah di dapatkan kemudian digambarkan dalam bentuk grafik menggunakan *microsoft ecxel* 2007. Grafik ini di buat untuk lebih mempermudah dalam melakukan analisis.



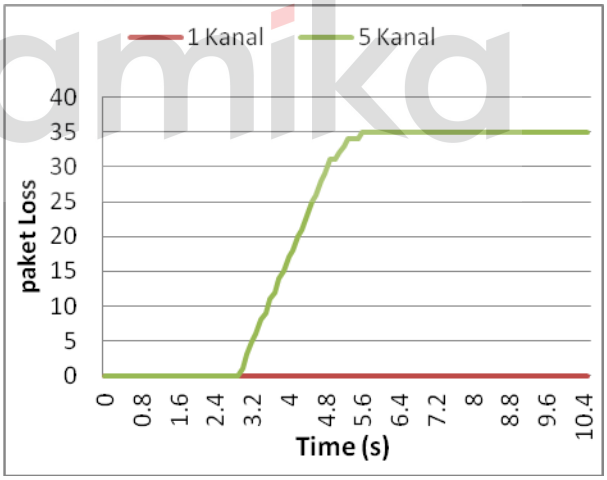
a. *Packet Loss* VOIP 100 ms-128Kb



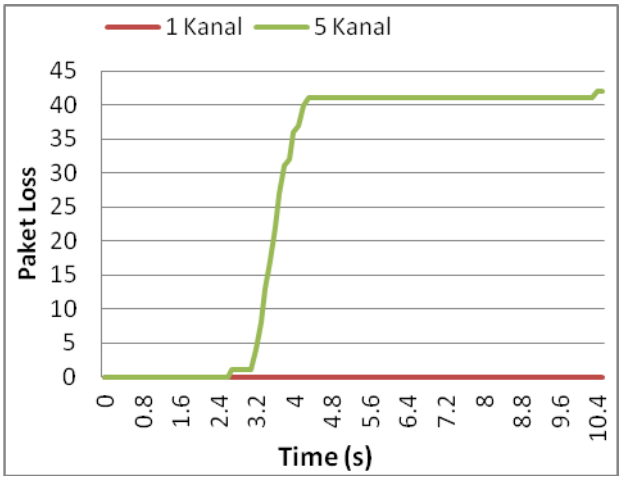
b. *Packet Loss* VOIP 100 ms – 384 Kb

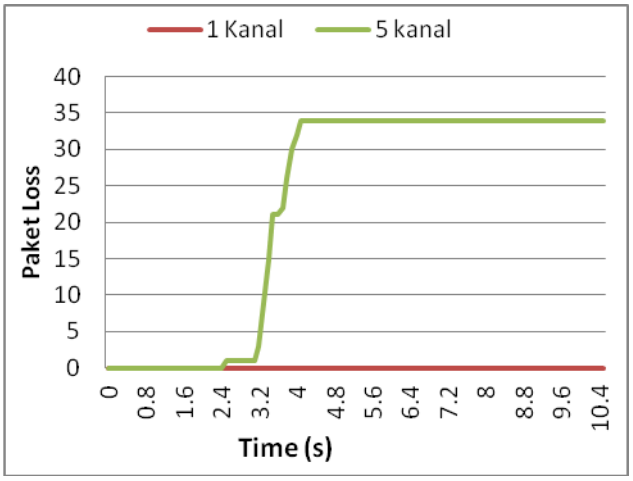


c. *Packet Loss* VOIP 100 ms – 512 Kb



d. *Packet Loss* VOIP 300 ms – 128 Kb

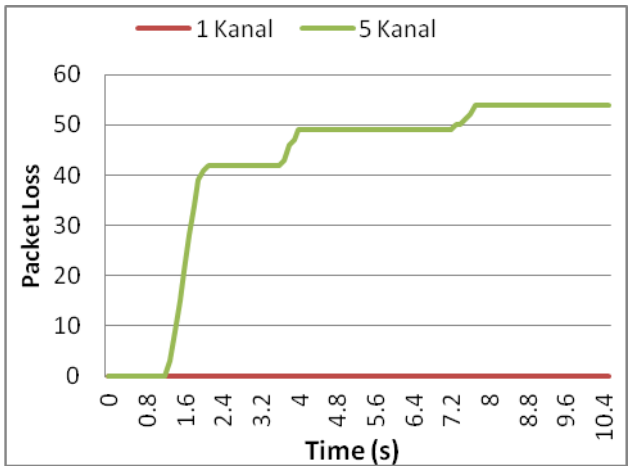


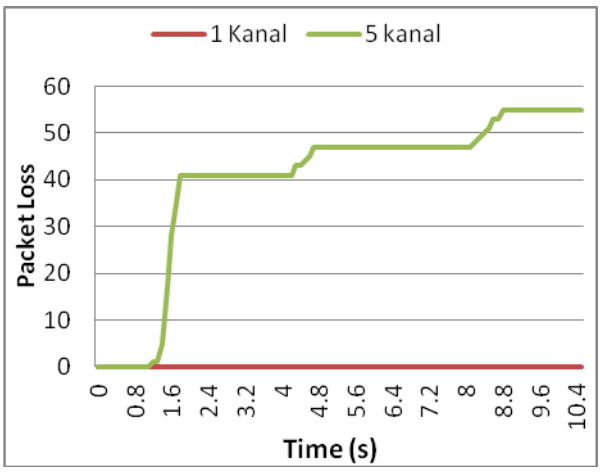


e. *Packet Loss* VOIP 300 ms -384 Kb f. *Packet Loss* VOIP 300ms – 512 Kb

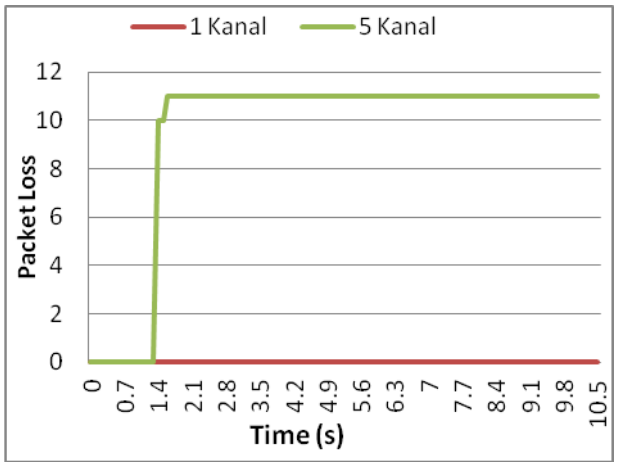
Gambar 4.8 *Packet Loss* Data VOIP

Pada Gambar 4.8 menunjukan nilai *packet loss* yang di dapat terhadap waktu (*time)* dari protokol TCP dengan data VOIP. Data *packet loss* bersifat kumulatif. Dari grafik yang ditampilkan menunjukan protokol TCP dengan menggunakan 1 kanal tidak mengalami kehilangan paket di jalur *bottleneck*. Sedangkan dengan 5 kanal menunjukan besar paket yang hilang naik secara kumulatif.

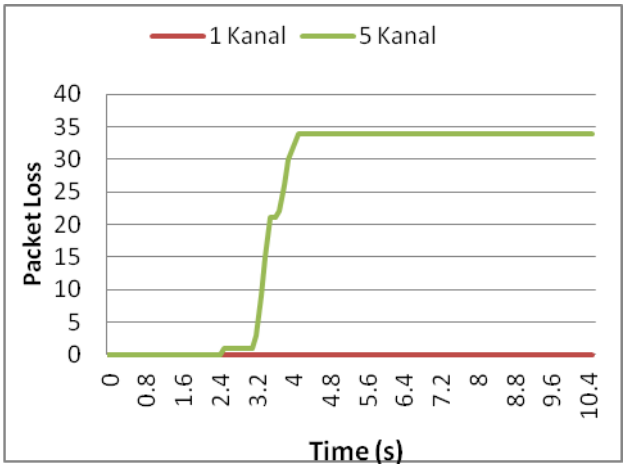




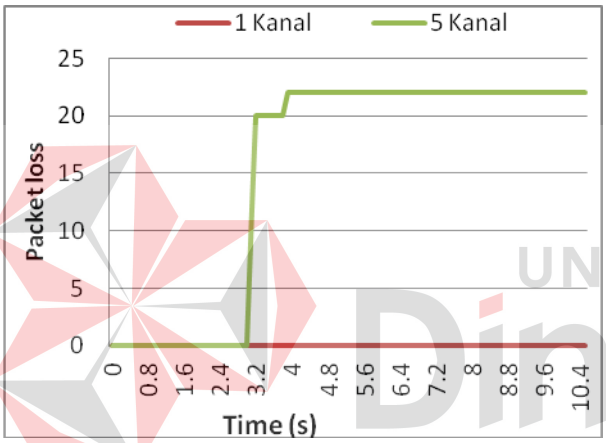
a. *Packet Loss* IPTV 100ms – 512 Kb b. *Packet Loss* IPTV 100ms – 1 Mb



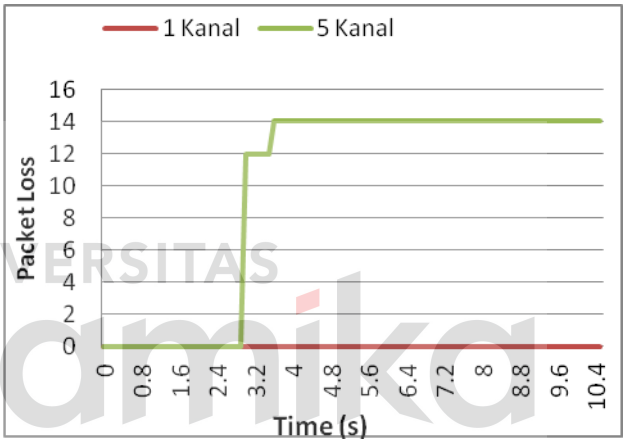
c. *Packet Loss* IPTV 100ms – 3 Mb



d. *Packet Loss* IPTV 300ms – 512 kb



c. *Packet Loss* IPTV 300ms – 1 Mb



d. *Packet Loss* IPTV 300ms – 3 Mb

Gambar 4.9 *Packet Loss* Data IPTV

Pada Gambar 4.9 menunjukan nilai *packet loss* yang di dapat terhadap waktu (*time*) dari protokol TCP dengan data IPTV. Data *packet loss* bersifat kumulatif. Dari grafik yang ditampilkan menunjukan protokol TCP dengan menggunakan 1 kanal tidak mengalami kehilangan paket di jalur *bottleneck.* Sedangkan dengan 5 kanal menunjukan besar paket yang hilang naik secara kumulatif.

4.2.2 *User Datagram Protocol* (UDP)

Dari Tabel 4.1 dilakukan dalam 2 kali percobaan, yaitu menggunakan 1 kanal, dan 5 kanal. Jumlah kanal digunakan untuk menunjukan kondisi jaringan saat data di jalankan tanpa adanya kongesti dengan menggunakan 1 kanal, dan dengan 5 kanal akan menunjukan kinerja dari jaringan saat jalur *bottleneck* harus berbagi data dengan yang lain. Percobaan dilakukan dengan menjalankan skrip *.tcl pada NS, yang hasilnya adalah *nam file, trace file* dan *.*out*, kemudian dengan menambahkan beberapa skrip perl untuk mendapatkan data yang diinginkan seperti *delay, paket loss* dan *queue*.

Di bawah ini merupakan skrip percobaan 1 yaitu udp_topologi.tcl dengan besar variasi ukuran *delay propagation/bandwidth*=100ms/128Kb menggunakan layanan data VOIP dan IPTV. Kedua data ini pada ns2 direpresentasikan oleh trafik cbr dengan ukuran paket yang berbeda. Ukuran paket yang digunakan untuk data VOIP sebesar 160 *byte*, sedangkan untuk data IPTV yaitu 1300 *byte*. Percobaan menggunakan 1 kanal dan 5 kanal. Untuk percobaan menggunakan 1 kanal, data VOIP direpresentasi oleh variabel cbr0 sehingga ketika akan mensimulasikannya waktu untuk proses simulasi cbr0 diaktifkan dan waktu proses simulasi untuk cbr1 yang merupakan variabel representasi dari IPTV di nonaktifkan dengan memberikan komentar pada awal baris dan sebaliknya. Untuk percobaan menggunakan 5 kanal, data VOIP dan IPTV direpresentasikan dengan variabel cbr0, cbr1, cbr2, cbr3, dan cbr4 dengan topologi yang terpisah, namun yang digunakan dalam proses pengambilan data hanya variabel cbr0, sedangkan variabel yang lain merupakan trafik yang digunakan untuk menghasilkan kongesti.

Hasil grafik sesuai dengan percobaan yang dilakukan yaitu sebagai berikut.

1. Percobaan dengan 1 kanal :

```
set ns [new Simulator]
set tr [open udp_topologi.tr w]
set f [open udp_topologi.nam w]
$ns trace-all $tr
$ns namtrace-all $f

proc finish {} {
 global ns f tr
 $ns flush-trace
 close $f
 close $tr
 exec nam udp_topologi.nam &
 exit 0
 }

set node0 [$ns node]
set node1 [$ns node]
set node2 [$ns node]
set node3 [$ns node]
set node4 [$ns node]
set node5 [$ns node]

$ns duplex-link $node0 $node2 1Mb 15ms DropTail
$ns duplex-link $node1 $node2 1Mb 15ms DropTail
$ns simplex-link $node2 $node3 128Kb 100ms DropTail
$ns simplex-link $node3 $node2 128Kb 100ms DropTail
$ns duplex-link $node3 $node4 1Mb 15ms DropTail
$ns duplex-link $node3 $node5 1Mb 15ms DropTail

$ns duplex-link-op $node0 $node2 orient right-down
$ns duplex-link-op $node1 $node2 orient right-up
$ns simplex-link-op $node2 $node3 orient right
$ns simplex-link-op $node3 $node2 orient left
$ns duplex-link-op $node3 $node4 orient right-up
$ns duplex-link-op $node3 $node5 orient right-down

$ns queue-limit $node2 $node3 20
$ns color 1 red
$ns color 2 blue

set udp0 [new Agent/UDP]
$ns attach-agent $node0 $udp0
$udp0 set fid_ 1
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 160
$cbr0 attach-agent $udp0
set sink0 [new Agent/Null]
$ns attach-agent $node4 $sink0
$ns connect $udp0 $sink0

set udp1 [new Agent/UDP]
$ns attach-agent $node1 $udp1
```

```
$udp1 set fid_ 2
set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 1300
$cbr1 attach-agent $udp1
set sink1 [new Agent/Null]
$ns attach-agent $node5 $sink1
$ns connect $udp1 $sink1

set qmon [$ns monitor-queue $node2 $node3 [open
udp_topologi.out w] 0.1];
[$ns link $node2 $node3] queue-sample-timeout;

$ns at 0.1 "$cbr0 start"
$ns at 10.0 "$cbr0 stop"

# di aktifkan jika ingin mensimulasikan data IPTV
$ns at 0.1 "$cbr1 start"
$ns at 10.0 "$cbr1 stop"

$ns at 10.5 "finish"
$ns run
```

2. Skrip percobaan dengan 5 kanal

```
set ns [new Simulator]
set tr [open udp_topologi.tr w]
set f [open udp_topologi.nam w]
$ns trace-all $tr
$ns namtrace-all $f

proc finish {} {
 global ns f tr
 $ns flush-trace
 close $f
 close $tr
 exec nam udp_topologi.nam &
 exit 0
 }

set node0 [$ns node]
set node1 [$ns node]
set node2 [$ns node]
set node3 [$ns node]
set node4 [$ns node]
set node5 [$ns node]
set node6 [$ns node]
set node7 [$ns node]
set node8 [$ns node]
set node9 [$ns node]
set node10 [$ns node]
set node11 [$ns node]

$ns duplex-link $node0 $node5 1Mb 15ms DropTail
$ns duplex-link $node1 $node5 1Mb 15ms DropTail
$ns duplex-link $node2 $node5 1Mb 15ms DropTail
$ns duplex-link $node3 $node5 1Mb 15ms DropTail
$ns duplex-link $node4 $node5 1Mb 15ms DropTail
$ns simplex-link $node5 $node6 128Kb 100ms DropTail
```

```
$ns simplex-link $node6 $node5 128Kb 100ms DropTail
$ns duplex-link $node6 $node7 1Mb 15ms DropTail
$ns duplex-link $node6 $node8 1Mb 15ms DropTail
$ns duplex-link $node6 $node9 1Mb 15ms DropTail
$ns duplex-link $node6 $node10 1Mb 15ms DropTail
$ns duplex-link $node6 $node11 1Mb 15ms DropTail
$ns duplex-link-op $node0 $node5 orient down
$ns duplex-link-op $node1 $node5 orient right-down
$ns duplex-link-op $node2 $node5 orient right
$ns duplex-link-op $node3 $node5 orient right-up
$ns duplex-link-op $node4 $node5 orient up
$ns simplex-link-op $node5 $node6 orient right
$ns simplex-link-op $node6 $node5 orient left
$ns duplex-link-op $node6 $node7 orient up
$ns duplex-link-op $node6 $node8 orient right-up
$ns duplex-link-op $node6 $node9 orient right
$ns duplex-link-op $node6 $node10 orient right-down
$ns duplex-link-op $node6 $node11 orient down

$ns queue-limit $node5 $node6 20
$ns color 1 red
$ns color 2 blue
$ns color 3 green
$ns color 4 pink
$ns color 5 black

set udp0 [new Agent/UDP]
$ns attach-agent $node0 $udp0
$udp0 set fid_ 1
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 160
$cbr0 attach-agent $udp0
set sink0 [new Agent/Null]
$ns attach-agent $node7 $sink0
$ns connect $udp0 $sink0

set udp1 [new Agent/UDP]
$ns attach-agent $node1 $udp1
$udp1 set fid_ 2
set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 160
$cbr1 attach-agent $udp1
set sink1 [new Agent/Null]
$ns attach-agent $node8 $sink1
$ns connect $udp1 $sink1

set udp2 [new Agent/UDP]
$ns attach-agent $node2 $udp2
$udp2 set fid_ 3
set cbr2 [new Application/Traffic/CBR]
$cbr2 set packetSize_ 160
$cbr2 attach-agent $udp2
set sink2 [new Agent/Null]
$ns attach-agent $node9 $sink2
$ns connect $udp2 $sink2

set udp3 [new Agent/UDP]
$ns attach-agent $node3 $udp3
$udp3 set fid_ 4
```

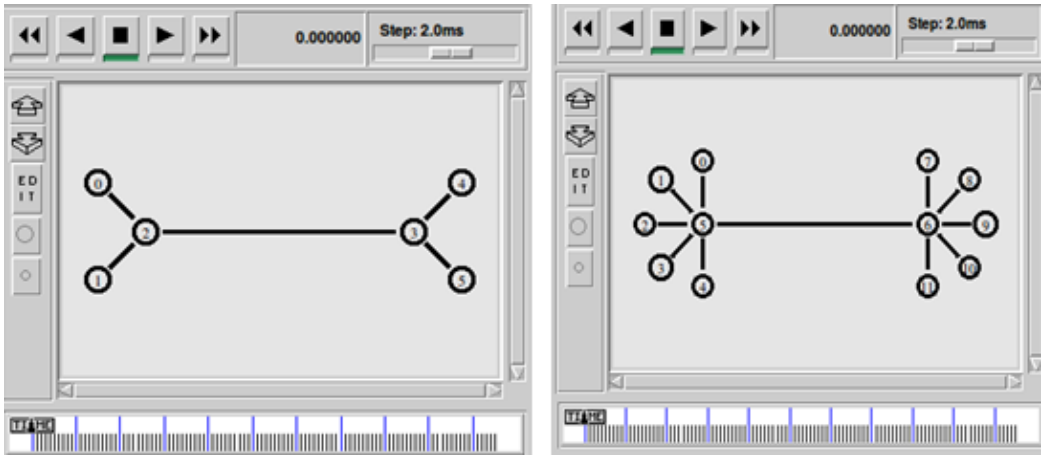
```
set cbr3 [new Application/Traffic/CBR]
$cbr3 set packetSize_ 160
$cbr3 attach-agent $udp3
set sink3 [new Agent/Null]
$ns attach-agent $node10 $sink3
$ns connect $udp3 $sink3

set udp4 [new Agent/UDP]
$ns attach-agent $node4 $udp4
$udp4 set fid_ 5
set cbr4 [new Application/Traffic/CBR]
$cbr4 set packetSize_ 160
$cbr4 attach-agent $udp4
set sink4 [new Agent/Null]
$ns attach-agent $node11 $sink4
$ns connect $udp4 $sink4

set qmon [$ns monitor-queue $node5 $node6 [open
udp_topologi.out w] 0.1];
[$ns link $node5 $node6] queue-sample-timeout;

$ns at 0.1 "$cbr0 start"
$ns at 10.0 "$cbr0 stop"
$ns at 0.1 "$cbr1 start"
$ns at 10.0 "$cbr1 stop"
$ns at 0.1 "$cbr2 start"
$ns at 10.0 "$cbr2 stop"
$ns at 0.1 "$cbr3 start"
$ns at 10.0 "$cbr3 stop"
$ns at 0.1 "$cbr4 start"
$ns at 10.0 "$cbr4 stop"

$ns at 10.5 "finish"
$ns run
```

Teks yang tercetak tebal menandakan parameter yang harus diubah-ubah. Besar perubahan *bandwidth* dan *delay propagation* bergantung dari nilai yang telah ditentukan dalam tabel. Sedangkan besar *packet size* bergantung jenis data yang digunakan, 160 *byte* untuk VOIP dan 1300 *byte* untuk IPTV. Simulasi dijalankan dengan mengetikan perintah berikut pada jendela terminal.

```
rinda@rinda:~$ ns udp_topologi.tcl
```

Setelah program simulasi dijalankan maka akan didapatkan file nam seperti pada Gambar 4.10.

Dengan 1 kanal Dengan 5 Kanal



Gambar 4.10 Tampilan Nam

Dari hasil nam diatas dapat dilihat topologi dari skrip yang telah di buat. Dengan topologi menggunakan 1 kanal, *node-node* yang dibutuhkan oleh sistem dalam melakukan simulasi yaitu sebanyak 6 *node* dengan bentuk topologi *dumb-bell* standar seperti pada Gambar 4.10. aliran dari *node* 0 akan membawa data VOIP sedang aliran data pada *node* 1 akan membawa data IPTV. Saat menjalankan program dengan data VOIP, maka aliran data IPTV dihentikan, dan begitu sebaliknya.

Pada hasil simulasi menggunakan 5 kanal, *node-node* yang dibutuhkan dalam melakukan proses simulasi menggunakan node berjumlah 12 *node*. *Node* 0,1,2,3,4 mengalirkan data cbr dengan besarnya ukuran paket bergantung dari jenis data yang di gunakan.

Simulasi berjalan saat tombol *play* di jalankan dan menghasilkan out.tr yang merekam seluruh kejadian dari pengiriman *node* sumber ke *node* tujuan, selain itu simulasi juga menghasilkan qm.out yang merupakan informasi kejadian pada jalur *bottleneck*. Selanjutnya dilakukan *parsing* file dengan menggunakan skrip perl *delay*, *queue* dan *packet loss* berikut:

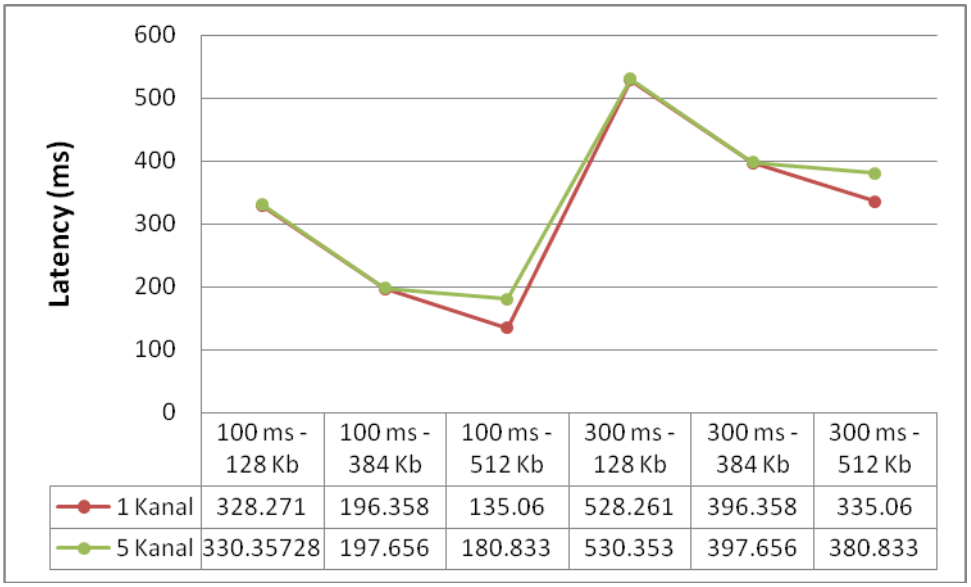
3. Skrip perl untuk *delay* sesuai dengan *flowchart* pada Gambar 3.9:

```
# type: perl delay.pl <trace file> > output file
$infile=$ARGV[0];
open (DATA,"<$infile") || die "Can't open $infile $!";
while (<DATA>)
{
 @x = split(' ');
 @UDP0;
 @UDP4;
 @hsl;
 if ($x[6] eq '-------')
 {
    if ($x[4] eq 'cbr')
    {
          if ($x[0] eq '+')
          {
                if ($x[2] eq '0')
                {
                   $UDP0[$x[11]]=$x[1];
                }
          }
          if ($x[0] eq 'r')
          {
                if ($x[3] eq '4')
                {
                   $UDP4[$x[11]]=$x[1];
              $hsl[$x[11]]=$UDP4[$x[11]]-$UDP0[$x[11]];
                   print STDOUT "$x[10] $hsl[$x[11]] \n";
                }
          }
    }
 }
}
close DATA;
exit(0);
```

Dari skrip perl tersebut dapat dipanggil dengan perintah berikut pada terminal:

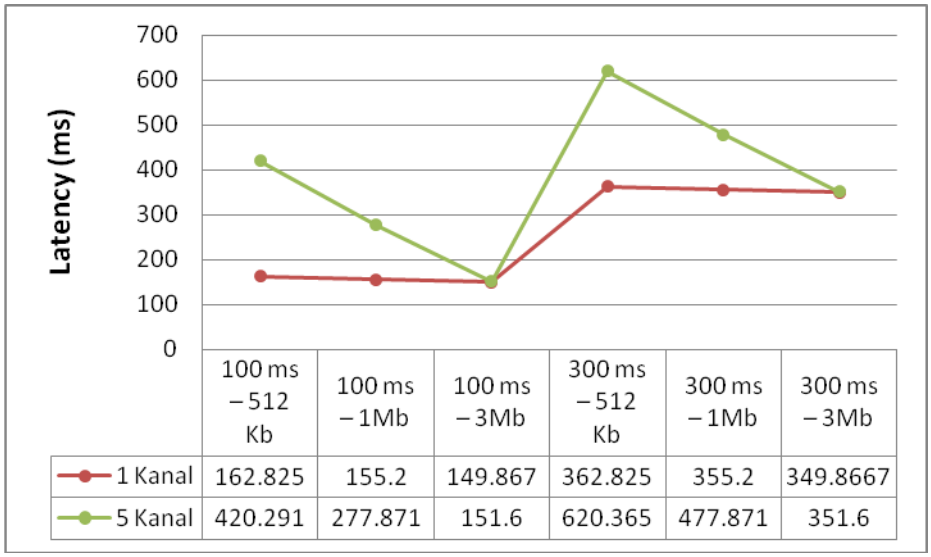
```
rinda@rinda:~$ perl delay.pl udp_topologi.tr > delay
```

Dari data *delay* yang diperoleh dengan melakukan *parsing* menggunakan perl, data tersebut diolah kembali untuk menghasilkan nilai *latency* dan *jitter* dengan menggunakan rumus yang telah di jelaskan pada bab sebelumnya menggunakan *microsoft ecxel* 2007. Data ini kemudian ditampilkan menggunakan grafik untuk lebih memperjelas dalam melakukan analisis.



Gambar 4.11 Besar *Latency* Data VOIP

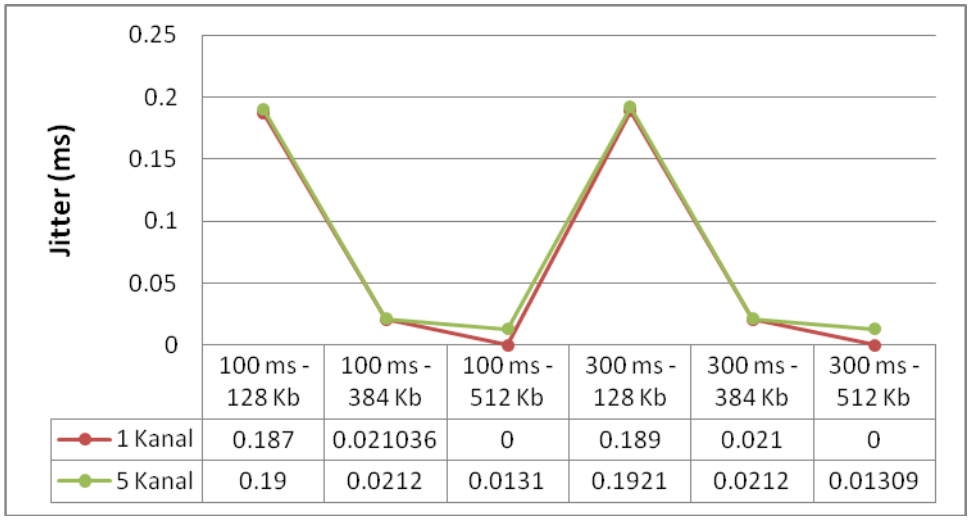
Gambar 4.11 menunjukan besar *latency* dari protokol UDP dengan menggunakan layanan dari data VOIP. Dari grafik tersebut dapat dilihat bahwa dengan menggunakan 1 kanal dan 5 kanal *latency* UDP dengan *bandwidth* 128Kb dan *delay propagation* 300 ms merupakan *latency* yang paling besar nilainya dibanding dengan yang lain. Sedangkan *latency* terendah terjadi saat penggunaan *bandwidth* 512 Kb dan *delay propagation* 100 ms.



	100 ms – 512 Kb	100 ms – 1Mb	100 ms – 3Mb	300 ms – 512 Kb	300 ms – 1Mb	300 ms – 3Mb
1 Kanal	162.825	155.2	149.867	362.825	355.2	349.8667
5 Kanal	420.291	277.871	151.6	620.365	477.871	351.6

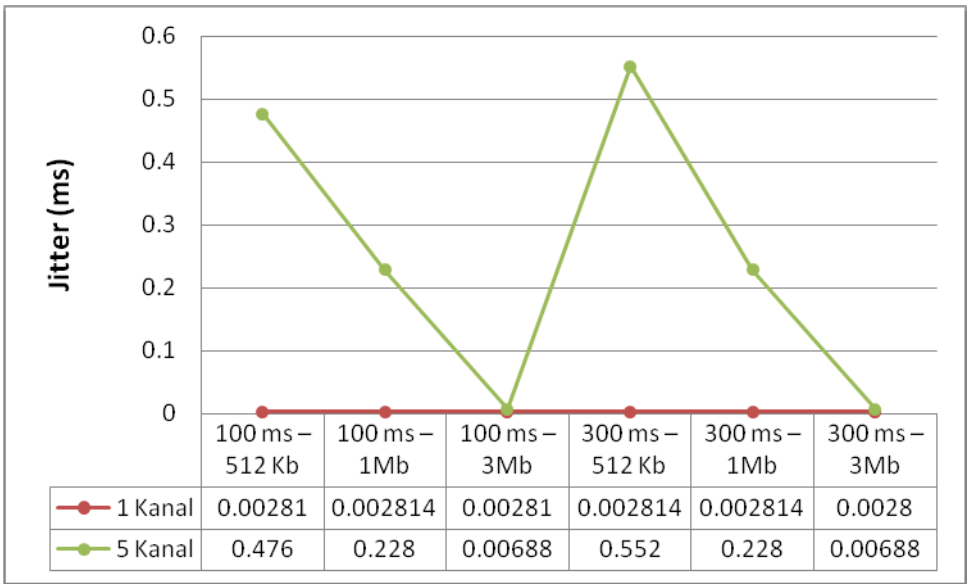
Gambar 4.12 Besar *Latency* Data IPTV

Gambar 4.12 menunjukan besar *latency* dari protokol UDP dengan menggunakan layanan dari data IPTV. Dari grafik tersebut dapat dilihat bahwa dengan menggunakan 1 kanal dan 5 kanal, *latency* UDP dengan *bandwidth* 512 Kb dan *delay propagation* 300 ms merupakan *latency* yang paling besar nilainya dibanding dengan yang lain. Sedangkan *latency* terendah terjadi saat penggunaan *bandwidth* 3 Mb dan *delay propagation* 100 ms. Pada *bandwidth* 3 Mb, *latency* pada 1 kanal dan 5 kanal menghasilkan perbedaan besar nilai yang terlalu signifikan.



Gambar 4.13 Besar *Jitter* Data VOIP

Pada Gambar 4.13 didapatkan bahwa besar nilai variasi *delay* atau *jitter* dari UDP yang menggunakan layanan data VOIP dengan 1 kanal maupun 5 kanal dipengaruhi oleh *bandwidth*, *sedangkan delay propagation* tidak terlalu berpengaruh pada besarnya *jitter* di UDP, terlihat dari nilai *jitter* yang tidak sigifikan untuk *delay propagation* yang berbeda.



Gambar 4.14 Besar *Jitter* Data IPTV

Pada Gambar 4.14 didapatkan bahwa besar nilai variasi *delay* atau *jitter* dari UDP yang menggunakan layanan data IPTV dengan 1 kanal maupun 5 kanal dipengaruhi oleh *bandwidth*, sedangkan *delay propagation* tidak terlalu berpengaruh pada besarnya *jitter* di UDP, terlihat dari nilai *jitter* yang tidak sigifikan untuk *delay propagation* yang berbeda. Penggunaan *jitter* dengan 1 kanal menghasilkan nilai yang rendah dibanding pada pengiriman dengan5 kanal.

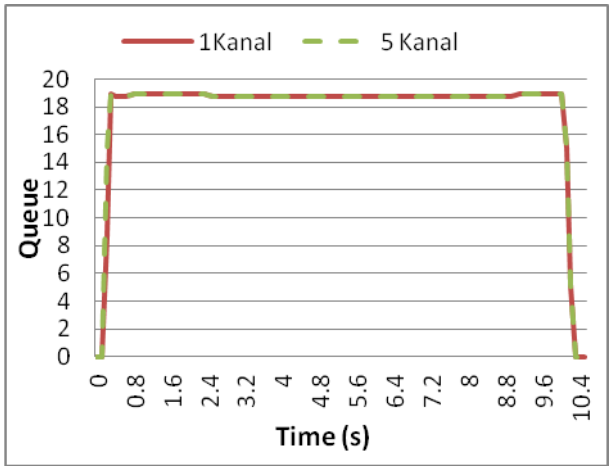
4. Skrip perl untuk *queue* sesuai dengan *flowchart* pada Gambar 3.10:

```
# perl queue.pl  <udp_topologi.out>  > queue
$infile=$ARGV[0];
open (DATA,"<$infile") || die "Can't open $infile $!";
while (<DATA>)
{
 @x = split(' ');
 print STDOUT "$x[0]  $x[4]\n";

}
close DATA;
exit(0);
```

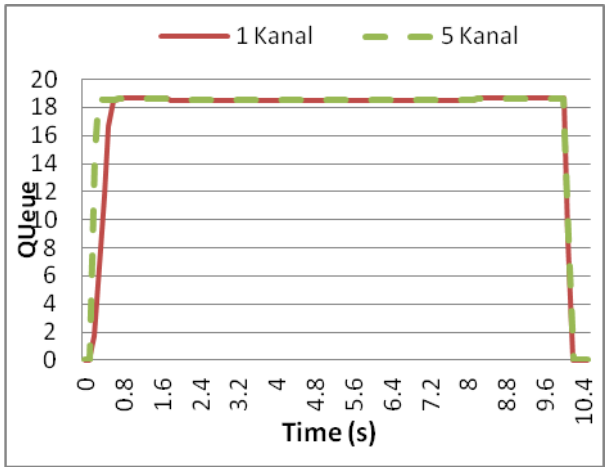
Dari skrip perl tersebut dapat dipanggil dengan perintah berikut pada terminal:

```
rinda@rinda:~$ perl queue.pl udp_topologi.out > queue
```

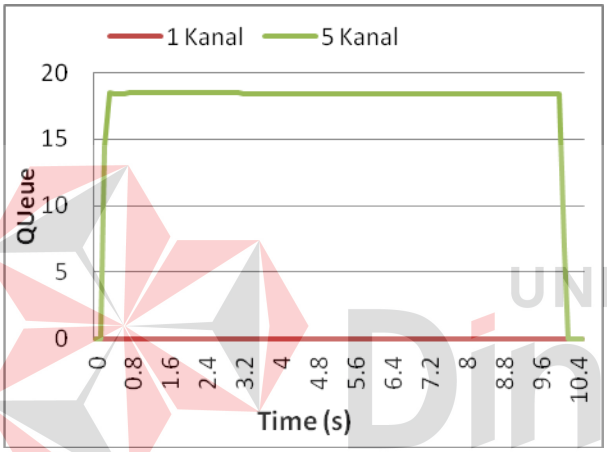
Data hasil queue yang telah di dapatkan kemudian digambarkan dalam bentuk grafik menggunakan *microsoft ecxel* 2007. Grafik ini di buat untuk leb\ih mempermudah dalam melakukan analisis.



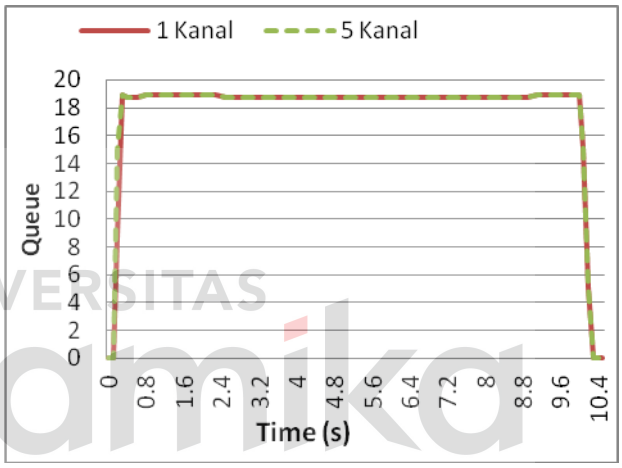
a. *Queue* VOIP 100ms – 128 Kb



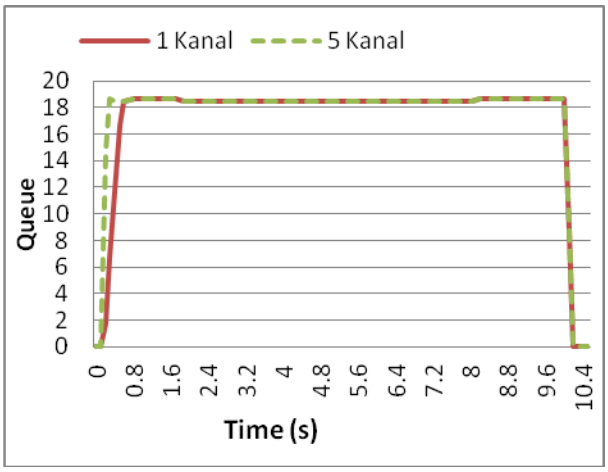
b. *Queue* VOIP 100ms – 384 Kb



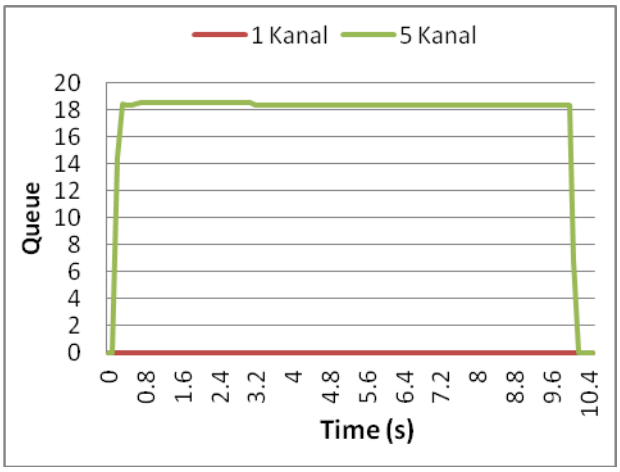
c. *Queue* VOIP 100ms – 512 Kb



d. *Queue* VOIP 300ms – 128 Kb



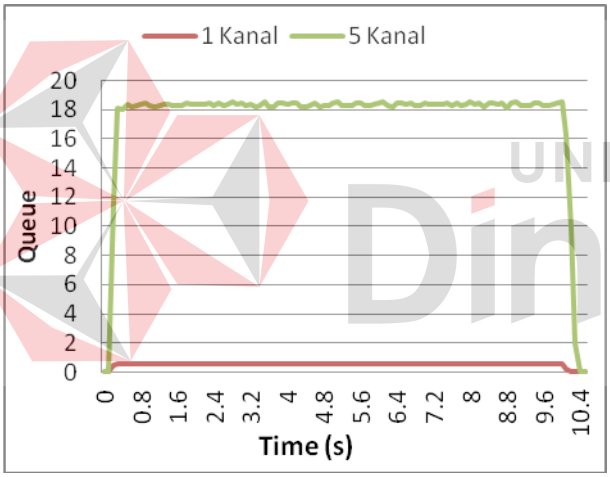
e. *Queue* VOIP 300ms – 384 Kb

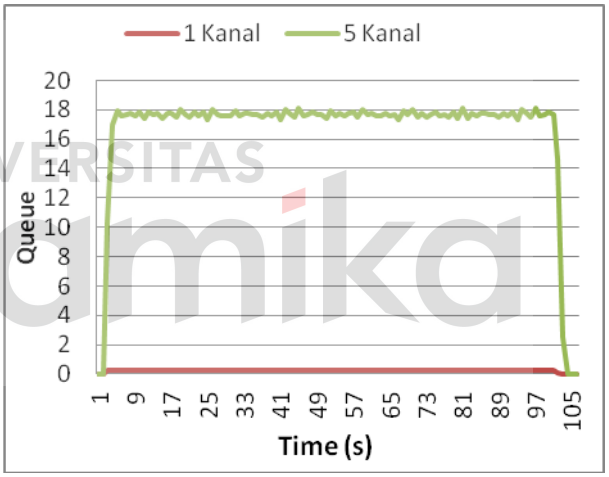


f. *Queue* VOIP 300ms – 512 Kb

Gambar 4.15 *Queue* Data VOIP

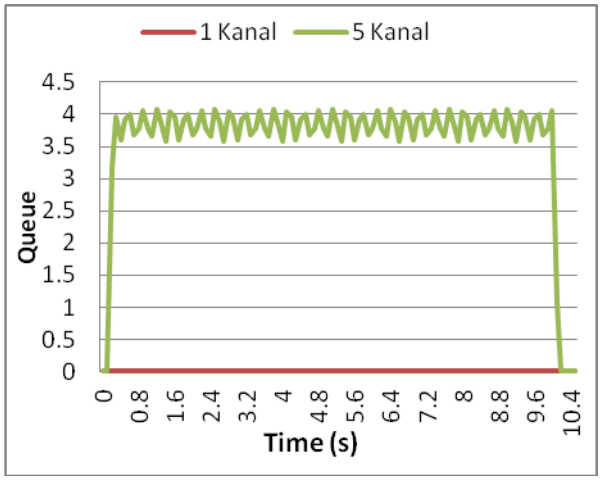
Pada Gambar 4.15 menunjukan nilai *queue* yang di dapat terhadap waktu (*time*) dari protokol UDP dengan data VOIP. Grafik yang naik menunjukan bahwa kapasitas antrian yang terpakai oleh sistem pada jalur *bottleneck* mengalami kenaikan untuk rentang waktu tersebut. Terlihat dari grafik tersebut bahwa penggunaan *queue* untuk protokol UDP pada 1 kanal maupun 5 kanal hampir maksimal, kecuali untuk *bandwidth* 512 Kb. *Queue* maksimum yang telah di seting adalah sebesar 20. Terlihat dari grafik bahwa penggunaan *queue* protokol UDP dengan 1 kanal dan 5 kanal sama-sama tinggi, kecuali saat penggunaan



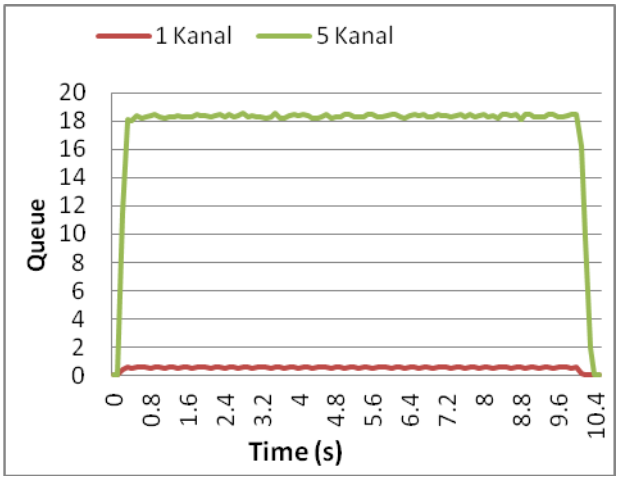


bandwidth 512.

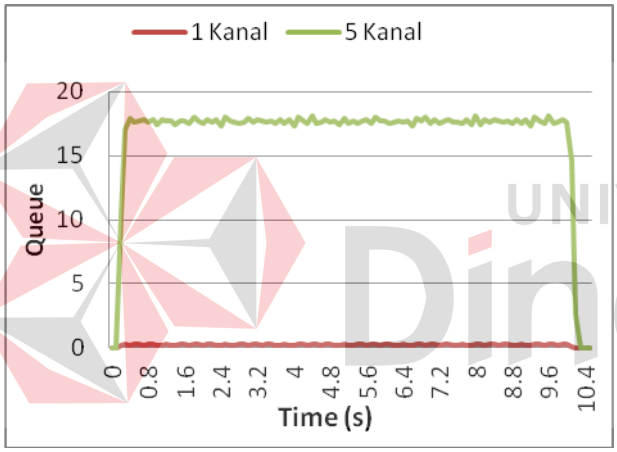
a. *Queue* IPTV 100ms – 512 Kb b. *Queue* IPTV 100ms – 384 Kb



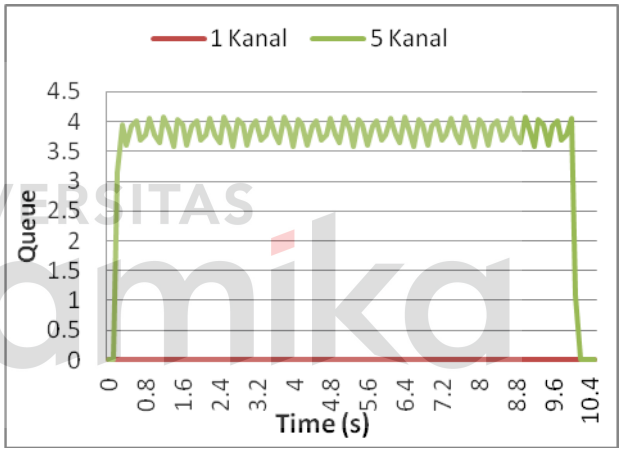
c. *Queue* IPTV 100ms – 3 Mb



d. *Queue* IPTV 300ms – 512 Kb



e. *Queue* IPTV 300ms – 1 Mb



f. *Queue* IPTV 300ms – 3 Mb

Gambar 4.16 Queue Data IPTV

Pada Gambar 4.16 menunjukan nilai *queue* yang di dapat terhadap waktu (*time*) dari protokol UDP dengan data IPTV. Grafik yang naik menunjukan bahwa kapasitas antrian yang terpakai oleh sistem pada jalur *bottleneck* mengalami kenaikan untuk rentang waktu tersebut. Pada saat menggunakan 1 kanal, penggunaan queue pada UDP dengan data IPTV rata-rata berada sangat rendah yang menunjukan bahwa tidak ada antrian yang penuh di jalur *bottleneck. Queue* maksimum yang telah di seting adalah sebesar 20.

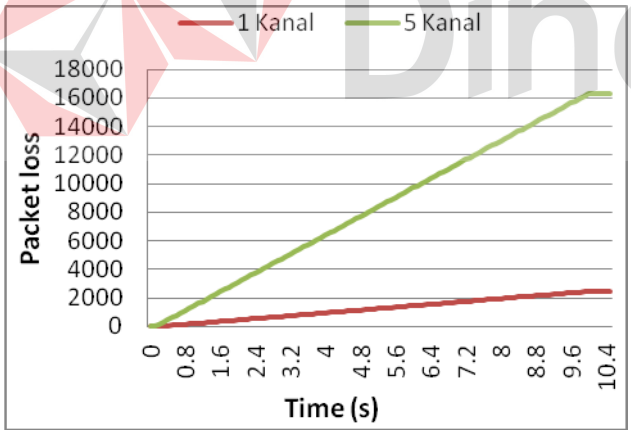
5. Skrip perl untuk *packet loss* sesuai dengan *flowchart* pada Gambar 3.11:

```
# perl packetloss.pl <udp_topologi.out> > loss
$infile=$ARGV[0];
open (DATA,"<$infile") || die "Can't open $infile $!";
while (<DATA>)
{
 @x = split(' ');
 print STDOUT "$x[0] $x[5] $x[7]\n";

}
close DATA;
exit(0);
```

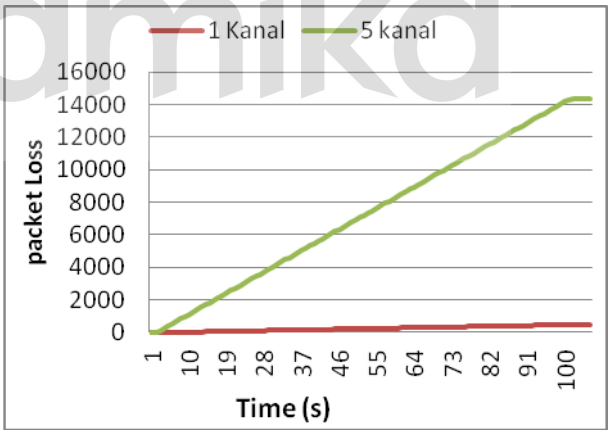
Dari skrip perl tersebut dapat dipanggil dengan perintah berikut pada terminal:

```
rinda@rinda:~$ perl packetloss.pl udp_topologi.out > loss
```

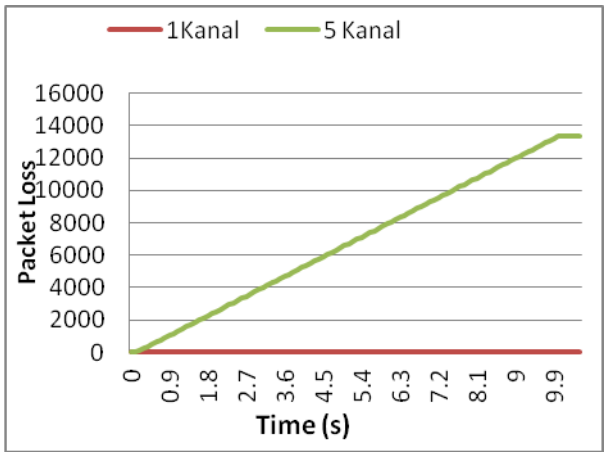
Data hasil *packet loss* yang telah di dapatkan kemudian digambarkan dalam bentuk grafik menggunakan *microsoft ecxel* 2007. Grafik ini di buat untuk lebih mempermudah dalam melakukan analisis.

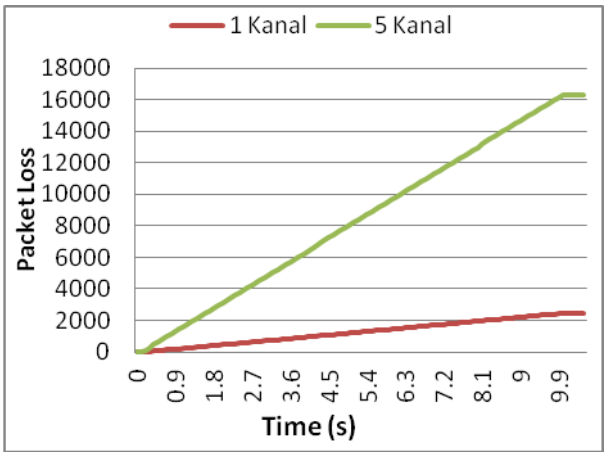


a. *Packet Loss* VOIP 100ms – 128 Kb

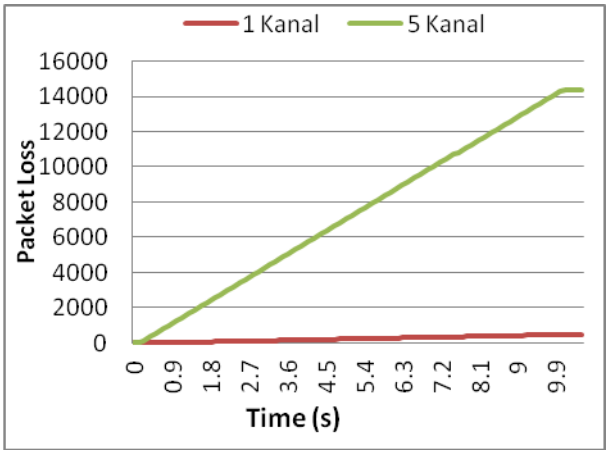


b. *Packet Loss* VOIP 100ms – 384Kb

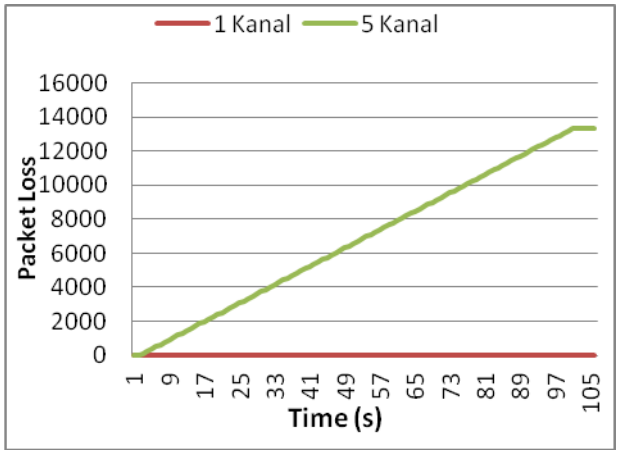




c. *Packet Loss* VOIP 100ms – 512 Kb



d. *Packet Loss* VOIP 300ms – 128 Kb

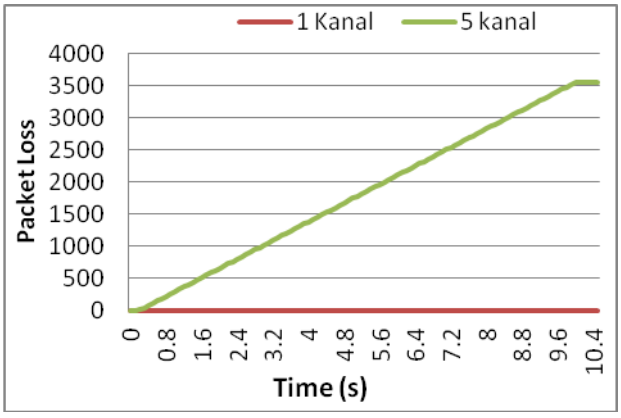


e. *Packet Loss* VOIP 300ms – 384 Kb

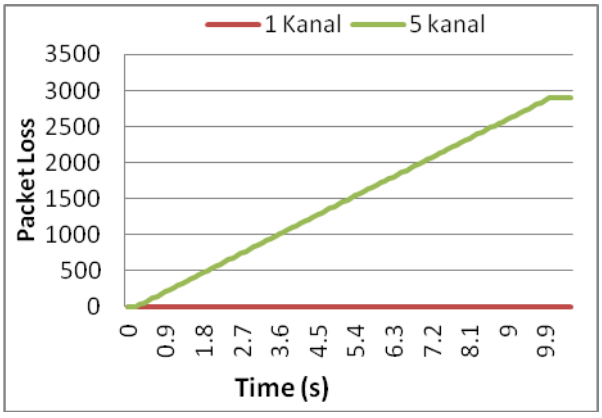
f. *Packet Loss* VOIP 300ms – 512Kb

Gambar 4.17 *Packet Loss* Data VOIP

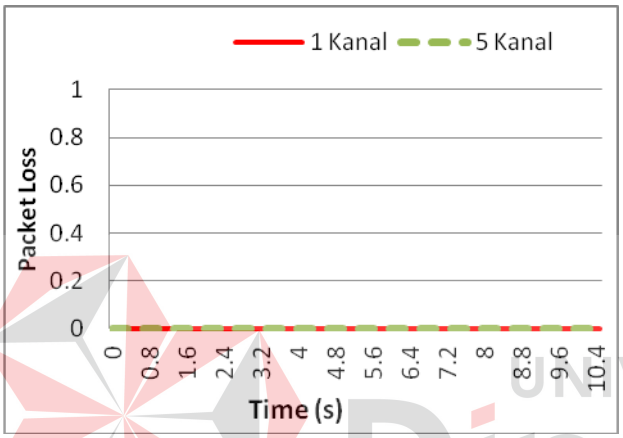
Pada Gambar 4.17 menunjukan nilai *packet loss* yang di dapat terhadap waktu (*time*) dari protokol UDP dengan data VOIP Data *packet loss* bersifat kumulatif. Dari grafik yang ditampilkan menunjukan protokol UDP dengan menggunakan 5 kanal mengalami kenaikan yang sangat tinggi di banding dengan menggunakan 1 kanal. Hal ini di sebabkan karena jalur *bottleneck* pada antrian digunakan oleh 5 data UDP dalam satu waktu yang berakibat pemenuhan kapasitas *queue.*



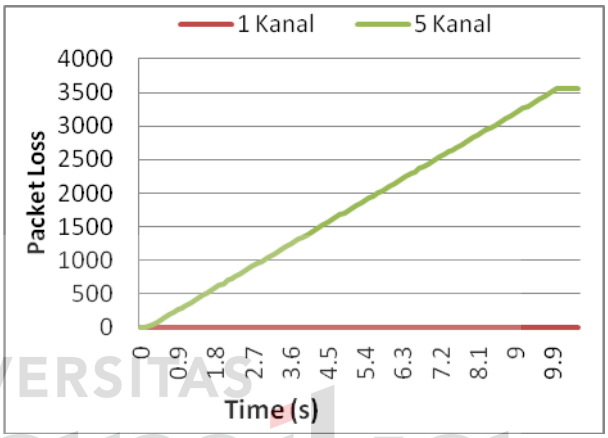
a. *Packet Loss* IPTV 100ms – 512 Kb



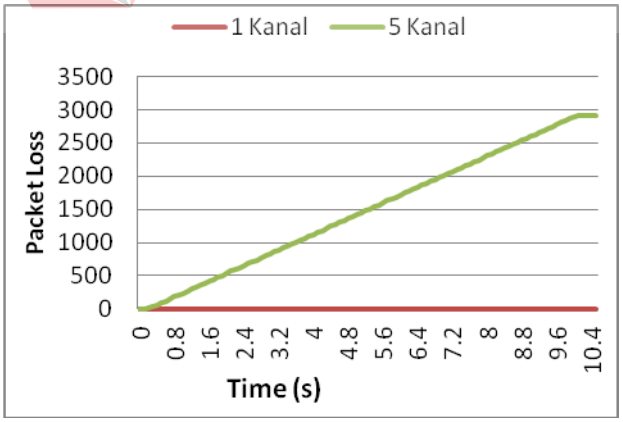
b. *Packet Loss* IPTV 100ms – 1 Mb



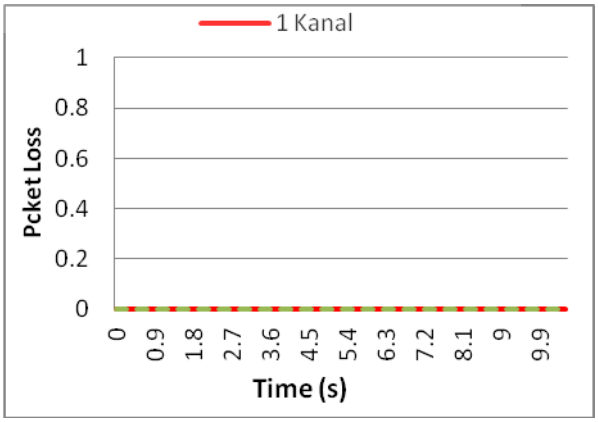
c. *Packet Loss* IPTV 100ms – 3 Mb



d. *Packet Loss* IPTV 300ms – 512Kb



e. *Packet Loss* IPTV 300ms – 1 Mb



f. *Packet Loss* IPTV 300ms – 3Mb

Gambar 4.18 *Packet Loss* Data IPTV

Pada Gambar 4.18 menunjukan nilai *packet loss* yang di dapat terhadap waktu (*time)* dari protokol UDP dengan data IPTV. Data *packet loss* bersifat

kumulatif. Trafik UDP dengan 1 kanal pada data IPTV tidak mengalami kehilangan paket, sedangkan dengan 5 kanal, protokol UDP tidak mengalami kehilangan paket pada *bandwidth* 3 Mb.

4.2.3 *Stream Control Transmission Protocol* (SCTP)

Dari Tabel 4.1 dilakukan dalam 2 kali percobaan, yaitu menggunakan 1 kanal, dan 5 kanal. Jumlah kanal digunakan untuk menunjukan kondisi jaringan saat data di jalankan tanpa adanya kongesti dengan menggunakan 1 kanal, dan dengan 5 kanal akan menunjukan kinerja dari jaringan saat jalur *bottleneck* harus berbagi data dengan yang lain. Percobaan dilakukan dengan menjalankan skrip *.tcl pada NS, yang hasilnya adalah *nam file, trace file* dan **.out*, kemudian dengan menambahkan beberapa skrip perl untuk mendapatkan data yang diinginkan seperti *delay, paket loss* dan *queue*.

Di bawah ini merupakan skrip percobaan 1 yaitu udp_topologi.tcl dengan besar variasi ukuran *delay propagation/bandwidth*=100ms/128Kb menggunakan layanan data VOIP dan IPTV. Kedua data ini pada ns2 direpresentasikan oleh trafik cbr dengan ukuran paket yang berbeda. Ukuran paket yang digunakan untuk data VOIP sebesar 160 *byte*, sedangkan untuk data IPTV yaitu 1300 *byte*. Percobaan menggunakan 1 kanal dan 5 kanal. Untuk percobaan menggunakan 1 kanal, data VOIP direpresentasi oleh variabel cbr0 sehingga ketika akan mensimulasikannya waktu untuk proses simulasi cbr0 diaktifkan dan waktu proses simulasi untuk cbr1 yang merupakan variabel representasi dari IPTV di nonaktifkan dengan memberikan komentar pada awal baris dan sebaliknya. Untuk percobaan menggunakan 5 kanal, data VOIP dan IPTV direpresentasikan dengan variabel cbr0, cbr1, cbr2, cbr3, dan cbr4 dengan topologi yang terpisah, namun

yang digunakan dalam proses pengambilan data hanya variabel cbr0, sedangkan variabel yang lain merupakan trafik yang digunakan untuk menghasilkan congesti.

Hasil grafik sesuai dengan percobaan yang dilakukan yaitu sebagai berikut.

1. Percobaan dengan 1 kanal :

```
set ns [new Simulator]

set tr [open sctp_topologi.tr w]
set f [open sctp_topologi.nam w]
$ns trace-all $tr
$ns namtrace-all $f

proc finish {} {
     global ns f tr
     $ns flush-trace
     close $f
     close $tr
     exec nam sctp_topologi.nam &
     exit 0
     }

set node0 [$ns node]
set node1 [$ns node]
set node2 [$ns node]
set node3 [$ns node]
set node4 [$ns node]
set node5 [$ns node]

$ns duplex-link $node0 $node2 1Mb 15ms DropTail
$ns duplex-link $node1 $node2 1Mb 15ms DropTail
$ns simplex-link $node2 $node3 128Kb 100ms DropTail
$ns simplex-link $node3 $node2 128Kb 100ms DropTail
$ns duplex-link $node3 $node4 1Mb 15ms DropTail
$ns duplex-link $node3 $node5 1Mb 15ms DropTail

$ns duplex-link-op $node0 $node2 orient right-down
$ns duplex-link-op $node1 $node2 orient right-up
$ns simplex-link-op $node2 $node3 orient right
$ns simplex-link-op $node3 $node2 orient left
$ns duplex-link-op $node3 $node4 orient right-up
$ns duplex-link-op $node3 $node5 orient right-down

$ns queue-limit $node2 $node3 20
$ns color 1 red
$ns color 2 blue

set sctp0 [new Agent/SCTP]
$ns attach-agent $node0 $sctp0
$sctp0 set fid_ 1
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 160
```

```
$cbr0 attach-agent $sctp0
set sctpsink0 [new Agent/SCTP]
$ns attach-agent $node4 $sctpsink0
$ns connect $sctp0 $sctpsink0

set sctp1 [new Agent/SCTP]
$ns attach-agent $node1 $sctp1
$sctp1 set fid_ 2
set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 1300
$cbr1 attach-agent $sctp1
set sctpsink1 [new Agent/SCTP]
$ns attach-agent $node5 $sctpsink1
$ns connect $sctp1 $sctpsink1

set qmon [$ns monitor-queue $node2 $node3 [open
sctp_topologi.out w] 0.1];
[$ns link $node2 $node3] queue-sample-timeout;

$ns at 0.1 "$cbr0 start"
$ns at 10.0 "$cbr0 stop"
# diaktifkan jika akan mensimulasikan data IPTV
$ns at 0.1 "$cbr1 start"
$ns at 10.0 "$cbr1 stop"

$ns at 10.5 "finish"
$ns run
```


2. Percobaan dengan 5 Kanal :

```
set ns [new Simulator]
set tr [open sctp_topologi.tr w]
set f [open sctp_topologi.nam w]
$ns trace-all $tr
$ns namtrace-all $f

proc finish {} {
     global ns f tr
     $ns flush-trace
     close $f
     close $tr
     exec nam sctp_topologi.nam &
     exit 0
     }

set node0 [$ns node]
set node1 [$ns node]
set node2 [$ns node]
set node3 [$ns node]
set node4 [$ns node]
set node5 [$ns node]
set node6 [$ns node]
set node7 [$ns node]
set node8 [$ns node]
set node9 [$ns node]
set node10 [$ns node]
```

```
set node11 [$ns node]

$ns duplex-link $node0 $node5 1Mb 15ms DropTail
$ns duplex-link $node1 $node5 1Mb 15ms DropTail
$ns duplex-link $node2 $node5 1Mb 15ms DropTail
$ns duplex-link $node3 $node5 1Mb 15ms DropTail
$ns duplex-link $node4 $node5 1Mb 15ms DropTail
$ns simplex-link $node5 $node6 128Kb 100ms DropTail
$ns simplex-link $node6 $node5 128Kb 100ms DropTail
$ns duplex-link $node6 $node7 1Mb 15ms DropTail
$ns duplex-link $node6 $node8 1Mb 15ms DropTail
$ns duplex-link $node6 $node9 1Mb 15ms DropTail
$ns duplex-link $node6 $node10 1Mb 15ms DropTail
$ns duplex-link $node6 $node11 1Mb 15ms DropTail

$ns duplex-link-op $node0 $node5 orient down
$ns duplex-link-op $node1 $node5 orient right-down
$ns duplex-link-op $node2 $node5 orient right
$ns duplex-link-op $node3 $node5 orient right-up
$ns duplex-link-op $node4 $node5 orient up
$ns simplex-link-op $node5 $node6 orient right
$ns simplex-link-op $node6 $node5 orient left
$ns duplex-link-op $node6 $node7 orient up
$ns duplex-link-op $node6 $node8 orient right-up
$ns duplex-link-op $node6 $node9 orient right
$ns duplex-link-op $node6 $node10 orient right-down
$ns duplex-link-op $node6 $node11 orient down

$ns queue-limit $node5 $node6 20

$ns color 1 red
$ns color 2 blue
$ns color 3 green
$ns color 4 pink
$ns color 5 black

set sctp0 [new Agent/SCTP]
$ns attach-agent $node0 $sctp0
$sctp0 set fid_ 1
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 160
$cbr0 attach-agent $sctp0
set sctpsink0 [new Agent/SCTP]
$ns attach-agent $node7 $sctpsink0
$ns connect $sctp0 $sctpsink0

set sctp1 [new Agent/SCTP]
$ns attach-agent $node1 $sctp1
$sctp1 set fid_ 2
set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 160
$cbr1 attach-agent $sctp1
set sctpsink1 [new Agent/SCTP]
$ns attach-agent $node8 $sctpsink1
$ns connect $sctp1 $sctpsink1

set sctp2 [new Agent/SCTP]
$ns attach-agent $node2 $sctp2
$sctp2 set fid_ 3
```

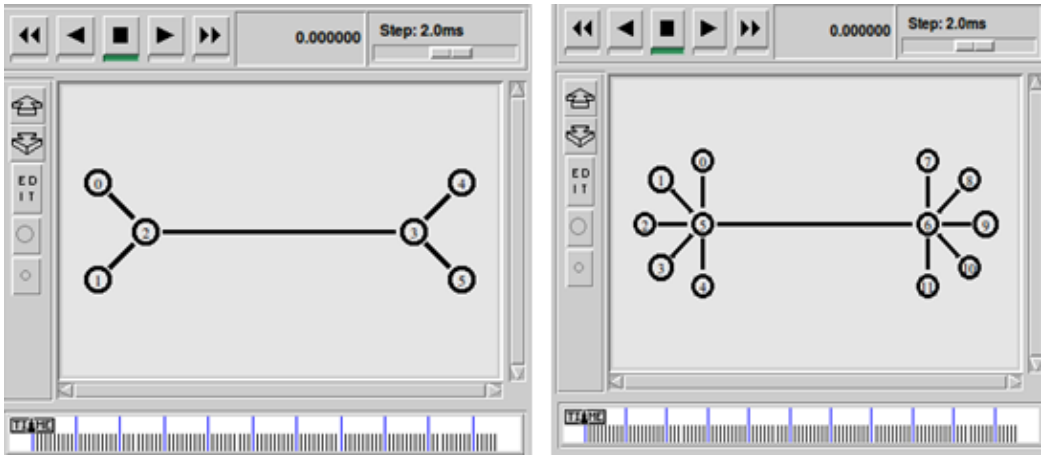
```
set cbr2 [new Application/Traffic/CBR]
$cbr2 set packetSize_ 160
$cbr2 attach-agent $sctp2
set sctpsink2 [new Agent/SCTP]
$ns attach-agent $node9 $sctpsink2
$ns connect $sctp2 $sctpsink2

set sctp3 [new Agent/SCTP]
$ns attach-agent $node3 $sctp3
$sctp3 set fid_ 4
set cbr3 [new Application/Traffic/CBR]
$cbr3 set packetSize_ 160
$cbr3 attach-agent $sctp3
set sctpsink3 [new Agent/SCTP]
$ns attach-agent $node10 $sctpsink3
$ns connect $sctp3 $sctpsink3

set sctp4 [new Agent/SCTP]
$ns attach-agent $node4 $sctp4
$sctp4 set fid_ 5
set cbr4 [new Application/Traffic/CBR]
$cbr4 set packetSize_ 160
$cbr4 attach-agent $sctp4
set sctpsink4 [new Agent/SCTP]
$ns attach-agent $node11 $sctpsink4
$ns connect $sctp4 $sctpsink4

set qmon [$ns monitor-queue $node5 $node6 [open
sctp_topologi.out w] 0.1];
[$ns link $node5 $node6] queue-sample-timeout;

$ns at 0.1 "$cbr0 start"
$ns at 10.0 "$cbr0 stop"
$ns at 0.1 "$cbr1 start"
$ns at 10.0 "$cbr1 stop"
$ns at 0.1 "$cbr2 start"
$ns at 10.0 "$cbr2 stop"
$ns at 0.1 "$cbr3 start"
$ns at 10.0 "$cbr3 stop"
$ns at 0.1 "$cbr4 start"
$ns at 10.0 "$cbr4 stop"

$ns at 10.5 "finish"
$ns run
```

Teks yang tercetak tebal menandakan parameter yang harus diubah-ubah. Besar perubahan *bandwidth* dan *delay propagation* bergantung dari nilai yang telah ditentukan dalam tabel. Sedangkan besar *packet size* bergantung jenis data yang digunakan, 160 *byte* untuk VOIP dan 1300 *byte* untuk IPTV. Simulasi dijalankan dengan mengetikan perintah berikut pada jendela terminal.

```
rinda@rinda:~$ ns SCTP_topologi.tcl
```

Setelah program simulasi dijalankan maka akan didapatkan file nam seperti pada Gambar 4.19.

Dengan 1 kanal Dengan 5 Kanal



Gambar 4.19 Tampilan Nam

Dari hasil nam diatas dapat dilihat topologi dari skrip yang telah di buat. Dengan topologi menggunakan 1 kanal, *node-node* yang dibutuhkan oleh sistem dalam melakukan simulasi yaitu sebanyak 6 *node* dengan bentuk topologi *dumb-bell* standar seperti pada Gambar 4.19. aliran dari *node* 0 akan membawa data VOIP sedang aliran data pada node 1 akan membawa data IPTV. Saat menjalankan program dengan data VOIP, maka aliran data IPTV dihentikan, dan begitu sebaliknya.

Pada hasil simulasi menggunakan 5 kanal, *node-node* yang dibutuhkan dalam melakukan proses simulasi menggunakan *node* berjumlah 12 *node*. *Node* 0,1,2,3,4 mengalirkan data cbr dengan besarnya ukuran paket bergantung dari jenis data yang di gunakan.

Simulasi berjalan saat tombol *play* di jalankan dan menghasilkan out.tr yang merekam seluruh kejadian dari pengiriman *node* sumber ke *node* tujuan,

selain itu simulasi juga menghasilkan qm.out yang merupakan informasi kejadian pada jalur *bottleneck*. Selanjutnya dilakukan *parsing* file dengan menggunakan skrip perl *delay*, *queue* dan *packet loss* berikut:

3. Skrip perl untuk *delay* sesuai *flowchart* pada Gambar 3.9:

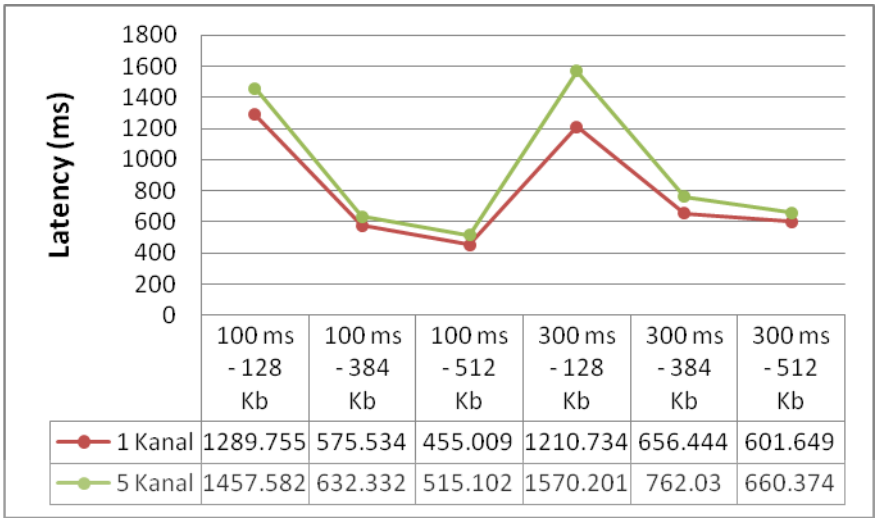
```
.# type: perl delay.pl <trace file> > output file
$infile=$ARGV[0];
open (DATA,"<$infile") || die "Can't open $infile $!";
while (<DATA>)
{
 @x = split(' ');
 @SCTP0;
 @SCTP4;
 @hsl;
 if ($x[6] eq '-------')
 {
    if ($x[4] eq 'SCTP')
    {
          if ($x[0] eq '+')
          {
                if ($x[2] eq '0')
                {
                   $SCTP0[$x[11]]=$x[1];
                }
          }
          if ($x[0] eq 'r')
          {
                if ($x[3] eq '4')
                {
                   $SCTP4[$x[11]]=$x[1];
              $hsl[$x[11]]=$SCTP4[$x[11]]-$SCTP0[$x[11]];
                   print STDOUT "$x[10] $hsl[$x[11]] \n";
                }
          }
     }
  }
}
close DATA;
exit(0);
```

Dari skrip perl tersebut dapat dipanggil dengan perintah berikut pada terminal:

```
rinda@rinda:~$ perl delay.pl sctp_topologi.tr > delay
```

Dari data *delay* yang diperoleh dengan melakukan *parsing* menggunakan perl, data tersebut diolah kembali untuk menghasilkan nilai *latency* dan *jitter*

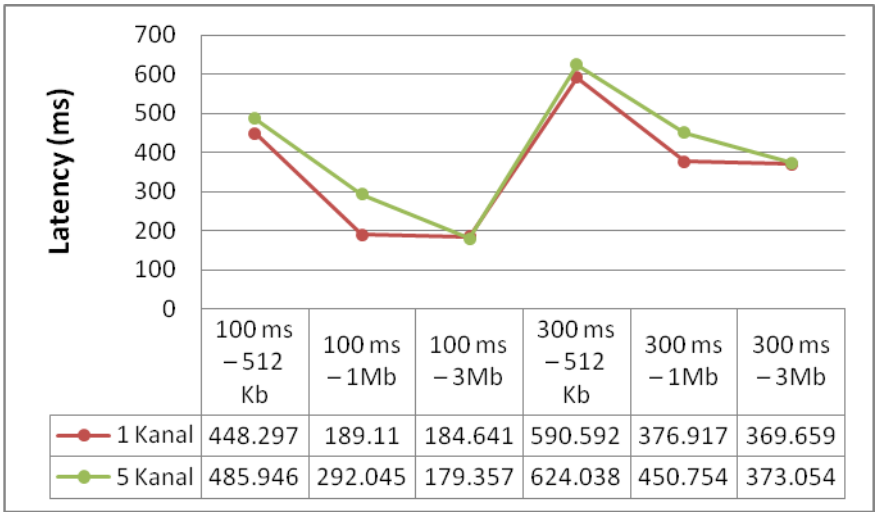
dengan menggunakan rumus yang telah di jelaskan pada bab sebelumnya menggunakan *microsoft excel* 2007. Data ini kemudian ditampilkan menggunakan grafik untuk lebih memperjelas dalam melakukan analisis.



	100 ms - 128 Kb	100 ms - 384 Kb	100 ms - 512 Kb	300 ms - 128 Kb	300 ms - 384 Kb	300 ms - 512 Kb
1 Kanal	1289.755	575.534	455.009	1210.734	656.444	601.649
5 Kanal	1457.582	632.332	515.102	1570.201	762.03	660.374

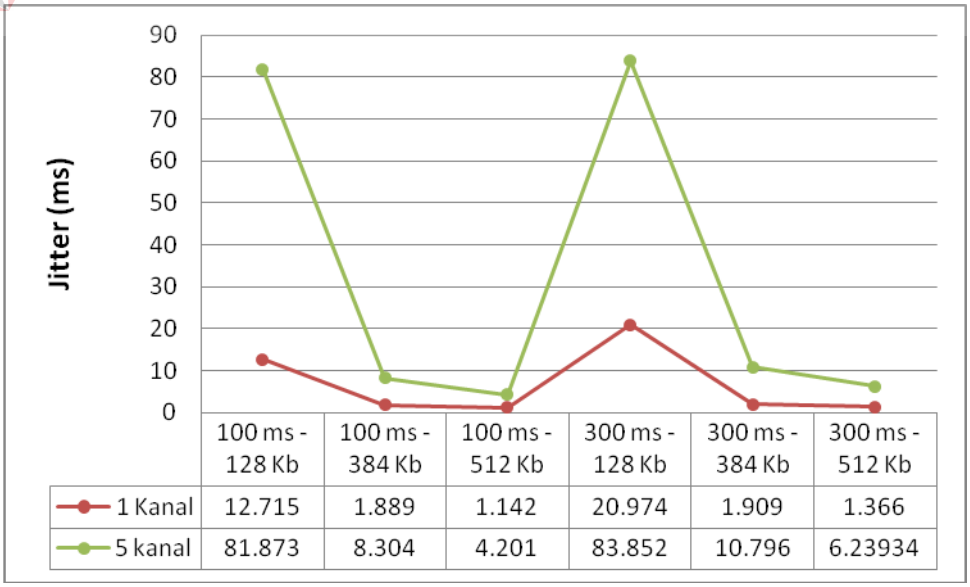
Gambar 4.20 Besar *Latency* Data VOIP

Gambar 4.20 menunjukan besar *latency* dari protokol SCTP dengan menggunakan layanan dari data VOIP. Dari grafik tersebut dapat dilihat bahwa dengan menggunakan 1 kanal, *latency* TCP dengan *bandwidth* 128 Kb dan *delay propagation* 100 ms merupakan *latency* yang paling besar nilainya. Sedangkan pada 5 kanal *latency* terbesar berada pada *bandwidth* 128 Kb dan *delay propagation* 300 ms.



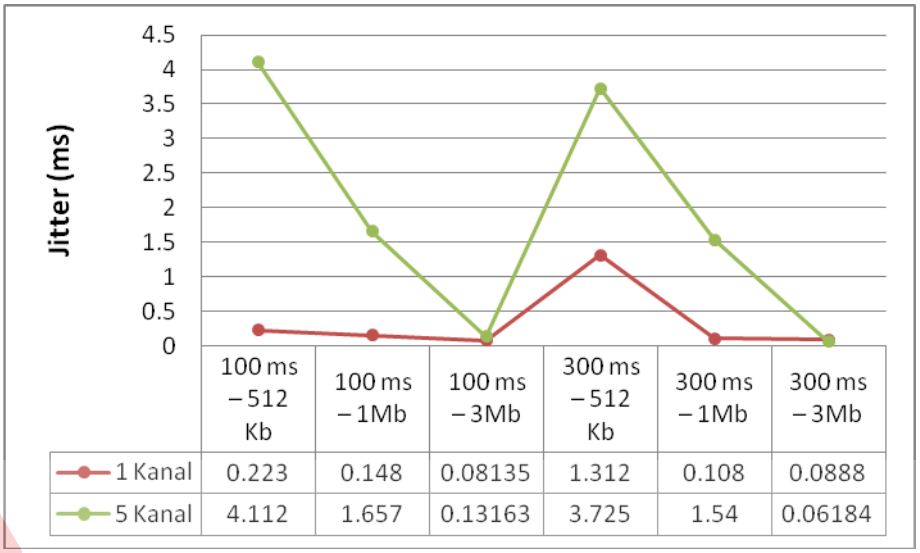
Gambar 4.21 Besar *Latency* Data IPTV

Gambar 4.21 menunjukan besar *latency* dari protokol SCTP dengan menggunakan layanan dari data IPTV. Dari grafik tersebut dapat dilihat bahwa dengan menggunakan 1 kanal dan 5 kanal latency SCTP dengan *bandwidth* 512 Kb dan *delay propagation* 300 ms merupakan *latency* yang paling besar nilainya dibanding dengan yang lain.



Gambar 4.22 Besar *Jitter* Data VOIP

Pada Gambar 4.22 didapatkan bahwa besar nilai variasi *delay* atau *jitter* dari TCP yang menggunakan layanan data VOIP dengan 1 kanal maupun 5 kanal terbesar berada pada *bandwidth* 128 Kb dan *delay propagation* 300 ms.



	100 ms – 512 Kb	100 ms – 1Mb	100 ms – 3Mb	300 ms – 512 Kb	300 ms – 1Mb	300 ms – 3Mb
1 Kanal	0.223	0.148	0.08135	1.312	0.108	0.0888
5 Kanal	4.112	1.657	0.13163	3.725	1.54	0.06184

Gambar 4.23 Besar *Jitter* Data IPTV

Pada Gambar 4.23 didapatkan bahwa besar nilai variasi *delay* atau *jitter* dari TCP yang menggunakan layanan data IPTV dengan 1 kanal terbesar berada pada *bandwidth* 512 Kb dan *delay propagation* 300 ms, sedangkan pada 5 kanal *jitter* terbesar berada pada *bandwidth* 512 Kb dan *delay propagation* 100 ms.

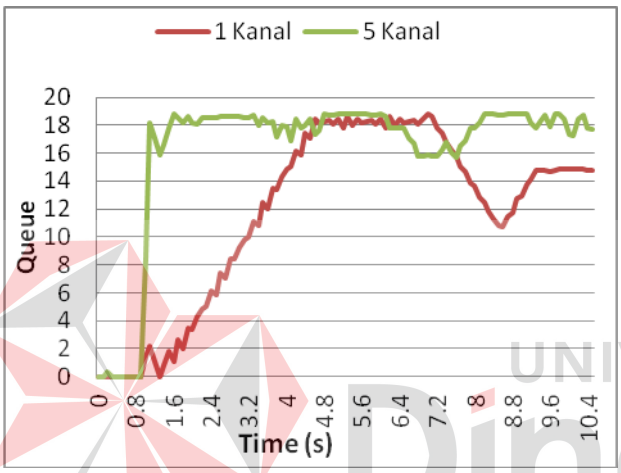
4. Skrip perl untuk *queue* sesuai *flowchart* pada Gambar 3.10:

```
# perl queue.pl  <sctp_topologi.out>  > queue
$infile=$ARGV[0];
open (DATA,"<$infile") || die "Can't open $infile $!";
while (<DATA>)
{
 @x = split(' ');
 print STDOUT "$x[0]  $x[4]\n";

}
close DATA;
exit(0);
```

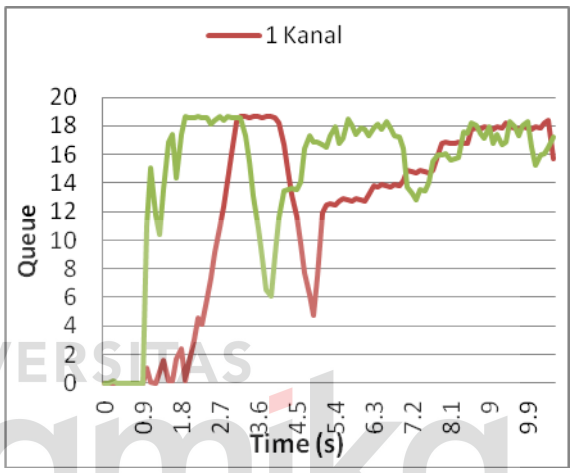
Dari skrip perl tersebut dapat dipanggil dengan perintah berikut pada terminal:

```
rinda@rinda:~$ perl queue.pl sctp_topologi.tr > queue
```

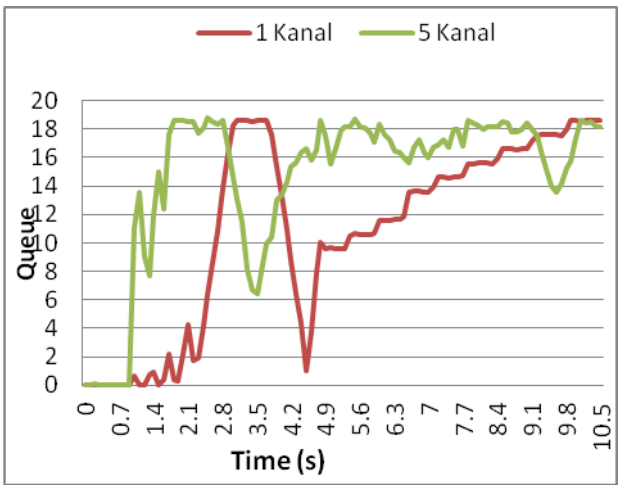
Data hasil *queue* yang telah di dapatkan kemudian digambarkan dalam bentuk grafik menggunakan *microsoft excel* 2007. Grafik ini di buat untuk lebih mempermudah dalam melakukan analisis.



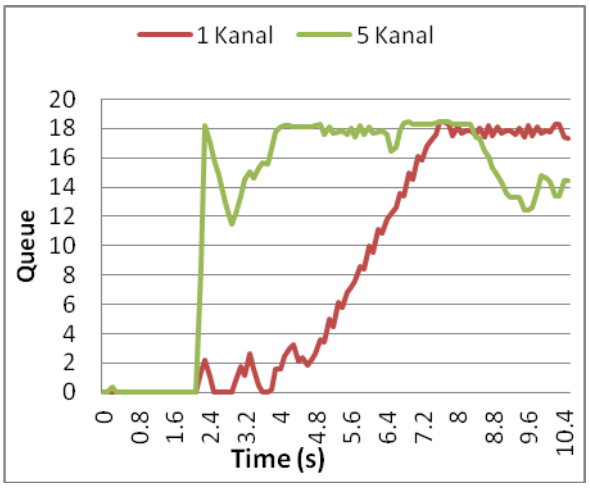
a. *Queue* VOIP 100ms – 128 Mb



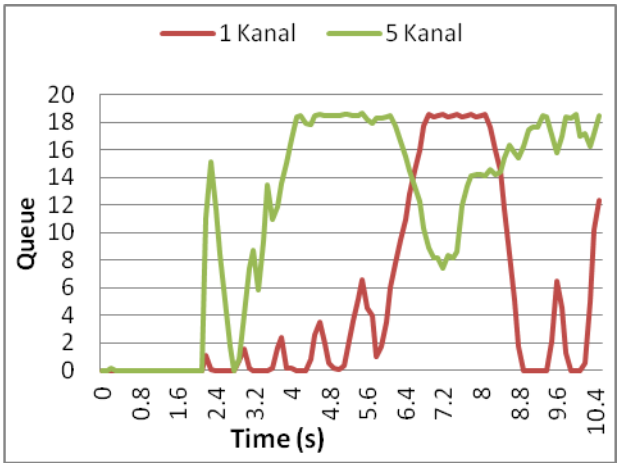
b. *Queue* VOIP 100ms – 384 Kb



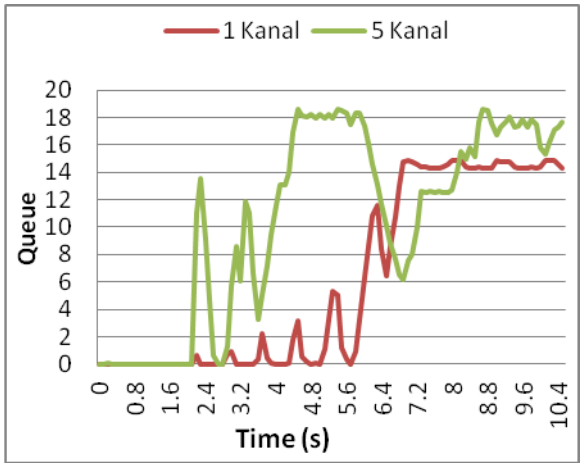
c. *Queue* VOIP 100ms – 512 Mb



b. *Queue* VOIP 300ms – 128 Kb



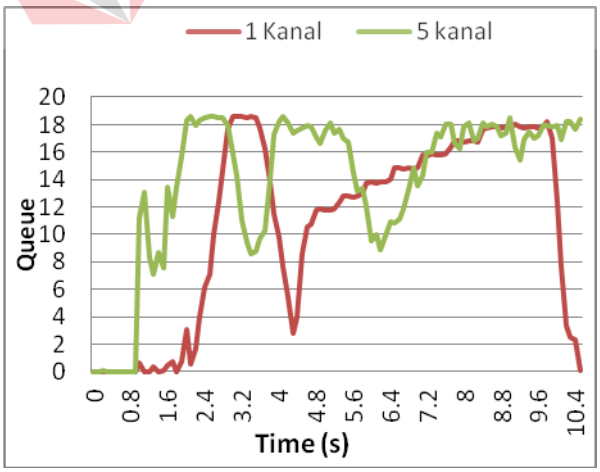
d. *Queue* VOIP 300ms – 384 Mb



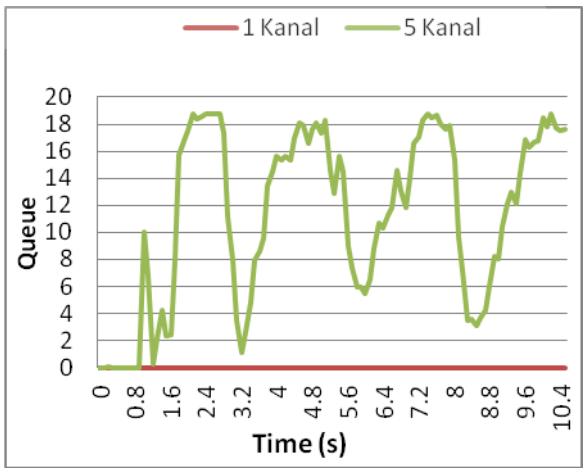
b. *Queue* VOIP 300ms – 512 Kb

Gambar 4.24 *Queue* Data VOIP

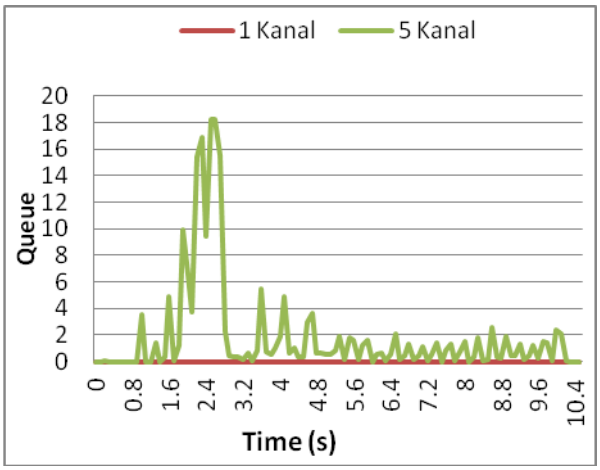
Pada Gambar 4.24 menunjukan nilai *queue* yang di dapat terhadap waktu (*time*) dari protokol SCTP dengan data VOIP. Grafik yang naik menunjukan bahwa kapasitas antrian yang terpakai oleh sistem pada jalur *bottleneck* mengalami kenaikan untuk rentang waktu tersebut. *Queue* maksimum yang telah di seting adalah sebesar 20.



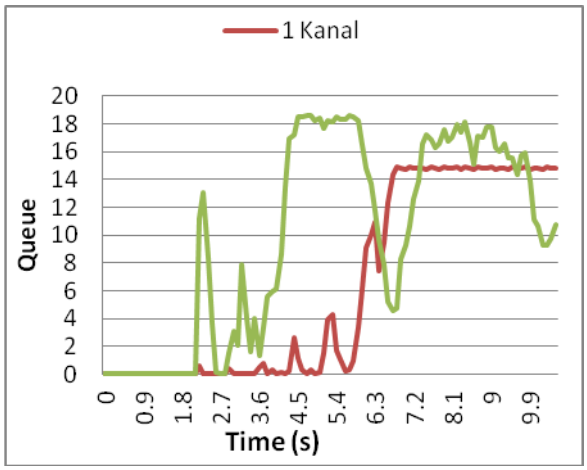
a. *Queue* IPTV 100ms – 512 Kb



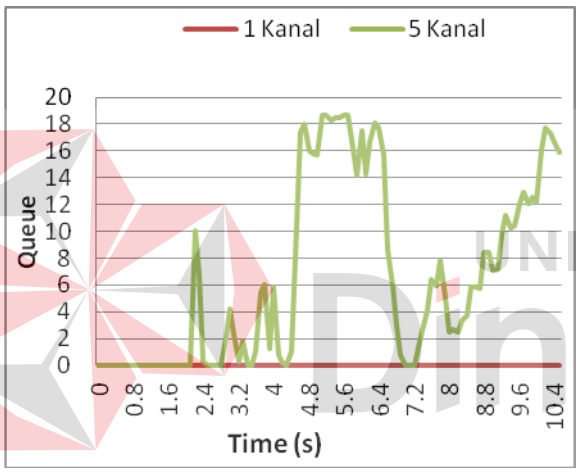
b. *Que*ue IPTV 100ms – 1 Mb



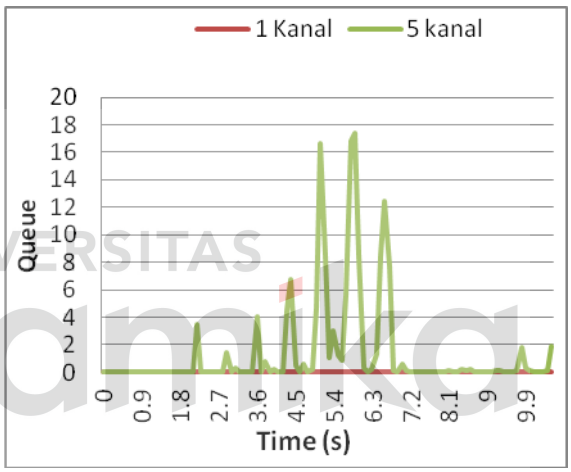
c. *Queue* IPTV 100ms – 3 Mb



d. *Queue* IPTV 300ms – 512 Kb



e. *Queue* IPTV 300ms – 1 Mb



f. *Queue* IPTV 300ms – 3Mb

Gambar 4.25 *Queue* Data IPTV

Pada Gambar 4.25 menunjukan nilai *queue* yang di dapat terhadap waktu (*time*) dari protokol SCTP dengan data IPTV. Grafik yang naik menunjukan bahwa kapasitas antrian yang terpakai oleh sistem pada jalur *bottleneck* mengalami kenaikan untuk rentang waktu tersebut. *Queue* maksimum yang telah di seting adalah sebesar 20.

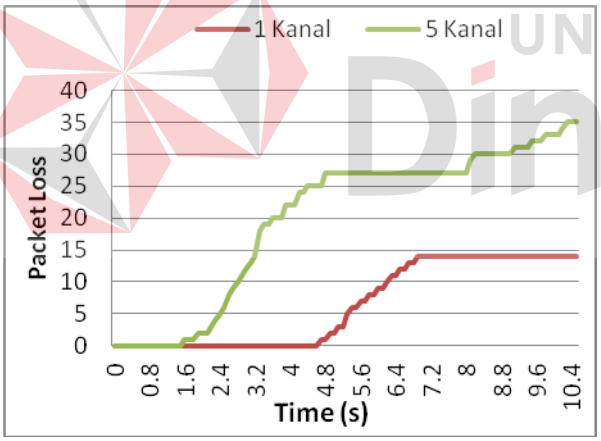
5. Skrip perl untuk *packet loss* sesuai *flowchart* pada Gambar 3.11:

```
# perl packetloss.pl <sctp_topologi.out> > loss
$infile=$ARGV[0];
open (DATA,"<$infile") || die "Can't open $infile $!";
while (<DATA>)
{
 @x = split(' ');
 print STDOUT "$x[0] $x[5] $x[7]\n";

}
close DATA;
exit(0);
```

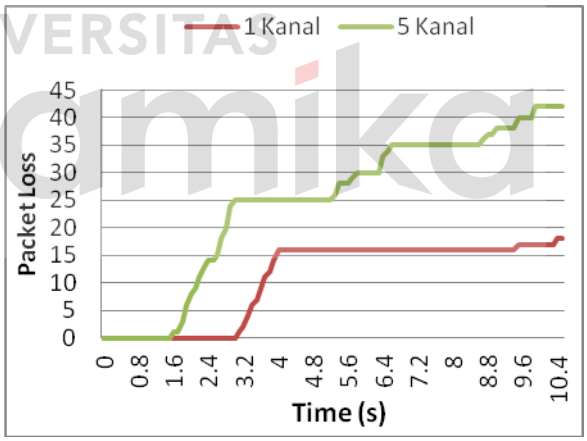
Dari skrip perl tersebut dapat dipanggil dengan perintah berikut pada terminal:

```
rinda@rinda:~$ perl packetloss.pl sctp_topologi.out > loss
```

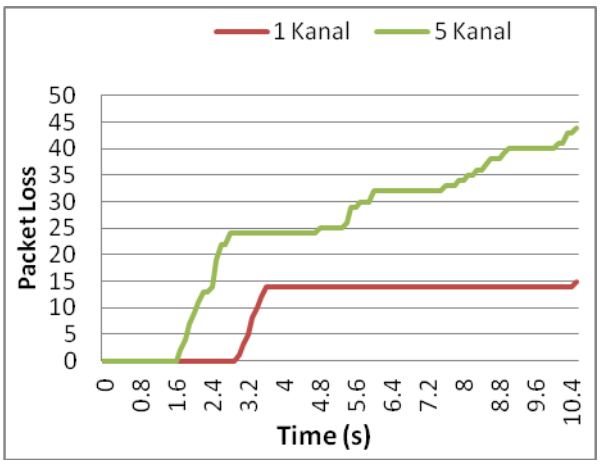
Data hasil *packet loss* yang telah di dapatkan kemudian digambarkan dalam bentuk grafik menggunakan *microsoft excel* 2007. Grafik ini di buat untuk lebih mempermudah dalam melakukan analisis.

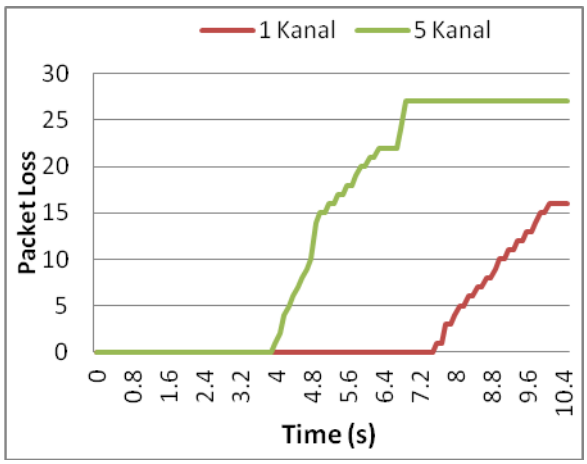


a. *Packet Loss* VOIP 100ms – 128 Kb

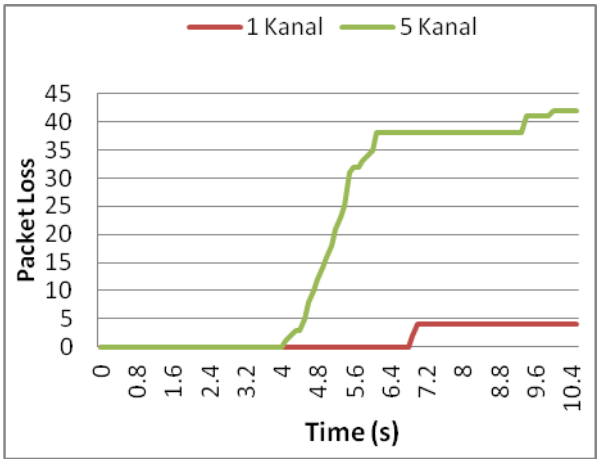


b. *Packet Loss* VOIP 100ms –384 Kb

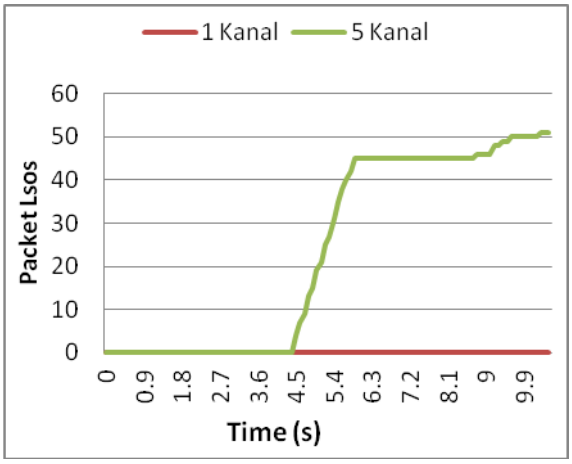




c. *Packet Loss* VOIP 100ms – 128 Kb



d. *Packet Loss* VOIP 100ms –384 Kb

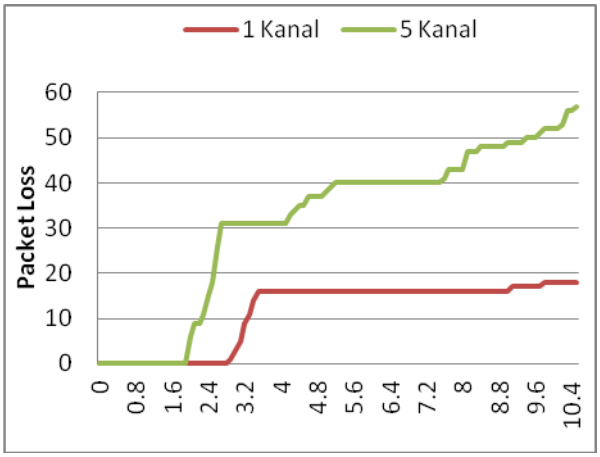


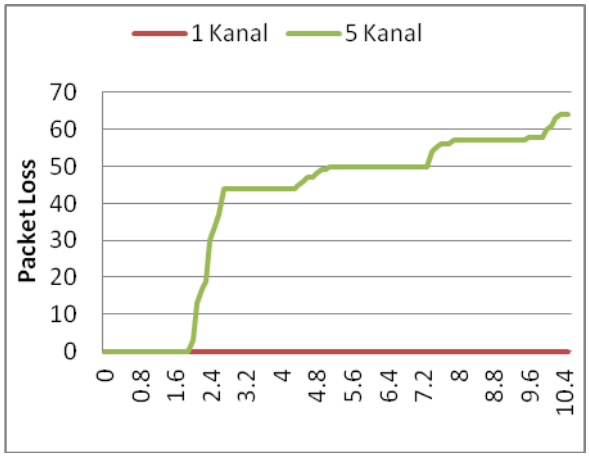
e. *Packet Loss* VOIP 100ms – 128 Kb

f. *Packet Loss* VOIP 100ms –384Kb

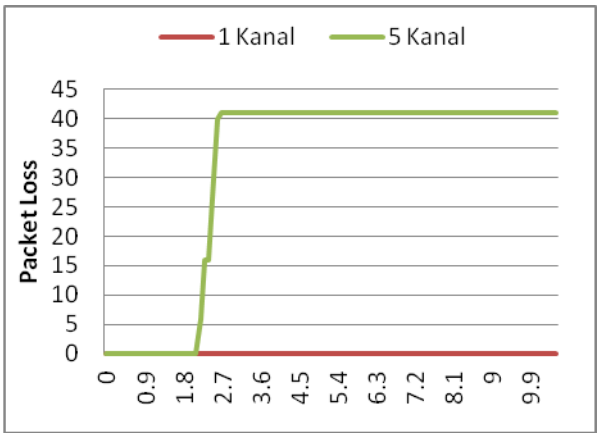
Gambar 4 26 *Packet Loss* Data VOIP

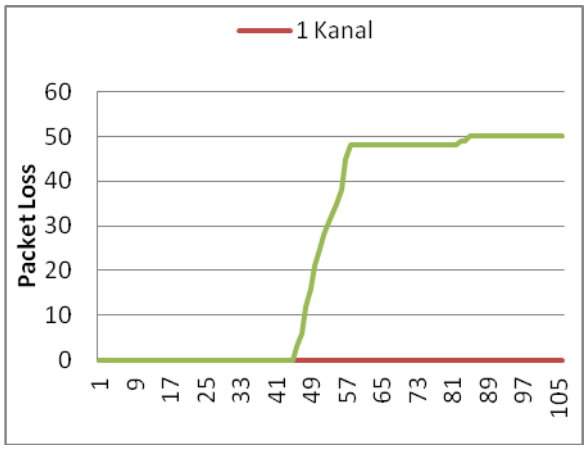
Pada Gambar 4.26 menunjukan nilai *packet loss* yang di dapat terhadap waktu (*time)* dari protokol SCTP dengan data VOIP. Data *packet loss* bersifat kumulatif. Dari grafik yang ditampilkan menunjukan protokol UDP dengan menggunakan 5 kanal mengalami kenaikan yang lebih tinggi di banding dengan menggunakan 1 kanal. Hal ini di sebabkan karena jalur *bottleneck* pada antrian digunakan oleh 5 data UDP dalam satu waktu.



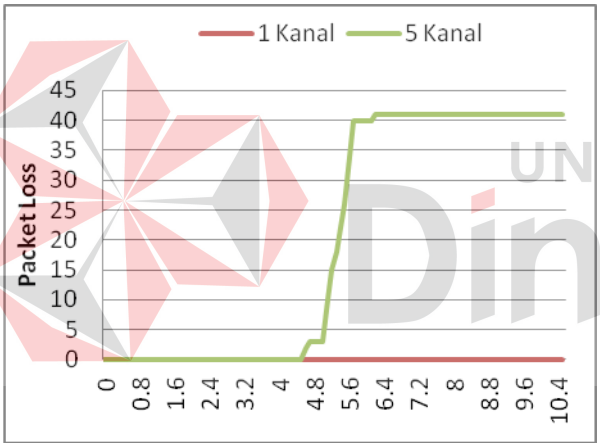


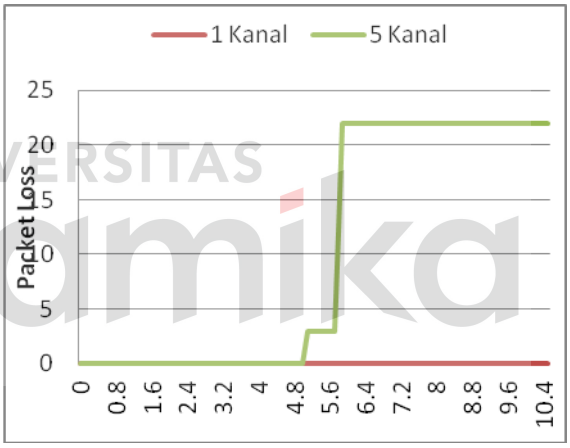
a. *Packet Loss* IPTV 100ms – 512Kb b. *Packet Loss* IPTV 100ms – 1 Mb





c. *Packet Loss* IPTV 100ms – 3 Mb d. *Packet Loss* IPTV 300ms –512 Kb





e. *Packet Loss* IPTV 300ms – 1 Mb f. *Packet Loss* IPTV 300ms –3 Mb

Gambar 4.27 *Packet Loss* Data IPTV

Pada Gambar 4.27 menunjukan nilai *packet loss* yang di dapat terhadap waktu (*time*) dari protokol SCTP dengan data IPTV. Data *packet loss* bersifat kumulatif. Dari grafik yang ditampilkan menunjukan protokol UDP dengan menggunakan 5 kanal mengalami kenaikan yang sangat tinggi di banding dengan menggunakan 1 kanal. Hal ini di sebabkan karena jalur *bottleneck* pada antrian digunakan oleh 5 data UDP dalam satu waktu, sehingga trafik akan penuh yang

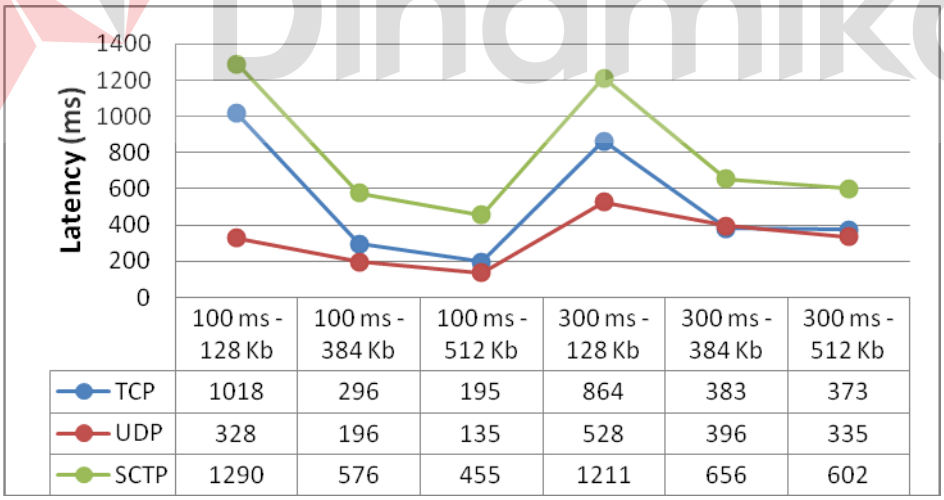
menyebabkan kapasitas *queue* meningkat sehingga tidak bisa menampung data yang masuk berikutnya

4.3 Hasil dan Pembahasan

Dari hasil simulasi akan dianalisis dan dibahas berdasarkan 4 parameter uji perbandingan yang telah ditetapkan yaitu *latency, jitter, packet loss* dan *queue*. Hasil masing-masing parameter disajikan dalam satu grafik untuk mengetahui perbandingan secara langsung, serta dilakukan pembahasan berdasarkan teori-teori yang mendukung. Analisis perbandingan dapat dilakukan melalui data yang berupa angka dan ditampilkan dalam grafik sehingga memudahkan kita dalam melakukan perbandingan unjuk kerja.

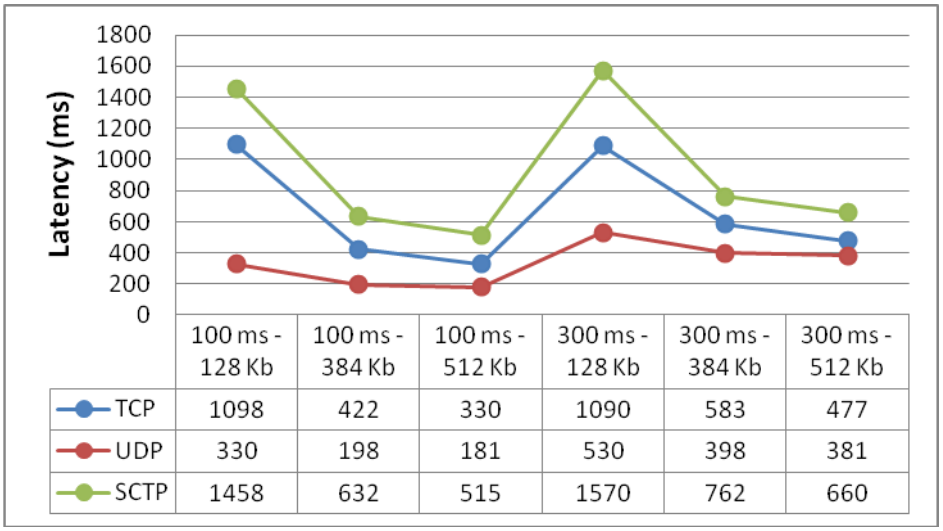
4.3.1 Analisis Menggunakan Data VOIP

1. Analisis *Latency*



	100 ms - 128 Kb	100 ms - 384 Kb	100 ms - 512 Kb	300 ms - 128 Kb	300 ms - 384 Kb	300 ms - 512 Kb
TCP	1018	296	195	864	383	373
UDP	328	196	135	528	396	335
SCTP	1290	576	455	1211	656	602

Gambar 4.28 *Latency* VOIP 1 Kanal



	100 ms - 128 Kb	100 ms - 384 Kb	100 ms - 512 Kb	300 ms - 128 Kb	300 ms - 384 Kb	300 ms - 512 Kb
TCP	1098	422	330	1090	583	477
UDP	330	198	181	530	398	381
SCTP	1458	632	515	1570	762	660

Gambar 4.29 *Latency* VOIP 5 Kanal

Pada perbandingan unjuk kerja TCP, UDP dan SCTP terhadap *latency*, enam simulasi di jalankan untuk masing-masing protokol dan masing-masing jumlah kanal. Jumlah kanal digunakan untuk menunjukan kondisi jaringan saat data di jalankan tanpa adanya kongesti dengan menggunakan 1 kanal, dan dengan 5 kanal akan menunjukan kinerja dari jaringan saat jalur *bottleneck* harus berbagi data dengan yang lain.

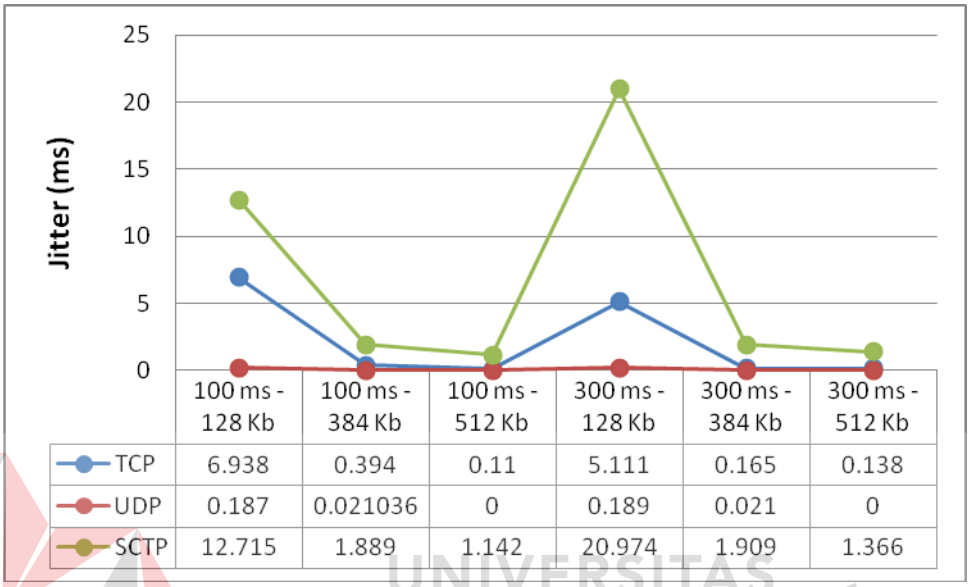
Ketika menggunakan jalur data dengan 1 kanal maupun 5 kanal seperti ditunjukan pada Gambar 4.28 dan Gambar 4.29, unjuk kerja UDP menghasilkan *latency* yang paling kecil. Jumlah *latency* yang kecil menunjukan sedikitnya waktu rata-rata pengiriman trafik data dari node sumber hingga sampai ke node tujuan. Semakin kecil *latency* akan semakin cepat data di kirim. L*atency* pada protokol TCP mempunyai nilai yang lebih besar dibanding pada protokol UDP, hal ini di karenakan oleh adanya mekanisme *three-way-handshake* pada proses inisialisasi koneksi. Di samping itu, ketika jaringan padat (kongesti sangat tinggi) menyebabkan adanya *time-out* dan TCP membutuhkan mekanisme *retransmisi*

sehingga *latency* pada TCP menjadi tinggi. (Sapura & Pramartha, n.d). *Latency* dari protokol SCTP pada Gambar 4.28 dan 4.29 menunjukan nilai terbesar pada jalur 1 kanal maupun 5 kanal. Ini karena pada SCTP menggunakan sistem pengiriman *best efford*, yang berarti bahwa SCTP akan berupaya mengirimkan data antar pengirim dan penerima dengan baik tetapi tidak menjamin urutan dan integritas data dari segmen yang dikirim. Sistem *best efford* sama seperti yang terjadi pada protokol IP. Selain karena hal itu, besarnya *latency* pada SCTP disebabkan juga karena pengiriman *acknowledgement* dari setiap paket SCTP membutuhkan waktu yang lama sehingga secara keseluruhan *delay* pada SCTP menjadi lebih tinggi. (Gangurde, et al., 2012). Untuk layanan data VOIP, *latency* berpengaruh terhadap kualitas layanan suara yang dihasilkan, sehingga dalam parameter *latency*, protokol UDP lah yang mempunyai nilai terkecil dari kedua protokol yang lain.

Menurut Gonia (2004) *latency* yang melebihi batas 500 ms dalam VOIP tidak bisa diterima, karena dengan penundaan diatas 500 ms akan terlihat seperti menanyakan pertanyaan yang tidak terdengar jawaban selama beberapa jam. Dari hasil menurut Gambar 4.28, *latency* pada protokol TCP tidak bisa diterima saat kondisi *bandwidth* yang diberikan bernilai 128 Kb, serta *latency* pada protokol UDP tidak bisa diterima saat kondisi *bandwidth* 128 Kb dengan *delay propagation* 300 ms, sedangkan *latency* pada protokol SCTP yang dapat diterima hanya saat kondisi *bandwidth* 512 Kb dan *delay propagation* 100 ms. Pada Gambar 4.29 yang merupakan hasil *latency* saat menggunakan 5 kanal, *latency* pada protokol TCP yang bisa diterima saat kondisi *bandwidth* 512 Kb, sedangkan *latency* pada protokol UDP yang tidak bisa diterima saat *bandwith* 128 Kb dengan

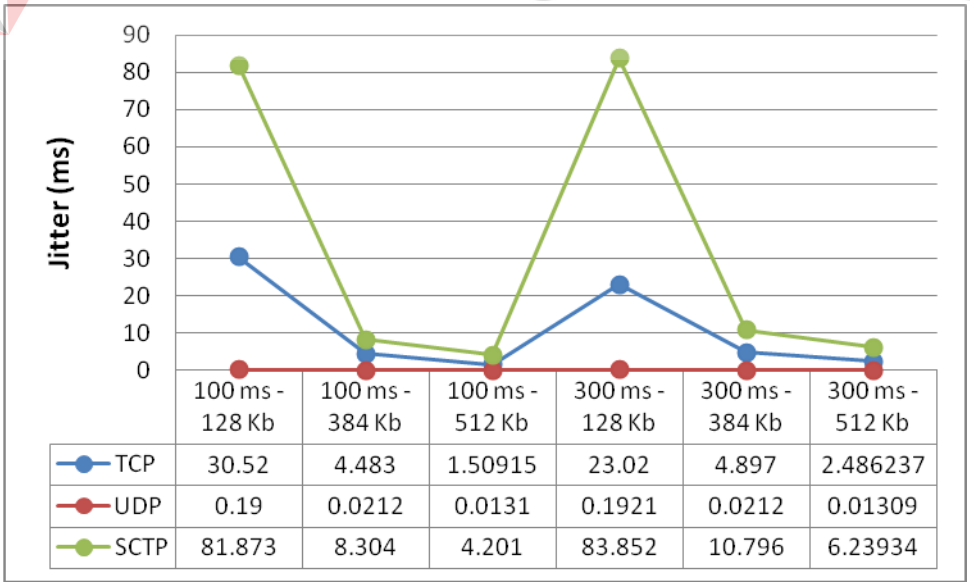
delay propagation 300 ms, dan *latency* pada protokol SCTP dengan keseluruhan kondisi tidak bisa diterima.

2. Analisis *Jitter*



	100 ms - 128 Kb	100 ms - 384 Kb	100 ms - 512 Kb	300 ms - 128 Kb	300 ms - 384 Kb	300 ms - 512 Kb
TCP	6.938	0.394	0.11	5.111	0.165	0.138
UDP	0.187	0.021036	0	0.189	0.021	0
SCTP	12.715	1.889	1.142	20.974	1.909	1.366

Gambar 4.30 *Jitter* VOIP 1 Kanal



	100 ms - 128 Kb	100 ms - 384 Kb	100 ms - 512 Kb	300 ms - 128 Kb	300 ms - 384 Kb	300 ms - 512 Kb
TCP	30.52	4.483	1.50915	23.02	4.897	2.486237
UDP	0.19	0.0212	0.0131	0.1921	0.0212	0.01309
SCTP	81.873	8.304	4.201	83.852	10.796	6.23934

Gambar 4 31 *Jitter* VOIP 5 Kanal

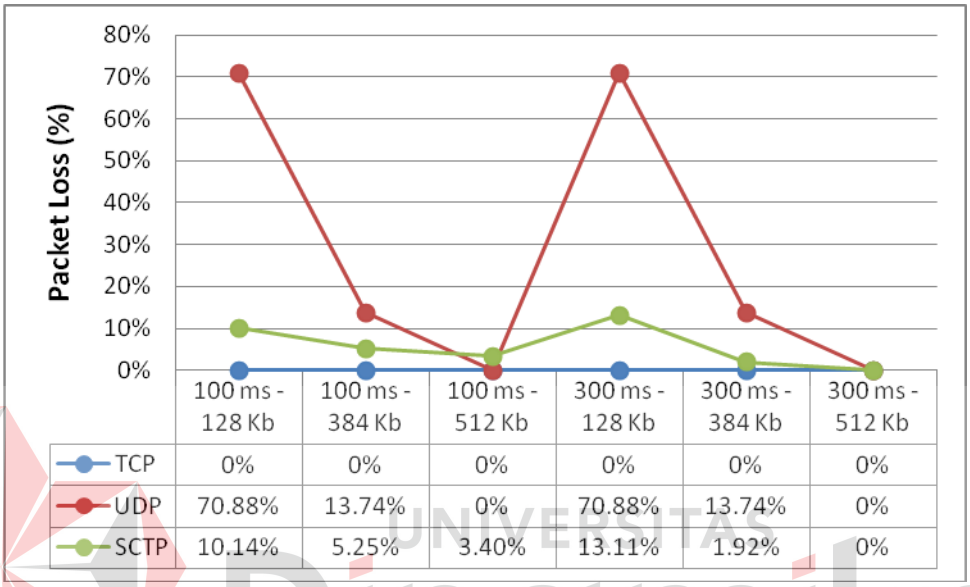
Pada perbandingan unjuk kerja TCP, UDP dan SCTP terhadap *jitter*, enam simulasi di jalankan untuk masing-masing protokol dan masing-masing jumlah kanal. Jumlah kanal digunakan untuk menunjukan kondisi jaringan saat data di jalankan tanpa adanya kongesti dengan menggunakan 1 kanal, dan dengan 5 kanal akan menunjukan kinerja dari jaringan saat jalur *bottleneck* harus berbagi data dengan yang lain.

Jitter lazimnya disebut variasi *delay*, berhubungan erat dengan *latency*, yang menunjukkan banyaknya variasi *delay* pada transmisi data di jaringan. (Nurhayati, n.d). Pada *jitter* dengan 1 kanal dan 5 kanal mengahasilkan analisis yang sama yaitu *jitter* yang paling besar adalah protokol SCTP. Pada protokol SCTP, variasi *delay* terendah berada saat kondisi *bandwidth* 512 Kb dan *delay propagation* 100 ms, sedangkan *jitter* tertinggi terjadi saat kondisi *bandwidth* 128 Kb dan *delay propagation* 300 ms. Rata-rata *jitter* tertinggi kedua yaitu pada protokol TCP. *Jitter* terendah pada TCP, terjadi saat *bandwidth* 512 Kb dan *delay propagation* 100 ms, sedangkan *jitter* tertinggi pada TCP terjadi saat kondisi *bandwidth* 128 Kb dan *delay propagation* 100 ms. Pada protokol UDP, *jitter* dipengaruhi oleh perubahan *bandwidth*, sedangkan *delay propagation* tidak begitu signifikan dalam menentukan nilai dari *jitter* pada UDP sehingga terlihat dengan *bandwidth* yang berbeda menghasilkan *jitter* dengan nilai yang berbeda.

Menurut referensi yang di dapat *peak jitter* yang bagus terletak anatara 0ms - 20ms, untuk *range* sedang yaitu 20ms - 50ms dan yang terburuk yaitu jika nilai *jitter* > 50ms. (Nurhayati, n.d). Dari referensi tersebut dapat disimpulkan bahwa dengan satu kanal pada Gambar 4.30 , *jitter* pada protokol TCP, UDP dan SCTP masi bisa diterima untuk membawa layanan VOIP, namun pada

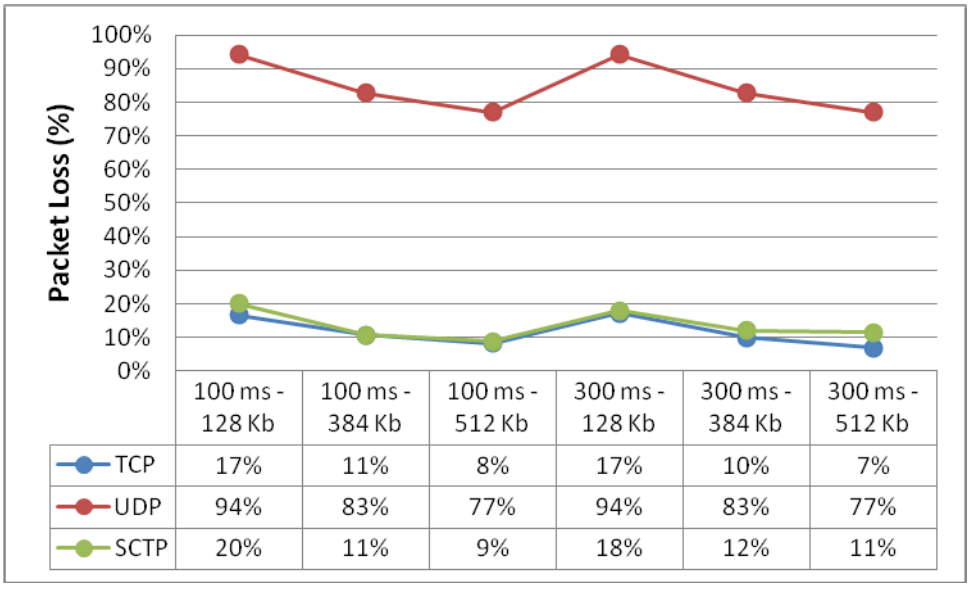
pengiriman dengan 5 kanal yang terlihat pada Gambar 4.31 , hanya *jitter* pada protokol SCTP dengan *bandwidth* 128 Kb tidak cocok untuk digunakan dalam pengiriman data VOIP sesuai dengan nilai standar.

3. Analisis *Paket Loss*



	100 ms - 128 Kb	100 ms - 384 Kb	100 ms - 512 Kb	300 ms - 128 Kb	300 ms - 384 Kb	300 ms - 512 Kb
TCP	0%	0%	0%	0%	0%	0%
UDP	70.88%	13.74%	0%	70.88%	13.74%	0%
SCTP	10.14%	5.25%	3.40%	13.11%	1.92%	0%

Gambar 4 32 *Packet loss* Pada VOIP 1 Kanal



	100 ms - 128 Kb	100 ms - 384 Kb	100 ms - 512 Kb	300 ms - 128 Kb	300 ms - 384 Kb	300 ms - 512 Kb
TCP	17%	11%	8%	17%	10%	7%
UDP	94%	83%	77%	94%	83%	77%
SCTP	20%	11%	9%	18%	12%	11%

Gambar 4.33 *Packet loss* Pada VOIP 5 Kanal

Pada perbandingan unjuk kerja TCP, UDP dan SCTP terhadap *packet loss*, enam simulasi di jalankan untuk masing-masing protokol dan masing-masing jumlah kanal. Jumlah kanal digunakan untuk menunjukan kondisi jaringan saat data di jalankan tanpa adanya kongesti dengan menggunakan 1 kanal, dan dengan 5 kanal akan menunjukan kinerja dari jaringan saat jalur *bottleneck* harus berbagi data dengan yang lain.

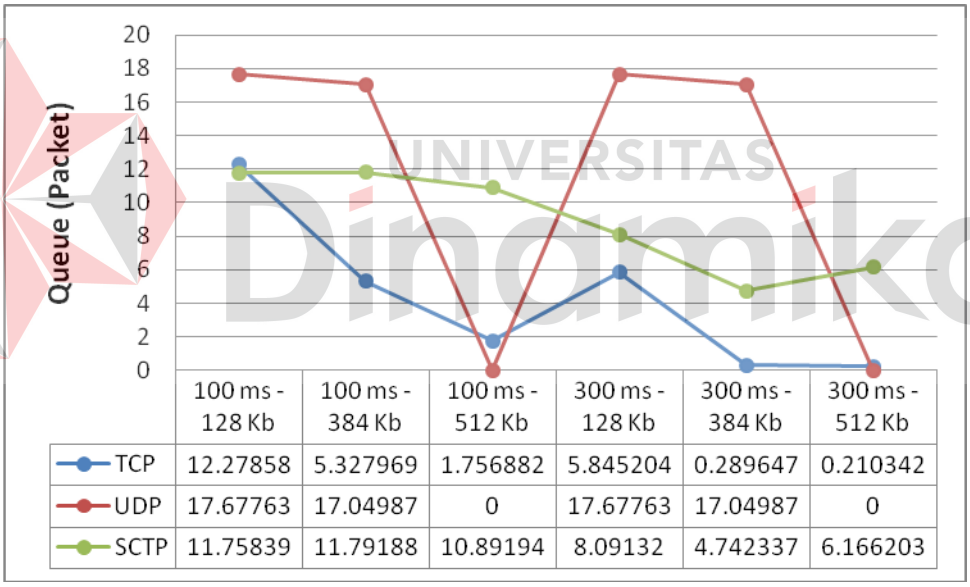
Pada percobaan mengggunakan satu kanal terlihat pada Gambar 4.32 bahwa TCP tidak mengalami *packet loss* pada semua kondisi yang telah di tentukan. Hal ini dikarenakan protokol TCP mempunyai *congestion control* yang dapat membatasi pengiriman saat jalur antrian mengalami kepadatan. Sedangkan pada UDP, dengan *bandwidth* yang rendah yaitu 128 Kb, data mengalami banyak *packet loss*, karena pada pengiriman dengan protokol UDP tidak terdapat *control congesti* yang berakibat paket-paket akan dikirimkan tanpa perlu dibatasi saat kapasitas *queue* sedang mengalami kepadatan. *Packet loss* berhubungan dengan kapasitas *queue* yang digunakan oleh protokol dalam mengirimkan data. Saat pemakaian *queue* tinggi, maka intesitas kehilangan paket juga akan tinggi.

Pada Gambar 4.33, pemakaian jalur *bottleneck* dibuat untuk *link* data dari 5 node sumber ke 5 node tujuan. Pada jalur ini, tidak bisa diketahui pengiriman paket yang datang berasal dari node mana dengan tujuan yang mana, sehingga *packet loss* pada jalur ini meningkat sangat banyak dikarenakan kapasitas *queue* juga mengalami peningkatan, karena pada jalur *bottleneck*, trafik yang dikirimkan harus berbagi tempat antrian yang cukup banyak antara node-node tujuan.

Dari Gambar 4.32 dan Gambar 4.33 terlihat bahwa dengan *bandwidth* yang lebih besar menghasilkan *packet loss* yang lebih lebih kecil. Saat pengiriman

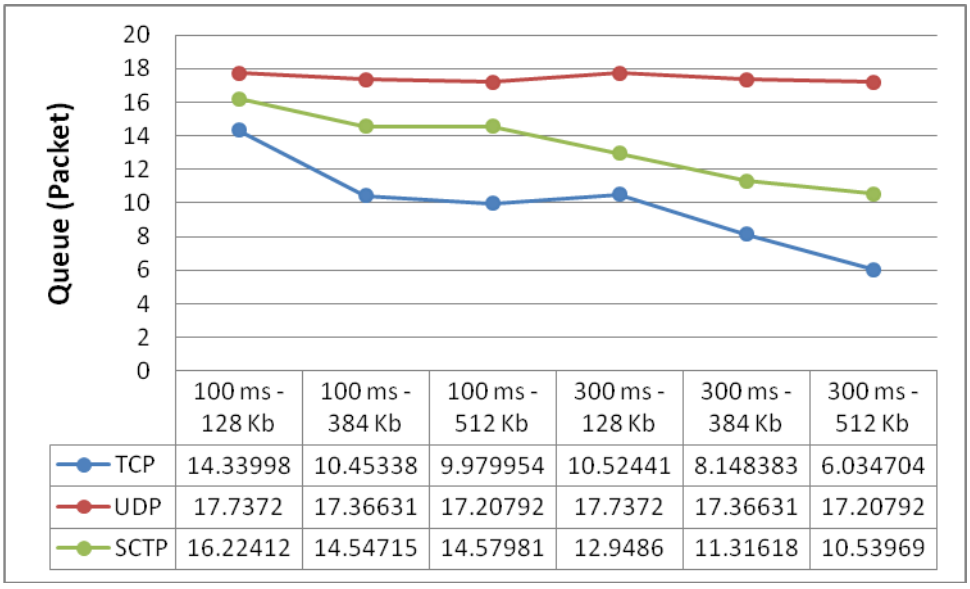
data menggunakan VOIP, menurut gonia (2004) kualitas percakapan dengan VOIP akan tertinggal jika *packet loss* mencapai 5 %. Dari hasil Gambar 4.32 menggunakan satu kanal, *packet loss* pada protokol TCP dengan semua kondisi bisa diterima, sedangkan pada potokol UDP *packet loss* yang bisa diterima saat kondisi *bandwidth* 512 Kb, dan pada protokol SCTP *packet loss* yang bisa diterima saat *delay propagatin* 300 ms dengan *bandwidth* masing-masing 384 Kb dan 512 Kb. Pada hasil Gambar 4.33, *packet loss* di semua protokol dengan semua kondisi tidak bisa diterima karena melebihi batas standar untuk percakapan VOIP.

4. Analisis *Queue*



	100 ms - 128 Kb	100 ms - 384 Kb	100 ms - 512 Kb	300 ms - 128 Kb	300 ms - 384 Kb	300 ms - 512 Kb
TCP	12.27858	5.327969	1.756882	5.845204	0.289647	0.210342
UDP	17.67763	17.04987	0	17.67763	17.04987	0
SCTP	11.75839	11.79188	10.89194	8.09132	4.742337	6.166203

Gambar 4.34 Rata-Rata *Queue* VOIP 1 Kanal



	100 ms - 128 Kb	100 ms - 384 Kb	100 ms - 512 Kb	300 ms - 128 Kb	300 ms - 384 Kb	300 ms - 512 Kb
TCP	14.33998	10.45338	9.979954	10.52441	8.148383	6.034704
UDP	17.7372	17.36631	17.20792	17.7372	17.36631	17.20792
SCTP	16.22412	14.54715	14.57981	12.9486	11.31618	10.53969

Gambar 4.35 Rata-Rata *Queue* VOIP 5 Kanal

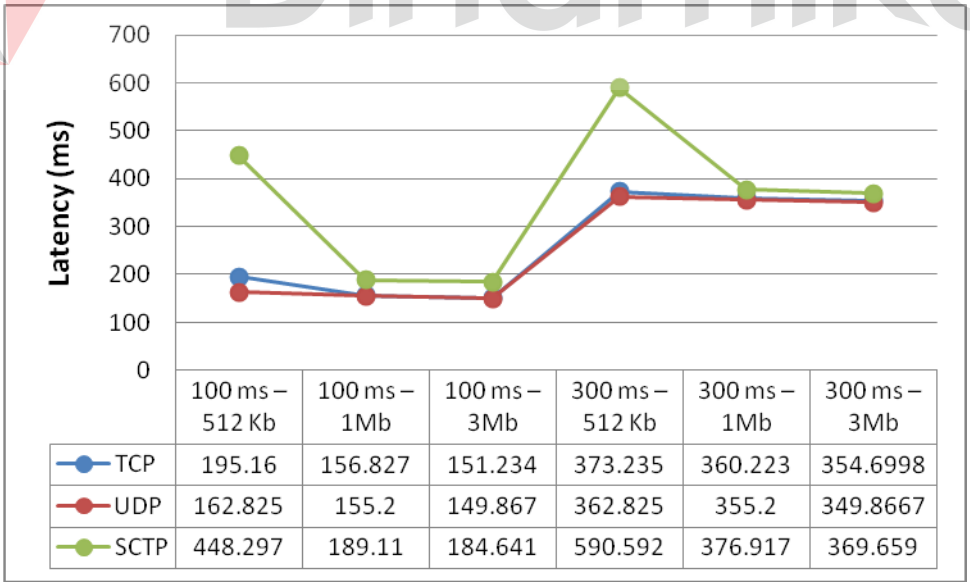
Pada perbandingan unjuk kerja TCP, UDP dan SCTP terhadap *queue*, enam simulasi di jalankan untuk masing-masing protokol dan masing-masing jumlah kanal. Jumlah kanal digunakan untuk menunjukan kondisi jaringan saat data di jalankan tanpa adanya kongesti dengan menggunakan 1 kanal, dan dengan 5 kanal akan menunjukan kinerja dari jaringan saat jalur *bottleneck* harus berbagi data dengan yang lain.

Pada pengiriman dengan 1 kanal, *queue* terbesar didapat dengan protokol UDP, namun saat *bandwidth* 512 Kb dan *delay propagation* 100ms serta pada bandwidth 512 Kb dengan *delay propagation* 300 ms, tidak ada kapasitas *queue* yang digunakan pada kondisi ini, artinya pengiriman berjalan lancar serta tidak ada segmen data yang diantrikan, segmen yang dikirim akan langsung di proses untuk di kirimkan ke node tujuan. Penggunaan *queue* pada UDP ini dipengaruhi oleh besarnya *bandwidth* tanpa mmperhitugkan *delay propagation* yang ada, terlihat dari Gambar 4.34 bahwa dengan *delay* yang sama tidak mengubah pemakaian *queue* pada protokol UDP. *Queue* terbesar kedua yaitu pada protokol

SCTP, protokol ini menggunakan kapasitas *queue* yang cukup besar dalam proses pengiriman trafik data. Semakin besar kapasitas *queue* yang digunakan, akan semakin bnyak paket data yang mengalami *loss*, karena saat kapasitas *queue* penuh, jalur pada *bottleneck* akan menutup akses masuknya data berikutnya hingga paket selesai dikirim ke tujuan dan *queue* mempunyai tempat kosong untuk penerimaan antrian berikutnya. Protokol yang menggunakan *queue* terendah adalah protokol TCP karena dia mempunyai kontrol kongesti yang mampu membatasi paket data saat terjadi kongesti atau peningkatan trafik pengiriman di jalur *bottleneck*. Pada pengiriman 5 kanal yang terlihat pada Gambar 4.35, *queue* pada protokol UDP berdasarkan beberapa kondisi yang telah ditentukan juga menunjukan grafik yang tertinggi dari kedua protokol yang lain.

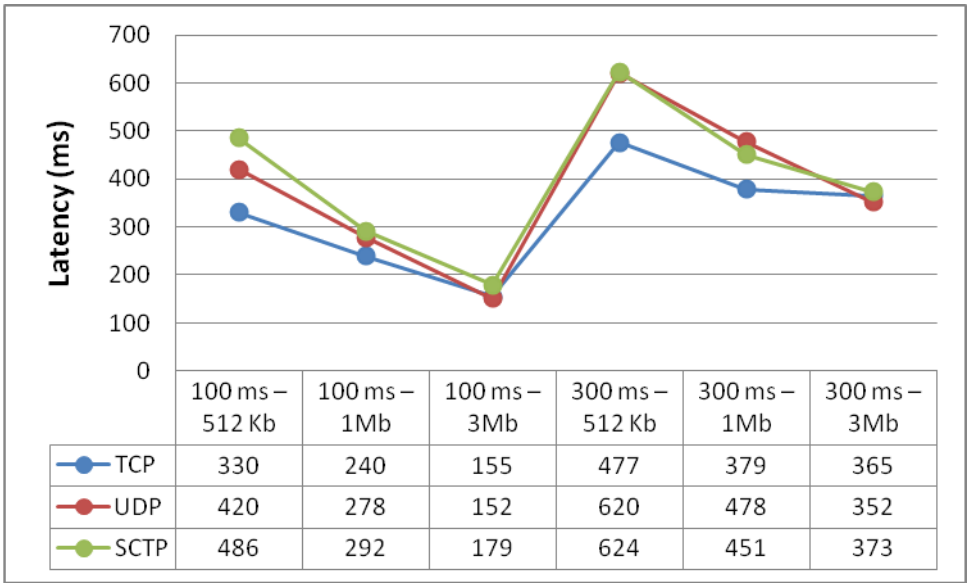
4.3.2 Analisis Menggunakan Data IPTV

1. Analisis *Latency*



	100 ms – 512 Kb	100 ms – 1Mb	100 ms – 3Mb	300 ms – 512 Kb	300 ms – 1Mb	300 ms – 3Mb
TCP	195.16	156.827	151.234	373.235	360.223	354.6998
UDP	162.825	155.2	149.867	362.825	355.2	349.8667
SCTP	448.297	189.11	184.641	590.592	376.917	369.659

Gambar 4.36 *Latency* IPTV 1 Kanal



	100 ms – 512 Kb	100 ms – 1Mb	100 ms – 3Mb	300 ms – 512 Kb	300 ms – 1Mb	300 ms – 3Mb
TCP	330	240	155	477	379	365
UDP	420	278	152	620	478	352
SCTP	486	292	179	624	451	373

Gambar 4.37 *Latency* IPTV 5 Kanal

Pada perbandingan unjuk kerja TCP, UDP dan SCTP terhadap *latency*, enam simulasi di jalankan untuk masing-masing protokol dan masing-masing jumlah kanal. Jumlah kanal digunakan untuk menunjukan kondisi jaringan saat data di jalankan tanpa adanya kongesti dengan menggunakan 1 kanal, dan dengan 5 kanal akan menunjukan kinerja dari jaringan saat jalur *bottleneck* harus berbagi data dengan yang lain.

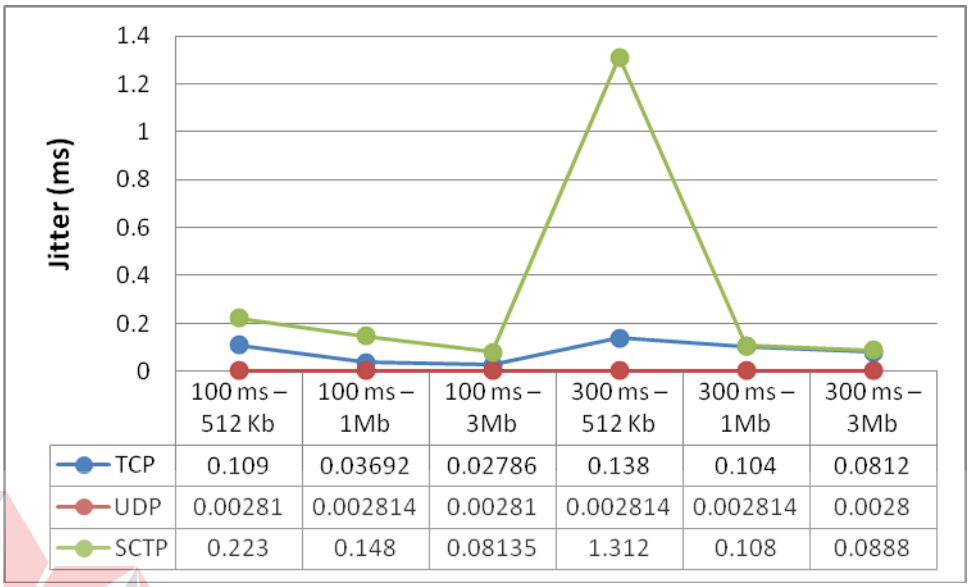
Ketika menggunakan jalur data dengan 1 kanal maupun 5 kanal seperti ditunjukan pada Gambar 4.36 dan Gambar 4.37, unjuk kerja UDP menghasilkan *latency* yang paling kecil. Jumlah *latency* yang kecil menunjukan sedikitnya waktu rata-rata pengiriman trafik data dari node sumber hingga sampai ke node tujuan. Semakin kecil *latency* akan semakin cepat data di kirim. L*atency* pada protokol TCP mempunyai nilai yang lebih besar dibanding UDP, hal ini di karenakan oleh adanya mekanisme *three-way-handshake* pada proses inisialisasi koneksi. Di samping itu, ketika jaringan padat (kongesti sangat tinggi)

menyebabkan adanya *time-out* dan TCP membutuhkan mekanisme *retransmisi* sehingga *latency* pada TCP menjadi tinggi . (Sapura & Pramartha, n.d). *Latency* dari protokol SCTP pada Gambar 4.36 dan Gambar 4.37 menunjukan nilai terbesar pada jalur 1 kanal maupun 5 kanal. Ini karena pada protokol SCTP menggunakan sistem pengiriman *best efford*, yang berarti bahwa SCTP akan berupaya mengirimkan data antar pengirim dan penerima dengan baik tetapi tidak menjamin urutan dan integritas data dari segmen yang dikirim. Sistem *best efford* sama seperti yang terjadi pada protokol IP. Selain karena hal itu, besarnya *latency* pada SCTP disebabkan juga karena pengiriman *acknowledgement* dari setiap paket SCTP membutuhkan waktu yang lama sehingga secara keseluruhan *delay* pada SCTP menjadi lebih tinggi. (Gangurde, et al., 2012). Untuk layanan data VOIP, *latency* berpengaruh terhadap kualitas layanan suara yang dihasilkan, sehingga dalam parameter *latency*, protokol UDP lah yang mempunyai nilai terkecil dari kedua protokol yang lain.

Mikael (2012) bahwa besar standar QoS untuk *latency* maksimal untuk data IPTV adalah 400 ms. Menurut Gambar 4.36 menggunakan 1 kanal, *packet loss* pada protokol TCP dan UDP dengan semua kondisi masih dapat diterima, namun pada protokol SCTP, besar *packet loss* yang tidak diterima pada kondisi *bandwidth* 512 Kb. Pada Gambar 4.37 hasil menggunakan 5 kanal, *packet loss* pada protokol TCP yang tidak bisa diterima saat kondisi *bandwidth* 512 Kb dengan *delay propagation* 300 ms, pada protokol UDP saat kondisi *bandwidth* 512 Kb dengan *delay propagation* masing-masing 100 ms dan 300 ms dan saat kondisi *bandwidth* 1 Mb dengan *delay propagation* 300 ms, sedangkan pada UDP

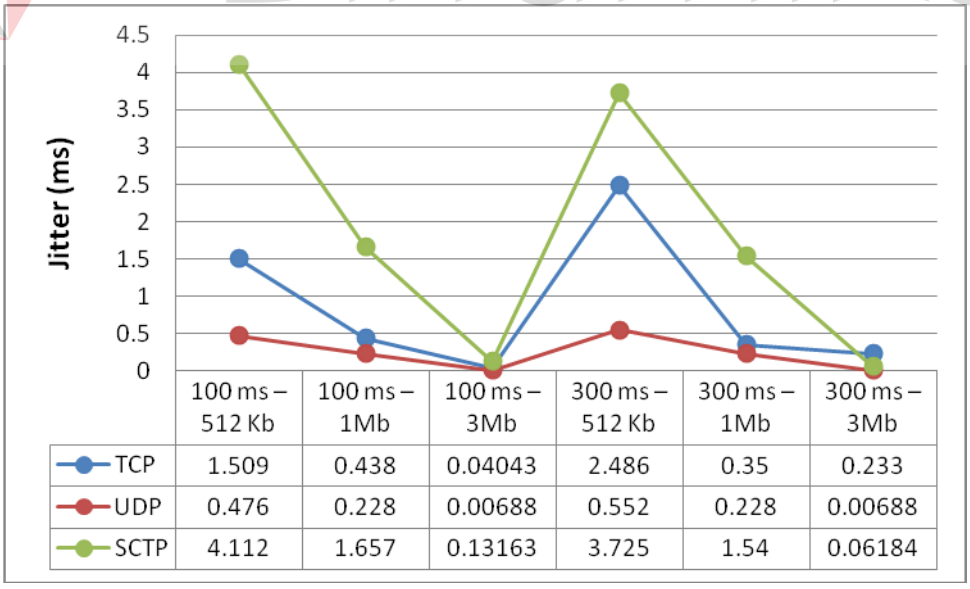
packet loss yang bisa diterima saat kondisi *bandwidth* 1 Mb dan 3 Mb dengan besar *delay propagation* masing-masing 100 ms.

.2. Analisis *Jitter*



	100 ms – 512 Kb	100 ms – 1Mb	100 ms – 3Mb	300 ms – 512 Kb	300 ms – 1Mb	300 ms – 3Mb
TCP	0.109	0.03692	0.02786	0.138	0.104	0.0812
UDP	0.00281	0.002814	0.00281	0.002814	0.002814	0.0028
SCTP	0.223	0.148	0.08135	1.312	0.108	0.0888

Gambar 4.38 *Jitter* IPTV dengan 1 Kanal



	100 ms – 512 Kb	100 ms – 1Mb	100 ms – 3Mb	300 ms – 512 Kb	300 ms – 1Mb	300 ms – 3Mb
TCP	1.509	0.438	0.04043	2.486	0.35	0.233
UDP	0.476	0.228	0.00688	0.552	0.228	0.00688
SCTP	4.112	1.657	0.13163	3.725	1.54	0.06184

Gambar 4.39 *Jitter* IPTV dengan 5 Kanal

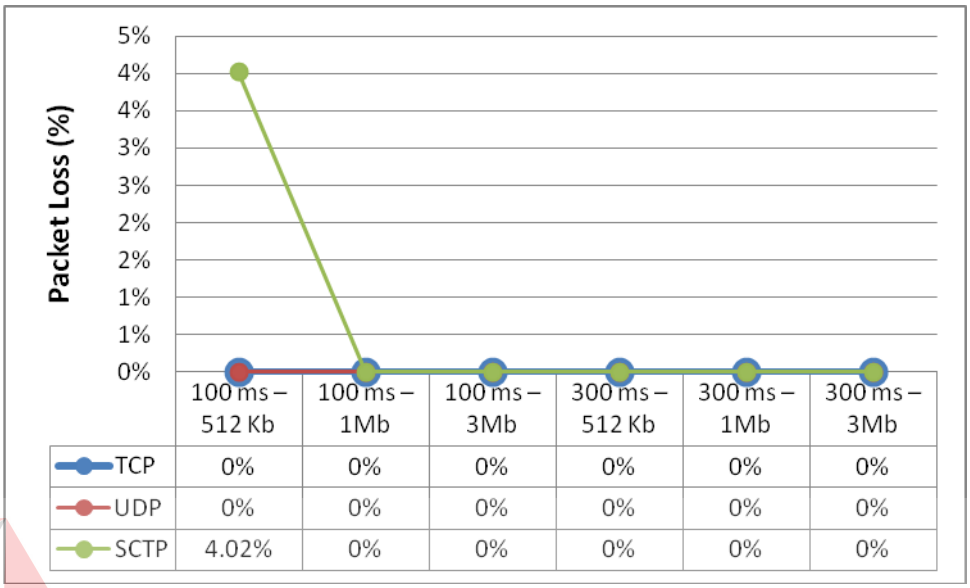
Pada perbandingan unjuk kerja TCP, UDP dan SCTP terhadap *jitter*, enam simulasi di jalankan untuk masing-masing protokol dan masing-masing jumlah kanal. Jumlah kanal digunakan untuk menunjukan kondisi jaringan saat data di jalankan tanpa adanya kongesti dengan menggunakan 1 kanal dan dengan banyak kanal akan menunjukan kinerja dari jaringan saat jalur *bottleneck* harus berbagi data dengan yang lain.

Jitter lazimnya disebut variasi *delay*, berhubungan erat dengan *latency*, yang menunjukkan banyaknya variasi *delay* pada transmisi data di jaringan. (Nurhayati, n.d). Pada *jitter* dengan 1 kanal dan 5 kanal mengahasilkan analisis yang sama yaitu *jitter* yang paling besar adalah protokol SCTP. Pada protokol SCTP, variasi *delay* terendah berada saat kondisi *bandwidth* 512 Kb dan *delay propagation* 100 ms, sedangkan *jitter* tertinggi terjadi saat kondisi *bandwidth* 128 Kb dan *delay propagation* 300 ms. Rata-rata *jitter* tertinggi kedua yaitu pada protokol TCP. *Jitter* terendah pada TCP, terjadi saat *bandwidth* 512 Kb dan *delay propagation* 100 ms, sedangkan *jitter* tertinggi saat kondisi *bandwidth* 128 Kb dan *delay propagation* 100 ms. Pada UDP *jitter* dipengaruhi oleh perubahan *bandwidth*, sedangkan *delay propagation* tidak begitu signifikan dalam menentukan nilai dari *jitter* pada UDP sehingga terlihat dengan *bandwidth* yang berbeda menghasilkan *jitter* yang berbeda pula, namun pada perbedaan *delay propagation*, nilai *jitter* tidak mengalami perubahan yang signifikan.

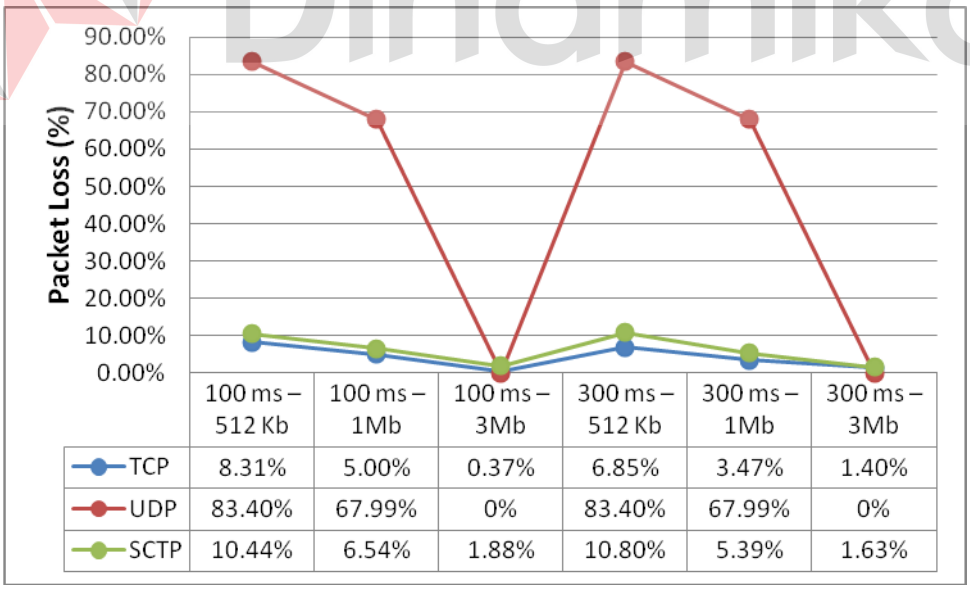
Menurut referensi *range jitter* yang masi bisa ditangani saat masih berada dibawah 50 ms. (Technologies, 2008). Dari referensi tersebut dapat disimpulkan bahwa dengan satu kanal maupun dengan 5 kanal, untuk semua protokol TCP, UDP dan SCTP dalam hal *jitter* masi bisa diterima untuk membawa layanan

IPTV. Dari ketiga protokol menunjukan UDP memiliki *jitter* yang lebih rendah dari kedua protokol yang lain

3. Analisis *Packet Loss*



Gambar 4.40 *Packet Loss* IPTV 1 Kanal



Gambar 4.41 *Packet Loss* IPTV 5 Kanal

Pada perbandingan unjuk kerja TCP, UDP dan SCTP terhadap *packet loss,* enam simulasi di jalankan untuk masing-masing protokol dan masing-masing jumlah kanal. Jumlah kanal digunakan untuk menunjukan kondisi jaringan saat data di jalankan tanpa adanya kongesti dengan menggunakan 1 kanal dan dengan 5 kanal akan menunjukan kinerja dari jaringan saat jalur *bottleneck* harus berbagi data dengan yang lain.

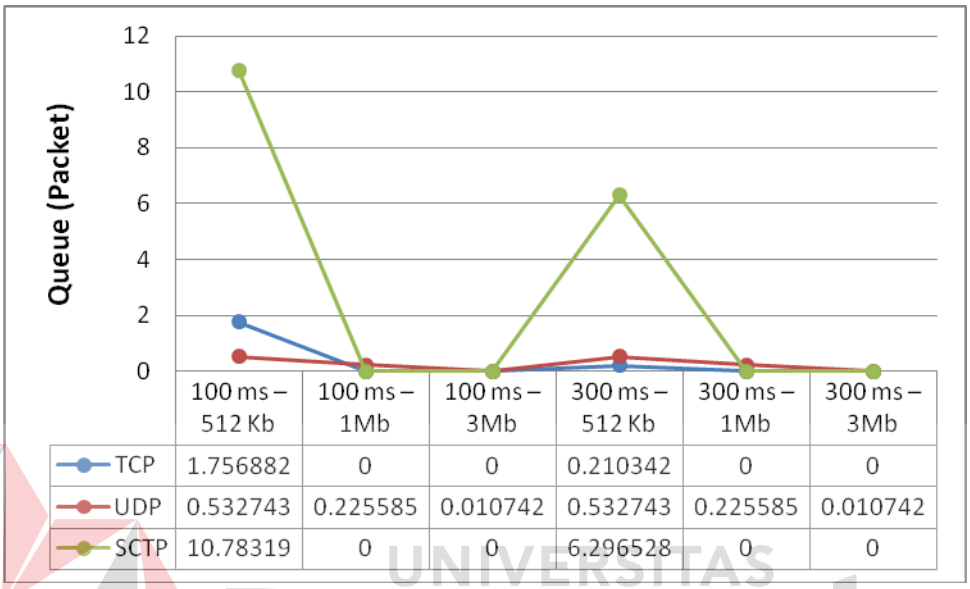
Pada percobaan mengggunakan satu kanal terlihat pada Gambar 4.40 bahwa TCP dan UDP tidak mengalami *packet loss* pada semua kondisi. Hal ini disebabkan kapasitas *queue* masi bisa menangani *packet* yang masuk ke jalur itu sehingga tidak ada paket yang harus dibuang untuk mengurangi kongesti. Sedangkan pada SCTP saat *bandwidth/delay propagation* 512 Kb/ 100ms, pada pengiriman data IPTV dengan 1 kanal menibulkan *packet loss* sebesar 4.02%.

Pada Gambar 4.41, pemakaian jalur *bottleneck* dibuat untuk *link* data dari 5 node sumber ke 5 node tujuan. Pada jalur ini, tidak bisa diketahui pengiriman paket yang datang berasal dari node mana dengan tujuan yang mana, sehingga *packet loss* pada jalur ini meningkat sangat banyak dikarenakan kapasitas *queue* juga mengalami peningkatan, karena pada jalur *bottleneck,* trafik yang dikirimkan harus berbagi tempat antrian yang cukup banyak antara node-node tujuan.

Dari Gambar 4.40 dan Gambar 4.41 terlihat bahwa dengan *bandwidth* yang lebih besar menghasilkan *packet loss* yang lebih lebih kecil. Menurut Mikael (2012), standard *packet loss* maksimal untuk layanan *video streaming* atau IPTV adalah 2 %. Dari hasil Gambar 4.40 dengan satu kanal, *packet loss* pada semua protokol dengan berbagai kondisi masih dapat diterima kecuali protokol SCTP saat kondisi *bandwidth* 128 Kb dengan *delay propagation* 100 ms. Pada Gambar

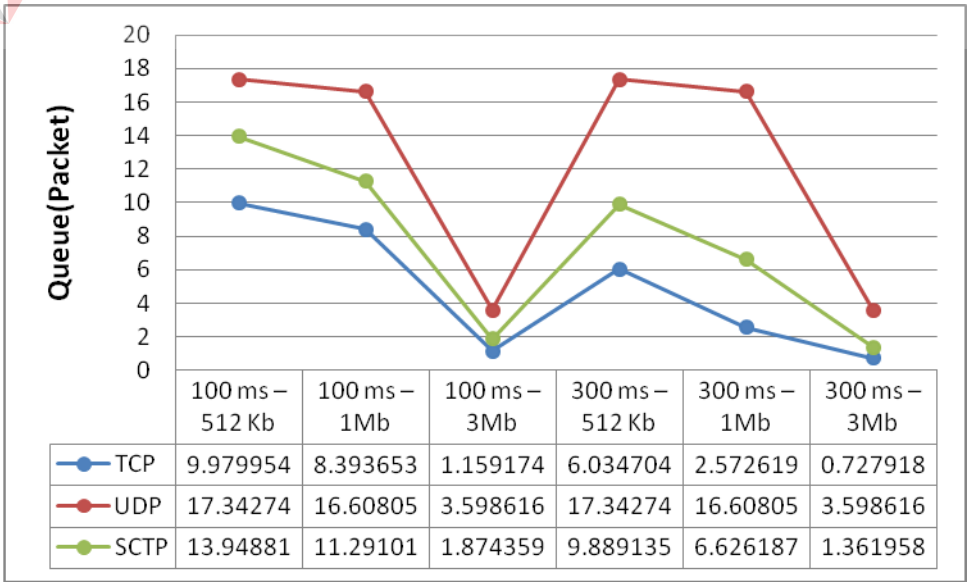
4.41 hasil *packet loss* dengan 5 kanal menunjukan bahwa *packet loss* untuk keseluruhan protokol yang bisa diterima adalah saat kondisi *bandwidth* 3 Mb.

4. Analisis *Queue*



	100 ms – 512 Kb	100 ms – 1Mb	100 ms – 3Mb	300 ms – 512 Kb	300 ms – 1Mb	300 ms – 3Mb
TCP	1.756882	0	0	0.210342	0	0
UDP	0.532743	0.225585	0.010742	0.532743	0.225585	0.010742
SCTP	10.78319	0	0	6.296528	0	0

Gambar 4.42 Rata-Rata *Queue* IPTV 1 Kanal



	100 ms – 512 Kb	100 ms – 1Mb	100 ms – 3Mb	300 ms – 512 Kb	300 ms – 1Mb	300 ms – 3Mb
TCP	9.979954	8.393653	1.159174	6.034704	2.572619	0.727918
UDP	17.34274	16.60805	3.598616	17.34274	16.60805	3.598616
SCTP	13.94881	11.29101	1.874359	9.889135	6.626187	1.361958

Gambar 4.43 Rata-Rata *Queue* IPTV 1 Kanal

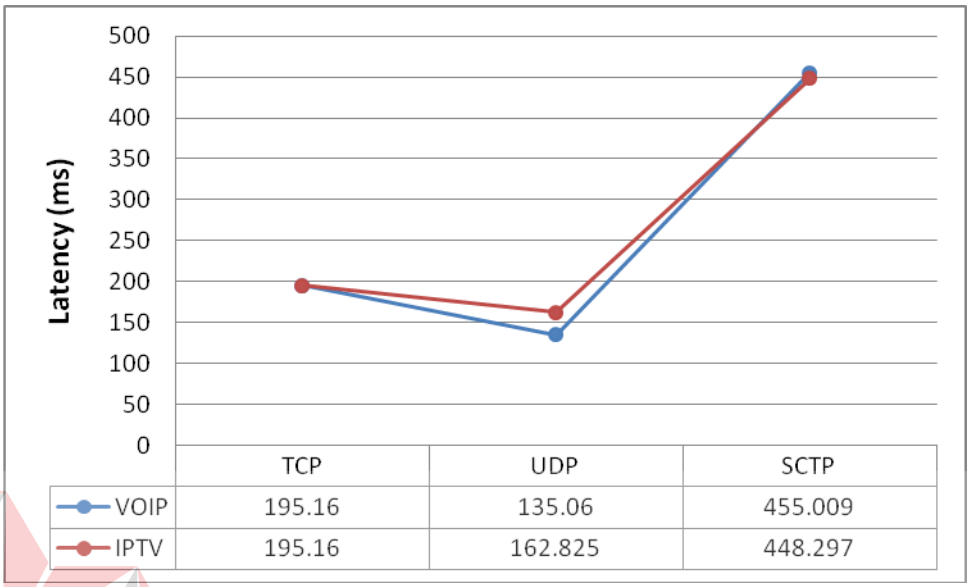
Pada perbandingan unjuk kerja TCP, UDP dan SCTP terhadap *queue*, enam simulasi di jalankan untuk masing-masing protokol dan masing-masing jumlah kanal. Jumlah kanal digunakan untuk menunjukan kondisi jaringan saat data di jalankan tanpa adanya kongesti dengan menggunakan 1 kanal dan dengan 5 kanal akan menunjukan kinerja dari jaringan saat jalur *bottleneck* harus berbagi data dengan yang lain.

Pada pengiriman dengan 1 kanal, pada *bandwidth* 512 Kb dan *delay propagation* 100 ms, besar kapasitas *queue* terbesar digunakan oleh protokol SCTP, setelah itu dia turun menjadi 0 yaitu pada kondisi ini tidak ada paket yang diantrikan pada *buffer queue*, kemudian naik kembali pada kondisi *bandwidth/delay propagation* 512Kb/300 ms. pada protokol TCP dan UDP, penggunaan *queue* tidak terlalu besar. Hal ini disebabkan karena pada protokol ini dengan berbagai kondisi di atas masih mampu melewatkan *queue* dengan baik. Pada pengiriman dengan 5 kanal, *queue* di semua protokol mengalami peningkatan di banding dengan menggunakan 1 kanal. Hal ini disebabkan adanya peningkatan beban trafik dan tumbukan antar paket yang meningkat sesuai dengan penambahan jumlah kanal.

4.3.2 Analisis *Latency* Dan *Jitter* Pada Kedua Data

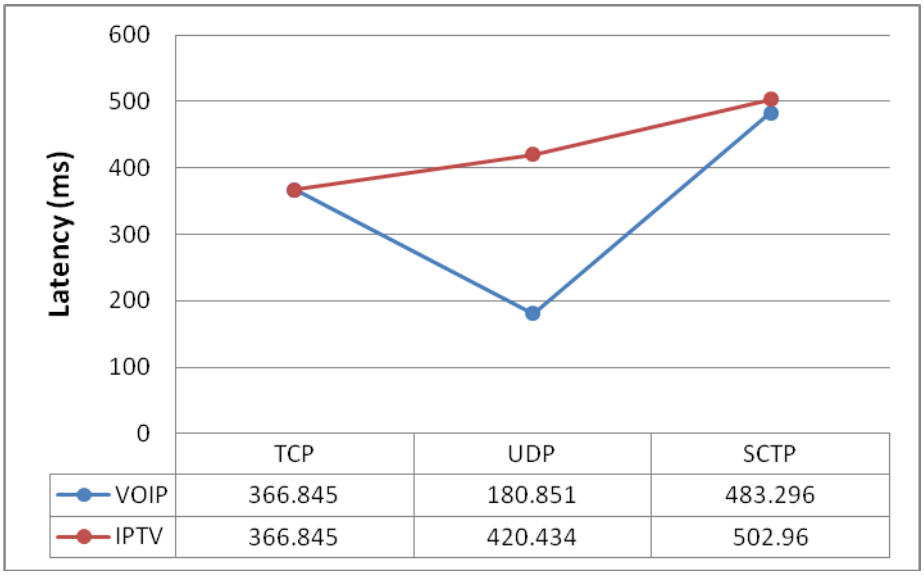
1. B*andwidth/Delay propagation* 512 Kb/100 ms

a. Analisis Latency



Gambar 4.44 *Latency* 1 Kanal Saat 512Kb-100ms

Analisis dilakukan dengan membandingkan *latency* kedua data dengan 1 kanal saat dikondisikan pada *bandwidth* dan *delay propagation* yang sama. Hasil berdasarkan grafik pada Gambar 4.44 dapat dilihat bahwa antara protokol TCP, UDP dan SCTP saat menggunakan data VOIP maupun IPTV dengan kondisi *bandwidth/delay propagation* yang sama menghasilkan perbedaan nilai yang tidak terlalu besar. Hal ini membuat kedua data VOIP dan IPTV mempunyai persamaan kualitas layanan dalam hal *latency* untuk kondisi tersebut.

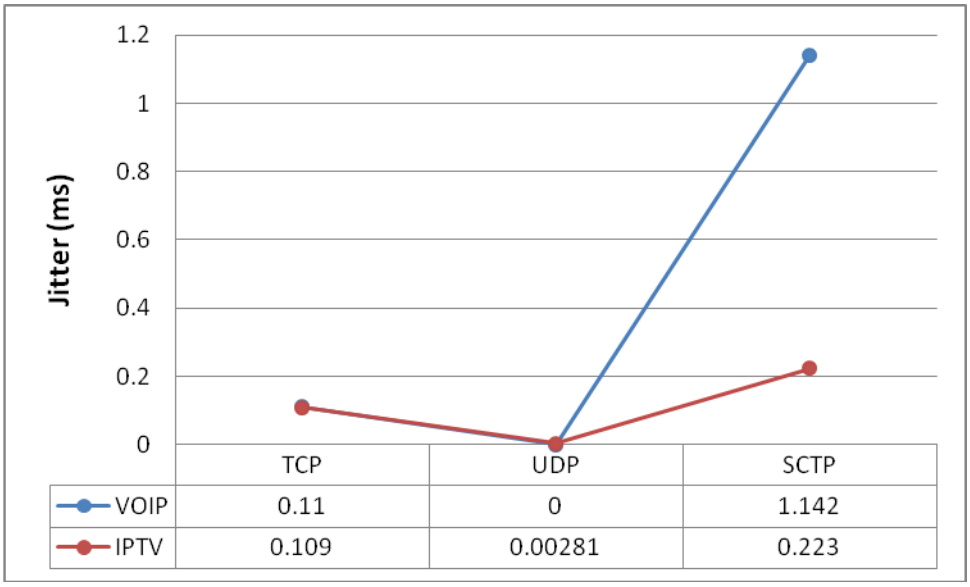


Gambar 4.45 *Latency* 5 Kanal Saat 512Kb-100ms

Analisis dilakukan dengan membandingkan kedua data dengan 5 kanal saat dikondisikan pada *bandwidth* dan *delay propagation* yang sama yang di dasarkan pada *latency* dari kedua data. Hasil berdasarkan grafik pada Gambar 4.45 dapat dilihat bahwa saat menggunakan data VOIP maupun IPTV dengan *bandwidth/delay propagation* 512Kb/100ms tidak ada perbedaan nilai latency pada TCP. Ini artinya bahwa waktu pengiriman rata-rata pada kedua data ini tidak berbeda. Pada UDP perbedaannya sangat besar, yaitu dengan data IPTV, delay yang dihasilkan akan lebih besar dari latency dengan data VOIP. Pada SCTP nilai *latency*

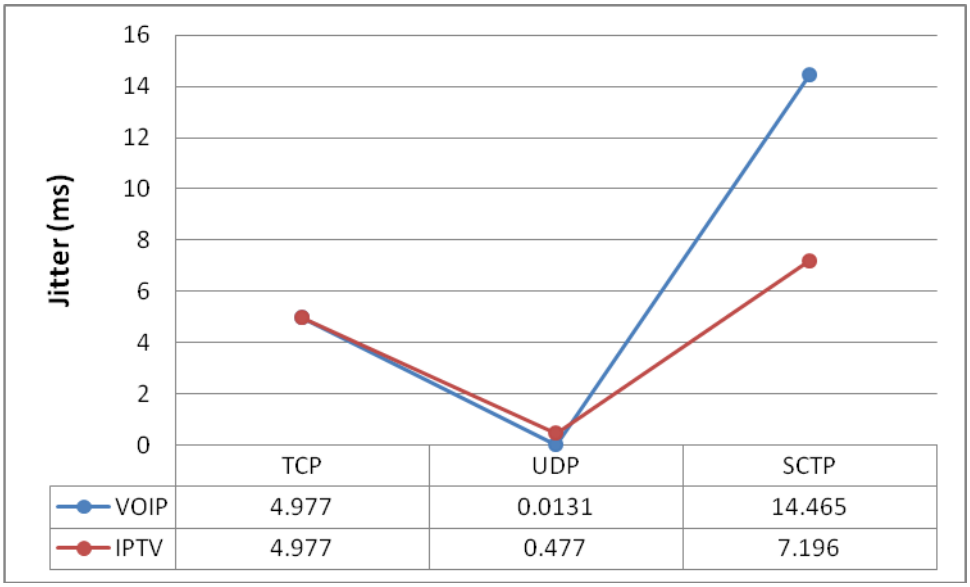
data IPTV sedikit lebih besar dari VOIP. Hal ini berarti bahwa pengiriman dengan data IPTV pada protokol UDP dan SCTP membuuhkan waktu yang lebih lama dibanding pada VOIP.

a. Analisis *jitter*



Gambar 4.46 *Jitter* 1 Kanal Saat 512Kb-100ms

Analisis dilakukan dengan membandingkan kedua data dengan 1 kanal saat dikondisikan pada *bandwidth* dan *delay propagation* yang sama yang di dasarkan pada *jitter* dari kedua data. Hasil berdasarkan grafik pada Gambar 4.46 dapat dilihat bahwa saat menggunakan data VOIP maupun IPTV dengan *bandwidth/delay propagation* 512Kb/100ms tidak ada perbedaan nilai *jitter* pada TCP dan UDP. Ini artinya bahwa tidak ada perbedaan variasi pengiriman pada kedua data dengan protokol TCP dan UDP. Pada SCTP perbedaannya cukup besar, yaitu dengan data IPTV, delay yang dihasilkan akan lebih kecil di bandingkan dengan menggunakan data VOIP. Dapat dirartikan bahwa variasi pengiriman di SCTP dengan data VOIP lebih besar di banding dengan data IPTV.

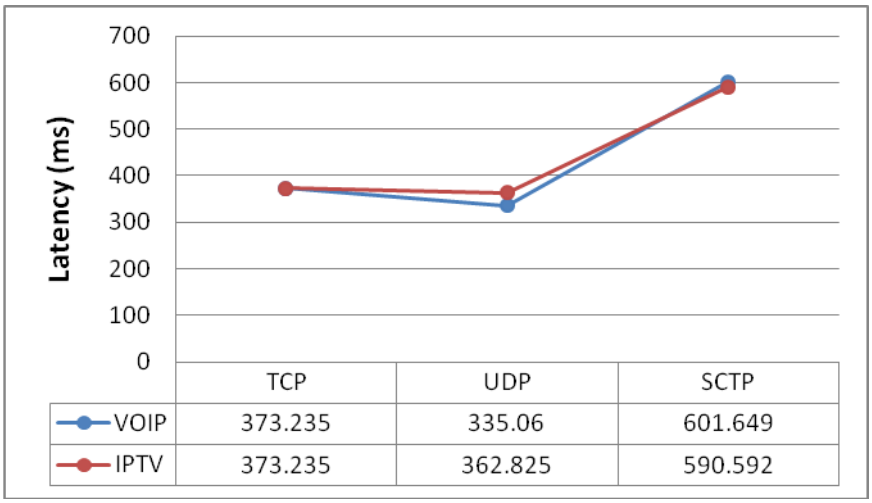


Gambar 4.47 *Jitter* 5 Kanal Saat 512Kb-100ms

Analisis dilakukan dengan membandingkan kedua data dengan 5 kanal saat dikondisikan pada *bandwidth* dan *delay propagation* yang sama yang di dasarkan pada *jitter* dari kedua data. Hasil berdasarkan grafik pada Gambar 4.47 dapat dilihat bahwa saat menggunakan data VOIP maupun IPTV dengan *bandwidth/delay propagation* 512Kb/100ms tidak ada perbedaan nilai *jitter* pada TCP. Ini artinya bahwa tidak ada perbedaan variasi pengiriman pada kedua data dengan protokol TCP, sedangkan untuk protokol UDP memiliki perbedaan *jitter* yang tidak terlalu besar pada kedua data. Pada SCTP perbedaannya cukup besar, yaitu dengan data IPTV, *delay* yang dihasilkan akan lebih kecil di bandingkan dengan menggunakan data VOIP. Dapat dirartikan bahwa variasi pengiriman dengan data VOIP lebih besar di banding dengan data IPTV

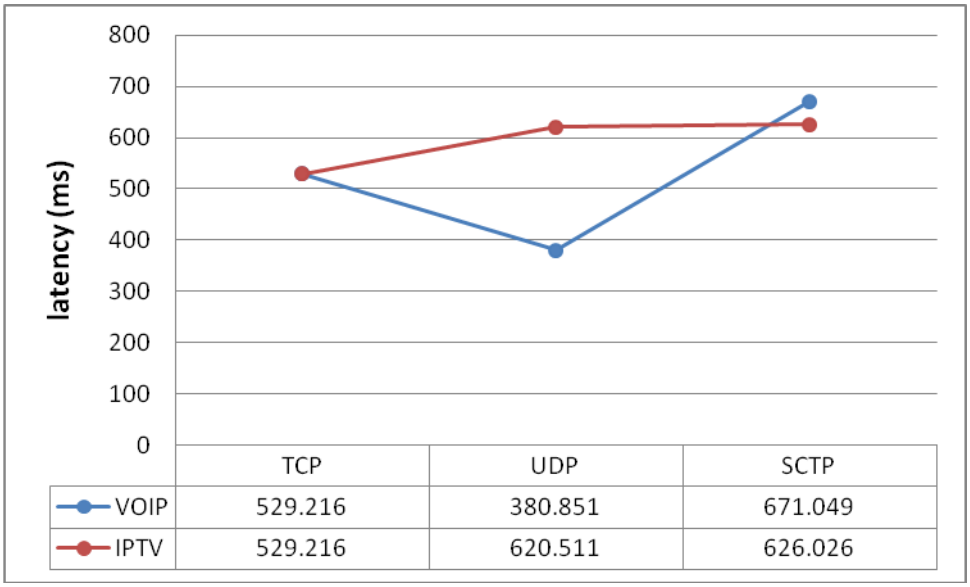
2. *Bandwidth/Delay propagation* 512 Kb/300 ms

a. Analisis *latency*



Gambar 4.48 *Latency* 1 Kanal Saat 512Kb-300ms

Analisis dilakukan dengan membandingkan kedua data dengan 1 kanal saat dikondisikan pada *bandwidth* dan *delay propagation* yang sama yang di dasarkan pada *latency* dari kedua data. Hasil berdasarkan grafik pada Gambar 4.48 dapat dilihat bahwa antara protokol TCP, UDP dan SCTP saat menggunakan data VOIP maupun IPTV dengan kondisi *bandwidth/delay propagation* yang sama menghasilkan perbedaan nilainya tidak terlalu besar. Hal ini membuat kedua data VOIP dan IPTV mempunyai persamaan kualitas layanan dalam hal *latency* untuk kondisi tersebut.

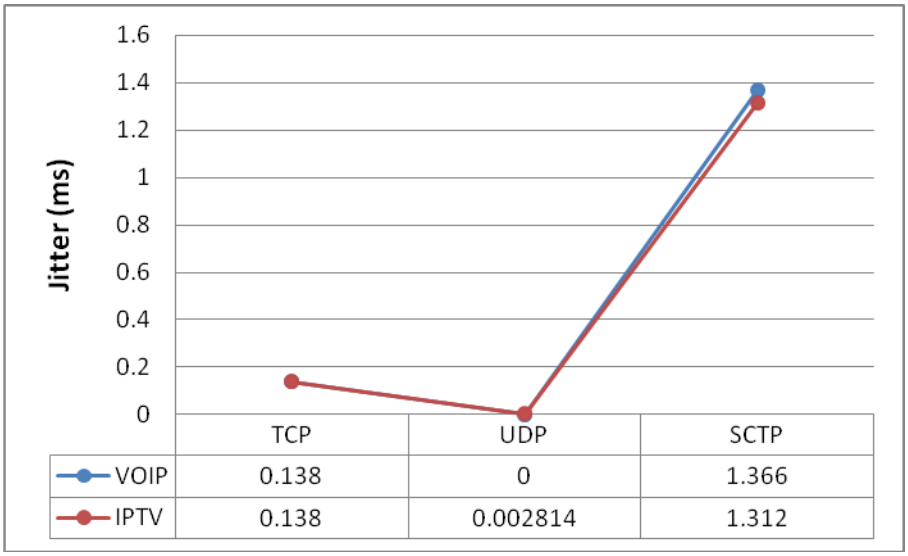


Gambar 4.49 *Latency* 5 Kanal Saat 512Kb-300ms

Analisis dilakukan dengan membandingkan kedua data dengan 5 kanal saat dikondisikan pada *bandwidth* dan *delay propagation* yang sama yang di dasarkan pada *latency* dari kedua data. Hasil berdasarkan grafik pada Gambar 4.49 dapat dilihat bahwa pada protokol TCP tidak ada perbedaan nilai *latency*, namun saat menggunakan protokol UDP perbedaan antara kedua data terlihat cukup besar. Ini dikarenakan dengan 5 kanal, pengiriman paket yang lebih besar dari VOIP akan menimbulkan besarnya nilai *latency*.

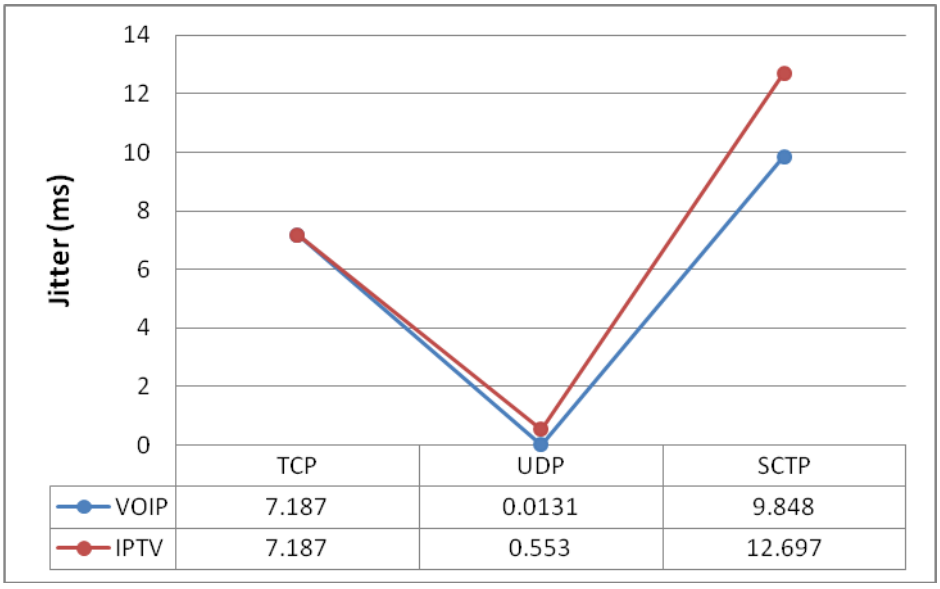
Dari Gambar 4.48 dan Gambar 4.49 menunjukan bahwa *latency* pada protokol UDP menunjukan nilai yang lebih baik dibanding *latency* pada protokol TCP dan SCTP, karena dengan penambahan *packet loss* tidak mempengaruhi jumlah *latency* yang dihasilkan pada protokol UDP. Hal ini sesuai dengan hasil penelitian yang dilakukan oleh Gangurde, et al. (2012).

b. Analisis *Jitter*



Gambar 4.50 *jitter* 1 Kanal Saat 512Kb-100ms

Analisis dilakukan dengan membandingkan *jitter* antara kedua data dengan 1 kanal saat dikondisikan pada *bandwidth* dan *delay propagation* yang sama. Hasil berdasarkan grafik pada Gambar 4.50 dapat dilihat bahwa *jitter* ketiga protokol terhadap kedua data dengan *bandwidth/delay propagation* 512Kb/300ms perbedaan nilainya tidak terlalu besar. Hal ini membuat kedua data VOIP dan IPTV mempunyai persamaan kualitas pengirirman dalam hal variasi *delay* untuk kondisi tersebut.



Gambar 4.51 *jitter* 5 Kanal Saat 512Kb-100ms

Analisis dilakukan dengan membandingkan *jitter* antara kedua data dengan 5 kanal saat dikondisikan pada *bandwidth* dan *delay propagation* yang sama. Hasil berdasarkan grafik pada Gambar 4.51 dapat dilihat bahwa *jitter* protokol SCTP pada data IPTV lebih besar dibanding pada data VOIP.

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Kesimpulan dari tugas akhir "ANALISIS PERBANDINGAN UNJUK KERJA PROTOKOL TCP, UDP, DAN SCTP MENGGUNAKAN SIMULASI LALU LINTAS DATA MULTIMEDIA" adalah :

1. Simulasi jaringan berhasil dibangun dengan menggunakan NS-2 yang mengimplementasikan protokol TCP, UDP dan SCTP, sedangkan untuk aplikasi terdiri atas dua aplikasi yaitu VOIP dan IPTV. Masing-masing aplikasi di uji coba untuk kondisi utilisasi *bandwidth* rendah menggunakan 1 kanal dan utilisasi *bandwidth* tinggi menggunakan 5 kanal.
2. Analisis dilakukan dengan membandingkan protokol TCP, UDP dan SCTP berdasarkan hasil yang didapat berdasarkan *latency*, *jitter*, *packet loss*, dan *queue*. Hasil dari simulasi ini yaitu :
   a. Pada data VOIP dan IPTV, menunjukan bahwa waktu pengirirman pada protokol SCTP memerlukan waktu yang lebih lama dibanding kedua protokol yang lain (ditunjukan oleh nilai *latency* yang lebih besar pada protokol SCTP dibandingkan dengan TCP dan UDP). Hal ini dikarenakan mekanisme *best efford* serta karena pengiriman *acknowledgement* dari setiap paket SCTP membutuhkan waktu yang lama. Besar perbandingan nilai *latency* dari ketiga protokol TCP, UDP dan SCTP dengan menggunakan satu kanal dapat dilihat pada lampiran 5.1 dan dengan lima kanal dapat dilihat pada lampiran 5.2.

b. Pada data VOIP dan IPTV, parameter *Jitter* berhubungan dengan besarnya nilai *latency*. Semakin tinggi nilai *latency* akan semakin tinggi pula nilai *jitter* yang dihasilkan. Berdasarkan hasil pengujian terhadap nilai *jitter*, protokol SCTP memiliki nilai *jitter* tertinggi dibandingkan dengan kedua protokol yang lain. Hal ini sesuai dengan hasil pada pengujian *latency*. Semakin tinggi nilai *jitter* yang dihasilkan akan mempengaruhi variasi dan ketidakseimbangan *delay* data yag diterima. Besar perbandingan nilai *jitter* dari ketiga protokol TCP, UDP dan SCTP dengan menggunakan satu kanal dapat dilihat pada lampiran 5.1 dan dengan lima kanal dapat dilihat pada lampiran 5.2.

c. Dengan kondisi nilai *bandwidth* dan *delay propagation* pada data VOIP dan IPTV sama untuk simulasi dengan menggunakan 1 kanal, didapatkan bahwa perbedaan nilai *latency*, *jitter* serta *packet loss* dari kedua data tidak terlalu besar. Hal ini membuat kedua data VOIP dan IPTV mempunyai persamaan kualitas layanan untuk kondisi tersebut. Besar perbandingan nilai *latency* dan *jitter* data VOIP dan IPTV dari ketiga protokol TCP, UDP dan SCTP dengan menggunakan satu kanal dapat dilihat pada lampiran 5.1 dan dengan lima kanal dapat dilihat pada lampiran 5.2.

d. Pada data VOIP dan IPTV, berdasarkan hasil pengujian didapatkan bahwa untuk *bandwidth* yang rendah menunjukan protokol UDP mempunyai rata-rata *packet loss* tertinggi dari kedua protokol yang lain. Semakin banyaknya *packet loss* akan menurunkan kualitas layanan dari suatu protokol, karena menggambarkan banyaknya paket yang hilang saat proses pengiriman. Besar perbandingan nilai *packet loss* dari ketiga protokol TCP, UDP dan

SCTP dengan menggunakan satu kanal dapat dilihat pada lampiran 5.3 dan dengan lima kanal dapat dilihat pada lampiran 5.4.

e. Pada data VOIP dan IPTV, Dari hasil pengujian didapatkan bahwa dengan nilai *bandwidth* yang rendah menunjukan UDP menggunakan *queue* lebih besar dari kedua protokol lain yang dibandingkan. Besarnya penggunaan *queue* menyebabkan banyaknya antrian paket data pada jalur *bottleneck* yang dapat mengakibatkan paket data yang masuk akan di *drop* saat penggunaan *queue* telah mencapai batas maksimum. Besar perbandingan nilai *queue* dari ketiga protokol TCP, UDP dan SCTP dengan menggunakan satu kanal dapat dilihat pada lampiran 5.6 dan dengan lima kanal dapat dilihat pada lampiran 5.7.

2. Dari kedua data VOIP dan IPTV dan ketiga protokol TCP, UDP, dan SCTP, untuk mendapatkan kualitas yang baik berdasarkan parameter latency dan jitter dibutuhkan nilai *bandwidth* yang tinggi serta nilai *delay propagation* yang minimum. Sedangkan, untuk menghasilkan *packet loss* dan penggunaan *queue* yang rendah dibutuhkan nilai *bandwidth* yang besar.

5.2 Saran

Agar diperoleh hasil simulasi yang lebih baik lagi dan sebagai pengembangan pada penelitian berikutnya sebaiknya simulasi dilakukan secara simultan untuk menguji *interfairness* dari masing-masing protokol TCP, UDP dan SCTP dalam satu jalur *bottleneck*. *Interfairness* menyatakan tingkat keadilan pembagian *bandwidth* antara sesi-sesi trafik yang terdiri dari dua atau lebih protokol yang berbeda. Untuk itu dibutuhkan pengukuran *interfairness* dari protokol tersebut terutama untuk protokol SCTP.

DAFTAR PUSTAKA

Alwi, E. I. & Syawie, ,. I., n.d. *SCTP (Stream Control Transmission Protocol)*, Yogyakarta: Fakultas Teknik UGM.

Andrew, R., Susanto, J. & Aprilianto, R., 2006. *ANALISIS DAN PERANCANGAN IP TELEPHONY DI JARINGAN KOMPUTER LOKAL KANTOR PUSAT PT. XYZ*, Jakarta: Universitas Bina Nusantara.

Anon., n.d. *Distribusi Multimedia.* [Online] Available at: http://elista.akprind.ac.id/staff/catur/Sistem%20Multimedia/11-Distribusi%20Multimedia.pdf [Accessed Februari 2013].

Budiardjo, B. & Thiotrisno, M., 2003. Simulasi Pengukuran Intrafairness dan Interfairness Protokol-Protokol Stream Control Transmission Protocol (SCTP) dan Transmission Control Protokol (TCP) Pada Jaringan Unicast. *Teknologi*, 7(2), pp. 55-62.

Budiraharjo, B., 2009. *ANALISI PERBANDINGAN TRANSMISSION CONTROL PROTOCOL (TCP) DENGAN STREAM CONTROL TRANSMISSION PROTOCOL (SCTP) PADA FILE TRANSFER*, Bandung: http://digilib.ittelkom.ac.id.

Computer Science Department, 2007. *NS-2 tutorial*, Pennsylvania: Carnegie Mellon.

CTTL, 2008. *IPTV CDN Traffic Modelling.* [Online] Available at: ftp://ftp.heanet.ie/disk1/sourceforge/g/project/gr/gridnetworksim/Development%20Reports/Original%20Project%20Reports/IPTV_CDN-Release1.pdf [Accessed 2013].

Darmawan, A., Alif, N. & Basuki, S., 2009. *Analisa QOS Pada Codec G711 dalam Jaringan VOIP Berbasis Protokol SIP*, Malang: Universitas Muhamadiyah Malang.

Deziel, C., 2013. *Minimum Bandwidth for VoIP*. [Online] Available at: http://techtips.salon.com/minimum-bandwidth-voip-20888.html [Accessed 2013].

Ferguson, P. & Huston, G., 1998. *Quality of Service.* s.l.:John Wiley & Sons Inc.

Forouzan, B. A., 2007. *Data Communications And Networking.* New York : McGraw-Hill.

gangurde, P., Waware, S. & Sarwade, N., 2012. Simulation of TCP, UDP, and SCTP with constant traffic for VOIP. *International Journal of Engineering Research and Application,* p. 1245.

Gangurde, P., Waware, S. & Sarwade, N., 2012. Simulation of TCP, UDP, and SCTP with constant traffic for VOIP. *International Journal of Engineering Research and Application,* p. 1245.

Gonia, K., 2004. *Latency and QoS for Voice over IP,* s.l.: SANS Institute.

Kartika, D. C., n.d. *RANCANG BANGUN LAYANAN PERSONAL VIDEO RECORDING (PVR) PADA INTERNET PTOTOCOL TELEVISION (IPTV),* Surabaya: Institut Teknologi Sepuluh Nopember,.

Kristanto, A., 2003. *Jaringan Komputer.* Yogyakarta: Graha Ilmu.

Kurose, J. F. & Ross, K. W., 2010. *Computer Networking.* Boston: Pearson.

LAKSONO, F. A., 2010. *ANALISIS PERFORMANSI IPTV (INTERNET PROTOKOL TELEVISION) OVER JARINGAN ADSL SPEEDY,* Bandung: IT Telkom Digital Library.

Nurhayati, O. D., n.d. *Sistem Komunikasi Multimedia.* [Online] Available at: http://eprints.undip.ac.id/20121/1/Persyaratan_Layanan_dan_Protokol_pert9.pdf [Accessed Februari 2012].

Riadi, I. & Wicaksono, W. P., 2011. *Implementasi Quality of Service Menggunakan Metode Hierarchical Token Bucket.* Yogyakarta: Universitas Ahmad Dahlan.

Rosalin, R. E., n.d. *Analisis dan Implementasi Algoritma Video Kompresi Pada Jaringan IPTV,* Bandung: Teknik Komputer Politeknik Telkom.

Santosa, B., 2004. *Manajemen bandwidth internet dan intranet.* [Online] Available at: http://stream.plasa.com/onno/gfe/view.php?file=referensi_bahasa_indonesia_2/network/bwmanagement.pdf

Sapura, I. W. A. & Pramartha, C. R. A., n.d. *PERANCANGAN FTP (FILE TRANSFER PROTOCOL) MELALUI SCTP (STREAM CONTROL TRANSMISION PROTOCOL) MENGGUNAKAN SOCKET PROGRAMMING,* Bali: Universitas Udayana.

Sapura, I. W. A. & Pramartha, C. R. A., n.d. *PERANCANGAN FTP (FILE TRANSFER PROTOCOL) MELALUI SCTP (STREAM CONTROL TRANSMISION PROTOCOL) MENGGUNAKAN SOCKET PROGRAMMING,* Bali: Universitas Udayana.

Sariningrum, L., 2008. *Analisa Perbandingan Performansi TCP dan SCTP Pada Jaringan Kabel,* Bandung: Perpustakaan Unikom.

Sofana, I., 2009. *CISCO CCNA & JARINGAN KOMPUTER.* Bandung: Informatika.

Stallings, W., 2000. *Komunikasi Data dan Komputer Jaringan Komputer.* New Jersey: Prentice-Hall.

Technologies, A., 2008. *IPTV QoE: Understanding and interpreting MDI values.* [Online] Available at: http://cp.literature.agilent.com/litweb/pdf/5989-5088EN.pdf [Accessed Februari 2013].

Telstra & Huston, G., n.d. *Future for TCP.* [Online] Available at: http://www.cisco.com/web/about/ac123/ac147/ac174/ac195/about_cisco_ipj_archive_article09186a00800c83f8.html [Accessed Desember 2012].

Unuth, N., 2013. *VoIP and Bandwidth - How Much Bandwidth Do I Need for VoIP?.* [Online] Available at: http://voip.about.com/od/requirements/a/bandwidth.htm [Accessed 2013].

Wang, G., Xia, Y. & Harrison, D., 2008. *An NS2 TCP Evaluation Tool.* [Online] Available at: ftp://ftp.heanet.ie/disk1/sourceforge/t/tc/tcpeval/tcpeval/v0.2/tcpeval-manual-0.2.pdf [Accessed 2013].

Wirawan, A. B. & Indarto, E., 2004. *Network Simulator-2 (NS-2).* Yogyakarta: Andi.

Yonathan, B., Bandung, Y. & Langi, A. Z., n.d. ANALISIS KUALITAS LAYANAN (QOS) AUDIO-VIDEO LAYANAN KELAS VIRTUAL DI JARINGAN DIGITAL LEARNING PEDESAAN. *DSP Research and Technology Group*, p. 5.

Zafar, M. S. & Gill, M. S., 2008. *Evaluation of UDP and SCTP for SIP-T and TCP, UDP and SCTP with constant traffic.* Karlskrona: Blekinge Institute of Technology.