Analisis Performa Komunikasi *Video Conference* Pada WebRTC dengan Menggunakan *Framework* RTCMultiConnection

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Lestari Sintika Syarah
NIM: 125150201111089

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2017

PENGESAHAN

Analisis Performansi Komunikasi *Video Conference* Pada WebRTC dengan Menggunakan *Framework* RTCMultiConnection

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Lestari Sintika Syarah
NIM: 125150201111089

Skripsi ini telah diuji dan dinyatakan lulus pada
23 Januari 2017
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Adhitya Bhawiyuga, S.Kom, M.S	Aswin Suharsono, S.T, M.T
NIK. 201405 890720 1 001	NIK. 201102 840919 1 001

Mengetahui
Ketua Jurusan Teknik Informatika

Tri Astoto Kurniawan, S.T, M.T, Ph.D
NIP. 197105182003121001

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar pustaka.

Apabila ternyata didalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 23 Januari 2017

Lestari Sintika Syarah

NIM: 125150201111089

KATA PENGANTAR

Syukur Alhamdulillah penulis ucapkan atas kehadirat Allah Subhanahu wa Ta'ala, karena berkat rahmat dan anugerah-Nya penulis dapat menyelesaikan skripsi yang berjudul "Analisis Performansi komunikasi *Video Conference* Pada WebRTC dengan Menggunakan *Framework* RTCMultiConnection" shalawat serta salam semoga selalu tercurah pada baginda Rasulullah Muhammad. Penulisan skripsi ini akan sulit terwujud apabila tidak ada rahmat dari sang pemilik langit dan bumi yaitu Allah Subhanahu wa Ta'ala serta bantuan dan dukungan dari berbagai pihak. Oleh karena itu, penulis ingin mengucapkan rasa terimakasih kepada:

1. Kedua orangtua serta adik yang selalu memberikan kasih sayang, dukungan yang terus menerus tanpa henti dan motivasi supaya penulis dapat menyelesaikan tugas akhir ini.
2. Bapak Adhitya Bhawiyuga, S.Kom, M.S dan Bapak Aswin Suharsono, S.T, M.T selaku dosen pembimbing yang telah banyak memberikan bimbingan, saran dan motivasi kepada penulis.
3. Bibi dan mamang yang telah membantu dalam semua proses menjalani kuliah dan selalu memberikan doa kepada penulis.
4. Keluarga Bumi palapa bi nuy, engkong, wa dede, neneng tya, atel, kakak, cika, yang selalu memberikan canda disaat putus asa, memberikan doa, memberikan dukungan kepada penulis.
5. Sahabat-sahabat penulis terutama neneng dita dan mba ell yang selalu sabar, yang selalu membantu, memberikan semangat, memberikan nasihat, memberikan canda dan tawanya dalam suka duka menjalani kuliah.
6. Dosen Fakultas Ilmu Komputer Universitas Brawijaya yang telah membantu penulis menjalani kuliah dan menyelesaikan skripsi.
7. Karyawan Fakultas Ilmu Komputer Universitas Brawijaya yang telah membantu penulis menjalani kuliah dan menyelesaikan skripsi.
8. Seluruh pihak yang telah membantu penulis secara langsung maupun tidak langsung yang tidak dapat disebutkan satu per satu.

Penulis menyadari bahwa skripsi ini tidaklah sempurna, kritik dan saran yang membangun sangat diharapkan darp para pembaca sehingga penulis dapat melakukan perbaikan yang diperlukan. Akhir kata, penulis berharap skripsi ini dapat bermanfaat bagi siapapun yang membacanya.

Malang, 23 Januari 2017

Penulis

lestarisintikasyarah@gmail.com

ABSTRAK

WebRTC adalah sekumpulan standar dari *World Wide Web Consortium* (WC3) dan *Internet Engineering Task* Force (IETF), WebRTC dapat melakukan proses komunikasi secara *real-time* pada *browser* yang telah mendukung fitur WebRTC. WebRTC dapat digunakan untuk layanan komunikasi *Video conference* dengan jumlah lebih dari dua pengguna. Pada penelitian ini webRTC yang dibangun menggunakan *framework* RTCMultiConnection dan node.js sebagai media signalingnya. Tujuan dari penelitian ini untuk mengetahui performansi komunikasi *video conference* pada WebRTC dengan melihat nilai parameter *delay*, *jitter*, *throughput* dan *packet loss* sebagai parameter uji. Hasil yang diperoleh dari penelitian komunikasi *video conference* pada WebRTC dengan menggunakan *framework* RTCMultiConnection menunjukkan bahwa semakin banyak jumlah *client* maka semakin meningkat nilai *delay, jitter, throughput* dan *packet loss* yang diperoleh. Dalam proses komunikasi dengan menggunakan minimum *bandwidth* sebesar 58 kbps dapat disimpulkan bahwa nilai parameter *delay* dan *jitter* dapat dikategorikan pada kategori baik menurut versi TIPHON, namun pada nilai parameter *throughput* dapat dikategorikan pada kategori kurang baik dan nilai *packets loss* masuk pada kategori sedang menurut versi TIPHON.

Kata kunci : WebRTC, RTCMultiConnection, *Video Conference*, Node.js, TIPHON.

ABSTRACT

WebRTC is standards from the World Wide Web Consortium (WC3) and Internet Engineering Task Force (IETF), WebRTC can be process real-time communication in Browser has supports WebRTC. WebRTC can be used for video conference communication services with more than users. In this research, WebRTC was built using RTCMultiConnection framework and node.js as media signaling. The purpose of this research is to determine the performance of the WebRTC video conference communication with seeing the value parameter delay, jitter, throughput and packet loss as test parameters. Results from the research of communication video conference on WebRTC using the framework RTCMultiConnection show that the more the number of clients it has increased the value of delay, jitter, throughput and packet loss. In the process of communication using a minimum bandwidth of 58 kbps can be concluded that the value of the parameter index delay and jitter can be categorized in good category by category version TIPHON, but on the value of the index throughput can be categorized in the unfavorable category and the index value of packets loss is entered in the category according to TIPHON version.

Keywords : WebRTC, RTCMultiConnectin, *Video Conference,* Node.js,TIPHON.

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR LAMPIRAN

BAB 1 PENDAHULUAN

1.1 Latar belakang

Salah satu teknologi komunikasi yang sekarang banyak digunakan adalah komunikasi yang dilakukan melalui layanan internet. Pemanfaatan teknologi komunikasi melalui jaringan internet menjadi komunikasi yang banyak diminati dan dibutuhkan oleh semua kalangan masyarakat. Hal ini dapat berlaku untuk perusahaan, kampus dan lain-lain (Aulia, 2006). Layanan komunikasi saat ini tidak hanya komunikasi melalui suara saja, melainkan terdapat layanan komunikasi yang dapat menggantikan tatap muka pada setiap pengguna, seperti layanan *video conference* yang sangat memudahkan pengguna untuk saling berinteraksi satu sama lain.

Video conference yaitu salah satu aplikasi multimedia yang memungkinkan komunikasi suara dan gambar yang bersifat *real time*. Bentuk aplikasi ini adalah percakapan lewat video dan audio antar pengguna secara langsung dan diharapkan dapat menggantikan fungsi tatap muka (Nurdiansyah, 2013). Sebelumnya untuk melakukan komunikasi *video conference* pengguna diharuskan memasang *plugin,* seperti pada layanan aplikasi *skype*. Saat ini telah dikembangkan suatu teknologi yang dapat digunakan untuk melakukan komunikasi, yaitu WebRTC. Teknologi tersebut berjalan pada *browser* yang bersifat *realtime* dan *reliable*.

Web Real Time Communication (WebRTC) merupakan sekumpulan standar dari *World Wide Web Consortium* (WC3) dan *Internet Engineering Task* Force (IETF), dimana WebRTC dapat melakukan proses komunikasi secara *real-time* pada *browser* yang telah mendukung fitur WebRTC. WebRTC memanfaatkan fitur dari HTML5 dan Javascript tanpa *plugin* tambahan, sehingga memungkinkan untuk mengaktifkan audio dan kamera yang terdapat pada perangkat keras. (Phil Edholm E. at.). WebRTC dapat digunakan untuk layanan komunikasi *Video conference* dengan jumlah lebih dari dua pengguna dalam melakukan proses komunikasi.

WebRTC akan merevolusi komunikasi web, namun teknologi ini masih dalam pengembangan dan standarisasi proses [Taheri, 2015]. Oleh karena itu diperlukan lebih banyak penelitian yang harus dilakukan untuk melihat kinerja dari WebRTC. Seperti dari proses pembentukan koneksi, media komunikasi dan data *channel*. Parameter yang perlu diteliti dalam implementasi WebRTC adalah pengaruh jumlah *client* WebRTC dan besar *bandwidth* dalam satu sesi *video conference* saat proses komunikasi sedang berlangsung.

Dari permasalahan diatas penulis melakukan analisis performansi terhadap layanan *video conference* WebRTC dengan menggunakan *framework* RTCMulticonnection. Penelitian ini terfokus pada proses komunikasi ketika sedang berlangsung, sehingga penulis dapat mengetahui *Quality of Service* pada layanan WebRTC. Parameter yang digunakan untuk mengetahui baik atau buruk dari layanan WebRTC adalah *delay, jitter, throughput* dan *packet loss,* kemudian hasil dari parameter penelitian dibandingkan dengan standar versi TIPHON.

1.2 Rumusan masalah

Berdasarkan pada permasalahan yang akan diangkat, maka rumusan masalah dikhususkan pada:

1. Bagaimana hasil dan analisis parameter QoS pada komunikasi *video conference* WebRTC dengan jumlah *client* dan *bandwidth* yang berbeda?
2. Apakah nilai *delay, jitter, throughput dan packets loss* pada komunikasi *video conference* WebRTC memenuhi standar komunikasi yang baik menurut standar TIPHON?

1.3 Tujuan

Tujuan dari penelitian ini adalah untuk mengetahui performansi dari komunikasi *video conference* pada WebRTC dengan jumlah *client* dan variasi *bandwith* yang berbeda.

1.4 Manfaat

Manfaat dari penelitian ini adalah:

1. Dapat mengetahui QoS dari WebRTC *video conference* dengan menggunakan *framework RTCMultiConnection*.
2. Dapat dijadikan sebagai referensi dalam membentuk jaringan *video conference.*
3. Memberikan informasi kualitas layanan *video conference* pada WebRTC dari hasil pengujian yang dilakukan.

1.5 Batasan masalah

Batasan masalah dalam penelitian diberikan dengan tujuan agar pembahasan tidak melebar dan lebih terperinci. Adapun penelitian ini dibatasi oleh hal-hal sebagai berikut:

1. Melakukan analisis WebRTC pada komunikasi *video conference* menggunakan *framework RTCMultiConnection*.
2. Pengujian komunikasi *video conference* dilakukan sebanyak dua, tiga, lima dan enam *client*.
3. Parameter yang dianalisis adalah *delay, jitter, packet loss* dan *throughput.*
4. Tidak membahas pada proses pembentukan terjadinya komunikasi, akan tetapi lebih terfokus pada proses komunikasi ketika telah berlangsung.
5. *Browser* yang digunakan adalah Mozilla Firefox dan Google Chrome.
6. *Software* analisis kinerja jaringan menggunakan Wireshark.
7. *Bandwidth* pengujian adalah 58 kbps dan 95 kbps.
8. Dalam proses penelitian *codec* yang digunakan adalan VP8.

1.6 Sistematika pembahasan

Sistematika penulisan dan gambaran untuk setiap bab pada skripsi ini adalah sebagai berikut:

BAB 1 PENDAHULUAN

Berisi tentang penjelasan latar belakang masalah yang terkait, rumusan masalah, batasan masalah, tujuan, manfaat dan sistematika penulisan.

BAB 2 LANDASAN KEPUSTAKAAN

Memuat kajian teori sebagai landasan penyusunan topik skripsi yang diawali dengan penjelasan mengenai pengertian, mekanisme WebRTC yang dapat melakukan proses *real-time communication* tanpa harus melakukan proses pemasangan *plugin* kemudian terdapat penjelasan mengenai protokol yang digunakan pada layanan *video conference*, serta penjelasan mengenai kualitas layanan *video conference* dengan menggunakan parameter *delay, jitter, throughput* dan *packet loss* sebagai parameter dari proses analisis dengan mengacu pada standar TIPHON.

BAB 3 METODOLOGI

Metodologi penelitian terdiri dari sekumpulan metode yang digunakan untuk menyelesaikan masalah dalam penelitian. Diawali dengan studi pendahuluan yang terdiri dari studi literatur, kemudian proses penerapan lingkungan penelitian yang terdiri dari proses pembangunan jaringan layanan *video conference* pada WebRTC, proses pembatasan *bandwidth* menggunakan *software* Netlimiter, selanjutnya adalah metode pengumpulan data dan proses mendapatkan data pada saat melakukan suatu skenario, untuk menjawab rumusan masalah, dilakukan beberapa skenario diterapkan sebagai metode pengujian yang dilakukan. Hasil pengujian diolah terlebih dahulu yang dapat dijadikan pada proses analisis.

BAB 4 LINGKUNGAN PENELITIAN

Lingkungan penelitian membahas mengenai hal yang harus diimplementasikan untuk mendukung penelitian.

BAB 5 PENGAMBILAN DATA

Pengambilan data membahas tentang teknik pengambilan data dari sistem yang telah diimplementasikan.

BAB 6 PEMBAHASAN

Pembahasan menjelaskan hasil analisis dari data hasil implementasi.

BAB 7 PENUTUP

Memuat kesimpulan yang merangkum jawaban dari rumusan masalah berdasarkan hasil penelitian yang sudah didapatkan dan saran untuk pengembangan penelitian lebih lanjut.

BAB 2 LANDASAN KEPUSTAKAAN

2.1 *Web Real - Time Communication* (WebRTC)

WebRTC adalah sekumpulan standar dari *World Wide Web Consortium* (WC3) dan *Internet Engineering Task* Force (IETF) yang dimulai sejak awal 2011, dimana WebRTC ini dapat melakukan proses *real-time communication* (RTC) pada browser yang telah mendukung fitur WebRTC. Gambar 2.1 menunjukan logo dari WebRTC.

Gambar 2.1 Logo WebRTC
Sumber: [webrtc.org]

WebRTC menyediakan *Application Programming Interface (API)* untuk mengakses perangkat asli, melakukan proses berbagi *stream* audio / video. WebRTC merupakan protokol komunikasi yang mendasari API JavaScript dalam *browser* untuk pengembangan web. WebRTC dapat digunakan untuk komunikasi *real time* antar *browser*, terutama dalam komunikasi audio dan video dengan berbasis *peer-to-peer*, WebRTC memiliki beberapa fungsi seperti:

1. Dapat mengakses *stream* audio dan video dari kamera dan mikrofon.
2. Dapat membentuk sebuah koneksi diantara *peer* dan *stream* audio dan video.
3. Dapat melakukan proses komunikasi transfer data.

2.1.1 API WebRTC

1. MediaStream (getUserMedia): Salah satu API yang menyediakan layanan untuk dapat memperoleh *stream* audio dan video yang didapatkan dari kamera dan mikrofon tanpa memerlukan *plugin*, karena HTML5 menyediakan API Javascript yang dapat mengakses *hardware* dari komputer. getUSerMedia akan meminta izin untuk mendapatkan akses mikrofon dan kamera dari pengguna, sehingga pengguna dapat berkomunikasi dengan suara (Manson Rob, 2013). Perintah yang digunakan untuk mengambil Video/Audio adalah navigator.getuserMedia, navigator.WebkitGetUserMedia, navigator. mozGetUserMedia berbeda-beda untuk setiap *browser*.
2. RTCPeersConnection: Menyediakan layanan untuk membentuk sebuah koneksi diantara *peers* dan *stream* audio atau video. Untuk menginisialisasi

proses RTCPeerConnection terdapat dua proses yang harus dilakukan yaitu memastikan kondisi dari media local seperti codec, informasi ini digunakan untuk bertukar informasi dalam mekanisme *offer* dan *answer*, mendapatkan *network address* yang dikenal dengan *finding candidate*.

3. RTCDataChannel: Menyediakan layanan untuk melakukan proses komunikasi data, terdiri dari sebuah API mirip dengan *WebSockets* untuk transfer data diantara peers. Hal ini dapat mendukung pengiriman yang *realiable* yang memanfaatkan protokol yang *reliable* (TCP) dan *unrealible* (UDP).

Pada Tabel 2.1 menjelaskan tentang macam – macam aplikasi dari WebRTC dan komponen yang mendukung terhadap aplikasi tersebut, seperti pada aplikasi *video conference* melibatkan fungsi dari GetUserMedia dan RTCPeerConnection.

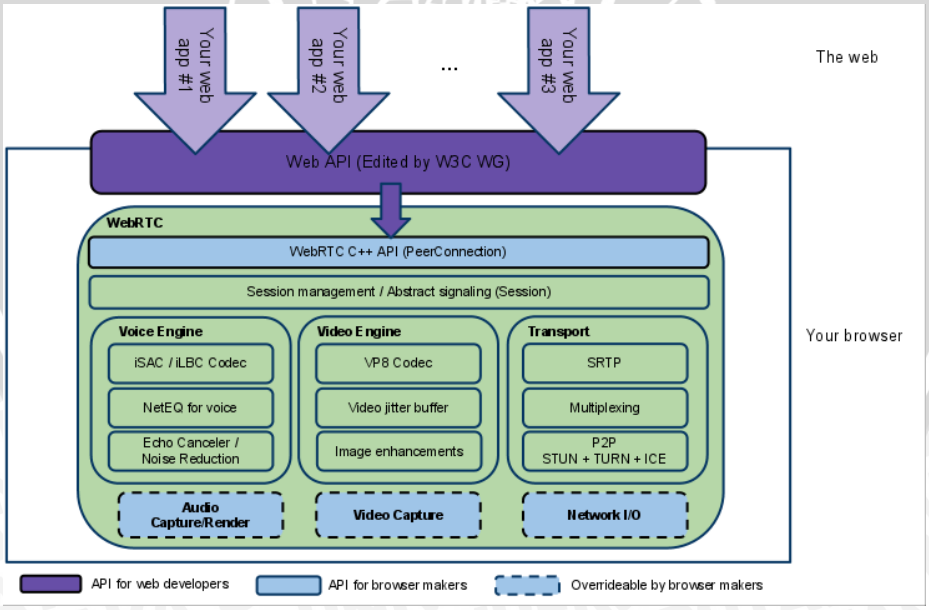
Tabel 2.1 Aplikasi WebRTC

Application	GM	PC	DC
Media Recording,Gesture Recognition, Voice Translation	✓		
Voice Call, video Conferencing, Screen Sharing	✓	✓	
Messaging, File Sharing, Data Transfer		✓	✓
Streaming Local Media Files		✓	

Sumber : (Sajjad, 2015)

2.1.2 Arsitektur WebRTC

WebRTC merupakan aplikasi web yang dapat melakukan komunikasi *real time* berbasis web. WebRTC bertujuan untuk membantu dalam proses pembentukan suatu *platform* yang *real time communication* yang bekerja di beberapa browser. Arsitektur dari WebRTC ditunjukan pada Gambar 2.2.



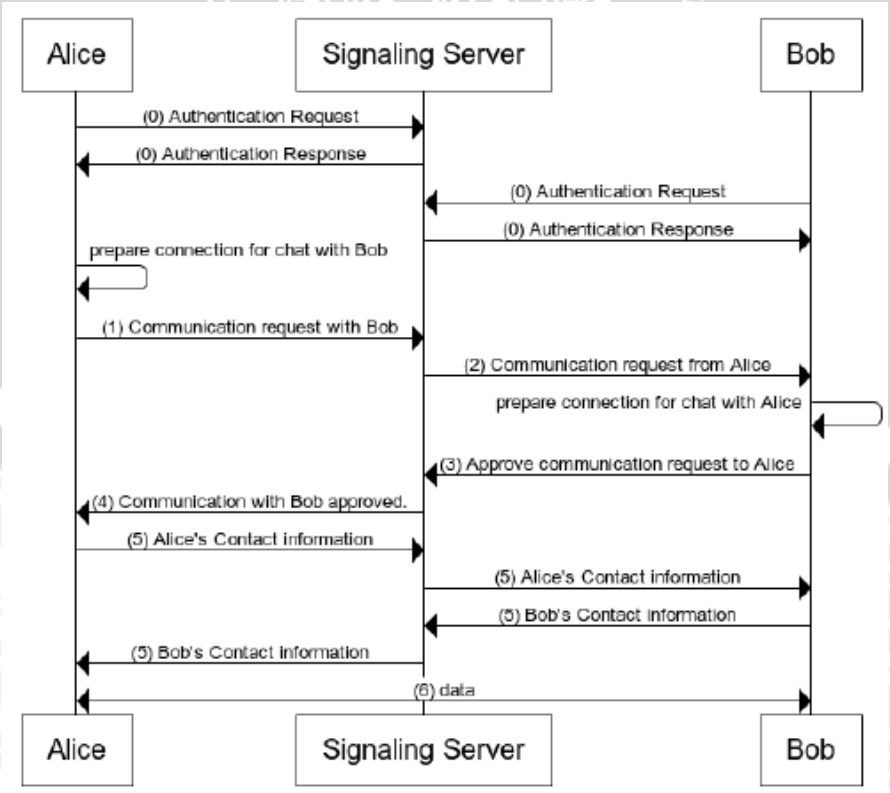
Gambar 2.2 Arsitektur WebRTC
Sumber: [Webrtc.org]

Pada Gambar 2.2 Terdapat dua layer yaitu pada layer Browser dan pada Web

- Web API: Digunakan *oleh third party developers* untuk membentuk sebuah web seperti aplikasi berbasis *video chat*.
- WebRTC Native C++ API: Sebuah layer API yang mendukung web yang membuat implementasi lebih mudah.
- STUN (*Session Traversal Utilities for NAT*): Digunakan untuk melakukan komunikasi *peer- to peer*. Protokol STUN digunakan untuk mendapatkan *external network address.*
- ICE dan TURN: Untuk membangun koneksi *peer to peer* digunakan untuk transfer data. Sebuah komponen yang memperbolehkan mekanisme STUN / ICE untuk membentuk suatu koneksi pada jaringan yang berbeda.
- *VideoEngine*: Sebuah framework video yang diambil dari kamera untuk ditransmisikan pada jaringan kemudian ditransmisikan pada layar.
- VP8: Video codec dari WebM *project*.
- Video *Jitter Buffer*: Jitter dinamis untuk video, membantu untuk menutupi efek dar *jitter* dan *packet loss* pada kualitas video.

2.1.3 Mekanisme WebRTC

Mekanisme WebRTC yang dilakukan oleh Alice yang mencoba melakukan proses panggilan kepada Bob ditunjukan pada Gambar 2.3.



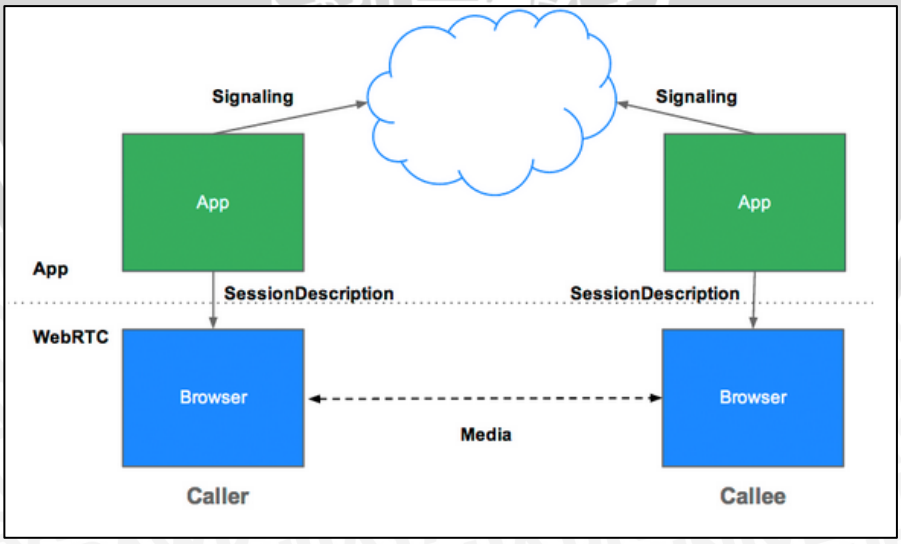
Gambar 2.3 Mekanisme WebRTC
Sumber: [webrtc.org]

Mekanisme WebRTC sesuai dengan gambar 2.3 adalah sebagai berikut:

1. Alice memulai koneksi dengan mengirimkan permintaan ke signaling server untuk melakukan komunikasi dengan Bob, pada proses ini Alice memberikan informasi tentang deskripsi komunikasi yang digunakan, isi dari *session deskripsi* tersebut berupa *codec* media yang digunakan.
2. Infomasi diatas akan dikirimkan ke server signaling dan setelah koneksi berlangsung maka informasi tersebut digunakan sebagai otentikasi antar kedua klien yang bertujuan untuk memastikan bahwa Alice dan Bob dapat melakukan komunikasi.
3. Signaling server mengotentikasi pesan dari Alice dan meneruskannya ke Bob.
4. Bob dapat melakukan proses panggilan atau menolak permintaan komunikasi dari Alice. Jika Bob menyetujui permintaan dari Alice, maka Bob membuka saluran komunikasinya dan mengirimkan respon ke server signaling.
5. Server signaling meneruskan pesan dari Bob ke Alice.
6. Misalkan Bob menyetujui permintaan dari Alice, maka mereka langsung melakukan pertukaran informasi komunikasi, termasuk IP yang digunakan untuk melakukan komunikasi, jenis komunikasi (TCP/UDP), dan informasi traversal seperti NAT (IP / Port Binding).
7. Kedua klien memulai *direct* komunikasi antar *peer*.

2.1.4 Tahap *Signalling*

Proses *signalling* menggunakan protokol SIP (*Session Initiation Protocol*). SIP bertujuan untuk menjadi protokol universal yang dapat mengintegrasikan suara dan data pada jaringan. SIP adalah protokol kontrol sesi atau panggilan yang didefinisikan oleh IETFRFC 3261 . Proses Signalling pada WebRTC ditunjukkan dalam Gambar 2.4.



Gambar 2.4 Proses *Signalling* pada WebRTC

Mekanisme signalling pada WebRTC sesuai gambar 2.4 adalah sebagai berikut:

1. Alice dan Bob perlu bertukar informasi mengenai jaringan, untuk menemukan jaringan dan port yang digunakan dengan menggunakan ICE Framework proses ini dinamakan dengan proses *finding candidate*. Dimana proses ini dilakukan dengan cara:
   - Alice membuat *peer object* untuk mendapatkan *candidates* menggunakan *ICE candidates* (Interactive Connectivity Establishment).
   - Alice akan mengirim *ICE candidates* menggunakan *signaling channel* seperti: Websocket. Untuk membuat koneksi antar dua *client* digunakan proses signaling, misalkan Alice ingin melakukan sebuah percakapan komunikasi permintaan ke signaling server untuk berkomunikasi dengan Bob, kemudian server meneruskan pesan Alice kepada Bob.
   - Kemudian Bob akan menyimpan *ICE candidates* yang diterima untuk kemudian digunakan.
2. Alice dan Bob perlu bertukar informasi mengenai media yang dimiliki proses ini dinamakan *offer-answer*.
   - Allice membuat *CreateOffer()* (*local Description*) tentang informasi media yang digunakan, kemudian dikirim kepada Bob.
   - Bob menyimpan local description yang dikirim Alice (remote Description).
   - Bob membuat *CreateOffer()* (*local Description*), kemudian dikirim ke Alice.
   - Alice menyimpan local description yang dikirim Alice (remote Description).

2.2 UDP

UDP merupakan mekanisme pengiriman datagram dari satu aplikasi ke aplikasi lain. UDP berfungsi untuk meyisipkan *field number port* sumber dan tujuan untuk layanan *multiplexing*. Tidak ada umpan balik untuk mengontrol tingkat infromasi yang dimiliki pesan. UDP memberikan transmisi kanal yang bersifat *unreliable* yaitu pada protokol ini data tidak dijamin akan sampai pada tujuan yang benar dan dalam kondisi yang benar, kemudian UDP bersifat *Connectionless* berarti tidak diperlukannya suatu bentuk hubungan terlebih dahulu untuk mengirimkan data. UDP menyediakan mekanisme untuk mengirim pesan-pesan ke sebuah protokol lapisan aplikasi aatu proses tertentu dimana di dalam sebuah host dalam jaringan yang menggunakan TCP/IP. Header UDP berisi *field source Process Identification* dan *Destination Process Identification.*

Pada sebuah pengiriman datagram yang menggunakan UDP harus mengetahui identitas tujuan berupa alamat IP dan port sumber dan tujuan.

UDP digunakan untuk multimedia *streaming* yang sangat memberikan toleransi kehilangan segment cukup baik dan yang sangat tidak sensitive terhadap kerusakan atau kehilangan segment. UDP merupakan transport yang agresif dan selalu mencoba untuk mengambil *bandwidth* yang tersedia.

2.3 Video

Video merupakan kumpulan gambar yang bergerak dengan kecepatan *frame* tertentu (*fps*).

2.3.1 Format Video

Video format mencakup dua perihal yaitu *Container*s atau *Wrappers* dan *Codecs*. *Container* mendeskripsikan struktur dari sebuah *file*, yakni dimana beberapa bagian yang disimpan, bagaimana mereka disisipkan, *Container* digunakan untuk mengemas video dan komponennya dan teridentifikasi dengan ekstensi *file* seperti .AVI, .MP4, atau .MOV. Sedangkan *Codec* adalah sebuah cara mengenkodekan suara atau video menjadi aliran bit-bit data.

2.3.2 *Video Conference*

Video conference yaitu salah satu aplikasi Multimedia yang memungkinkan komunikasi data, suara dan gambar secara *real time*. Bentuk aplikasi ini adalah percakapan via video dan audio antar pengguna secara langsung dan diharapkan dapat menggantikan fungsi tatap muka secara langsung [Nurdiansyah, 2013]. Karakteristik dari sebuah video yaitu *high bit rate*, video membutuhkan *bit rate* yang tinggi untuk proses transfer data, video didistribusikan antara range 100 kbps untuk kualitas rendah dan 3 Mbps untuk kualitas gambar yang tinggi.

Video conference memiliki dua tipe, yakni:

1. *Video conference point-to-point* merupakan metode sederhana yang menggunakan dua buah komputer untuk saling terhubung menggunakan single IP address. Biasanya *point-to-point* digunakan untuk *video phone calls* yang sifatnya lebih personal. *Video conference point-to-point* menggunakan *webcam* dan *microphone* yang telah terpasang pada personal computer pengguna, sehingga bisa berbicara dan saling melihat lawan bicara.
2. *Video conference multipoint* merupakan metode untuk melakukan *video conference* dengan melibatkan lebih dari dua pengguna menggunakan satu *point* sebagai *server* dan beberapa pengguna lainya sebagai *client* agar dapat terhubung dan melakukan panggilan video.

2.3.3 Standar Kompresi Data Video

Kompresi video adalah suatu proses mengurangi dan menghapus data video sehingga file video dapat secara efektif dikirim dan disimpan. Proses ini melibatkan penerepan algoritma untuk membuat file terkompresi yang siap untuk ditransmisikan atau disimpan, algoritma diterapkan untuk menghasilkan sebuah video yang menunjukan konten yang sama sebagai sumber asli video, waktu yang dibutuhkn untuk kompresi, dekompresi dan menampilkan file disebut *latency*. Pada WebRTC khusunya pada layanan *video conference* terdapat codec yang yang digunakan sebagai kompresi data video yaitu codec Vp8.

VP8 adalah *video compression format* yang dimiliki oleh Google dan dibuat oleh On2 Technologies sebagai penerus VP7 yang memiliki lisensi public bebas

royalti. VP8 menawarkan kualitas *real time* pengiriman video yang sangat tinggi dan daya yang dibutuhkan oleh CPU akan tetap.

Dengan teknik kompresi yang ada pada saat ini, kita dapat menghemat sebuah kanal video sekitar 30 kbps dan kanal suara menjadi 6 kbps, artinya sebuah saluran internet yang tidak terlalu cepat sebetulnya dapat digunakan untuk menyalurkan video dan audio sekaligus, artinya minimal sekali kita harus menggunakan kanal 64 kbps ke Internet

2.4 Parameter *Quality of Service* (QoS)

Quality of Service (Qos) adalah kemampuan suatu jaringan untuk menyediakan layanan yang lebih baik pada *traffic* dan data tertentu pada berbagai jenis *platform* teknologi (Saputra, 2010). QoS tidak diperoleh langsung dari infrastruktur yang ada, tetapi diperoleh langsung dengan mengimplementasikannya pada jaringan bersangkutan. Tujuan utama dari QoS adalah untuk mendukung prioritas termasuk *bandwidth* yang aktual, pengontrolan, jitter dan latency.

Penelitian ini menggunakan empat parameter QoS yaitu *delay*, *jitter*, *packet loss* dan *throughput* yang diujikan dalam *bandwidth* dan jumlah *client* yang berbeda. Penulis menggunakan standarisasi TIPHON untuk mengetahui hasil dari analisisjaringan.

2.4.1 *Delay*

Delay adalah waktu yang dibutuhkan untuk mengirimkan data dari sumber ke tujuan. Adanya variasi waktu tunda dalam transmisi layanan aplikasi internet menimbulkan masalah tersendiri, yaitu paket-paket yang datang terlambat yang dapat mengganggu data yang diterima oleh pengguna. Dalam perancangan jaringan *video conference, delay* merupakan salah satu permasalah yang harus diperhitungkan karena kualitas suara dan gambar menjadi baik sangat tergantung dari waktu *delay* yang terjadi. Delay adalah waktu yang dibutuhkan untuk mengirim sebuah paket dari sumber menuju ke tujuan (ujung ke ujung). Besarnya *delay* maksimum yang direkomendasikan TIPHON untuk aplikasi suara dan gambar adalah 150 ms, sedangkan *delay* maksimum dengan kualitas suara dan gambar yang masih dapat diterima pengguna adalah 300 ms. Kategori kinerja jaringan berdasarkan nilai *delay* dapat dilihat pada Tabel 2.2

Tabel 2.2 Kategori Kualitas *Delay*

Kategori Delay	Besar Delay
Sangat Baik	< 150 ms
Baik	150 – 300 ms
Sedang	300 – 450 ms
Jelek	> 450 ms

Sumber: [TIPHON, 2012]

Untuk mencari besar *delay* dalam komunikasi *video conference* dapat dicari menggunakan persamaan 2.1.

$$\boldsymbol{Delay = \frac{waktu\ pengamatan}{jumlah\ paket}} \quad (2.1)$$

Waktu pengamatan didapatkan dari berapa lama waktu yang digunakan dalam proses *capture* jaringan sedangkan jumlah paket merupakan jumlah dari paket yang telah sampai pada tujuan.

2.4.2 *Jitter*

Jitter merupakan variasi *delay* yang terjadi akibat adanya selisih waktu atau interval antara kedatangan paket di penerima. Untuk mengatasi *jitter* paket data yang datang dikumpulkan dulu dalam *jitter buffer* selama waktu yang telah ditentukan sampai paket dapat diterima pada sisi penerima dengan urutan yang benar. Tabel 2.3 merupakan kategori yang digunakan dalam menentukan kategori nilai *jitter* pada jaringan yang akan diamati.

Tabel 2.3 Kategori *Jitter*

Kategori	Nilai Jitter
Sangat Baik	0 ms
Baik	0 – 75 ms
Sedang	76 – 125 ms
Jelek	125 – 225 ms

Sumber: TIPHON, 2012

Untuk mencari besar *jitter* dapat dicari menggunakan Persamaan 2.2.

$$\boldsymbol{Jitter\ rata-rata = \frac{Total\ variasi\ delay}{Jumlah\ paket\ yang\ diterima}} \quad (2.2)$$

Total variasi *delay* merupakan total dari selisih antara paket satu dengan paket kedua begitupun selanjutnya kemudian total variasi tersebut dibagi dengan jumlah paket yang diterima.

2.4.3 *Packet Loss*

Packet Loss merupakan jumlah paket yang hilang dibandingkan dengan paket yang diterima (Tanenbaum, 2003). Paket yang hilang dapat disebabkan oleh beberapa kemungkinan, antara lain adalah:

a. Terjadinya *over load* dalam jaringan.
b. Tabrakan atau tumbukan dalam jaringan.
c. *Error* yang terjadi pada media fisik.
d. Pengiriman data pada waktu yang bersamaan dengan menggunakan sebuah saluran secara bersama-sama.

Semakin kecil nilai *packet loss* dalam suatu jaringan maka semakin baik pula kinerja yang dimiliki jaringan tersebut. *Packet loss* biasanya dinyatakan dalam ukuran persen (%) untuk mencari besar *packet loss* dalam komunikasi. Biasanya *packet loss* yang dapat ditolerir adalah sebesar 10%. Secara umum terdapat empat kategori penurunan performansi jaringan berdasarkan nilai *packet loss*

sesuai dengan versi TIPHON *Telecomunications and Internet Protocol Harmonization Over Networks*, yaitu seperti ditunjukan pada Tabel 2.4.

Tabel 2.4 Kategori *Packet Loss*

Kategori Degradasi	Packet loss
Sangat Baik	0%
Baik	3%
Sedang	15%
Jelek	25%

Sumber: TIPHON, 2012

Untuk mencari besar *packet loss* dapat dicari menggunakan Persamaan 2.3

$$\textbf{Paket loss}\ (\%) = \frac{\textbf{paket yang dikirim} - \textbf{paket yang diterima}}{\textbf{Paket yang dikirim}} \textbf{x}\ \textbf{100}\ \% \qquad (2.3)$$

2.4.4 *Throughput*

Throughput didefinisikan sebagai jumlah paket yang diterima di sisi penerima dengan benar tiap satuan waktu. Tabel 2.5 merupakan kategori yang digunakan dalam menentukan kategori *throughput* pada jaringan yang akan diamati.

Tabel 2.5 Kategori *Throughput*

Kategori *Throughput*	*Throughput*
Sangat Baik	100%
Baik	75%
Sedang	50%
Jelek	< 25%

Sumber: TIPHON, 2012

Besarnya nilai *throughput* pada aplikasi *video conference* dapat ditentukan dengan persamaan 2.4.

$$\textbf{Throughput} = \frac{\textbf{paket data yang diterima}}{\textbf{lama pengamatan}} \qquad \textbf{(2.3)}$$

2.5 Wireshark

Wireshark adalah sebuah alat ukur penganalisispaket jaringan. *Network packet analyzer* akan mencoba menangkap paket-paket jaringan dan berusaha untuk menampilkan semua informasi di paket tersebut sedetail mungkin. *Network packet analyzer* dapat diumpamakan sebagai alat untuk memeriksa apa yang sebernanya sedang terjadi di dalam kabel jaringan. *Wireshark* merupakan tool *open source* gratis.

Wireshark memiliki fungsi sebagai *packet sniffer*, yang memiliki 3 proses utaman, antara lain:

1. *Capture.* Wireshark menangkap paket data yang melewati adapter kabel jaringan maupun wireless pada pengguna.

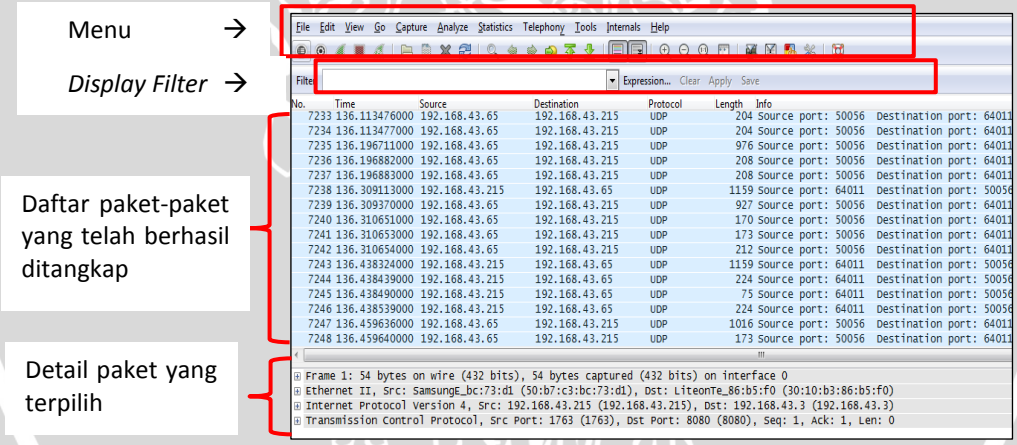
2. *Decode. Decode* atau decode adalah proses mengubah suatu protocol menjadi sebuah informasi yang dapat diketahui.
3. *Analyze*. *Analyze* membuat wireshark menjadi program paling disenangi *Network Analyze Wireshark* karena dapat menampilkan beberapa parameter terkait performansi maupun karakteristik paket data.

Cara pengoperasian dari *Wireshark* adalah sebagai berikut:

1. Aktifkan aplikasi *Network packet analyzer* pada masing-masing PC client

   Start → All Programs → aplikasi *Network Analyze Wireshark*.
2. Pilih *interface* dengan klik tab ***capture* → *interface*.**
3. Lakukan pengamatan selama 2 menit.
4. Pilih jenis *protocol* yang akan dianalisis. Contoh yang akan dianalisis adalah *protocol* UDP.
5. Untuk menganalisis dan rekapitukasi dari tangkapan paket klik ***Statistics* → *Summary*.**

Pada Gambar 2.5 merupakan contoh tampilan *Wireshark* yang sedang meng-*capture* paket-paket pada suatu sistem jaringan.



Gambar 2.5 Contoh *wireshark* yang sedang Meng-*capture* paket-paket pada suatu sistem jaringan.

Sumber : Tutorial *Wireshark*

Keterangan:

1. Menu : Menu-menu yang tersedia pada *Wireshark.*
2. *Display Filter* : Sebuah kolom yang dapat diisi dengan sintaks-sintaks untuk membatasi paket-paket apa saja yang akan ditampilkan pada *list* paket.
3. Daftar Paket : Menampilkan paket-paket yang berhasil ditangkap oleh aplikasi *Wireshark,* berurutan dari paket pertama yang ditangkap, dan seterusnya.

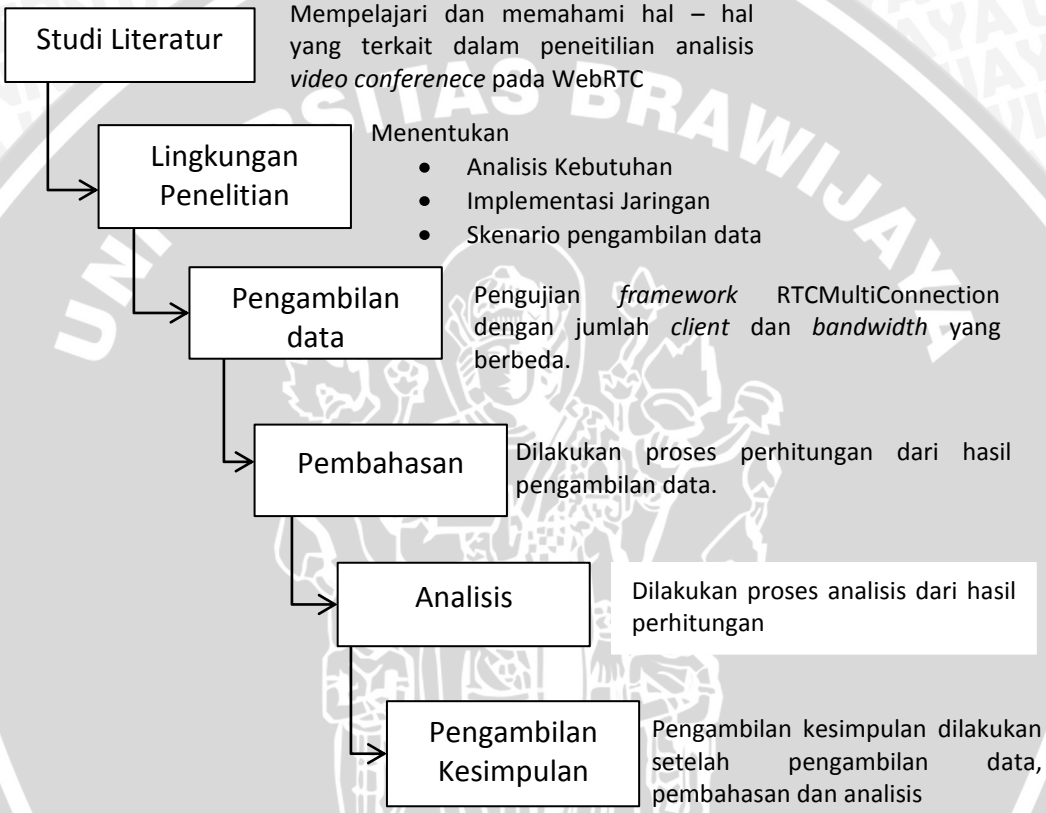
4. Detail Paket : Menampilkan detail paket yang terpilih pada daftar paket

Pada daftar bagian daftar paket, terdapat kolom-kolom seperti berikut ini :

1. *Time* : Menampilkan waktu saat paket-paket tersebut ditangkap.
2. *Source* : Menampilkan alamat IP sumber dari paket data tersebut.
3. *Destination* : Menampilkan alamat IP tujuan dari paket data tersebut.
4. *Protocol* :Menampilkan protokol yang digunakan pada sebuah paket data.
5. *Info* : Menampilkan informasi secara detail tentang paket data tersebut.

BAB 3 METODOLOGI

Bab tiga menjelaskan metode yang digunakan dalam penelitian ini. Alur metode penelitian dapat dilihat pada Gambar 3.1. Gambar 3.1 menjelaskan tahapan atau langkah-langkah yang digunakan dalam penelitian. Penelitian ini merupakan penelitian non-implementatif analitik karena penelitian ini bermaksud untuk mengetahui hasil dari kualitas jaringan layanan *video conference* pada WebRTC dengan jenis *bandwidth* dan jumlah *client* yang berbeda.



Gambar 3. 1 Diagram Alir Strategi Penelitian

3.1 Studi literatur

Studi literatur dilakukan untuk mempelajari dan memahami hal-hal yang terkait dalam penelitian analisis performansi *video conference* pada layanan WebRTC. Studi literatur yang dilakukan adalah mengenai karakteristik, parameter, konsep dan mekanisme WebRTC, protokol yang digunakan pada layanan komunikasi *video conference* serta teori pendukung lain yang menunjang dalam penulisan skripsi ini.

Beberapa jurnal yang penulis jadikan referensi dalam penelitian penulis seperti jurnal pada penelitian Sajjad Taheri dengan judul "WebRTCBench: A Bencmark for Performance Assessment of WebRTC Implementations" yang menjelaskan bagaimana mengimplementasikan klien SIP berbasis web

menggunakan HTML5 dan node.js serta menghitung quality of service dari audio video tersebut yang menggunakan codec iSAC dan VP8.

3.2 Analisis Kebutuhan

Dalam proses perancangan komunikasi *video conference* WebRTC terdapat beberapa kebutuhan dari perangkat lunak yang penulis gunakan yaitu:

- Ubuntu 12.04 LTS (sebagai OS untuk server signaling).
- *Framework* RTCMulticonnection (untuk membuat webrtc *video conference*).
- Node.js (digunakan sebagai proses signaling).
- Wireshark v1.12.8 (digunakan untuk mencapture paket data yang berjalan pada jaringan lokal).
- Microsoft excel (digunakan untuk menghitung quality of service).

Untuk perangkat keras penulis menggunakan empat *Personal Computer* yang digunakan untuk melakukan proses pengujian. Spesifikasi dari perangkat keras yang digunakan adalah sebagai berikut

a. Laptop 1

Laptop 1 digunakan sebagai server dari *framework* RTCMultiConnection.

Jenis Laptop : Samsung
Processor : AMD A6-4400M APU with Radeon(tm) (2.70 GHz)
RAM : 2.00 GB
OS : Ubuntu 14.04
Tipe Sistem : 32-bit Operating System

b. Laptop 2

Laptop 2 digunakan sebagai client A yang akan melakukan komunikasi *video conference* dengan client B.

Jenis Laptop : Asus
Processor : core i3
RAM : 2.00 GB
OS : Windows 7
Tipe Sistem : 32-bit Operating System

c. Laptop 3

Laptop 3 merupakan laptop yang digunakan sebagai client B yang akan melakukan komunikasi *video conference* dengan client A.

Jenis Laptop : Asus
Processor : Intel Core i3
RAM : 2.00 GB
OS : Windows 8
Tipe Sistem : 64-bit Operating System

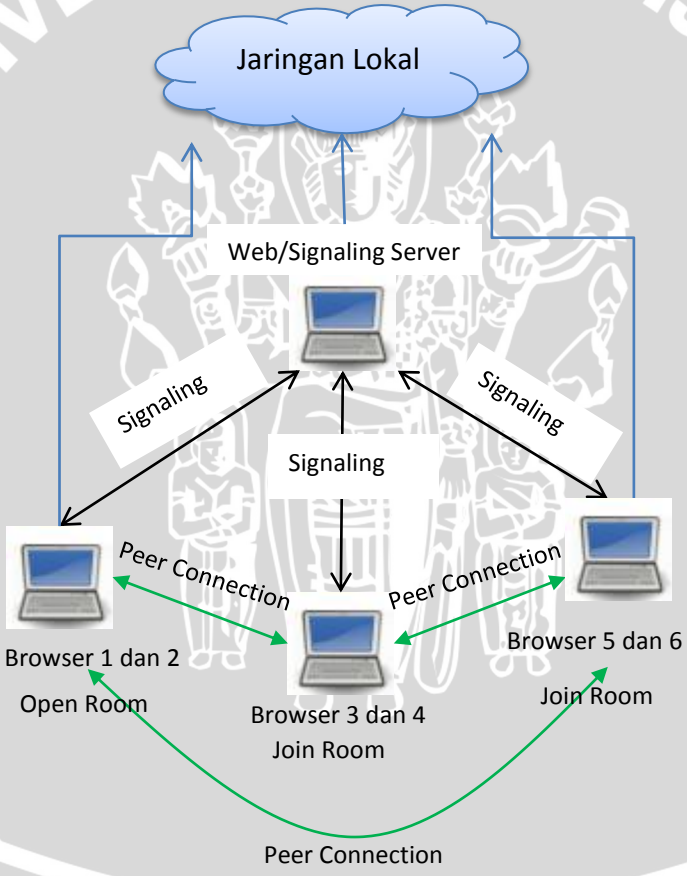
D. Laptop 4

Laptop 4 merupakan laptop yang digunakan sebagai client c yang akan melakukan komunikasi *video conference* dengan client A.

Jenis Laptop : Asus
Processor : Intel Core i3
RAM : 2.00 GB
OS : Windows 8

3.3 Perancangan Sistem

Perancangan sistem *video conference* menggunakan *framework* RTCMultiConnection yang dibangun dengan Node.js, karena *library* WebRTC dibangun kedalam beberapa *browser* yang tidak membutuhkan sebuah *plug-in*. Dalam *framework* ini menggunakan node.js untuk melakukan komunikasi. Gambar 3.2 merupakan Poses perancangan pada komunikasi WebRTC.



Gambar 3.2 Perancangan Sistem

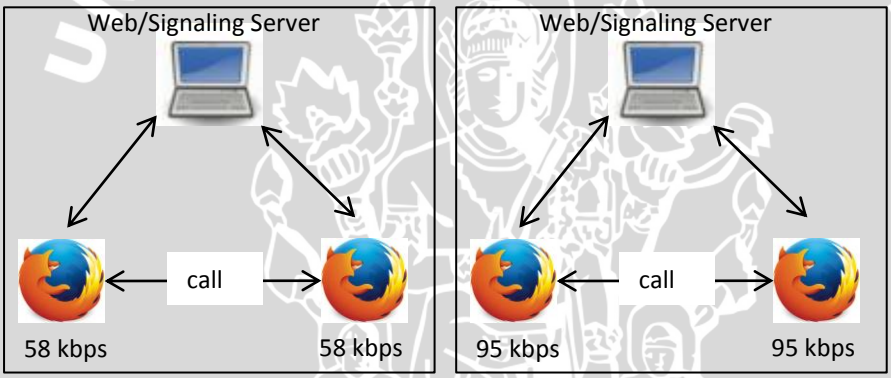
Pada *framework* RTCMultiConnection mekanisme komunikasi yang terjadi adalah seperti ditunjukan dan gambar 3.2. Semua perangkat terhubung pada jaringan yang sama, kemudian semua *browser* dapat mengakses halaman Web yang telah dibuat dengan memanggil IP_Server:9001 (port) pada web/signaling server.

Setelah semua browser mengakses halaman html, kemudian salah satu *browser* dapat melakukan "Open Room" atau membuka suatu komunikasi dengan memasukan kode atau nama unik untuk berbagi dengan *browser* lain. kemudian *browser* dapat melakukan proses akses getusermedia (akses webcam dan mikrofon). Apabila telah diperbolehkan mengakses media maka akan mucul tampilan video.

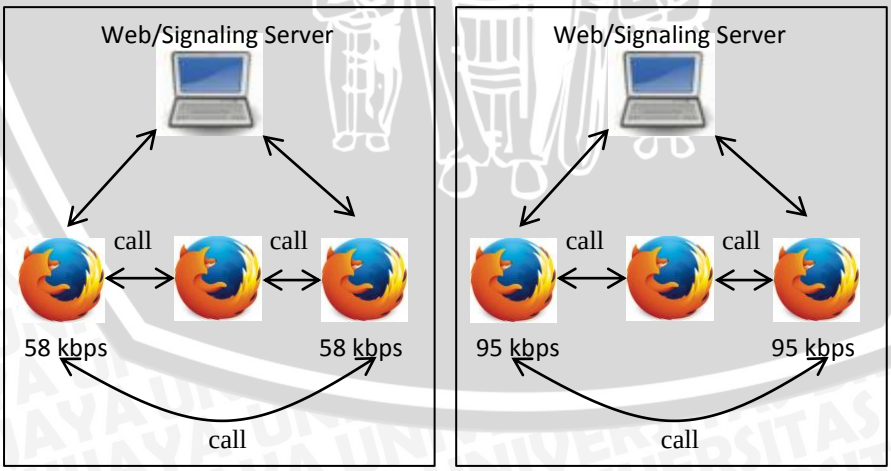
Untuk *browser* yang ingin bergabung dengan layanan *video conference* maka *browser* lainnya bisa memasukan kode yang sama dengan *browser yang* melakukan proses *"*Open Room*"*. Penulis menggunakan *framework* RTCMultiCOnnection sebagai *interface client*, dan node.js sebagai media signaling yang sudah terdapat modul socket.io.

3.4 Skenario Pengambilan Data

Pada penelitian ini terdapat proses komunikasi yang akan berjalan. Pada saat *client* sedang melakukan komunikasi akan dilakukan pengambilan data dengan melakukan *capture* data menggunakan Wireshark. Gambar 3.3 dan Gambar 3.4 menunjukan contoh skema panggilan pada Mozilla Firefex.

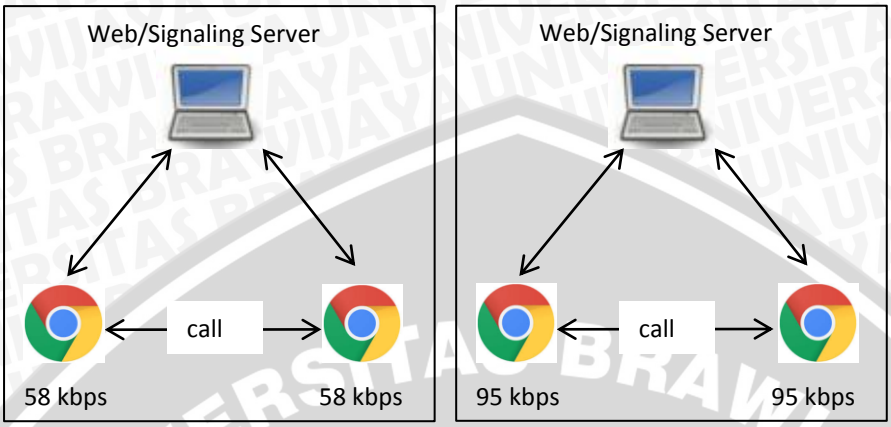


Gambar 3.3 Skema *video conference* Dua *Client*

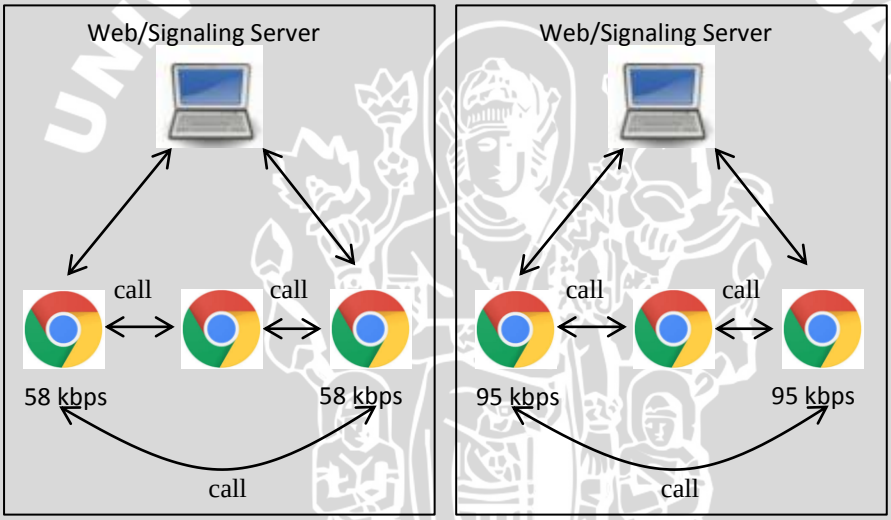


Gambar 3.4 Skema *video conference* Tiga *Client*

Gambar 3.5 dan Gambar 3.6 menunjukan skema panggilan pada Google Chrome.



Gambar 3. 5 Skema *video conference* Dua *Client*



Gambar 3.6 Skema *video conference* Tiga *Client*

Pengumpulan data pengujian dilakukan dengan melakukan proses komunikasi menggunakan *framework* RTCMultiConnection dengan variasi *bandwidth* dan *client* yang berbeda. Pengujian komunikasi *video conference* akan dilakukan dengan melakukan komunikasi antar *client* selama empat menit dengan tiga kali pengujian untuk setiap kondisi *bandwidth*.

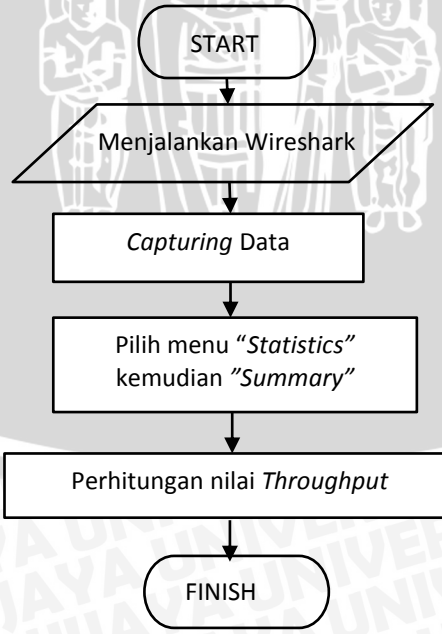
Ketika proses komunikasi berlangsung akan dilakukan pengambilan data menggunakan Wireshark. Dari hasil *capture* wireshark tersebut dapat dianalisis nilai dari masing-masing parameter. Pengambilan data kualitas *video conference* akan dilakukan selama pengujian komunikasi sedang berlangsung yaitu ketika proses media komukasi telah dilakukan, tanpa melakukan pengambilan data ketika proses pembentukan komunikasi. Terdapat skenario pengujian yang terbagi dalam dua kondisi *bandwidth* dan jumlan *client* yang berbeda dan pada tiap skenario akan dilakukan tiga kali pengujian, Tabel 3.1 merupakan tabel skenario pengujian yang dilakukan.

Tabel 3.1 Skenario Pengujian

Browser	Jumlah *Client*	*Bandwidth*
Mozilla Firefox	Dua	58 kbps
		95 kbps
	Tiga	58 kbps
		95 kbps
	Lima	58 kbps
		95 kbps
	Enam	58 kbps
		95 kbps
Google Chrome	Dua	58 kbps
		95 kbps
	Tiga	58 kbps
		95 kbps
	Lima	58 kbps
		95 kbps
	Enam	58 kbps
		95 kbps

Setelah proses pengambilan data, kemudian dilakukan proses perhitungan untuk mengetahui hasil dari parameter *throughput*, *delay*, *packet loss* dan *jitter* langkah dalam mengolah data yang diamati adalah sebagai berikut:

1. Throughput



Gambar 3.7 Proses Pengambilan nilai *Throughput*

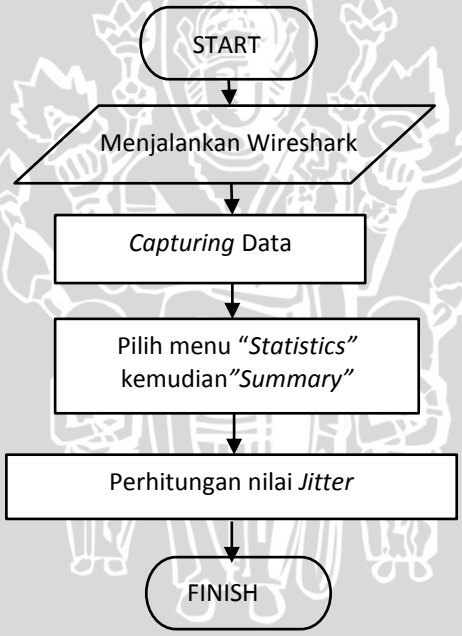
Gambar 3.8 menjelasakan alur mencari nilai *Throughput, p*ada Wireshark. Pada wireshark terdapat beberapa menu bar yaitu *File, Edit, Capture, Statistics, Interfaces, Analyze, Telephony, Help dan Window*. Untuk menghitung nilai *throughput* didapatkan data pada menu bar "Statistics" terdapat pilihan *Summary* untuk menampilkan ringkasan komunikasi yang telah ditangkap. Pada kotak dialog *Summary* terdapat berbagai macam informasi, yakni *packets*, *Time first packet into last packet, Avg Mbit/sec* dan lain-lain. Untuk menghasilkan nilai *throuhgput* dapat dilihat dari jumlah paket yang diterima (*packets*) dibagi dengan nilai lama pengamatan (*between first and last packet*).

2. *Delay*

Untuk menganalisis nilai *Delay* terlebih dahulu dilakukan *filtering* paket kemudian pilih menu *Statistics – Summary*, nilai delay ditunjukan pada *traffic Between first* and *last packet* dibagi jumlah paket yang diterima.

3. Jitter

Jitter merupakan variasi *delay* yang terjadi akibat adanya selisih waktu atau interval antara kedatangan paket di penerima.



Gambar 3.8 Proses Pengambilan nilai *Jitter*

Gambar 3.9 menjelasakan alur mencari nilai *Throughput, p*ada Wireshark. Pada wireshark terdapat beberapa menu bar yaitu *File, Edit, Capture, Statistics, Interfaces, Analyze, Telephony, Help dan Window*. Untuk menghitung nilai *throughput* didapatkan data pada menu bar " Statistics " terdapat pilihan *Summary* untuk menampilkan ringkasan komunikasi yang telah ditangkap, kemudian untuk mengetahui hasil dari *jitter* kita lakukan convert data hasil *capture* pada Microsoft excel kemudian dapat dihitung menggunakan persamaan 2.2.

3.5 Analisis hasil pengujian

Setelah melakukan proses pengujian maka data yang didapatkan dikumpulkan menjadi hasil dari proses pengujian. Selanjutnya dianalisis untuk menjawab permasalahan dalam proses penelitian. Analisis dilakukan dengan melihat hasil dari parameter *delay, jitter, throughput* dan *packet loss* untuk melihat performansi WebRTC pada layanan *video conference*. Selain itu dapat dilakukan analisis apakah nilai parameter masih memenuhi standar versi TIPHON.

3.6 Pengambilan Kesimpulan

Pengambilan kesimpulan dilakukan setelah melakukan pengambilan data, pembahasan dan analisis. Kesimpulan merupakan hasil akhir yang diperoleh dan didasarkan pada kesesuaian dengan teori dan praktek yang telah dilakukan. Isi dari kesimpulan diharapkan dapat mengetahui pengaruh jumlah *client* dan *bandwidth* terhadap parameter kualitas *video conference*. Kemudian untuk mengetahui baik dan buruk kualitas parameter *video conference,* maka disesuaikan dengan standarisasi versi TIPHON.

BAB 4 LINGKUNGAN PENELITIAN

Pada bab lingkungan penelitian ini membahas mengenai hal yang harus diimplementasikan untuk mendukung penelitian. Sesuai dengan rancangan pada Bab III, komunikasi *video conference* ini dibangun dan diujikan pada jaringan lokal. Pada tahap ini akan dilakukan implementasi antara lain instalasi dan konfigurasi perangkat lunak dan implementasi layanan *video conference* pada WebRTC dengan menggunakan *framework* RTCMulticonnection.

4.1 Implementasi Modul Node.js dan server menggunakan *framework* RTCMultiConnection

Proses pemasangan server RTCMultiConnection pada sistem operasi Ubuntu 14.04 adalah sebagai berikut:

1. Download file Zip framework RTCMultiConnection di situs https://github.com/muaz-khan/RTCMultiConnection kemudian ekstrak folder RTCMultiConnection dari file ZIP.

   Pada proses pemasangan server diperlukan modul node.js sebagai proses *signaling,* pada penelitian ini node.js dipasang pada komputer dengan OS Ubuntu 14.04 yang digunakan sebagai server. Berikut ini adalah langkah-langkah dalam menginstall node.js untuk Linux.

   ```
   #curl -sL https://deb.nodesource.com/setup_4.x |sudo -E bash-
   #sudo apt-get install -y nodejs
   ```

   Sumber : https://nodejs.org/en/download/package-manager/

2. Apabila node.js telah dipasang langkah selanjutnya adalah installasi *framework* RTCMultiConnection pada folder yang telah digunakan, kemudian pada langkah ini disimpan pada direktori Download kemudian masuk pada folder tersebut.

   ```
   #sudo su
   #cd /Download/RTCMultiConnection
   #npm install
   ```

3. Kemudian langkah selanjutnya menJalankan server RTCMultiConnection:

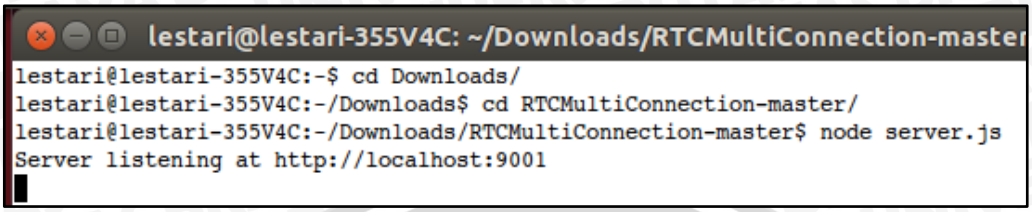
   Server yang digunakan dalam penelitian ini adalah satu unit server yang telah diinstal pada sistem operasi Ubuntu 14.04 dengan spesifikasi sebagai berikut:

   CPU : AMD A6-4400M with Radem (tm) Vision A6
   Hardisk : 197 GB
   Memory : 2 GB
   Operating System : Ubuntu 14.04

   Untuk menjalankan server RTCMultiConnection dapat dilakukan dengan perintah:

   ```
   #node server.js
   ```

Setelah selesai melakukan langkah-langkah diatas, maka server RTCMultiConnection dapat dijalankan dan *client* dapat mengakses server tersebut dengan cara mengetikan IP_server: 9001 pada *browser* yang digunakan.



Gambar 4.1 Server RTCMultiConnection

Gambar 4.1 merupakan server RTCMultiConnection yang telah diinstall pada OS Ubuntu 14.04.

Kemudian setelah selesai proses installasi server, penulis merubah isi index.html sesuai dengan kebutuhan pada lingkungan pengujian, *file* ini terletak pada direktori /Downloads/RTCMultiConnection-master/.

4.2 getUserMedia pada *framework* RTCMultiConnection

MediaStream API digunakan untuk mengakses *stream* dari media seperti *stream* yang diambil dari kamera dan mikrofon untuk disinkronkan dengan video dan audio *tracks*. Setiap MediaStream memiliki input yang akan dilewatkan pada elemen video atau RTCPeerConnection. Navigator.mozGetUserMedia digunakan untuk mengakses kamera pada browser Mozilla firefox sedangkan navigator.webkitGetUserMedia digunakan pada browser Google Chrome.

Tabel 4.1 getuserMedia untuk Mozilla Firefox

No	Source Code
1	`if (Navigator.mozGetUserMedia)`
2	`Console.log ("This appears to be Firefox");`
3	`webrtcDetectedBrowser = "firefox";`

Tabel 4.1 menjelaskan tentang proses pengambilan stream gambar dan audio pada perangkat keras dengan menggunakan browser Mozilla Firefox. Baris 1 digunakan untuk mengakses stream pada perangkat keras melewati *browser* Mozilla Firefox. Baris 2 untuk menampilkan *console*.

Tabel 4.2 getuserMedia untuk Google Chrome

No	Source Code
1	`Else if (Navigator.webkitGetUserMedia){`
2	`Console.log ("This appears to be Chrome");`
3	`webrtcDetectedBrowser = "chrome";`

Tabel 4.2 menjelaskan tentang proses pengambilan stream gambar dan audio pada perangkat keras dengan menggunakan *browser* google chrome. Baris 1 digunakan untuk mengakses stream pada perangkat keras melewati *browser* google chrome. Baris 2 untuk menampilkan console.

4.3 RTCPeerConnection pada *framework* RTCMultiConnection

Menyediakan layanan untuk membentuk sebuah koneksi diantara *peers* dan *stream* audio atau video. Untuk menginisialisasi proses RTCPeerConnection terdapat dua proses yang harus dilakukan yaitu memastikan kondisi dari media local seperti *codec* yang digunakan, informasi ini digunakan untuk bertukar informasi dalam mekanisme *offer* dan *answer*, mendapatkan *network address* yang dikenal dengan *finding candidate*.

a. Caller

Membuat RTCPeerConnectin baru dan menambahkan stream yang didapatkan dari getUserMedia().

Tabel 4.3 RTCPeerConnection

No	Source Code
1	`Pc1 = new webkitRTCPeerConnection (servers) ;`
2	`Pc1.addStream (localStream);`

Tabel 4.4 Membuat offer dan setting menjadi *local description* dari Pc1 dan *remote description* pada Pc2.

Tabel 4.4 Proses *offer*

No	Source Code
1	`Pc1 = createOffer(gotDescription1);`
2	`Function gotDescription(desc){`
3	`  Pc1.setLocalDescription1(desc){`
4	`  Pc2.setRemoteDescription(desc);`
5	`  Pc2.createAnswer(gotDescription)`

b. Calle

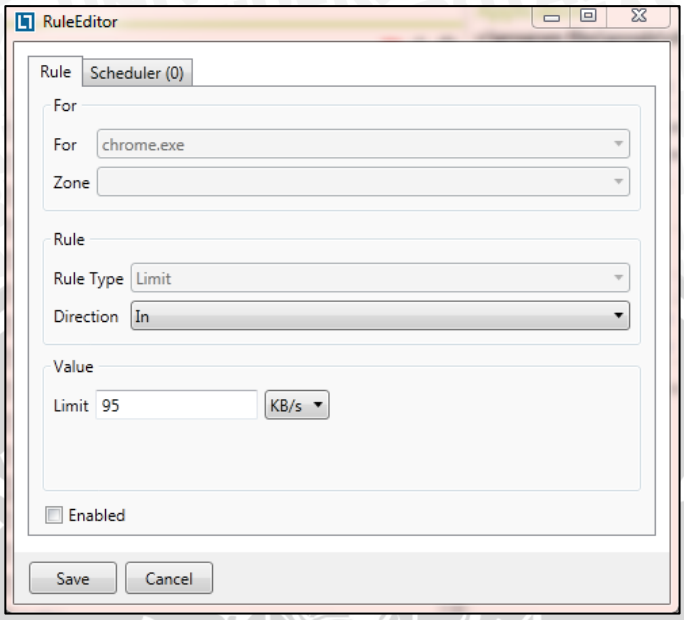
Membuat Pc2 dapat melakukan proses pengambilan *stream* kemudian *stream* dari Pc1 ditambahkan, dan *display* didalam sebuah video element.

No	Source Code
1	`Pc2 = new webkitRTCPeerConnection(servers);`
2	`Pc2.onaddstream = gotRemoterStream;`
3	`Function gotRemoteStream(e){`
4	`  Vid2.src=URL.createObjectURL(e.stream);`

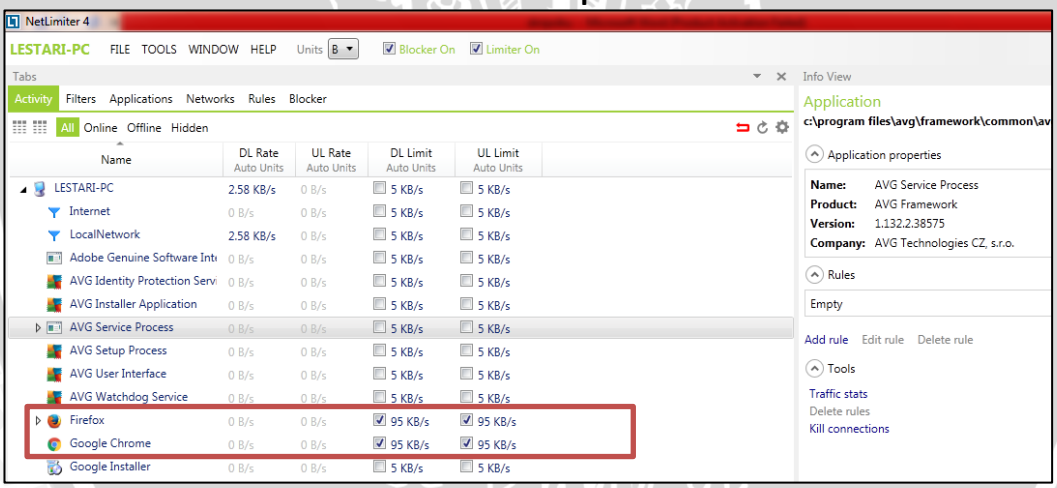
4.4 Proses pembatasan *bandwidth* menggunakan *Software* Netlimiter

Netlimiter digunakan untuk proses pembatasan *bandwidth*, *software* ini akan membatasi *bandwidth* yang diinginkan pada setiap aplikasi yang digunakan pada penelitian ini penulis menggunakan Netlimiter versi 4.0.19.0 untuk proses pengujian. Netlimiter diinstal pada sistem operasi Windows pada tiap-tiap *client*. Netlimiter dapat digunakan untuk melakukan proses *setting bandwidth* terhadap kondisi jaringan komputer pada setiap *client*, untuk *setting bandwidth* kita dapat merubahnya pada menu *Rule Editor* terdapat nilai limit yang akan di *setting*

kemudian *save*, maka *bandwidth* yang digunakan tidak akan membatasi *bandwidth* yang telah ditentukan. Gambar 4.2 dan gambar 4.3 menunjukan proses konfigurasi pada Netlimiter.



Gambar 4.2 *Rule Editor* pada NetLimiter



Gambar 4.3 Netlimiter *Setting*

BAB 5 PENGAMBILAN DATA

Pada bab ini akan dilakukan pengujian dan pengambilan data terhadap komunikasi *video conference*. Teknik pengambilan data dilakukan dalam tahap seperti yang telah dijelaskan pada Bab III, yaitu pengujian parameter *delay, jitter, throughput* dan *packet loss* pada Browser Mozilla Firefox dan Google Chrome dengan *bandwidth* 58 kbps dan 95 kbps. Tabel 5.1 menunjukan waktu pengujian dan pengambilan data komunikasi *video conference*.

Tabel 5.1 Waktu Pengujian

Browser	Jumlah *Client*	*Bandwidth*	Tanggal	Waktu
Mozilla Firefox	Dua	58 kbps	2 November 2016	10.30 – 11.20
		95 kbps	2 November 2016	12.00 – 14.00
	Tiga	58 kbps	2 November 2016	15.00 – 17.30
		95 kbps	2 November 2016	19.00 – 19.30
	Lima	58 kbps	2 November 2016	21.00 – 22.00
		95 kbps	2 November 2016	22.15 – 22.30
	Enam	58 kbps	3 November 2016	07.00 – 09.00
		95 kbps	3 November 2016	09.15 – 10.30
Google Chrome	Dua	58 kbps	3 November 2016	13.00 – 15.30
		95 kbps	3 November 2016	15.45 – 18.30
	Tiga	58 kbps	4 November 2016	07.00 – 09.00
		95 kbps	4 November 2016	09.15 – 10.30
	Lima	58 kbps	5 November 2016	10.30 – 11.20
		95 kbps	6 November 2016	12.00 – 14.00
	Enam	58 kbps	7 November 2016	14.15 – 18.30
		95 kbps	9 November 2016	15.00 – 18.30

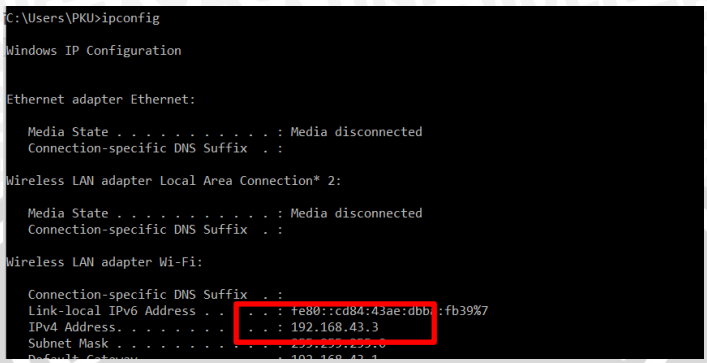
5.1 Pengujian Framework RTCMultiConnection

Pengujian ini dilakukan untuk mengetahui performa WebRTC dengan menggunakan *framework* RTCMultiConnection dalam melakukan komunikasi *video conference* dengan menggunakan Browser Mozilla Firefox dan Google Chrome. Pengujian dilakukan sesuai dengan skenario pengujian yang telah dijelaskan pada Bab tiga. Masing masing skenario dilakukan sebanyak tiga kali percobaan dengan waktu komunikasi sekitar empat menit kemudian diambil nilai rata-rata dari tiga percobaan tersebut.

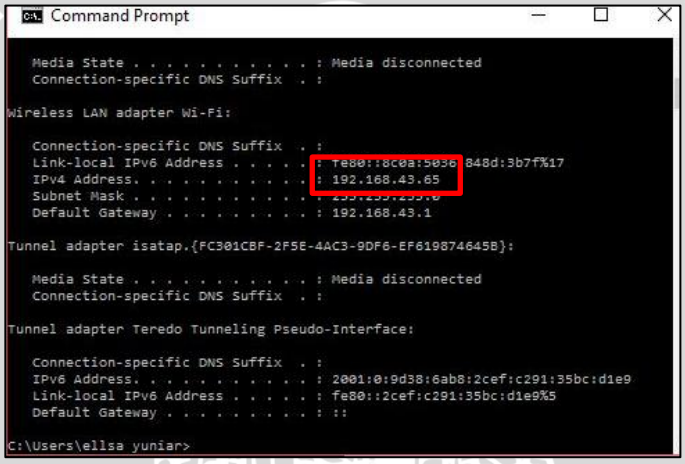
Jaringan komputer yang digunakan dalam pengujian adalah jaringan komputer lokal yang hanya dapat digunakan untuk IP pada setiap komputer yang digunakan untuk pengujian, tidak ada IP lain yang terhubung pada jaringan, hal ini dilakukan untuk memastikan bahwa jaringan yang di*capture* hanya IP yang digunakan pada proses pengujian.

Pada penelitian ini, *server* RTCMultiConnection diletakan pada Laptop Samsung dengan OS Ubuntu 14.04. Host yang bekerja sebagai *server* menggunakan alamat IP 192.168.43.215 dengan port 9001.

Sedangkan *host* yang bekerja sebagai *client* menggunakan IP 192.168.43.65dan 192.168.43.3 Konfigurasi IP dari ketiga host dapat dilihat pada Gambar 5.1 dan Gambar 5.2

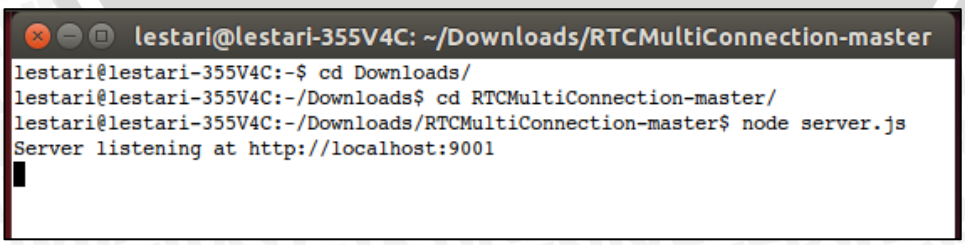


Gambar 5.1 *Client* A



Gambar 5.2 *Client* B

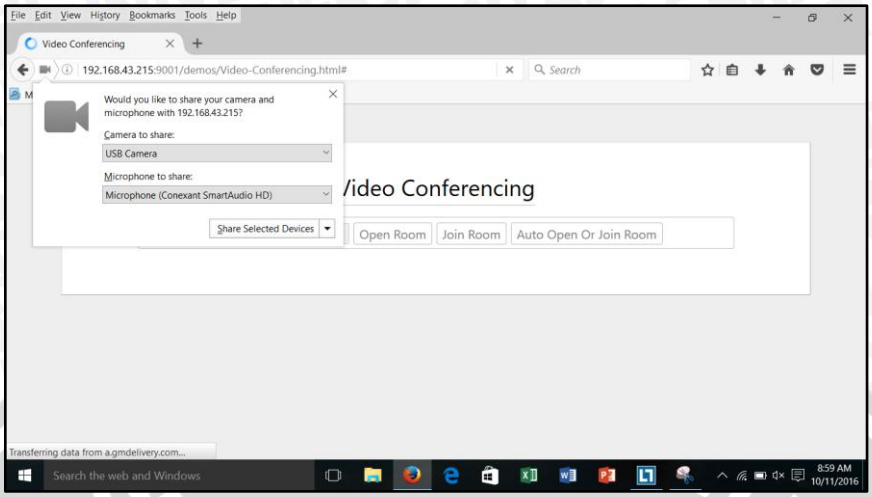
Pertama untuk menjalankan *server* dengan cara masuk pada direktori tempat menyimpan server.js yang telah diinstal node.js kemudian lakukan perintah "node server.js", setelah menjalankan proses tersebut maka *server* dapat digunakan dan proses komunikasi dapat dilakukan, kemudian *client* A dan *client* B dapat membuka *browser* yang akan digunakan dan menuliskan IP_server:9001.



Gambar 5.3 Server RTCMultiConnection

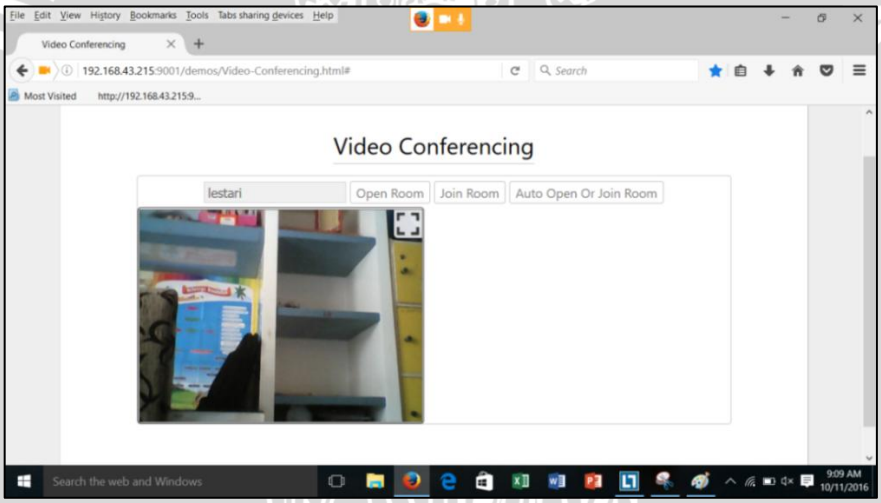
Kemudian kedua *client* mengakses halaman *video conference* masukan kode pada field text *yang* digunakan sebagai kode untuk bisa terhubung antara *client* A dan *client* B , misalkan disini *client* A memberikan kode dan melakukan proses

Open Room berarti *client* A membuka komunikasi terhadap *client* lain dan akan mucul tampilan untuk mengakses kamera dan mikrofon seperti pada Gambar 5.4



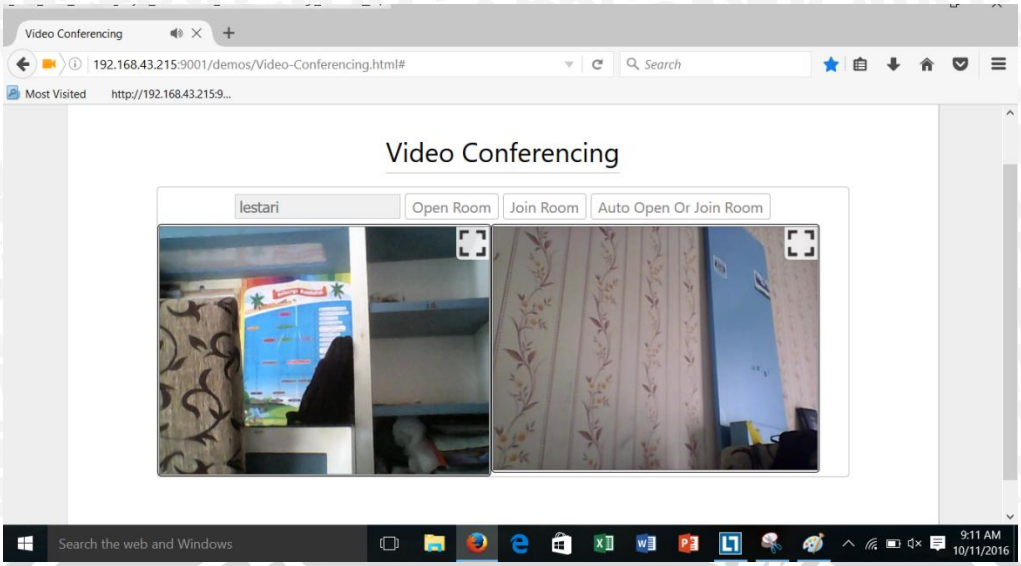
Gambar 5.4 Akses kamera dan mikrofon

Setelah *client* mengijinkan untuk mengakses kamera dan mikrofon maka akan otomatis muncul kamera yang digunakan oleh perangkat keras itu sendiri.



Gambar 5.5 GetUserMedia

Kemudian *client* A dapat memberitahu *client* B kode yang digunakan dan *client* B dapat melakukan proses *Join Room* dengan menggunakan kode yang telah diberikan *client* A, maka kedua *client* akan saling terhubung untuk melakukan komunikasi *video conference* seperti yang terlihat pada Gambar 5.6. *Client* A dan *client* B sudah dapat saling terhubung satu sama lain



Gambar 5.6 Komunikasi *video conference*

Skenario pengujian yang dilakukan yaitu dengan melakukan percobaan WebRTC pada *browser* yang berbeda. Hal ini bertujuan untuk mengetahui hasil dari parameter yang telah ditentukan pada setiap browser, setelah komunikasi dapat berjalan kemudian dilakukan proses pengujian dengan melakukan skenario untuk mendapatkan data yang akan dianalisa.

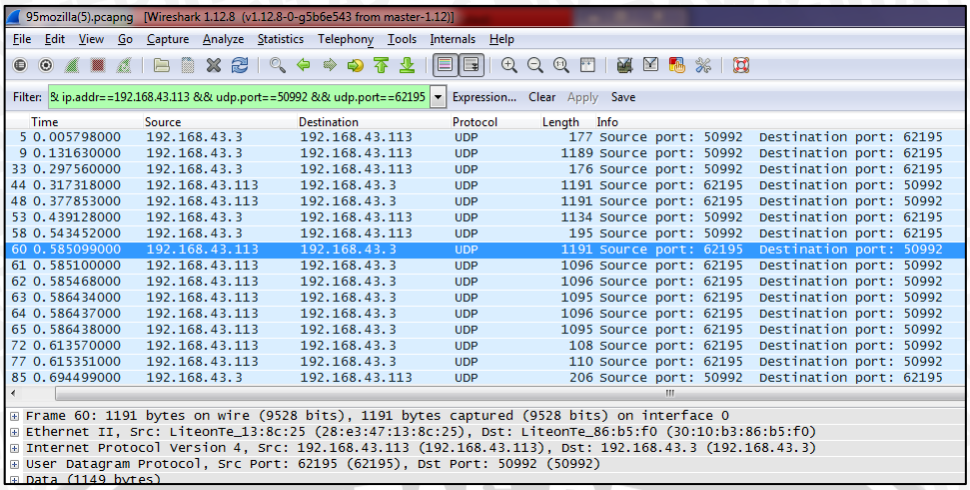
5.2 Pengambilan Data pada *Video Conference*

Pengambilan data parameter *delay, jitter, throughput* dan *packet loss* akan dilakukan pada proses pengujian terhadap komunikasi *video conference*. Teknik pengambilan data dilakukan dalam tahap seperti yang telah dijelaskan pada Bab III, yaitu pengujian menggunakan *browser* Mozilla Firefox dan Google Chrome dengan kondisi *bandwidth* dan jumlah *client* yang berbeda.

Pada tahap ini akan dilakukan pengujian *video conference* pada WebRTC antar *client* dengan menggunakan Wireshark sebagai alat untuk *capturing* data. Tahap pengujian ini dilakukan dengan melakukan panggilan komunikasi *video conference* antar *client* selama empat menit dengan tiga kali pengujian untuk setiap kondisi *bandwidth* 58 kbps dan 95 kbps pada Browser Mozilla firefox kemudian hasil pengujian dirata-rata. Hal ini bertujuan untuk mengetahui nilai parameter pada *bandwidth* dan jumlah *client* yang berbeda.

5.2.1 Pengambilan Data Dua *client* dengan *Bandwidth* 95 kbps

Pada pengujian *bandwidth* 58 kbps penulis melakukan tiga kali percobaan dengan waktu sekitar empat menit. Gambar 5.7 merupakan hasil *capture* jaringan pada saat melakukan proses komunikasi menggunakan wireshark.



Gambar 5.7 Hasil *Capture* Data Dua *Client Bandwidth* 95 kbps

Gambar 5.7 merupakan hasil *capture* Data pada *bandwidth* 95 kbps, sintak yang digunakan untuk memfilter paket data pada Wireshark.

```
Ip.addr == 192.168.43.3 && ip.addrs== 192.168.43.113
&& udp.port == 50992 && udp.port == 62195
```

Dapat dilihat dari proses *capturing* protokol yang digunakan adalah UDP. Pada penelitian ini terfokus pada protokol UDP karena penelitian ini memfokuskan pada proses pertukaran data *mediastream* dan *peers* pada saat komunikasi telah terhubung satu sama lain. Dalam proses pertukaran data *Framework* RTCMultiConnection berjalan dengan memanfaatkan protokol UDP. Pengambilan data untuk kualitas *video conference* diambil dari dua sumber yaitu *client* dengan IP 192.168.43.3 menuju *client* dengan IP 192.168.43.113 dengan nomor port 50992 dan 62195.

Untuk mengetahui paket data yang didapatkan setelah proses *filtering* maka dapat dilihat pada sub menu *summary* pada Wireshark dari sini penulis dapat menghitung nilai parameter dari setiap percobaan yang dilakukan.

Traffic	Captured	Displayed	Displayed %	Marked	Marked %
Packets	31453	30813	97.965%	0	0.000%
Between first and last packet	231.313 sec	231.313 sec			
Avg. packets/sec	135.976	133.209			
Avg. packet size	457 bytes	462 bytes			
Bytes	14386803	14225617	98.880%	0	0.000%
Avg. bytes/sec	62196.296	61499.465			
Avg. MBit/sec	0.498	0.492			

Gambar 5.8 *Summary Filtering* paket data

Pada menu *summary* dapat dilihat informasi dari komunikasi yang dilakukan yaitu total paket data yang dikirim total paket data yang diterima dan lama pengamatan. Untuk langkah selanjutnya dilakukan proses perhitungan parameter dari tiga kali percobaan dengan proses komunikasi yang dilakukan sekitar empat menit.

A. *Delay*

Dari *capture* data yang telah dilakukan dengan Wireshark maka didapatkan rata-rata *delay* dengan menggunakan Persamaan 2.1.

$$Delay\ Rata-rata = \frac{Total\ Delay}{Total\ Paket\ yang\ diterima}$$

$$Delay\ Rata-rata = \frac{241{,}578}{39932} = 0{,}00604\ \text{sec} = 6{,}050\ ms$$

Tabel 5.2 *Delay* pada Pengujian ke-1

Delay	
Total *Delay*	241,578
Total Paket yang diterima	39932
rata - rata *delay* (s)	0,0060
rata - rata *delay* (ms)	6,050

Tabel 5.3 *Delay* pada Pengujian ke-2

Delay	
Total *Delay*	238,650
Total Paket yang diterima	34446
rata - rata *delay* (s)	0,0069
rata - rata *delay* (ms)	6,928

Tabel 5.4 *Delay* pada Pengujian ke-3

Delay	
Total *Delay*	231,313
Total Paket yang diterima	30813
rata - rata *delay* (s)	0,0075
rata - rata *delay* (ms)	7,549

Tabel 5.2, 5.3 dan Tabel 5.4 merupakan hasil perhitungan rata-rata *delay* yang diambil dari *summary* wireshark, total *delay* (*second*) didapatkan dari hasil penjumlahan pengiriman paket data yang terdapat pada tabel "*time delta from previous displayed frame*" pada wireshark. Tabel tersebut adalah waktu kedatangan pada tiap paket data waktu, kemudian untuk total paket data yang diterima didapatkan pada *summary* wireshark pada baris *packets* dan kolom *displayed*. Kemudian setelah mendapatkan nilai tersebut maka dapat dilakukan proses perhitungan menggunakan persamaan 2.1 maka didapatkan rata-rata *delay* dari hasil perhitungan persamaan tersebut.

B. Jitter

Dari *capture* data yang telah dilakukan dengan wireshark maka didapatkan nilai *jitter* dengan menggunakan Persamaan 2.2 untuk melakukan proses pengambilan data.

$$Jitter = \frac{Total\ Variasi\ Delay}{Total\ Paket\ yang\ diterima}$$

Total variasi *delay* diperoleh dari penjumlahan:

$$delay2 - delay1\ +\ delay3 - delay2\ + \cdots + (delay\ n - delay\ (n-1)$$

Untuk menghitung nilai *jitter* penulis juga melakukan tiga kali percobaan selama empat menit. Berikut nilai *jitter* yang penulis dapatkan dari tiga kali percobaan.

Tabel 5.5 *Jitter* pada Pengujian ke-1

Jitter	
Total variasi *delay*	0,000055
Total paket yang diterima	39932
Nilai *Jitter* (s)	0,000000
Nilai *Jitter* (ms)	0,000001

Tabel 5.6 *Jitter* pada Pengujian ke-2

Jitter	
Total variasi *delay*	0,07957
Total paket yang diterima	34446
Nilai *Jitter* (s)	0,000002
Nilai *Jitter* (ms)	0,002310

Tabel 5.7 *Jitter* pada Pengujian ke-3

Jitter	
Total variasi *delay*	0,028624
Total paket yang diterima	30813
Nilai *Jitter* (s)	0,000001
Nilai *Jitter* (ms)	0,000007

Tabel 5.5, 5.6 dan Tabel 5.7 merupakan hasil pengambilan data dan hasil dari proses perhitungan *jitter* yang diambil dari proses *convert* data wireshark kedalam excel, total variasi *delay* (*second*) didapatkan dengan menjumlahkan keseluruhan selisih *delay* yang ada antara paket satu dengan paket selanjutnya kemudian dibagi dengan total paket yang diterima.

C. Throughput

Dari *capture* data yang telah dilakukan dengan wireshark maka didapatkan nilai *troughput* dengan menggunakan Persamaan 2.4.

$$Throughput = \frac{Paket\ data\ yang\ diterima}{lama\ pengamatan}$$

$$Throughput = \frac{14727978\ bytes}{241{,}577\ sec} = 60965{,}8\ \text{bps} = 60{,}966\ \text{kbps}$$

Tabel 5.8 *Throughput* pada Pengujian ke-1

Throughput	
paket yang dikirim (bytes)	14727978
lama pengamatan (s)	241,577
Nilai *Throughput* (bps)	60965,8
Throughput (kbps)	60,966

Tabel 5.9 *Throughput* pada Pengujian ke-2

Throughput	
paket yang dikirim (bytes)	17328240
lama pengamatan (s)	238,650
Nilai *Throughput* (bps)	72609,3
Throughput (kbps)	72,609

Tabel 5.10 *Throughput* pada Pengujian ke-3

Throughput	
paket yang dikirim (bytes)	13550160
lama pengamatan (s)	231,313
Nilai *Throughput* (bps)	58102,5
Throughput (kbps)	58,103

Tabel 5.8, 5.9 dan Tabel 5.10 merupakan perhitungan *throughput* yang diambil dari *summary* pada wireshark dimana total paket data yang dikirim dibagi dengan lama pengamatan.

Pada contoh perhitungan ini menggunakan *bandwidth* 95 kbps, dari *throughput* yang dihasilkan adalah 66.045 kbps, maka diperlukannya persamaan untuk mengubah menjadi nilai persentase, dikarenakan standarisasi *throughput* versi TIPHON bernilai persentase. Berikut ini adalah contoh perhitungan untuk mengubah nilai *throughput* kedalam presentase.

$$Throughput \ \% \ = \frac{\text{Throughput}}{Bandwidth} x100$$

$$\% \ = \frac{60{,}966}{95} x100 = 64{,}17 \ \%$$

Hasil perhitungan *throughput* dari satuan kbps ke satuan persentase menghasilkan nilai 64,17%. hasil tersebut menunjukkan bahwa dari *bandwidth* 95 kbps menghasilkan nilai *throughput* 66,054 kbps. Sehingga jika persentasekan mendapatkan nilai *throughput* 64,17%.

D. Packet Loss

Untuk menghitung nilai *packet loss* penulis juga melakukan tiga kali percobaan selama empat menit. Berikut nilai *packets loss* yang penulis dapatkan dari tiga kali percobaan. Untuk mengetahui *paket loss* yang terjadi pada komunikasi WebRTC menggunakan persamaan 2.3.

$$\text{Pa}ket \ loss = \frac{paket \ data \ yang \ dikirim \ - \ paket \ yang \ diterima}{Paket \ data \ yang \ dikirim} x \ 100 \ \%$$

$$Paket \ loss \ = \frac{40206 - 39932}{40206} x \ 100 \ \% = 0{,}68 \ \%$$

Tabel 5.11 *packet loss* pada Pengujian ke-1

Packet Loss	
Paket yang dikirim	40206
paket yang diterima	39932
nilai p*acket loss*	0,68

Tabel 5.12 *packet loss* pada Pengujian ke-2

Packet Loss	
Paket yang dikirim	35121
paket yang diterima	34446
nilai *packet loss*	1,92

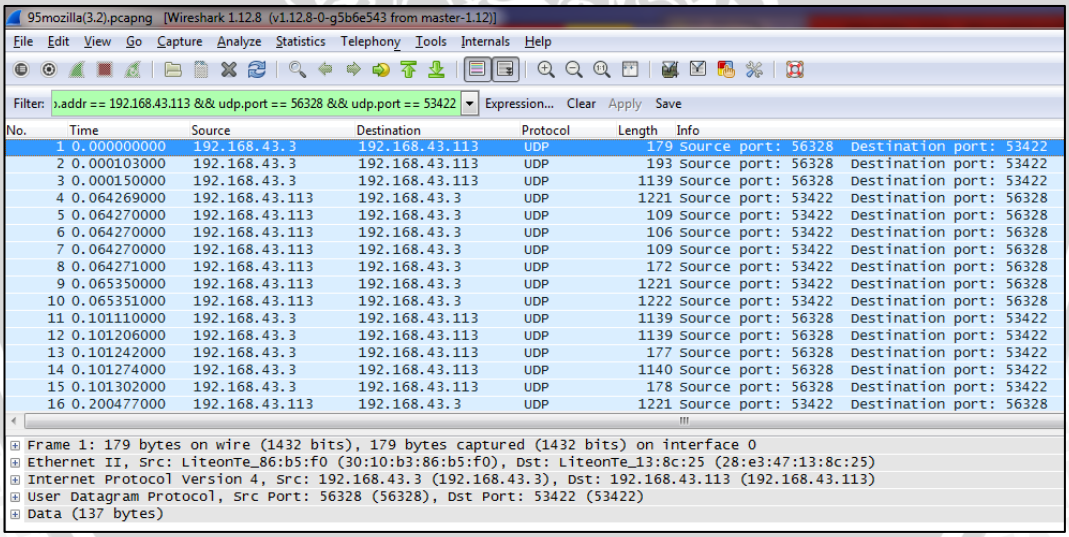
Tabel 5.13 *packet loss* pada Pengujian ke-3

Packet Loss	
Paket yang dikirim	32570
paket yang diterima	30813
nilai *packet loss*	5,16

Tabel 5.11, 5.12 dan Tabel 5.13 merupakan perhitungan *packet loss* yang diambil dari *summary* pada wireshark dimana total paket yang dikirim dan total paket yang diterima terdapat pada menu *summary.*

5.2.2 Pengambilan Data Tiga *client* dengan *Bandwidth* 95 kbps

Pada pengujian bandwidth 95 kbps penulis melakukan tiga kali percobaan dengan waktu sekitar empat menit. Gambar 5.9 merupakan hasil *capture* pada saat melakukan proses komunikasi menggunakan wireshark.



Gambar 5.9 Hasil *Capture* Wireshark Tiga *client Bandwidth* 95 kbps

Gambar 5.9 merupakan hasil *capture* wireshark pada *bandwidth* 95 kbps, dapat dilihat bahwa pada *framework* RTCMultiConnection komunikasi berjalan dengan memanfaatkan protokol UDP. Pengujian data untuk kualitas *video conference* diambil dari dua sumber yaitu client dengan IP 192.168.43.3 menuju client dengan IP 192.168.43.113.

```
Ip.addr == 192.168.43.3 && ip.addrs== 192.168.43.113
&& udp.port == 56328 && udp.port == 53422
```

Untuk mengetahui paket data yang didapatkan setelah proses *filtering* maka dapat dilihat pada sub menu *summary* pada wireshark dari sini penulis dapat menghitung *quality of service* dari setiap percobaan yang dilakukan.

Traffic	Captured	Displayed	Displayed %	Marked	Marked %
Packets	31722	31653	99.782%	0	0.000%
Between first and last packet	249.339 sec	249.339 sec			
Avg. packets/sec	127.225	126.948			
Avg. packet size	483 bytes	483 bytes			
Bytes	15307448	15303998	99.977%	0	0.000%
Avg. bytes/sec	61392.231	61378.394			
Avg. MBit/sec	0.491	0.491			

Gambar 5.10 *Summary Filtering* paket data

Pada menu *summary* dapat dilihat informasi dari komunikasi yang dilakukan yaitu total paket data yang dikirim total paket data yang diterima dan lama pengamatan. Untuk langkah selanjutnya dilakukan proses perhitungan parameter dari tiga kali percobaan dengan proses komunikasi yang dilakukan sekitar empat menit.

A. Delay

Dari *capture* data yang telah dilakukan dengan wireshark maka didapatkan rata-rata delay dengan menggunakan Persamaan 2.1.

$$Delay\ Rata-rata = \frac{Total\ Delay}{Total\ Paket\ yang\ diterima}$$

$$Delay\ Rata-rata = \frac{239{,}941}{34326} = 0{,}007\ \text{sec} = 6{,}990\ ms$$

Tabel 5.14 *Delay* pada Pengujian ke-1

Delay	
Total *Delay*	239,941
Total Paket yang diterima	34326
rata - rata *delay* (s)	0,007
rata - rata *delay* (ms)	6,990

Tabel 5.15 *Delay* pada Pengujian ke-2

Delay	
Total *Delay*	231,573
Total Paket yang diterima	29927
rata - rata *delay* (s)	0,008
rata - rata *delay* (ms)	7,738

Tabel 5.16 *Delay* pada Pengujian ke-3

Delay	
Total *Delay*	249,339
Total Paket yang diterima	31653
rata - rata *delay* (s)	0,008
rata - rata *delay* (ms)	7,877

Tabel 5.14, 5.15 dan Tabel 5.16 merupakan hasil perhitungan rata-rata *delay* yang diambil dari *summary* wireshark, total *delay* (*second*) didapatkan dari hasil penjumlahan pengiriman paket data yang terdapat pada tabel "*time delta from previous displayed frame*" pada wireshark. Tabel tersebut adalah waktu kedatangan pada tiap paket data waktu, kemudian untuk total paket data yang diterima didapatkan pada *summary* wireshark pada baris *packets* dan kolom *displayed*. Kemudian setelah mendapatkan nilai tersebut maka dapat dilakukan proses perhitungan menggunakan persamaan 2.1 maka didapatkan rata-rata *delay* dari hasil perhitungan persamaan tersebut.

B. Jitter

Dari *capture* data yang telah dilakukan dengan wireshark maka didapatkan rata-rata *delay* dengan menggunakan Persamaan 2.2.

$$Jitter = \frac{Total\ Variasi\ Delay}{Total\ Paket\ yang\ diterima}$$

Total variasi *delay* diperoleh dari penjumlahan:

$$delay2 - delay1 \ + \ delay3 - delay2 \ + \cdots + (delay\ n - delay\ (n-1))$$

Untuk menghitung nilai *jitter* penulis juga melakukan tiga kali percobaan selama dua menit. Berikut nilai *jitter* yang penulis dapatkan dari tiga kali percobaan.

Tabel 5.17 *Jitter* pada Pengujian ke-1

Jitter	
Total variasi *jitter*	0.035801
Total paket yang diterima	34326
Nilai *Jitter* (s)	0.000001
Nilai *Jitter* (ms)	0.001043

Tabel 5.18 *Jitter* pada Pengujian ke-2

Jitter	
Total variasi *jitter*	0.000513
Total paket yang diterima	29927
Nilai *Jitter* (s)	0.000000
Nilai *Jitter* (ms)	0.000017

Tabel 5.19 *Jitter* pada Pengujian ke-3

Jitter	
Total variasi *jitter*	0.039985
Total paket yang diterima	31653
Nilai *Jitter* (s)	0.000001
Nilai *Jitter* (ms)	0.001263

Tabel 5.17, 5.18 dan Tabel 5.19 merupakan proses perhitungan *jitter* yang diambil dari *convert* data wireshark, total variasi *delay* (*second*) didapatkan dengan menjumlahkan keseluruhan selisih *delay* yang ada antara paket satu dengan paket yang lainnya kemudian dibagi dengan total paket yang diterima.

C. *Throughput*

Dari *capture* data yang telah dilakukan dengan wireshark maka didapatkan nilai *troughput* dengan menggunakan Persamaan 2.4.

$$Throughput = \frac{Paket\ data\ yang\ diterima}{lama\ pengamatan}$$

$$Throughput = \frac{15846891\ bytes}{239.941\ sec} = 56244{,}76\ \text{bps} = 66.045\ \text{kbps}$$

Tabel 5.20 *Throughput* pada Pengujian ke-1

Throughput	
paket yang dikirim (bytes)	15846891
lama pengamatan (s)	239.941
Nilai *Throughput* (bps)	66044.85
Throughput (kbps)	66.045

Tabel 5.21 *Throughput* pada Pengujian ke-2

Throughput	
paket yang dikirim (bytes)	14153498
lama pengamatan (s)	231.573
Nilai *Throughput* (bps)	61118.89
Throughput (kbps)	62.919

Tabel 5.22 *Throughput* pada Pengujian ke-3

Throughput	
paket yang dikirim (bytes)	15303998
lama pengamatan (s)	239.941
Nilai *Throughput* (bps)	63782.24
Throughput (kbps)	63.982

Tabel 5.20, 5.21 dan Tabel 5.22 merupakan perhitungan *throughput* yang diambil dari *summary* pada wireshark dimana total paket data yang dikirim dibagi dengan lama pengamatan.

Pada contoh perhitungan ini menggunakan *bandwidth* 95 kbps, dari *throughput* yang dihasilkan adalah 66.045 kbps, maka diperlukannya persamaan untuk mengubah menjadi nilai persentase, dikarenakan standarisasi *throughput* versi TIPHON bernilai persentase. Berikut ini adalah contoh perhitungan untuk mengubah nilai *throughput* kedalam presentase.

$$Throughput\ \ \% \ = \frac{\text{Throughput}}{Bandwidth} x100$$

$$\% \ = \frac{66{,}054}{95} x100 = 69.53\ \%$$

Hasil perhitungan *throughput* dari satuan kbps ke satuan persentase menghasilkan nilai 69,53%. hasil tersebut menunjukkan bahwa dari *bandwidth* 95 kbps menghasilkan nilai *throughput* 66,054 kbps. Sehingga jika persentasekan mendapatkan nilai *throughput* 69,53%

Jika dibandingkan dengan standarisasi *throughput* pada versi TIPHON masuk kedalam indeks sedang, dikarenakan nilai *throughput* dari penelitian ini kurang

dari 75%. Dapat disimpulkan bahwa dengan menggunakan *bandwidth* 95 kbps webrtc *video conference* dengan menggunakan *framework* RTCMultiConnection dengan node.js sebagai media signalingnya, dalam melakukan pengiriman data yang sukses sampai ketujuan masuk kedalam kategori kurang baik.

D. Packet Loss

Untuk menghitung nilai *packets loss* penulis juga melakukan tiga kali percobaan selama empat menit. Berikut nilai *packets loss* yang penulis dapatkan dari tiga kali percobaan. Untuk mengetahui *packet loss* yang terjadi pada komunikasi WebRTC menggunakan persamaan 2.3.

$$\text{Pa}ket\ loss = \frac{paket\ data\ yang\ dikirim\ -\ paket\ yang\ diterima}{Paket\ data\ yang\ dikirim} x\ 100\ \%$$

$$Paket\ loss\ = \frac{34778 - 34326}{34778} x\ 100\ \% = 1.30\ \%$$

Tabel 5.23 *packet loss* pada Pengujian ke-1

Packet Loss	
Paket yang dikirim	34778
paket yang diterima	34326
nilai p*acket loss*	1.30

Tabel 5.24 *packet loss* pada Pengujian ke-2

Packet Loss	
Paket yang dikirim	30324
paket yang diterima	29927
nilai p*acket loss*	1.31

Tabel 5.25 *packet loss* pada Pengujian ke-3

Packet Loss	
Paket yang dikirim	31722
paket yang diterima	31653
nilai p*acket loss*	0.22

Tabel 5.23, 5.24 dan Tabel 5.25 merupakan perhitungan *packet loss* yang diambil dari *summary* pada wireshark dimana total paket yang dikirim terdapat pada menu *summary.*

BAB 6 PEMBAHASAN

Berdasarkan pengambilan data yang telah dilakukan, kemudian dilakukan analisis dari data tersebut. Analisis dilakukan terhadap nilai *delay*, *jiter*, *throughput* dan *packet loss.* Data hasil analisis dibedakan menjadi dua yaitu pada kondisi *bandwidth* 58 kbps dan 95 kbps untuk setiap jumlah *client* yang berbeda.

6.1 *Delay*

Dari pengambilan data yang telah dilakukan diperoleh nilai *delay* pada komunikasi *video conference* dengan *bandwidth* 58 kbps dan 95 kbps pada Mozilla Firefox dan Google Chrome dengan jumlah *client* yang berbeda. Tabel 6.1 menunjukan rata-rata nilai *delay* dari komunikasi *video conference* pada kondisi *bandwidth* 58 kbps dengan pengujian pada dua *browser* yang berbeda yaitu Mozilla Firefox dan Google Chrome. Tabel 6.2 menunjukan rata-rata nilai *delay* dari komunikasi *video conference* pada kondisi *bandwidth* 95 kbps.

Tabel 6.1 *Delay* dengan *Bandwidth* 58 kbps

58 kbps		
Jumlah *Client*	**mozilla Firefox (ms)**	**Google Chrome (ms)**
Dua	7,609	7,286
Tiga	8,824	7,423
Lima	9,530	7,747
Enam	9,889	8,819

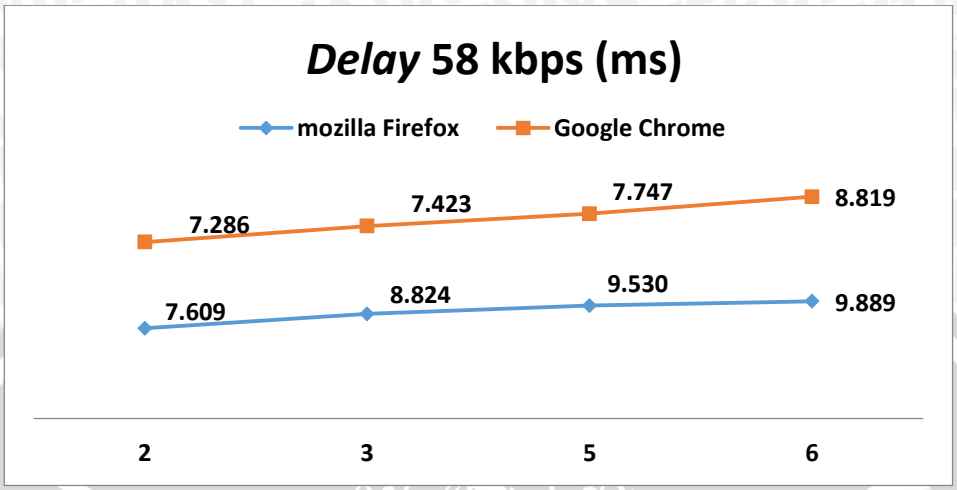
Hasil dari Tabel 6.1 dapat dilihat bahwa implementasi *video conference* dengan kondisi *bandwidth* 58 kbps pada *browser* Mozilla Firefox dan Google Chrome dengan jumlah *client* dua menghasilkan *delay* sebesar 7,607 ms, untuk jumlah *client* tiga menghasilkan *delay* sebesar 8,824 ms, untuk jumlah *client* lima menghasilkan *delay* sebesar 9,530 ms, dan untuk jumlah *client* sebanyak enam menghasilkan *delay* sebesar 9,889 ms. Pada *browser* Google Chrome dengan jumlah *client* dua menghasilkan *delay* sebesar 7,286 ms, untuk jumlah *client* tiga menghasilkan *delay* sebesar 7,423 ms, untuk jumlah *client* lima menghasilkan *delay* sebesar 7,747 ms, dan untuk jumlah *client* sebanyak enam menghasilkan *delay* sebesar 8,819 ms.

Tabel 6. 2 *Delay* dengan *Bandwidth* 95 kbps

95 kbps		
Jumlah *Client*	**mozilla Firefox (ms)**	**Google Chrome (ms)**
Dua	6,837	6,684
Tiga	7,535	7,275
Lima	8,069	8,454
Enam	8,471	31,267

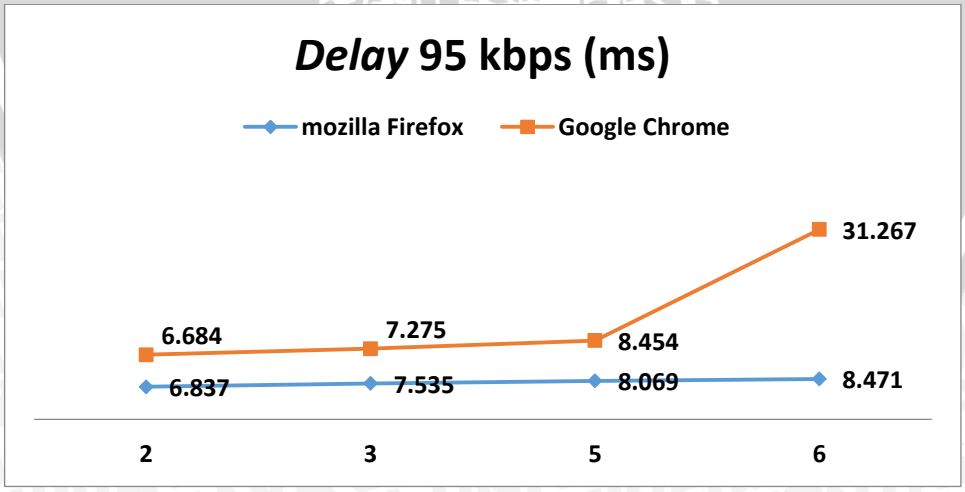
Hasil dari Tabel 6.2 dapat dilihat bahwa implementasi *video conference* dengan kondisi *bandwidth* 95 kbps pada *browser* Mozilla Firefox dan Google Chrome dengan jumlah *client* dua menghasilkan *delay* sebesar 6,684 ms, untuk jumlah *client* tiga menghasilkan *delay* sebesar 7,275 ms, untuk jumlah *client* lima menghasilkan *delay* sebesar 8,454 ms, dan untuk jumlah *client* sebanyak enam

menghasilkan *delay* sebesar 31,267 ms. Pada *browser* Google Chrome dengan jumlah *client* dua menghasilkan *delay* sebesar 6,837 ms, untuk jumlah *client* tiga menghasilkan *delay* sebesar 7,535 ms, untuk jumlah *client* lima menghasilkan *delay* sebesar 8,069 ms, dan untuk jumlah *client* sebanyak enam menghasilkan *delay* sebesar 8,471 ms.



Gambar 6.1 *Delay* 58 kbps

Gambar 6.1 menunjukan grafik *delay* dengan *bandwidth* 58 kbps pada *browser* Mozilla Firefox dan Google Chrome. Dari grafik terlihat bahwa nilai *delay* komunikasi *video conference* pada kondisi *bandwidth* 58 kbps pada Mozilla Firefox, cenderung lebih tinggi dibandingkan pada *browser* Google Chrome. Semakin banyak jumlah *client* pada saat komunikasi berlangsung maka semakin tinggi nilai *delay,* apabila dibandingkan dengan *delay* pada standarisasi TIPHON nilai *delay* komunikasi *video conference* masuk kedalam kategori baik, dikarenakan nilai rata-rata *delay* pada komunikasi *video conference* kurang dari 150 ms.



Gambar 6.2 *Delay* 95 kbps

Gambar 6.2 menunjukan grafik *delay* dengan *bandwidth* 95 kbps pada *browser* Mozilla Firefox dan Google Chrome. Dari grafik terlihat bahwa nilai *delay* komunikasi *video conference* pada kondisi *bandwidth* 95 kbps pada Mozilla Firefox cenderung lebih tinggi dibandingkan pada *browser* Google Chrome, akan

tetapi pada kondisi jumlah *client* sebanyak enam, pada saat kondisi ini nilai delay meningkat drastic. Hal ini disebabkan terdapat beberapa paket yang mempunyai waktu yang sangat lama untuk sampai ke tujuan, apabila dibandingkan dengan *delay* pada standarisasi TIPHON nilai *delay* komunikasi *video conference* masuk kedalam kategori baik, dikarenakan nilai rata-rata *delay* pada komunikasi *video conference* kurang dari 150 ms.

6.2 *Jitter*

Dari pengujian yang dilakukan diperoleh nilai *jitter* yang dihasilkan dalam proses komunikasi *video conference* dengan *bandwidth* dan jumlah *client* yang berbeda. Tabel 6.3 menunjukan nilai *jitter* dari komunikasi *video conference* dengan *bandwidth* 58 kbps.

Tabel 6.3 *Jitter* dengan *Bandwidth* 58 kbps

58 kbps		
Jumlah Client	**mozilla Firefox (ms)**	**Google Chrome (ms)**
Dua	0,000002	0,000129
Tiga	0,000004	0,000433
Lima	0,000469	0,000558
Enam	0,000499	0,008478

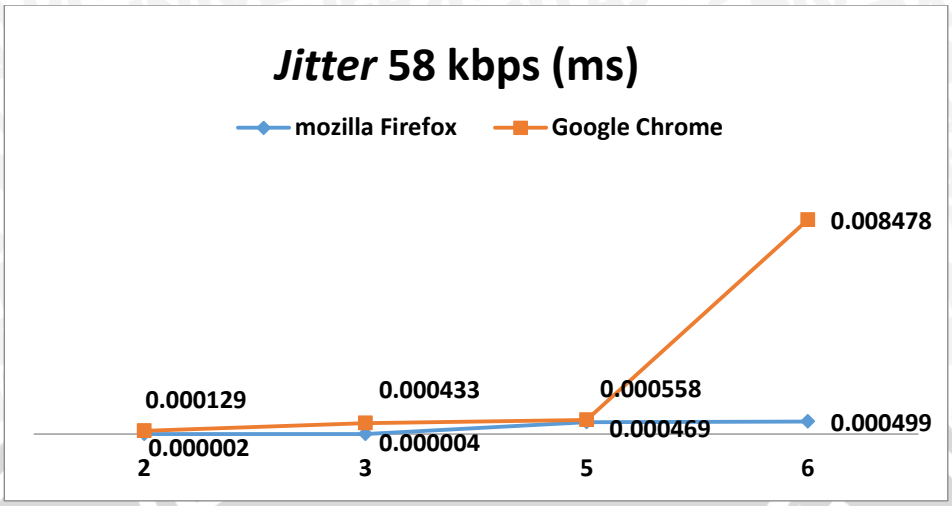
Tabel 6.3 dapat dilihat bahwa implementasi *video conference* dengan kondisi *bandwidth* 58 kbps pada *browser* Mozilla Firefox dan Google Chrome dengan jumlah *client* dua menghasilkan *jitter* sebesar 0,000002 ms, untuk jumlah *client* tiga menghasilkan *Jitter* sebesar 0,000004 ms, untuk jumlah *client* lima menghasilkan *Jitter* sebesar 0,000469 ms, dan untuk jumlah *client* sebanyak enam menghasilkan *Jitter* sebesar 0,000499 ms. Pada *browser* Google Chrome dengan jumlah *client* dua menghasilkan *Jitter* sebesar 0,000129 ms, untuk jumlah *client* tiga menghasilkan *Jitter* sebesar 0,000433 ms, untuk jumlah *client* lima menghasilkan *Jitter* sebesar 0,000558 ms, dan untuk jumlah *client* sebanyak enam menghasilkan *Jitter* sebesar 0,008478 ms.

Tabel 6.4 *Jitter* dengan *Bandwidth* 95 kbps

95 kbps		
Jumlah Client	**mozilla Firefox (ms)**	**Google Chrome (ms)**
Dua	0,000773	0,000344
Tiga	0,000788	0,000391
Lima	0,000949	0,000507
Enam	0,001231	0,967151

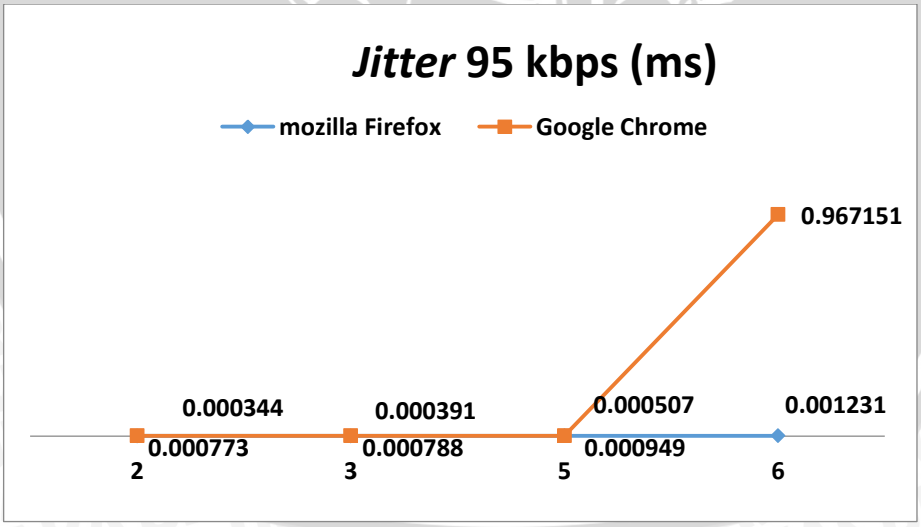
Tabel 6.4 dapat dilihat bahwa implementasi *video conference* dengan kondisi *bandwidth* 95 kbps pada *browser* Mozilla Firefox dan Google Chrome dengan jumlah *client* dua menghasilkan *jitter* sebesar 0,000773 ms, untuk jumlah *client* tiga menghasilkan *Jitter* sebesar 0,000788 ms, untuk jumlah *client* lima menghasilkan *Jitter* sebesar 0,000949 ms, dan untuk jumlah *client* sebanyak enam menghasilkan *Jitter* sebesar 0,001231 ms. Pada *browser* Google Chrome dengan jumlah *client* dua menghasilkan *Jitter* sebesar 0,000344 ms, untuk jumlah *client* tiga menghasilkan *Jitter* sebesar 0,000391 ms, untuk jumlah *client* lima

menghasilkan *Jitter* sebesar 0,000507 ms, dan untuk jumlah *client* sebanyak enam menghasilkan *Jitter* sebesar 0,967151 ms.



Gambar 6.3 *Jitter* 58 kbps

Gambar 6.3 menunjukan grafik *jitter* dengan *bandwidth* 58 kbps pada *browser* Mozilla Firefox dan Google Chrome. Dari grafik terlihat bahwa nilai *jitter* komunikasi *video conference* pada kondisi *bandwidth* 58 kbps pada Mozilla Firefox cenderung lebih tinggi dibandingkan pada *browser* Google Chrome dan semakin banyak jumlah *client* pada saat komunikasi berlangsung maka semakin tinggi nilai *jitter,* apabila dibandingkan dengan *jitter* pada standarisasi TIPHON nilai *jitter* komunikasi *video conference* masuk kedalam kategori baik, dikarenakan nilai rata-rata *jitter* pada komunikasi *video conference* kurang dari 75 ms.



Gambar 6.4 *Jitter* 95 kbps

Gambar 6.4 menunjukan grafik *jitter* dengan *bandwidth* 95 kbps pada *browser* Mozilla Firefox dan Google Chrome. Dari grafik terlihat bahwa nilai *jitter* komunikasi *video conference* pada kondisi *bandwidth* 95 kbps pada Mozilla Firefox cenderung lebih tinggi dibandingkan pada *browser* Google Chrome dan semakin banyak jumlah *client* pada saat komunikasi berlangsung maka semakin

tinggi nilai *jitter*. Faktor yang mempengaruhi nilai *jitter* disebabkan karena semakin banyaknya variasi *delay* yang terjadi pada proses komunikasi ketika sedang berlangsung.

6.3 *Throughput*

Dari pengujian yang dilakukan diperoleh *throughput* yang dihasilkan dalam proses komunikasi *video conference* dengan *bandwidth* dan *browser* yang berbeda. Tabel 6.5 menunjukan nilai *throughput* dari komunikasi *video conference* dengan *bandwidth* 58 kbps.

Tabel 6.5 *Throughput* dengan *Bandwidth* 58 kbps

58 kbps		
Jumlah Client	**mozilla Firefox (kbps)**	**Google Chrome (kbps)**
Dua	43,615	48,204
Tiga	44,386	48,253
Lima	44,942	54,491
Enam	49,146	55,935

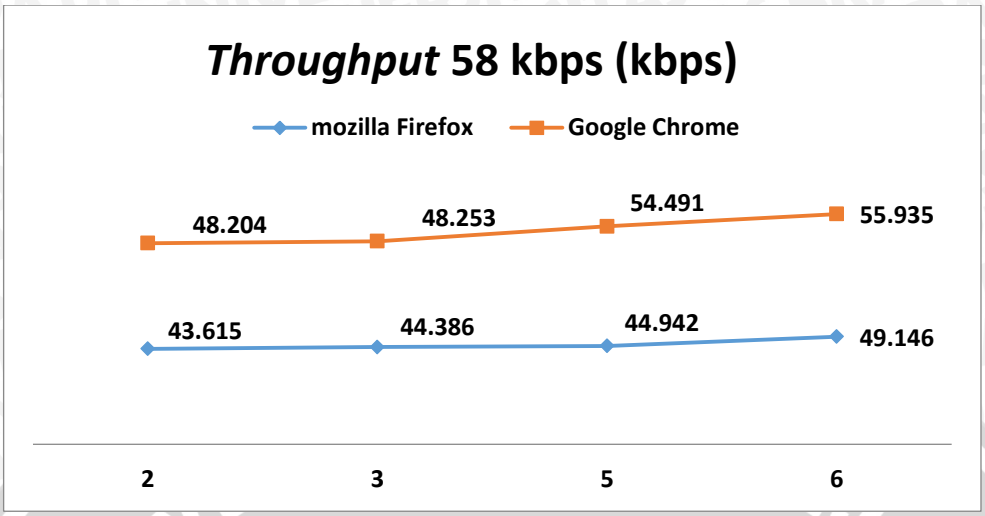
Tabel 6.5 dapat dilihat bahwa implementasi *video conference* dengan kondisi *bandwidth* 58 kbps pada *browser* Mozilla Firefox dan Google Chrome dengan jumlah *client* dua menghasilkan *throughput* sebesar 43,615 kbps, untuk jumlah *client* tiga menghasilkan *throughput* sebesar 44,386 kbps, untuk jumlah *client* lima menghasilkan *throughput* sebesar 44,942 kbps, dan untuk jumlah *client* sebanyak enam menghasilkan *throughput* sebesar 49,146 kbps. Pada *browser* Google Chrome dengan jumlah *client* dua menghasilkan *throughput* sebesar 48,204 kbps, untuk jumlah *client* tiga menghasilkan *throughput* sebesar 48,253 kbps, untuk jumlah *client* lima menghasilkan *throughput* sebesar 54,491 kbps, dan untuk jumlah *client* sebanyak enam menghasilkan *throughput* sebesar 55,935 kbps.

Tabel 6.6 *Throughput* dengan *Bandwidth* 95 kbps

95 kbps		
Jumlah Client	**mozilla Firefox (kbps)**	**Google Chrome (kbps)**
Dua	63,438	79,926
Tiga	64,615	80,879
Lima	78,661	85,326
Enam	78,993	91,165

Tabel 6.6 dapat dilihat bahwa implementasi *video conference* dengan kondisi *bandwidth* 95 kbps pada *browser* Mozilla Firefox dan Google Chrome dengan jumlah *client* dua menghasilkan *throughput* sebesar 63,438 kbps, untuk jumlah *client* tiga menghasilkan *throughput* sebesar 64,615 kbps, untuk jumlah *client* lima menghasilkan *throughput* sebesar 78,661 kbps, dan untuk jumlah *client* sebanyak enam menghasilkan *throughput* sebesar 78,993 kbps. Pada *browser* Google Chrome dengan jumlah *client* dua menghasilkan *throughput* sebesar 79,926 kbps, untuk jumlah *client* tiga menghasilkan *throughput* sebesar 80,879 80,879 kbps, untuk jumlah *client* lima menghasilkan *throughput* sebesar 85,326

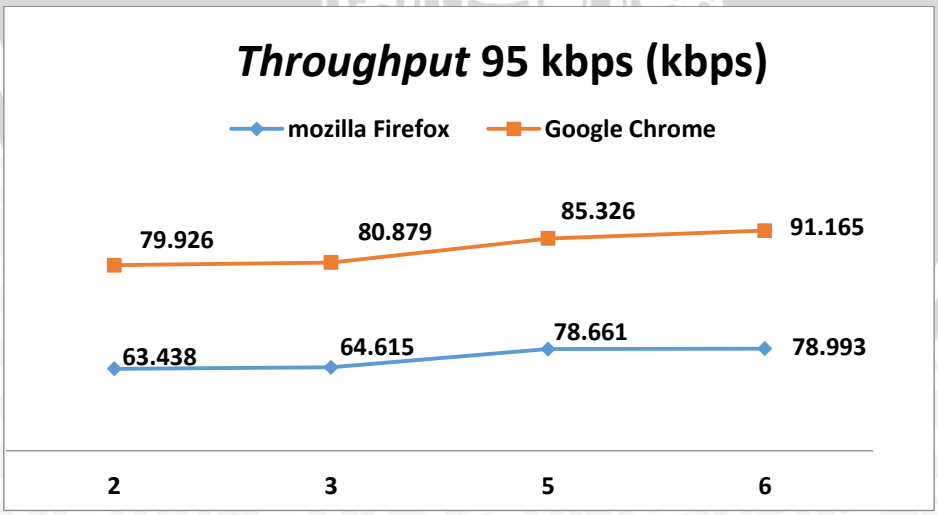
kbps, dan untuk jumlah *client* sebanyak enam menghasilkan *throughput* sebesar 91,165 kbps.



Gambar 6. 5 *Throughput* 58 kbps

Gambar 6.5 menunjukan grafik *throughput* dengan *bandwidth* 58 kbps pada *browser* Mozilla Firefox dan Google Chrome. Dari grafik terlihat bahwa nilai *throughput* komunikasi *video conference* pada kondisi *bandwidth* 58 kbps menghasilkan kesimpulan semakin banyak banyak jumlah *client* pada saat komunikasi berlangsung maka semakin tinggi nilai *throughput* yang dihasilkan.

Jika dibandingkan dengan standarisasi *throughput* pada versi TIPHON dapat disimpulkan bahwa dengan menggunakan *bandwidth* 58 kbps pada Mozilla Firefox masuk kedalam kategori kurang baik, dikarenakan nilai *throughput* dari penelitian ini kurang dari 75%, sedangkan untuk Google Chrome Dapat disimpulkan bahwa dengan menggunakan *bandwidth* 95 kbps dalam melakukan pengiriman data yang sukses sampai ketujuan masuk pada kategori baik.



Gambar 6. 6 *Throughput* 58 kbps

Gambar 6.6 menunjukan grafik *throughput* dengan *bandwidth* 95 kbps pada *browser* Mozilla Firefox dan Google Chrome. Dari grafik terlihat bahwa nilai *throughput* komunikasi *video conference* pada kondisi *bandwidth* 95 kbps

mendapatkan kesimpulan semakin banyak jumlah *client* pada saat komunikasi berlangsung maka semakin tinggi nilai *throughput* yang dihasilkan.

Jika dibandingkan dengan standarisasi *throughput* pada versi TIPHON dapat disimpulkan bahwa dengan menggunakan *bandwidth* 95 kbps pada Mozilla Firefox masuk kedalam kategori kurang baik, dikarenakan nilai *throughput* dari penelitian ini kurang dari 75%, sedangkan untuk Google Chrome Dapat disimpulkan bahwa dengan menggunakan *bandwidth* 95 kbps dalam melakukan pengiriman data yang sukses sampai ketujuan masuk pada kategori baik.

6.4 *Packet Loss*

Dari pengambilan data yang telah dilakukan diperoleh nilai *packet loss* yang dibutuhkan dalam komuikasi *video conference* dengan *bandwidth* 58 kbps dan 95 kbps pada Mozilla Firefox dan Google Chrome. Tabel 6.7 menunjukan rata-rata nilai *packet loss* dari tiga kali pengujian pada setiap kondisi. Tabel 6.8 menunjukan rata-rata nilai *packet loss* dari komunikasi *video conference* pada kondisi *bandwidth* 95 kbps.

Tabel 6.7 *Packet Loss* dengan *Bandwidth* 58 kbps

58 kbps		
Jumlah Client	**mozilla Firefox (%)**	**Google Chrome (%)**
Dua	4,964	7,493
Tiga	7,690	2,424
Lima	7,493	10,696
Enam	7,493	1,137

Tabel 6.7 dapat dilihat bahwa implementasi video conference dengan kondisi bandwidth 58 kbps pada browser Mozilla Firefox dan Google Chrome dengan jumlah *client* dua menghasilkan *packet loss* sebesar 4,964 %, untuk jumlah *client* tiga menghasilkan *packet loss* sebesar 7,690 %, untuk jumlah *client* lima menghasilkan *packet loss* sebesar 7,493 %, dan untuk jumlah *client* sebanyak enam menghasilkan *packet loss* sebesar 7,493 %. Pada browser Google Chrome dengan jumlah *client* dua menghasilkan *packet loss* sebesar 7,493 %, untuk jumlah *client* tiga menghasilkan *packet loss* sebesar 2,424 %, untuk jumlah *client* lima menghasilkan *packet loss* sebesar 10,696 %, dan untuk jumlah *client* sebanyak enam menghasilkan *packet loss* sebesar 1,137 %.

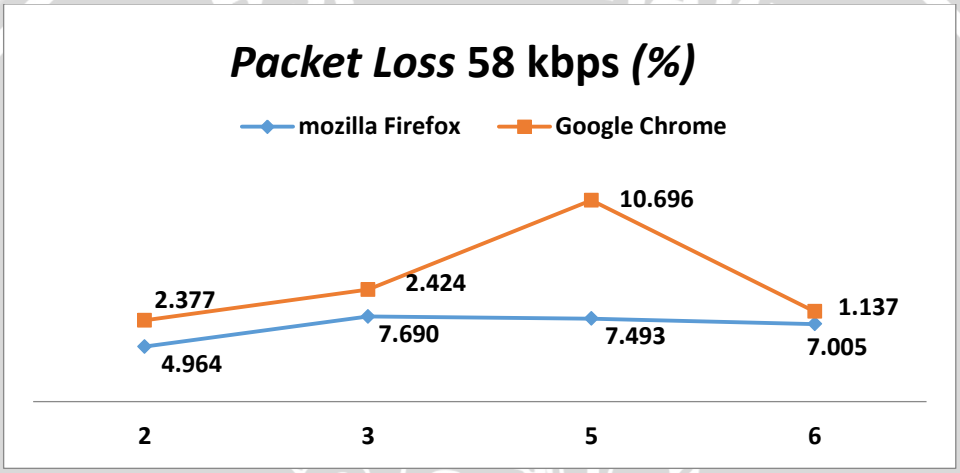
Tabel 6.8 *Packet Loss* dengan *Bandwidth* 95 kbps

95 kbps		
Jumlah Client	**mozilla Firefox (%)**	**Google Chrome (%)**
Dua	2,586	0,897
Tiga	0,942	1,012
Lima	2,621	0,918
Enam	6,898	30,233

Tabel 6.8 dapat dilihat bahwa implementasi video conference dengan kondisi bandwidth 95 kbps pada browser Mozilla Firefox dan Google Chrome dengan jumlah *client* dua menghasilkan *packet loss* sebesar 2,586 %, untuk jumlah *client* tiga menghasilkan *packet loss* sebesar 0,942 %, untuk jumlah *client* lima

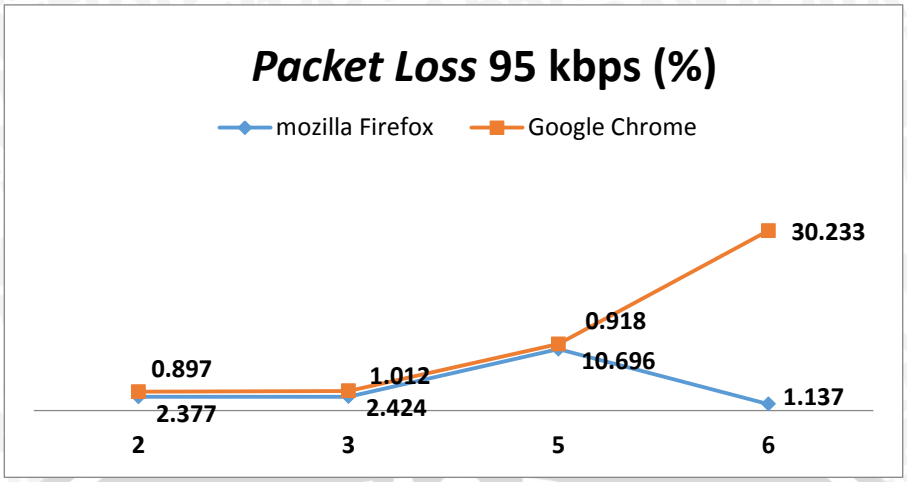
menghasilkan *packet loss* sebesar 2,621 %, dan untuk jumlah *client* sebanyak enam menghasilkan *packet loss* sebesar 6,898 %. Pada browser Google Chrome dengan jumlah *client* dua menghasilkan throughput *packet loss* sebesar 0,897 %, untuk jumlah client tiga menghasilkan *packet loss* sebesar 1,012 %, untuk jumlah *client* lima menghasilkan *packet loss* throughput sebesar 0,918 %, dan untuk jumlah client sebanyak enam menghasilkan *packet loss* sebesar 30,233 %.

Gambar 6.7 menunjukan grafik *packet loss* dengan *bandwidth* 58 kbps pada *browser* Mozilla Firefox dan Google Chrome. Dari grafik terlihat bahwa nilai *packet loss* komunikasi *video conference* pada kondisi *bandwidth* 58 kbps pada Mozilla Firefox cenderung lebih tinggi dibandingkan pada *browser packet loss*, apabila dibandingkan dengan *packet loss* pada standarisasi TIPHON nilai *packet loss* komunikasi *video conference* masuk kedalam kategori baik, dikarenakan nilai rata-rata *packet loss* pada komunikasi *video conference* kurang dari 10 %.



Gambar 6.7 *Packet Loss* 58 kbps

Gambar 6.8 menunjukan grafik *packet loss* dengan *bandwidth* 95 kbps pada *browser* Mozilla Firefox dan Google Chrome. Dari grafik terlihat bahwa nilai *packet loss* komunikasi *video conference* pada kondisi *bandwidth* 95 kbps pada Mozilla Firefox cenderung lebih tinggi dibandingkan pada *browser packet loss*, akan tetapi pada saat jumlah *client* enam pada Google Chrome mengalami kenaikan yang sangat tinggi hal ini disebabkan pada pengiriman UDP tidak terdapat *control congesti* yang berakibat paket-paket akan dikirimkan tanpa perlu dibatasi saat kapasitas antrian sedang mengalami kepadatan.



Gambar 6.8 *Packet Loss* 95 kbps

BAB 7 PENUTUP

7.1 Kesimpulan

Berdasarkan hasil pengujian dan analisis yang telah dilakukan dapat diambil kesimpulan sebagai berikut:

1. Terdapat empat parameter *quality of service* yang telah diuji pada jumlah *client* dan *bandwidth* yang berbeda
   - *Delay* dan *jitter*

     Dapat disimpulkan bahwa nilai *delay* dan *jitter* dari pengujian menghasilkan bahwa semakin banyak jumlah *client* pada saat komunikasi berlangsung maka semakin tinggi nilai *delay* dan *jitter.* Pada saat jumlah *client* meningkat maka Kecepatan terima dan pengiriman paket dari setiap node menyebabkan *jitter* semakin meningkat juga.

   - *Throughput*

     Nilai *throughput* komunikasi *video conference* pada kondisi *bandwidth* 58 dan 95 kbps, semakin banyak jumlah *client* pada saat komunikasi berlangsung maka semakin tinggi nilai *throughput* yang dihasilkan.

   - *Packets Loss*

     Nilai *packet loss* komunikasi *video conference* pada kondisi *bandwidth* 58 dan 95 kbps, semakin banyak jumlah *client* pada saat komunikasi berlangsung maka semakin tinggi nilai *packet loss* yang dihasilkan. Faktor yang menyebabkan terjadinya *packets loss* karena pada pengiriman UDP tidak terdapat *control congesti* yang berakibat paket-paket dikirimkan tanpa dibatasi saat kapasitas *queue* sedang mengalami kepadatan.

2. Dari hasil parameter *delay*, *jitter*, *throughput* dan *packet loss* pada komunikasi *video conference* WebRTC dengan menggunakan *framework* RTCMultiConnection, dapat disimpulkan bahwa nilai parameter *delay* dan *jitter* dapat dikategorikan pada kategori baik menurut versi TIPHON, namun pada nilai parameter *throughput* dapat dikategorikan pada kategori kurang baik dan nilai *packets loss* masuk pada kategori sedang menurut versi TIPHON.

7.2 Saran

Saran yang dapat disampaikan untuk pengembangan lebih lanjut adalah sebagai berikut:

1. Perlu dilakukan adanya Pengujian yang dimulai ketika terbentuknya suatu proses pembentukan komunikasi akan berlangsung.
2. Perlu dilakukan penelitian lebih lanjut untuk pengujian selain menggunakan codec VP8

DAFTAR PUSTAKA

Agassi, Mohammad, Vicky. 2017. Analisis Skalabilitas Dan Quality of Service komunikasi Audio Call Pada WebRTC dengan menggunakan Framework EasyRTC. Universitas Brawijaya.

Aulia, Nur, Hendra, 2006. *Perancangan MAC untuk layanan video conference sebagai alat bantu perkuliahan.* STT Telkom Bandung. www.scribd.com. Diakses pada tanggal 02 Februari 2016.

Bidelman, Eric, 2012. Capturing Audio & Video in HTML5. Tersedia di : < http://www.html5rocks.com/en/tutorials/getusermedia/intro/> [Diakses 30 Januari 2016>.

Phil Edholm, E. Brent Kelly, Ph.D., Matt Krebs, November 2014, "*WebRTC Ecosystem "Overview" A Description of the Ecosystem Framework*", pedholm@pkeconsulting.com, bkelly@kelcor.com, mattkrebs@kelcor.com.

Priaksa, Indradito, Boby. *Implementasi dan Analisis Performansi Mobicents Sebagai Server Aplikasi pada Arsitektur IP Multimedia Subsistem untuk Layanan Video Conference.* Universitas Telkom Bandung.

Primadasa, I Gede, Putu, Bagus. 2011. Kompresi Video Conference Degan Standar H-263 Dan H261. Universitas Udayana

Nurdiansyah Deby Cahya, 2013. *Implementasi Video Conference Pada Jaringan Hsupa (High SpeedUplink Packet Access) Dengan MediaIpv6 Menggunakan Simulator Opnet Modeler V.14.5*. Universitas Brawijaya.

Rhinow, Florian, Veloso, Pablo, Porto, Barrett Stephen & Nuollain, O, Eamonn. *P2P Live Video Streaming in WebRTC*. School pf Computer Scienceand Statisitcs, Trinity College Dublin.

Sepasthika, Rizki. 2015. Analisis Kinerja Real-Time Transport protokol dan Real-Time Transport control protokol pada VoIP menggunakan Codec GSM & G711. Universitas Brawijaya.

Taheri, Sajjad, Beni, Laleh, Aghababaie, Veidenbaum, V., Alexander & Nicolau Alexandru, 2015. *WebRTCBench : A Benchmark for Performansi Assessment of WebRTC Implementation*. Dept. of Computer Science, UC Irvine, Irvine, CA.

Tiphon. "*Telecommunication and Internet Protocol Harmonization Over Network (TIPHON) General Aspec of Quality of Service (QoS)*", DTR/TIPHON-05006 (cb0010cs.PDF).1999.

WebRTC. WebRTC Architecture. Tersedia di : https://webrtc.org/ [Diakses 01 Februari 2016].

LAMPIRAN A HASIL PENGUJIAN MOZILLA FIREFOX *BANDWIDTH* 95 KBPS

A.1 Pengujian Dua *Client*

Percobaan 1

Delay	
Total *Delay*	241.5777
Total Paket yang diterima	39932
rata - rata *delay* (s)	0.0060
rata - rata *delay* (ms)	6.0497

Jitter	
Total variasi *delay*	0.000055
Total paket yang diterima	39932
Nilai *Jitter* (s)	0.000000
Nilai *Jitter* (ms)	0.000001

Throughput	
paket yang dikirim (bytes)	14727978
lama pengamatan (s)	241.57769
Nilai *Throughput* (bps)	60965.804
Throughput (kbps)	60.966

Packet Loss	
Paket yang dikirim	40206
paket yang diterima	39932
nilai p*acket loss*	0.68

Percobaan 2

Delay	
Total *Delay*	238.6504
Total Paket yang diterima	34446
rata - rata *delay* (s)	0.0069
rata - rata *delay* (ms)	6.9282

Jitter	
Total variasi *delay*	0.07957
Total paket yang diterima	34446
Nilai *Jitter* (s)	0.000002
Nilai *Jitter* (ms)	0.002310

Throughput	
paket yang dikirim (bytes)	17328240
lama pengamatan (s)	238.65038
Nilai *Throughput* (bps)	72609.3
Throughput (kbps)	72.609

Packet Loss	
Paket yang dikirim	35121
paket yang diterima	34446
nilai p*acket loss*	1.92

Percobaan 3

Delay	
Total *Delay*	233.2113
Total Paket yang diterima	30891
rata - rata *delay* (s)	0.0075
rata - rata *delay* (ms)	7.5495

Jitter	
Total variasi *delay*	0.028624
Total paket yang diterima	30891
Nilai *Jitter* (s)	0.000001
Nilai *Jitter* (ms)	0.000007

Throughput	
paket yang dikirim (bytes)	13550160
lama pengamatan (s)	233.2113
Nilai *Throughput* (bps)	58102.5
Throughput (kbps)	58.103

Packet Loss	
Paket yang dikirim	32570
paket yang diterima	30891
nilai p*acket loss*	5.16

A.2 Pengujian Tiga *Client*

Percobaan 1

Delay	
Total *Delay*	239.941
Total Paket yang diterima	34326
rata - rata *delay* (s)	0.007
rata - rata *delay* (ms)	6.990

Jitter	
Total variasi *delay*	0.035801
Total paket yang diterima	34326
Nilai *Jitter* (s)	0.000001
Nilai *Jitter* (ms)	0.001043

Throughput	
paket yang dikirim (bytes)	15846891
lama pengamatan (s)	239.941
Nilai *Throughput* (bps)	66044.85
Throughput (kbps)	66.045

Packet Loss	
Paket yang dikirim	34778
paket yang diterima	34326
nilai p*acket loss*	1.30

Percobaan 2

Delay	
Total *Delay*	231.573
Total Paket yang diterima	29927
rata - rata *delay* (s)	0.008
rata - rata *delay* (ms)	7.738

Jitter	
Total variasi *delay*	0.000513
Total paket yang diterima	29927
Nilai *Jitter* (s)	0.000000
Nilai *Jitter* (ms)	0.000017

Throughput	
paket yang dikirim (bytes)	14153498
lama pengamatan (s)	231.573
Nilai *Throughput* (bps)	61118.89
Throughput (kbps)	62.919

Packet Loss	
Paket yang dikirim	30324
paket yang diterima	29927
nilai p*acket loss*	1.31

Percobaan 3

Delay	
Total *Delay*	249.339
Total Paket yang diterima	31653
rata - rata *delay* (s)	0.008
rata - rata *delay* (ms)	7.877

Jitter	
Total variasi *delay*	0.039985
Total paket yang diterima	31653
Nilai *Jitter* (s)	0.000001
Nilai *Jitter* (ms)	0.001263

Throughput	
paket yang dikirim (bytes)	15303998
lama pengamatan (s)	239.941
Nilai *Throughput* (bps)	63782.24
Throughput (kbps)	63.982

Packet Loss	
Paket yang dikirim	31722
paket yang diterima	31653
nilai p*acket loss*	0.22

A.3 Pengujian Lima *Client*

Delay	
Total *Delay*	258.880
Total Paket yang diterima	36009
rata - rata *delay* (s)	0.007
rata - rata *delay* (ms)	7.189

Jitter	
Total variasi *delay*	0.034535
Total paket yang diterima	36009
Nilai *Jitter* (s)	0.000001
Nilai *Jitter* (ms)	0.001159

Throughput	
paket yang dikirim (bytes)	20899409
lama pengamatan (s)	258.880
Nilai *Throughput* (bps)	80730.24
Throughput (kbps)	80.730

Packet Loss	
Paket yang dikirim	36458
paket yang diterima	36009
nilai *Paket Loss*	1.23

Delay	
Total *Delay*	251.968
Total Paket yang diterima	31108
rata - rata *delay* (s)	0.008
rata - rata *delay* (ms)	8.100

Jitter	
Total variasi *delay*	0.036548
Total paket yang diterima	31108
Nilai *Jitter* (s)	0.000001
Nilai *Jitter* (ms)	0.001175

Throughput	
paket yang dikirim (bytes)	19170602
lama pengamatan (s)	251.968
Nilai *Throughput* (bps)	76083.48
Throughput (kbps)	76.083

Packet Loss	
Paket yang dikirim	31923
paket yang diterima	31108
nilai *Paket Loss*	2.55

Delay	
Total *Delay*	237.336
Total Paket yang diterima	48270
rata - rata *delay* (s)	0.005
rata - rata *delay* (ms)	8.917

Jitter	
Total variasi *delay*	0.024734
Total paket yang diterima	48270
Nilai *Jitter* (s)	0.000001
Nilai *Jitter* (ms)	0.000512

Throughput	
paket yang dikirim (bytes)	20213924
lama pengamatan (s)	237.336
Nilai *Throughput* (bps)	85169.97
Throughput (kbps)	80.170

Packet Loss	
Paket yang dikirim	50322
paket yang diterima	48270
nilai *Paket Loss*	4.08

A.4 Pengujian Enam *Client*

Percobaan 1

Delay	
Total *Delay*	251.931
Total Paket yang diterima	30721
rata - rata *delay* (s)	0.008
rata - rata *delay* (ms)	8.201

Jitter	
Total variasi *delay*	0.036548
Total paket yang diterima	30721
Nilai *Jitter* (s)	0.000001
Nilai *Jitter* (ms)	0.001190

Throughput	
paket yang dikirim (bytes)	19119308
lama pengamatan (s)	251.931
Nilai *Throughput* (bps)	75891.03
Throughput (kbps)	75.891

Packet Loss	
Paket yang dikirim	31923
paket yang diterima	30721
nilai *Paket Loss*	3.77

Percobaan 2

Delay	
Total *Delay*	252.931
Total Paket yang diterima	29721
rata - rata *delay* (s)	0.009
rata - rata *delay* (ms)	8.510

Jitter	
Total variasi *delay*	0.036548
Total paket yang diterima	29721
Nilai *Jitter* (s)	0.000001
Nilai *Jitter* (ms)	0.001230

Throughput	
paket yang dikirim (bytes)	19119308
lama pengamatan (s)	252.931
Nilai *Throughput* (bps)	75590.98
Throughput (kbps)	80.591

Packet Loss	
Paket yang dikirim	31923
paket yang diterima	29721
nilai *Paket Loss*	6.90

Percobaan 3

Delay	
Total *Delay*	249.931
Total Paket yang diterima	28721
rata - rata *delay* (s)	0.009
rata - rata *delay* (ms)	8.702

Jitter	
Total variasi *delay*	0.036548
Total paket yang diterima	28721
Nilai *Jitter* (s)	0.000001
Nilai *Jitter* (ms)	0.001273

Throughput	
paket yang dikirim (bytes)	19119308
lama pengamatan (s)	249.931
Nilai *Throughput* (bps)	76498.32
Throughput (kbps)	80.498

Packet Loss	
Paket yang dikirim	31923
paket yang diterima	28721
nilai *Paket Loss*	10.03

LAMPIRAN B HASIL PENGUJIAN MOZILLA FIREFOX *BANDWIDTH* 58 KBPS

B.1 Pengujian Dua *Client*

Percobaan ke -1

Delay	
Total *Delay*	241.029
Total Paket yang diterima	30838
rata - rata *delay* (s)	0.008
rata - rata *delay* (ms)	7.816

Jitter	
Total variasi *delay*	0.000038
Total paket yang diterima	30838
Nilai *Jitter* (s)	0.000000
Nilai *Jitter* (ms)	0.000001

Throughput	
paket yang dikirim (bytes)	11455274
lama pengamatan (s)	241.029
Nilai *Throughput* (bps)	47526.519
Throughput (kbps)	47.527

Packet Loss	
Paket yang dikirim	32872
paket yang diterima	30838
nilai *paket Loss*	6.188

Percobaan ke -2

Delay	
Total *Delay*	244.885
Total Paket yang diterima	32146
rata - rata *delay* (s)	0.008
rata - rata *delay* (ms)	7.618

Jitter	
Total variasi *delay*	0.025468
Total paket yang diterima	32146
Nilai *Jitter* (s)	0.000001
Nilai *Jitter* (ms)	0.000002

Throughput	
paket yang dikirim (bytes)	10733381
lama pengamatan (s)	244.885
Nilai *Throughput* (bps)	43830.214
Throughput (kbps)	43.830

Packet Loss	
Paket yang dikirim	33152
paket yang diterima	32146
nilai *paket Loss*	3.035

Percobaan ke -3

Delay	
Total *Delay*	242.858
Total Paket yang diterima	32848
rata - rata *delay* (s)	0.007
rata - rata *delay* (ms)	7.393

Jitter	
Total variasi *delay*	0.000059
Total paket yang diterima	32848
Nilai *Jitter* (s)	0.000000
Nilai *Jitter* (ms)	0.000002

Throughput	
paket yang dikirim (bytes)	9589710
lama pengamatan (s)	242.858
Nilai *Throughput* (bps)	39486.921
Throughput (kbps)	39.487

Packet Loss	
Paket yang dikirim	34822
paket yang diterima	32848
nilai *paket Loss*	5.669

B.2 Pengujian Tiga *Client*

Percobaan ke -1

Delay	
Total *Delay*	285.436
Total Paket yang diterima	30844
rata - rata *delay* (s)	0.009
rata - rata *delay* (ms)	9.254

Jitter	
Total variasi *delay*	0.000055
Total paket yang diterima	30844
Nilai *Jitter* (s)	0.000000
Nilai *Jitter* (ms)	0.000002

Throughput	
paket yang dikirim (bytes)	12660536
lama pengamatan (s)	285.436
Nilai *Throughput* (bps)	44355.017
Throughput (kbps)	44.355

Packet Loss	
Paket yang dikirim	31398
paket yang diterima	30844
nilai *paket Loss*	1.764

Percobaan ke -2

Delay	
Total *Delay*	258.903
Total Paket yang diterima	26562
rata - rata *delay* (s)	0.010
rata - rata *delay* (ms)	9.247

Jitter	
Total variasi *delay*	0.000203
Total paket yang diterima	26562
Nilai *Jitter* (s)	0.000000
Nilai *Jitter* (ms)	0.000008

Throughput	
paket yang dikirim (bytes)	11319502
lama pengamatan (s)	258.903
Nilai *Throughput* (bps)	43721.091
Throughput (kbps)	43.721

Packet Loss	
Paket yang dikirim	30938
paket yang diterima	26562
nilai *paket Loss*	14.144

Percobaan ke -3

Delay	
Total *Delay*	242.661
Total Paket yang diterima	32488
rata - rata *delay* (s)	0.007
rata - rata *delay* (ms)	7.469

Jitter	
Total variasi *delay*	0.000088
Total paket yang diterima	32488
Nilai *Jitter* (s)	0.000000
Nilai *Jitter* (ms)	0.000003

Throughput	
paket yang dikirim (bytes)	10939366
lama pengamatan (s)	242.661
Nilai *Throughput* (bps)	45080.882
Throughput (kbps)	45.081

Packet Loss	
Paket yang dikirim	34994
paket yang diterima	32488
nilai *paket Loss*	7.161

B.3 Pengujian Lima *Client*

Percobaan ke -1

Delay	
Total *Delay*	228.015
Total Paket yang diterima	22036
rata - rata *delay* (s)	0.010
rata - rata *delay* (ms)	10.347

Jitter	
Total variasi *delay*	0.00777
Total paket yang diterima	22036
Nilai *Jitter* (s)	0.00000
Nilai *Jitter* (ms)	0.00035

Throughput	
paket yang dikirim (bytes)	10334211
lama pengamatan (s)	228.015
Nilai *Throughput* (bps)	45322.507
Throughput (kbps)	45.323

Packet Loss	
Paket yang dikirim	24396
paket yang diterima	22036
nilai *paket Loss*	9.674

Percobaan ke -2

Delay	
Total *Delay*	241.877
Total Paket yang diterima	26528
rata - rata *delay* (s)	0.009
rata - rata *delay* (ms)	9.118

Jitter	
Total variasi *delay*	0.00070
Total paket yang diterima	26528
Nilai *Jitter* (s)	0.00000
Nilai *Jitter* (ms)	0.00003

Throughput	
paket yang dikirim (bytes)	11174834
lama pengamatan (s)	241.877
Nilai *Throughput* (bps)	46200.572
Throughput (kbps)	46.201

Packet Loss	
Paket yang dikirim	28808
paket yang diterima	26528
nilai *paket Loss*	7.914

Percobaan ke -3

Delay	
Total *Delay*	235.301
Total Paket yang diterima	22682
rata - rata *delay* (s)	0.010
rata - rata *delay* (ms)	9.530

Jitter	
Total variasi *delay*	0.03237
Total paket yang diterima	22682
Nilai *Jitter* (s)	0.00000
Nilai *Jitter* (ms)	0.00103

Throughput	
paket yang dikirim (bytes)	10024818
lama pengamatan (s)	235.301
Nilai *Throughput* (bps)	42604.288
Throughput (kbps)	42.604

Packet Loss	
Paket yang dikirim	24492
paket yang diterima	22682
nilai *paket Loss*	7.390

B.4 Pengujian Enam *Client*

Percobaan ke -1

Delay	
Total *Delay*	271.636
Total Paket yang diterima	28604
rata - rata *delay* (s)	0.009
rata - rata *delay* (ms)	9.496

Jitter	
Total variasi *delay*	0.00555
Total paket yang diterima	28604
Nilai *Jitter* (s)	0.00000
Nilai *Jitter* (ms)	0.00019

Throughput	
paket yang dikirim (bytes)	11245838
lama pengamatan (s)	271.636
Nilai *Throughput* (bps)	41400.460
Throughput (kbps)	41.400

Packet Loss	
Paket yang dikirim	28966
paket yang diterima	28602
nilai *paket Loss*	1.257

Percobaan ke -2

Delay	
Total *Delay*	251.728
Total Paket yang diterima	45872
rata - rata *delay* (s)	0.005
rata - rata *delay* (ms)	9.496

Jitter	
Total variasi *delay*	0.00899
Total paket yang diterima	45872
Nilai *Jitter* (s)	0.00000
Nilai *Jitter* (ms)	0.00020

Throughput	
paket yang dikirim (bytes)	15559553
lama pengamatan (s)	251.728
Nilai *Throughput* (bps)	61811.063
Throughput (kbps)	61.811

Packet Loss	
Paket yang dikirim	47470
paket yang diterima	45872
nilai *paket Loss*	3.366

Percobaan ke -3

Delay	
Total *Delay*	244.071
Total Paket yang diterima	50756
rata - rata *delay* (s)	0.005
rata - rata *delay* (ms)	10.774

Jitter	
Total variasi *delay*	0.00538
Total paket yang diterima	50756
Nilai *Jitter* (s)	0.00000
Nilai *Jitter* (ms)	0.00011

Throughput	
paket yang dikirim (bytes)	10794278
lama pengamatan (s)	244.071
Nilai *Throughput* (bps)	44225.986
Throughput (kbps)	44.226

Packet Loss	
Paket yang dikirim	51472
paket yang diterima	50756
nilai *paket Loss*	1.391

LAMPIRAN C HASIL PENGUJIAN GOOGLE CHROME *BANDWIDTH* 95 KBPS

C.1 Pengujian Dua *Client*

Percobaan 1

Delay	
Total *Delay*	220.883
Total Paket yang diterima	26314
rata - rata *delay* (s)	0.008
rata - rata *delay* (ms)	7.394

Jitter	
Total variasi *delay*	0.005010
Total paket yang diterima	26314
Nilai *Jitter* (s)	0.0000002
Nilai *Jitter* (ms)	0.000190

Throughput	
paket yang dikirim (bytes)	19623484
lama pengamatan (s)	220.883
Nilai *Throughput* (bps)	88841.22
Throughput (kbps)	88.841

Packet Loss	
Paket yang dikirim	26534
paket yang diterima	26314
nilai *paket Loss*	0.83

Percobaan 2

Delay	
Total *Delay*	241.181
Total Paket yang diterima	36623
rata - rata *delay* (s)	0.007
rata - rata *delay* (ms)	6.586

Jitter	
Total variasi *delay*	0.000060
Total paket yang diterima	36623
Nilai *Jitter* (s)	0.0000000
Nilai *Jitter* (ms)	0.000002

Throughput	
paket yang dikirim (bytes)	18035074
lama pengamatan (s)	241.181
Nilai *Throughput* (bps)	74778.11
Throughput (kbps)	74.778

Packet Loss	
Paket yang dikirim	36836
paket yang diterima	36623
nilai *paket Loss*	0.58

Percobaan 3

Delay	
Total *Delay*	245.436
Total Paket yang diterima	40421
rata - rata *delay* (s)	0.006
rata - rata *delay* (ms)	6.072

Jitter	
Total variasi *delay*	0.034011
Total paket yang diterima	40421
Nilai *Jitter* (s)	0.000001
Nilai *Jitter* (ms)	0.000841

Throughput	
paket yang dikirim (bytes)	18692285
lama pengamatan (s)	245.436
Nilai *Throughput* (bps)	76159.57
Throughput (kbps)	76.160

Packet Loss	
Paket yang dikirim	40947
paket yang diterima	40421
nilai *paket Loss*	1.28

C.2 Pengujian Tiga *Client*

Delay	
Total *Delay*	258.478
Total Paket yang diterima	30678
rata - rata *delay* (s)	0.008
rata - rata *delay* (ms)	8.426

Jitter	
Total variasi *delay*	0.005389
Total paket yang diterima	30678
Nilai *Jitter* (s)	0.000000
Nilai *Jitter* (ms)	0.000176

Throughput	
paket yang dikirim (bytes)	22974219
lama pengamatan (s)	258.478
Nilai *Throughput* (bps)	88882.53
Throughput (kbps)	88.883

Packet Loss	
Paket yang dikirim	30805
paket yang diterima	30678
nilai *paket Loss*	0.4122707

Delay	
Total *Delay*	234.267
Total Paket yang diterima	34562
rata - rata *delay* (s)	0.007
rata - rata *delay* (ms)	6.778

Jitter	
Total variasi *delay*	0.00243
Total paket yang diterima	34562
Nilai *Jitter* (s)	0.000000
Nilai *Jitter* (ms)	0.000903

Throughput	
paket yang dikirim (bytes)	13947595
lama pengamatan (s)	154.013
Nilai *Throughput* (bps)	90561.25
Throughput (kbps)	90.561

Packet Loss	
Paket yang dikirim	35069
paket yang diterima	34562
nilai *paket Loss*	1.445721292

Delay	
Total *Delay*	228.185
Total Paket yang diterima	34460
rata - rata *delay* (s)	0.007
rata - rata *delay* (ms)	6.622

Jitter	
Total variasi *delay*	0.003270
Total paket yang diterima	34460
Nilai *Jitter* (s)	0.000000
Nilai *Jitter* (ms)	0.000095

Throughput	
paket yang dikirim (bytes)	14419617
lama pengamatan (s)	228.185
Nilai *Throughput* (bps)	63192.59
Throughput (kbps)	63.193

Packet Loss	
Paket yang dikirim	34871
paket yang diterima	34460
nilai *paket Loss*	1.17863

C.3 Pengujian Lima *Client*

Delay	
Total *Delay*	221.782
Total Paket yang diterima	24279
rata - rata *delay* (s)	0.009
rata - rata *delay* (ms)	9.135

Delay	
Total *Delay*	246.346
Total Paket yang diterima	40175
rata - rata *delay* (s)	0.006
rata - rata *delay* (ms)	7.132

Delay	
Total *Delay*	220.782
Total Paket yang diterima	24276
rata - rata *delay* (s)	0.009
rata - rata *delay* (ms)	9.095

Jitter	
Total variasi *delay*	0.014170
Total paket yang diterima	24279
Nilai *Jitter* (s)	0.000001
Nilai *Jitter* (ms)	0.000584

Jitter	
Total variasi *delay*	0.014256
Total paket yang diterima	40175
Nilai *Jitter* (s)	0.000000
Nilai *Jitter* (ms)	0.000355

Jitter	
Total variasi *delay*	0.014170
Total paket yang diterima	24276
Nilai *Jitter* (s)	0.000001
Nilai *Jitter* (ms)	0.000584

Throughput	
paket yang dikirim (bytes)	19978539
lama pengamatan (s)	221.782
Nilai *Throughput* (bps)	90081.83
Throughput (kbps)	90.082

Throughput	
paket yang dikirim (bytes)	18576197
lama pengamatan (s)	246.346
Nilai *Throughput* (bps)	75407.06
Throughput (kbps)	75.407

Throughput	
paket yang dikirim (bytes)	19978539
lama pengamatan (s)	220.782
Nilai *Throughput* (bps)	90489.85
Throughput (kbps)	90.490

Packet Loss	
Paket yang dikirim	24445
paket yang diterima	24279
nilai *paket Loss*	0.68

Packet Loss	
Paket yang dikirim	40484
paket yang diterima	40175
nilai *paket Loss*	0.76

Packet Loss	
Paket yang dikirim	24445
paket yang diterima	24276
nilai *paket Loss*	0.69

LAMPIRAN D HASIL PENGUJIAN GOOGLE CHROME *BANDWIDTH* 58 KBPS

D.1 Pengujian Dua *Client*

Percobaan 1

Delay	
Total *Delay*	264.432
Total Paket yang diterima	29798
rata - rata *delay* (s)	0.009
rata - rata *delay* (ms)	5.074

Jitter	
Total variasi *delay*	0.00023
Total paket yang diterima	29798
Nilai *Jitter* (s)	0.00000
Nilai *Jitter* (ms)	0.00001

Throughput	
paket yang dikirim (bytes)	12783193
lama pengamatan (s)	264.432
Nilai *Throughput* (bps)	48342.103
Throughput (kbps)	48.342

Packet Loss	
Paket yang dikirim	30826
paket yang diterima	29798
nilai *paket Loss*	3.335

Percobaan 2

Delay	
Total *Delay*	255.430
Total Paket yang diterima	29842
rata - rata *delay* (s)	0.009
rata - rata *delay* (ms)	8.559

Jitter	
Total variasi *delay*	0.00060
Total paket yang diterima	29842
Nilai *Jitter* (s)	0.00000
Nilai *Jitter* (ms)	0.00002

Throughput	
paket yang dikirim (bytes)	12354989
lama pengamatan (s)	255.430
Nilai *Throughput* (bps)	48369.328
Throughput (kbps)	48.369

Packet Loss	
Paket yang dikirim	29866
paket yang diterima	29842
nilai *paket Loss*	0.080

Percobaan 3

Delay	
Total *Delay*	253.372
Total Paket yang diterima	28700
rata - rata *delay* (s)	0.009
rata - rata *delay* (ms)	8.828

Jitter	
Total variasi *delay*	0.01027
Total paket yang diterima	28700
Nilai *Jitter* (s)	0.00000
Nilai *Jitter* (ms)	0.00036

Throughput	
paket yang dikirim (bytes)	12136426
lama pengamatan (s)	253.372
Nilai *Throughput* (bps)	47899.681
Throughput (kbps)	47.900

Packet Loss	
Paket yang dikirim	29808
paket yang diterima	28700
nilai *paket Loss*	3.717

D.2 Pengujian Tiga *Client*

Percobaan ke-1

Delay	
Total *Delay*	247.135
Total Paket yang diterima	30556
rata - rata *delay* (s)	0.008
rata - rata *delay* (ms)	8.088

Jitter	
Total variasi *delay*	0.00264
Total paket yang diterima	30556
Nilai *Jitter* (s)	0.00000
Nilai *Jitter* (ms)	0.00009

Throughput	
paket yang dikirim (bytes)	11661422
lama pengamatan (s)	247.135
Nilai *Throughput* (bps)	47186.385
Throughput (kbps)	47.186

Packet Loss	
Paket yang dikirim	30968
paket yang diterima	30556
nilai *paket Loss*	1.330

Percobaan ke-2

Delay	
Total *Delay*	237.649
Total Paket yang diterima	29962
rata - rata *delay* (s)	0.008
rata - rata *delay* (ms)	7.132

Jitter	
Total variasi *delay*	0.01336
Total paket yang diterima	29962
Nilai *Jitter* (s)	0.00000
Nilai *Jitter* (ms)	0.00045

Throughput	
paket yang dikirim (bytes)	10109306
lama pengamatan (s)	237.649
Nilai *Throughput* (bps)	42538.878
Throughput (kbps)	42.539

Packet Loss	
Paket yang dikirim	30648
paket yang diterima	29962
nilai *paket Loss*	2.238

Percobaan ke-3

Delay	
Total *Delay*	253.890
Total Paket yang diterima	30776
rata - rata *delay* (s)	0.008
rata - rata *delay* (ms)	7.050

Jitter	
Total variasi *delay*	3.00566
Total paket yang diterima	30776
Nilai *Jitter* (s)	0.00010
Nilai *Jitter* (ms)	0.09766

Throughput	
paket yang dikirim (bytes)	11179810
lama pengamatan (s)	253.890
Nilai *Throughput* (bps)	44033.988
Throughput (kbps)	44.034

Packet Loss	
Paket yang dikirim	31960
paket yang diterima	30776
nilai *paket Loss*	3.705

D.3 Pengujian Lima *Client*

Percobaan ke-1

Delay	
Total *Delay*	236.258
Total Paket yang diterima	37120
rata - rata *delay* (s)	0.006
rata - rata *delay* (ms)	6.365

Jitter	
Total variasi *delay*	0.00124
Total paket yang diterima	37120
Nilai *Jitter* (s)	0.00000
Nilai *Jitter* (ms)	0.00003

Throughput	
paket yang dikirim (bytes)	12697372
lama pengamatan (s)	236.258
Nilai *Throughput* (bps)	53743.749
Throughput (kbps)	53.744

Packet Loss	
Paket yang dikirim	40440
paket yang diterima	37120
nilai *paket Loss*	8.210

Percobaan ke-2

Delay	
Total *Delay*	235.307
Total Paket yang diterima	29100
rata - rata *delay* (s)	0.008
rata - rata *delay* (ms)	8.086

Jitter	
Total variasi *delay*	0.00018
Total paket yang diterima	29100
Nilai *Jitter* (s)	0.00000
Nilai *Jitter* (ms)	0.00001

Throughput	
paket yang dikirim (bytes)	13287830
lama pengamatan (s)	235.307
Nilai *Throughput* (bps)	56470.301
Throughput (kbps)	56.470

Packet Loss	
Paket yang dikirim	33440
paket yang diterima	29100
nilai *paket Loss*	12.978

Percobaan ke-3

Delay	
Total *Delay*	253.287
Total Paket yang diterima	41670
rata - rata *delay* (s)	0.006
rata - rata *delay* (ms)	9.090

Jitter	
Total variasi *delay*	0.00029
Total paket yang diterima	41670
Nilai *Jitter* (s)	0.00000
Nilai *Jitter* (ms)	0.00163

Throughput	
paket yang dikirim (bytes)	13490093
lama pengamatan (s)	253.287
Nilai *Throughput* (bps)	53260.150
Throughput (kbps)	53.260

Packet Loss	
Paket yang dikirim	46768
paket yang diterima	41670
nilai *paket Loss*	10.901

D.4 Pengujian Enam *Client*

Percobaan ke-1

Delay	
Total *Delay*	227.703
Total Paket yang diterima	33060
rata - rata *delay* (s)	0.007
rata - rata *delay* (ms)	7.888

Jitter	
Total variasi *delay*	0.0002
Total paket yang diterima	33060
Nilai *Jitter* (s)	0.0000
Nilai *Jitter* (ms)	0.0000

Throughput	
paket yang dikirim (bytes)	15296101
lama pengamatan (s)	227.703
Nilai *Throughput* (bps)	67175.595
Throughput (kbps)	67.176

Packet Loss	
Paket yang dikirim	33348
paket yang diterima	33060
nilai *paket Loss*	0.864

Percobaan ke-2

Delay	
Total *Delay*	230.259
Total Paket yang diterima	31898
rata - rata *delay* (s)	0.007
rata - rata *delay* (ms)	10.219

Jitter	
Total variasi *delay*	0.0052
Total paket yang diterima	31898
Nilai *Jitter* (s)	0.0000
Nilai *Jitter* (ms)	0.0002

Throughput	
paket yang dikirim (bytes)	10036667
lama pengamatan (s)	230.259
Nilai *Throughput* (bps)	43588.654
Throughput (kbps)	43.589

Packet Loss	
Paket yang dikirim	32452
paket yang diterima	31898
nilai *paket Loss*	1.707

Percobaan ke-3

Delay	
Total *Delay*	247.993
Total Paket yang diterima	29692
rata - rata *delay* (s)	0.008
rata - rata *delay* (ms)	8.352

Jitter	
Total variasi *delay*	0.7503
Total paket yang diterima	29692
Nilai *Jitter* (s)	0.0000
Nilai *Jitter* (ms)	0.0253

Throughput	
paket yang dikirim (bytes)	11666012
lama pengamatan (s)	247.993
Nilai *Throughput* (bps)	47041.758
Throughput (kbps)	47.042

Packet Loss	
Paket yang dikirim	29944
paket yang diterima	29692
nilai *paket Loss*	0.842