Analisa Kompatibilitas Dan *Quality Of Service* Komunikasi *Audio Call* Pada WebRTC dengan Menggunakan *Framework* EasyRTC

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Mohammad Vicky Agassi
NIM: 125150207111004

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2016

PENGESAHAN

ANALISA KOMPATIBILITAS DAN *QUALITY OF SERVICE* KOMUNIKASI AUDIO CALL PADA WEBRTC DENGAN MENGGUNAKAN *FRAMEWORK* EASYRTC

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Mohammad Vicky Agassi
NIM: 125150207111004

Skripsi ini telah diuji dan dinyatakan lulus pada
22 Desember 2016
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Rakhmadhany Primananda, S.T, M.Kom	Barlian Henryranu Prasetio, S.T, M.T
NIK: 201609 860406 1 001	NIK: 201102 821024 1 001

Mengetahui
Ketua Jurusan Teknik Informatika

Tri Astoto Kurniawan, S.T, M.T, Ph.D
NIP : 19710518 200312 1 001

PERNYATAAN ORISINALITAS

Penulis menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan penulis, 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, penulis bersedia skripsi ini digugurkan dan gelar akademik yang telah penulis 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, 2 Desember 2016

Mohammad Vicky Agassi

NIM: 125150207111004

KATA PENGANTAR

Puji dan rasa syukur yang penulis panjatkan kehadirat Allah SWT, telah melimpahkan rahmat, hidayah, dan inayah-Nya maka skripsi ini dapat terselesaikan dengan sebaik-baiknya. Salam serta shalawat semoga selalu tercurah pada baginda Rasulullah Muhammad SAW, kepada keluarganya, para sahabatnya, hingga kepada seluruh umatnya hingga akhir zaman, aamiin.

Penulis mengucapkan rasa terimakasih yang sebesar-besarnya atas semua bantuan, bimbingan serta dukungan yang telah diberikan sehingga skripsi ini dapat terselesaikan. Oleh karena itu penulis dengan senang hati mengucapkan rasa terimakasih kepada yang terhormat :

- Bapak Rakhmadhany Primananda, S.T, M.Kom dan Bapak Bapak Barlian Henryranu Prasetio, S.T, M.T selaku dosen pembimbing I dan II yang telah memberikan bimbingan, bantuan dan menyemangati dalam meneliti topik yang dibahas serta memberi saran-saran yang luar biasa, Sehingga membuat penulis pantang meyerah dan selalu bersemangat dalam menyelesaikan skripsi ini.
- Bapak Tri Astoto Kurniawan, S.T, M.T, Ph.D selaku Ketua Jurusan Informatika/Ilmu Komputer.
- Seluruh dosen dan staff Fakultas Ilmu Komputer, Universitas Brawijaya atas ilmu, bimbingan dan bantuannya hingga penulis dapat menyelesaikan skripsi ini.
- Kedua Orang tua penulis yang selalu mendukung, karena tanpa dukungannya penulis tidak akan mampu menempuh masa-masa perkuliahan ini.
- Lestari Sintika Syarah yang banyak membantu dalam proses menyusun skripsi dari awal sampai akhir dan Ellsa Yuniar yang telah bersedia membantu meminjamkan sarana untuk tahap pengujian skripsi penulis.
- Semua teman kos Joyo, kos kembang kertas 9A dan TIF H semester 1 yang telah memberikan inspirasi serta berbagi pengalaman selama masa perkuliahan ini.

Penulis menyadari skripsi ini tidaklah sempurna. Oleh karena itu kritik dan saran yang membangun dan penelitian lebih lanjut sangat diharapkan untuk menutupi kekurangan pada skripsi ini.

Malang, 2 Desember 2016

Penulis

mohammadvickyagassi@rocketmail.com

ABSTRAK

WebRTC adalah sebuah teknologi aplikasi berbasis web yang bersifat *opensource*, memungkinkan pengguna untuk berkomunikasi secara *real-time* tanpa perlu menginstall *plugin* namun hanya menggunakan *browser* yang support untuk WebRTC. Pada penelitian ini WebRTC yang dibangun menggunakan framework easyRTC dan node.js sebagai media signalingnya, codec audio yang digunakan oleh framework easyRTC adalah codec opus.

Tujuan dari penelitian ini adalah untuk mengetahui kompatibilitas dari WebRTC audio call dari framework easyRTC dapat berjalan pada browser Chrome v52.0, Mozilla v38.0.5, Opera v38.0, Safari v5.1.7 dan Internet Explorer v8.0, dan untuk mengetahui *quality of service* (QoS) dari WebRTC audio call, apakah nilai QoS yang dihasilkan dapat memenuhi standarisasi komunikasi yang baik dengan menurut TIPHON. Parameter *quality of service* yang digunakan dalam penelitian ini adalah rata-rata *delay*, *jitter*, *throughput* dan *packets loss*.

Hasil yang diperoleh dari penelitian ini menujukkan bahwa hanya browser Mozilla v38.0.5. quality of service yang dihasilkan mempunyai rata-rata *delay* dengan nilai 9.977 ms menunjukkan indeks sangat baik dan nilai *jitter* 0.000314 ms yang dihasilkan juga masuk dalam indeks baik menurut versi TIPHON, namun masih mempunyai presentase *packets loss* 4.255% yang menunjukkan kategori indeks sedang dan nilai *throughput* 14.019 kbps yang dihasilkan masuk dalam indeks jelek menurut versi TIPHON. Sehingga tergolong cukup untuk kategori komunikasi menurut versi TIPHON.

Kata kunci : WebRTC, easyRTC, node.js, *quality of service* (QoS).

ABSTRACT

WebRTC is a web-based application technology which has a characteristic that is open source, allowing users to communicate in real-time without needing to install the plugin but they just need to use the browser that support for WebRTC. In this research, WebRTC was built by using framework easyRTC and node.js as its signaling media, audio codec that is used by the framework easyRTC is codec opus.

The purpose of this study was to determine the scalability of WebRTC audio call from easyRTC framework can run on v52.0 Chrome browser, Mozilla v38.0.5, v38.0 Opera, Safari and Internet Explorer v5.1.7 v8.0, and to determine the quality of service (QoS) on WebRTC audio call, whether the QoS values generated can be standardized according to a good communication with TIPHON. Quality of service parameters used in this study is the average delay, jitter, throughput and packets loss.

The results obtained from this study showed that only Mozilla v38.0.5. quality of service produced having an average delay with a value of 9977 ms indicates the index is excellent and the value of jitter 0.000314 ms generated is also included in the index according to versions of TIPHON is good, but it still has a percentage of packets loss 4.255% which indicates the category of the index according to the version of TIPHON is medium and throughput 14.019 kbps generated in the index according to the version of TIPHON is bad. Therefore, it is quite enough for the category of communications under TIPHON version.

Keywords : WebRTC, easyRTC, node.js, *quality of service* (QoS).

DAFTAR ISI

PENGESAHAN ii
PERNYATAAN ORISINALITAS iii
KATA PENGANTAR iv
ABSTRAK v
ABSTRACT vi
DAFTAR ISI vii
DAFTAR TABEL ix
DAFTAR Persamaan xi
DAFTAR Source Code xii
DAFTAR GAMBAR xiii
BAB 1 PENDAHULUAN 1
1.1 Latar Belakang 1
1.2 Rumusan Masalah 2
1.3 Tujuan 2
1.4 Manfaat 2
1.5 Batasan Masalah 2
1.6 Sistematika Pembahasan 3
BAB 2 LANDASAN KEPUSTAKAAN 4
2.1 WebRTC 4
2.1.1 Tahap Signaling 5
2.1.2 Api Javascript Pada WebRTC 6
2.1.3 Codec Audio 7
2.2 Perhitungan Quality of Service 7
BAB 3 METODOLOGI 10
3.1 Tahapan Penelitian 10
3.2 Penjelasan Tahapan Penelitian 11
3.2.1 Studi Literatur 11
3.2.2 Analisa Kebutuhan 11
3.2.3 Perancangan Sistem 11
3.2.4 Implementasi 12
3.2.5 Pengujian dan Analisa 13

3.2.6 Pengambilan Kesimpulan 15
BAB 4 Perancangan Sistem 16
4.1 Rancangan Arsitektur 16
BAB 5 Implementasi 21
5.1 Topologi 21
5.1.1 Implementasi Modul Node.js dan Framework EasyRTC pada Server 21
5.1.2 Distribusi Sistem 25
BAB 6 Pengujian dan Analisa 27
6.1 Skenario Pengujian 27
6.2 Pengujian Kompatibilitas Sistem 28
6.2.1 Browser Chrome v52.0 28
6.2.2 Browser Mozilla v38.0.5 29
6.2.3 Browser Opera v38.0 30
6.2.4 Browser Safari v5.1.7 30
6.2.5 Browser Internet Explorer v8.0 31
6.3 Hasil Analisa Kompatibilitas Sistem 31
6.4 Pengujian *Quality of Service* 32
6.4.1 Pengujian 2 menit 32
6.4.2 Pengujian 4 menit 40
6.4.3 Pengujian 6 menit 46
6.4.4 Pengujian 8 menit 53
6.4.5 Hasil Pengujian Dari rentan Waktu 2 s/d 8 menit 59
6.4.6 Analisis Hasil Pengujian Dari rentan Waktu 2 s/d 8 menit 64
BAB 7 PENUTUP 66
7.1 Kesimpulan 66
7.2 Saran 68
DAFTAR PUSTAKA 69

DAFTAR TABEL

Tabel 3.1 Indeks Parameter QoS (Sumber TIPHON) 13
Tabel 3.2 Standarisasi Delay (Sumber TIPHON) 13
Tabel 3.3 Standarisasi Packets Loss (Sumber TIPHON) 13
Tabel 3.4 Standarisasi Throughput (Sumber TIPHON) 14
Tabel 3.5 Standarisasi Jitter (Sumber TIPHON) 14
Tabel 6.1 IP *Address* Skenario Pengujian 27
Tabel 6.2 Hasil Analisa Kompatibilitas 32
Tabel 6.3 Hasil Perhitungan Rata-rata *Delay* Percobaan 1 34
Tabel 6.4 Hasil Perhitungan Rata-rata *Delay* Percobaan 2 34
Tabel 6.5 Hasil Perhitungan Rata-rata *Delay* Percobaan 3 34
Tabel 6.6 Hasil Perhitungan Nilai *Jitter* Percobaan 1 35
Tabel 6.7 Hasil Perhitungan Nilai *Jitter* Percobaan 2 36
Tabel 6.8 Hasil Perhitungan Nilai *Jitter* Percobaan 3 36
Tabel 6.9 Hasil Perhitungan Nilai *Throughput* Percobaan 1 37
Tabel 6.10 Hasil Perhitungan Nilai *Throughput* Percobaan 2 37
Tabel 6.11 Hasil Perhitungan Nilai *Throughput* Percobaan 3 37
Tabel 6.12 Hasil Perhitungan *Packets Loss* Percobaan 1 38
Tabel 6.13 Hasil Perhitungan *Packets Loss* Percobaan 2 38
Tabel 6.14 Hasil Perhitungan *Packets Loss* Percobaan 3 39
Tabel 6.15 *Sample* Nilai Parameter uji QoS Selama 2 menit 40
Tabel 6.16 Hasil Perhitungan Rata-rata *Delay* Percobaan 1 40
Tabel 6.17 Hasil Perhitungan Rata-rata *Delay* Percobaan 2 40
Tabel 6.18 Hasil Perhitungan Rata-rata *Delay* Percobaan 3 41
Tabel 6.19 Hasil Perhitungan Nilai *Jitter* Percobaan 1 42
Tabel 6.20 Hasil Perhitungan Nilai *Jitter* Percobaan 2 42
Tabel 6.21 Hasil Perhitungan Nilai *Jitter* Percobaan 3 42
Tabel 6.22 Hasil Perhitungan Nilai *Throughput* Percobaan 1 43
Tabel 6.23 Hasil Perhitungan Nilai *Throughput* Percobaan 2 43
Tabel 6.24 Hasil Perhitungan Nilai *Throughput* Percobaan 3 44
Tabel 6.25 Hasil Perhitungan *Packets Loss* Perocbaan 1 45
Tabel 6.26 Hasil Perhitungan *Packets Loss* Percobaan 2 45

Tabel 6.27 Hasil Perhitungan *Packets Loss* Percobaan 3 45
Tabel 6.28 Sample Nilai Parameter uji QoS Selama 4 menit 46
Tabel 6.29 Hasil Perhitungan Rata-rata *Delay* Percobaan 1 47
Tabel 6.30 Hasil Perhitungan Rata-rata *Delay* Percobaan 2 47
Tabel 6.31 Hasil Perhitungan Rata-rata *Delay* Percobaan 3 47
Tabel 6.32 Hasil Perhitungan Nilai *Jitter* Percobaan 1 48
Tabel 6.33 Hasil Perhitungan Nilai *Jitter* Percobaan 2 48
Tabel 6.34 Hasil Perhitungan Nilai *Jitter* Percobaan 3 49
Tabel 6.35 Hasil Perhitungan Nilai *Throughput* Percobaan 1 50
Tabel 6.36 Hasil Perhitungan Nilai *Throughput* Percobaan 2 50
Tabel 6.37 Hasil Perhitungan Nilai *Throughput* Percobaan 3 50
Tabel 6.38 Hasil Perhitungan Nilai *Packets Loss* Percobaan 1 51
Tabel 6.39 Hasil Perhitungan Nilai *Packets Loss* Percobaan 2 51
Tabel 6.40 Hasil Perhitungan Nilai *Packets Loss* Percobaan 3 52
Tabel 6.41 *Sample* Nilai Parameter Uji QoS Selama 6 menit 53
Tabel 6.42 Hasil Perhitungan Rata-rata *Delay* Percobaan 1 53
Tabel 6.43 Hasil Perhitungan Rata-rata *Delay* Percobaan 2 53
Tabel 6.44 Hasil Perhitungan Rata-rata *Delay* Percobaan 3 54
Tabel 6.45 Hasil Perhitungan Nilai *Jitter* Percobaan 1 55
Tabel 6.46 Hasil Perhitungan Nilai *Jitter* Percobaan 2 55
Tabel 6.47 Hasil Perhitungan Nilai *Jitter* Percobaan 3 55
Tabel 6.48 Hasil Perhitungan Nilai *Throughput* Percobaan 1 56
Tabel 6.49 Hasil Perhitungan Nilai *Throughput* Percobaan 2 56
Tabel 6.50 Hasil Perhitungan Nilai *Throughput* Percobaan 3 57
Tabel 6.51 Hasil Perhitungan Nilai *Packets Loss* Percobaan 1 58
Tabel 6.52 Hasil Perhitungan Nilai *Packets Loss* Percobaan 2 58
Tabel 6.53 Hasil Perhitungan Nilai *Packets Loss* Percobaan 3 58
Tabel 6.54 *Sample* Nilai Parameter Uji QoS Selama 8 menit 59
Tabel 6.55 Nilai Parameter QoS Yang Dihasilkan Pada Rentan Waktu 2 s/d 8 menit 63

DAFTAR PERSAMAAN

Persamaan 1 *Bandwidth* 7
Persamaan 2 *Delay* 8
Persamaan 3 *Jitter* 8
Persamaan 4 Variasi *Delay* 8
Persamaan 5 *Throughput* 8
Persamaan 6 *Packets Loss* 9
Persamaan 7 Menghitung Rata-rata Nilai dari Keempat Parameter Qos 60
Persamaan 8 Persentase Throughput 65

DAFTAR SOURCE CODE

Source Code 1 Install Node.js 21
Source Code 2 Install Framework EasyRTC 21
Source Code 3 Konfigurasi Folder Pada EasyRTC 22
Source Code 4 Konfigurasi File index.html 24
Source Code 5 Codec Opus 25
Source Code 6 Konfigurasi File server.js 25
Source Code 7 Sintak Menjalankan node.js 25

DAFTAR GAMBAR

Gambar 2.1 Proses *Signaling* (Sumber Feher Ben etal, 2016) 5
Gambar 3.1 Tahapan Penelitian 10
Gambar 3.2 Skema Komunikasi pada WebRTC 12
Gambar 4.1 Flowchart Proses Login 17
Gambar 4.2 Flowchart Proses Terjadinya Komunikasi 18
Gambar 4.3 Flowchart Proses Hangup 19
Gambar 5.1 Proses *Signaling* 26
Gambar 6.1 Skenario Pengujian 27
Gambar 6.2 Akses Media Mikrofon 28
Gambar 6.3 Pengujian Kompatibilitas Pada *Browser* Chrome v52.0 29
Gambar 6.4 Pengujian Kompatibilitas Pada Browser Mozilla v38.0.5 29
Gambar 6.5 Pengujian Kompatibilitas Pada *Browser* Opera v38.0 30
Gambar 6.6 Pengujian Kompatibilitas Pada *Browser* Safari v5.1.7 31
Gambar 6.7 Pengujian Kompatibilitas Pada *Browser* Internet Explorer v8.0 31
Gambar 6.8 Hasil Capture Data Selama 2 menit 33
Gambar 6.9 Summary Hasil *Filtering* Paket Data Pada Wireshark 34
Gambar 6.10 Grafik Rata-rata *Delay* Dari Tiga Kali Percobaan 2 menit 35
Gambar 6.11 Grafik Nilai *Jitter* Dari Tiga Kali Percobaan 2 menit 37
Gambar 6.12 Grafik Nilai *Throughput* Dari Tiga Kali Percobaan 2 menit 38
Gambar 6.13 Grafik Nilai *Packets Loss* Dari Tiga Kali Percobaan 2 menit 39
Gambar 6.14 Grafik Rata-rata *Delay* Dari Tiga Kali Percobaan 4 menit 41
Gambar 6.15 Garfik Nilai *Jitter* Dari Tiga Kali Percobaan 4 menit 43
Gambar 6.16 Grafik Nilai *Throughput* Dari Tiga Kali Percobaan 4 menit 44
Gambar 6.17 Grafik Nilai *Packets Loss* Dari Tiga Kali Percobaan 4 menit 46
Gambar 6.18 Grafik Rata-rata *Delay* Dari Tiga Kali Percobaan 6 menit 48
Gambar 6.19 Grafik Nilai *Jitter* Dari Tiga Kali Percobaan 6 menit 49
Gambar 6.20 Grafik Nilai *Throughput* Dari Tiga Kali Percobaan 6 menit 51
Gambar 6.21 Grafik Nilai *Packets Loss* Dari Tiga Kali Percobaan 6 menit 52
Gambar 6.22 Grafik Nilai Rata-rata *Delay* Dari Tiga Kali Percobaan 8 menit 54
Gambar 6.23 Grafik Nilai *Jitter* Dari Tiga Kali Percobaan 8 menit 56
Gambar 6.24 Grafik Nilai *Throughput* Dari Tiga Kali Percobaan 8 menit 57

Gambar 6.25 Grafik Nilai *Packets Loss* Dari Tiga Kali Percobaan 8 menit 58
Gambar 6.26 Grafik Nilai Rata-rata *Delay* Pada Rentan Waktu 2 s/d 8 menit 60
Gambar 6.27 Grafik Nilai Rata-rata *Jitter* Pada Rentan Waktu 2 s/d 8 menit 61
Gambar 6.28 Grafik Nilai Rata-rata *Throughput* Pada Rentan Waktu 2 s/d 8 menit 62

BAB 1 PENDAHULUAN

1.1 Latar Belakang

Kemajuan teknologi dari masa ke masa telah berkembang begitu cepat, terutama dalam bidang komunikasi. Teknologi komunikasi sangat diperlukan dalam kehidupan dikarenakan dapat mempermudah komunikasi antar manusia walaupun dengan perbedaan jarak dan waktu. Saat ini telah banyak dikembangkan teknologi komunikasi yang menggunakan IP dengan kata lain *Voice Over Internet Protocol* atau biasa disebut dengan VoIP (Muhammad Iqbal dkk, 2012). Sebelumnya, kebanyakan untuk mendapatkan komunikasi VoIP pengguna harus menginstall *plugin* terlebih dahulu agar dapat menggunakan layanan VoIP tersebut. Contoh layanan VoIP yang harus menginstall *plugin* adalah Skype, Facebook Messenger, Line dan sejenisnya. Namun sekarang telah dikembangkan suatu teknologi yang dapat menggunakan layanan VoIP tanpa harus menginstall *plugin*, teknologi tersebut adalah WebRTC (*Web Real-Time Communication*) (Edholm Phil et al, 2014).

WebRTC merupakan sebuah teknologi aplikasi berbasis web yang bersifat *opensource*, memungkinkan pengguna untuk berkomunikasi secara *real-time* tanpa perlu menginstall *plugin*, hanya menggunkan *browser* yang support untuk WebRTC, sehingga pengguna dapat berkomunikasi dengan pihak lain. WebRTC memanfaatkan fitur dari HTML5 dan javascript sehingga memungkinkan untuk mengaktifkan *audio* yang tertanam pada *browser* dan dapat berkomunikasi secara visual dalam browser (Edholm Phil et al, 2014). Meskipun WebRTC adalah *peer-to-peer* namun dalam arsitektur WebRTC memerlukan server sebagai media untuk *signaling* (Manson Rob, 2013). Pada penelitian WebRTC ini menggunakan module Node.js yang didalamnya sudah terinstall socket.io sebagai media untuk *signaling* pada WebRTC sehingga antar pengguna WebRTC dapat terhubung dan dapat berkomunikasi (McLaughlin Brett, 2011).

Saat terjadinya proses komunikasi antar pengguna satu dengan pengguna lainnya, proses pengiriman data akan terus berjalan dan akan berhenti jika salah satu pengguna menghentikan komunikasi tersebut. Dengan begitu data akan semakin banyak atau semakin menumpuk, hal ini memungkinkan nilai jitter semakin meningkat sehingga komunikasi antar pengguna akan terganggu dan mengakibatkan pengguna tidak nyaman melakukan komunikasi pada WebRTC (Muhammad Iqbal C. R. dkk, 2012).

Dari permasalahan diatas penulis ingin melakukan analisa terhadap layanan VoIP WebRTC dengan *framework easyRTC* yang menggunakan node.js sebagai media *signaling* serta untuk mengetahui *browser* apa saja yang *support* dengan *framework easyRTC* yang menggunakan node.js sebagai media *signaling*. WebRTC *audio call* diimplementasikan berjalan pada jaringan lokal yang mempunyai *bandwidth* 128 kbps. Untuk mengetahui baik atau buruk dari layanan VoIP menggunakan metode *Quality Of Service* atau biasa disebut dengan *QoS* pada layanan WebRTC. Parameter QoS yang digunakan adalah rata-rata

delay, throughput, jitter, dan packets loss. Parameter *QoS* tersebut mengacu pada versi TIPHON untuk mengetahui apakah layanan WebRTC yang diteliti telah memenuhi syarat *QoS* yang baik.

1.2 Rumusan Masalah

1. Browser apa yang dapat menjalankan WebRTC *audio call* menggunakan *framework easyRTC*?
2. Berapa nilai rata-rata *delay, jitter, throughput dan packets loss* yang dihasilkan dari rentan waktu komunikasi 2 menit sampai dengan 8 menit?
3. Apakah nilai rata-rata *delay, jitter, throughput dan packets loss* dari *audio call* WebRTC memenuhi standarisasi komunikasi yang baik menurut versi TIPHON?

1.3 Tujuan

Tujuan dari Analisa Kompatibilitas Dan *Quality of Service* Komunikasi Audio pada WebRTC dengan Menggunakan Node.js Sebagai Media *Signaling* ini adalah :

1. Dapat mengetahui nilai rata-rata *delay*, *jitter*, *throughput*, dan *packets loss* dari WebRTC *audio call* menggunakan codec opus.
2. Dapat mengetahui *browser* apa saja yang dapat menjalankan WebRTC *audio call* dengan *framework easyRTC* yang menggunakan node.js sebagai media *signaling*.
3. Dapat mengetahui *quality of service* dari WebRTC *audio call* menggunakan *framework easyRTC*.

1.4 Manfaat

Semoga dengan penelitian ini penulis berharap dapat bermanfaat bagi berbagai pihak, khususnya pengembang WebRTC. Manfaat dari penelitian penulis ini adalah dapat mengetahui *quality of service*, serta *browser* apa saja yang *support* dengan WebRTC yang diimplementasikan pada *framework easyRTC* menggunakan node.js sebagai media *signaling*.

1.5 Batasan Masalah

Dalam menyelesaikan suatu permasalahan diatas maka diperlukannya batasan masalah agar permasalahan diatas dapat lebih mudah diselesaikannya, batasan masalah yang diambil adalah

1. Melakukan analisa WebRTC pada *framework easyRTC* dengan node.js sebagai media *signaling*.
2. Melakukan uji coba komunikasi dengan waktu 2 menit, 4 menit, 6 menit dan 8 menit.
3. Melakukan analisa QoS dengan parameter uji rata-rata *delay*, *jitter*, *throughput* dan *packets loss*.
4. Melakukan uji coba dengan ukuran *bandwidth* 128 kbps.

5. Melakukan uji coba WebRTC pada *browser* mozila firefox.
6. Menggunakan software wireshark untuk menganalisa jaringan.
7. Menggunakan standarisasi *Quality of Service* versi TIPHON sebagai tolak ukur komunikasi yang baik.
8. Menggunakan netbalancer dan netlimiter untuk membatasi pemakaian *bandwidth*.

1.6 Sistematika Pembahasan

Sistematika penulisan yang dikerjakan menggunakan kerangka sebagai berikut :

a) BAB I PENDAHULUAN

Didalam BAB I memuat latar belakang, rumusan masalah, tujuan, manfaat, batasan masalah, dan sistematika penulisan.

b) BAB II TINJAUAN PUSTAKA

Menguraikan kajian pustaka dan dasar teori yang berkaitan dengan kinerja WebRTC.

c) BAB III METODOLOGI PENELITIAN

Membahas tentang metodologi yang akan digunakan untuk menganalisa *Quality Of Service* pada WebRTC.

d) BAB IV Analisis Kebutuhan

Bagian ini berisi kebutuhan-kebutuhan untuk mewujudkan Implementasi WebRTC dengan memanfaatkan Node.js sebagai media *signaling*.

e) BAB V Implementasi

Membahas tentang implementasi dari sistem.

f) BAB VI Pengujian dan Analisa

Memuat proses dan hasil pengujian terhadap sistem yang telah direalisasikan.

g) BAB VII Penutup

Memuat kesimpulan serta saran yang diperoleh dari pembuatan dan pengujian sistem untuk pengembangan lebih lanjut.

BAB 2 LANDASAN KEPUSTAKAAN

Pada bab ini akan dibahas mengenai penelitian sebelumnya yang terkait dengan WebRTC. Berikut adalah data mengenai penelitian-penelitian sejenis yang pernah dilakukan.

2.1 WebRTC

WebRTC (*Web Real-Time Communication*) adalah sebuah proyek *opensource* yang dapat melakukan komunikasi *real-time* dengan menggunakan *browser* tanpa perlu menginstall *plugin*. Untuk mewujudkan komunikasi harus terdapat javascript API yang digunakan untuk pemanfaatan suara, video dan konektivitas (Novianti K dkk, 2014).

Chating merupakan salah satu bagian dari WebRTC, dengan aplikasi *live chat* dapat memudahkan pengguna yang mempunyai keterbatasan penglihatan (Novianti K dkk, 2014). Pada aplikasi *live chat* WebRTC ini hanya menggunakan dua API utama, yaitu

1. RTCPeerConnection

   RTCPeerConnection merupakan javascript API yang bertugas untuk mengatur komunikasi data *streaming* antar *peers* (Novianti K dkk, 2014).

2. RTCDataChannel

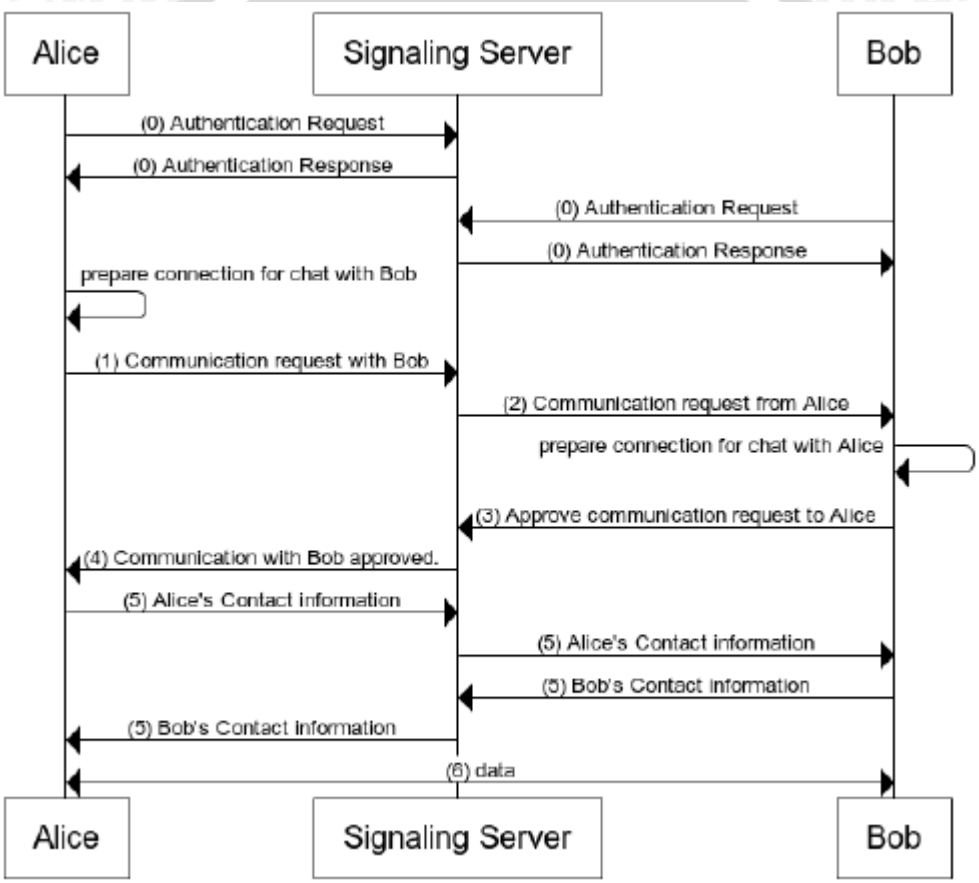
   RTCDataChannel bertugas untuk membantu proses pertukaran data antar *peers* dengan tingkat *delay* yang rendah dan *throughput* yang tinggi (Novianti K dkk, 2014).

Menurut penulis, aplikasi WebRTC *live chat* masih kurang membantu untuk pengguna yang mempunyai keterbatasan penglihatan, karena pengguna jika ingin mengirimkan pesan kepada pengguna lain, pengguna tersebut harus mengetikkan beberapa huruf, sehingga pengguna tersebut masih kesulitan dalam berkomunikasi dengan aplikasi *live chat* tersebut. Pada penelitian penulis ini mengimplementasikan WebRTC *audio call*, sehingga pengguna dapat dapat berkomunikasi dengan suara dan tidak perlu mengetikkan beberapa huruf untuk berkomunikasi antar *peers*.

Untuk mewujudkan WebRTC audio call memerlukan tiga javascript API, yaitu RTCPeerConnection, RTCDataChannel dan UserGetMedia. Penjelasan RTCPeerConnection dan RTCDataChannel sudah dijelaskan seperti diatas. UserGetMedia merupakan javascript API yang dapat memanfaatkan mikrofon dari laptop atau komputer pengguna. UserGetMedia akan meminta izin untuk mendapatkan akses mikrofon dari pengguna, sehingga pengguna dapat berkomunikasi dengan suara (Manson Rob, 2013).

2.1.1 Tahap Signaling

Tahap *signaling* adalah tahap untuk setup percakapan antar dua klien. Ketika dua klien ingin melakukan komunikasi, kedua klien harus mengatur saluran informasi yang akan mereka gunakan. Jika klien A ingin berkomunikasi dengan klien B, maka klien A harus menyampaikan alamatnya kepada klien B agar klien B dapat menghubungi klien A. Gambar 2.1 menunjukkan bahwa skema umum dari proses klien A yang ingin menghubungi klien B.



Gambar 2.1 Proses *Signaling* (Sumber Feher Ben etal, 2016)

(1) Klien A membuka koneksi dan mengirimkan permintaan ke *signaling* server untuk berkomunikasi dengan klien B. Permintaan komunikasi dari klien A ini memberikan informasi tentang kemampuan klien untuk melakukan komunikasi, isi dari informasi tersebut seperti (codec media yang tersedia, versi API, dll) serta kode kode unik dari klien A seperti :

a. Unique session

b. Session waktu mulai/akhir

c. Versi session

d. Media stream (audio atau video)

e. Ice ufrag/ ice pwd

Nilai-nilai diatas akan dikirimkan ke server *signaling* dan setelah koneksi tersambung, maka nilai-nilai tersebut digunakan sebagai otentikasi antar kedua klien yang bertujuan untuk memastikan bahwa klien A dapat memastikan bahwa yang dihubunginya adalah klien B

(2) *Signaling* server mengotentikasi pesan dari klien A dan meneruskannya ke klien B.

(3) Klien B dapat menyetujui atau menolak permintaan komunikasi dari klien A. Jika klien B menyetujui permintaan dari klien A, maka klien B membuka saluran komunikasinya dan mengirimkan respon ke server *signaling*.

(4) Server *signaling* meneruskan pesan dari klien B ke klien A.

(5) Dengan asumsi klien B menyetujui permintaan dari klien A, 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).

(6) Kedua klien memulai direct komunikasi antar peer.

Proses *signaling* diatas 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 (Ansuman Kar, 2014).

2.1.2 Api Javascript Pada WebRTC

WebRTC bergantung pada tiga API, yang masing-masing melakukan fungsi tertentu untuk melakukan komunikasi *real-time* dalam aplikasi web. Ketiga API tersebut adalah getUserMedia, RTCPeerConection, RTCDataChannel (Manson Rob, 2013).

- getUserMedia() adalah untuk meminta persetujuan dari pengguna untuk mengakses kamera atau mikrofon yang nantinya akan disinkronisasikan sehingga dapat melakukan *audio streaming*.
- RTCPeerConnection() adalah inti dari koneksi *peer-to-peer* yang digunakan untuk melakukan proses *streaming* data antar *browser*, namun untuk melakukan *streaming* tersebut diperlukannya sebuah mekanisme untuk koordinasi antar peer yaitu yang dikenal dengan *signaling*.
- RTCDataChannel() berfungsi untuk melakukan pertukaran semua jenis data. RTCDataChannel ini telah dirancang khusus sehingga mempunyai fungsi yang sama dengan API WebSocket melalui metode send() dan *onmessage event*.

2.1.3 Codec Audio

WebRTC memerlukan untuk mengkompresi data *audio* dan video pada komunikasi. Adapun beberapa contoh codec *audio* antara lain G.711, G.726 dan iSAC, sedangkan codec untuk video antara lain H.264 dan VP8.

Dalam penelitian ini penulis mengunakan codec *audio* Opus, codec Opus tersebut sudah ada pada *library framework easyRTC*. Codec opus belakangan ini menjadi standarisasi pada RFC6716. Codec opus menargetkan untuk keperluan aplikasi internet *real-time* (Valin Jean-Marc, 2013). mempunyai spesifikasi sebagai berikut :

- *Support* bitrates dari 6 kbps sampai 510 kbps.
- Mempunyai *frame size* dari 2.5 ms sampai 60 ms.
- *Support* audio dan music.
- *Support* mono dan stereo audio.

2.2 Perhitungan Quality of Service

Terdapat beberapa rumus untuk menghitung beberapa parameter dari QoS yaitu Bandwidth*, Delay, Jitter, Throughput, Packets Loss*. Berikut adalah rumus-rumus yang digunakan dari jurnal yang berjudul "ANALISIS QOS (*QUALITY OF SERVICE*) PADA JARINGAN INTERNET (STUDI KASUS: FAKULTAS TEKNIK UNIVERSITAS TANJUNGPURA)" (Yanto, 2014).

- *Bandwidth*

  Bandwidth adalah sebagai tolak ukur kecepatan dari banyaknya informasi yang dapat mengalir pada *source* ke *destination*. Jenis *bandwidth* ini biasanya diukur dalam bentuk kbps (kilo bit per *second*). Rumus untuk menghitung bandwidth dapat dilihat pada persamaan 1 dibawah ini.

$$Bandwidth = \frac{\text{ukuran data}}{\text{waktu unduh terbaik}}$$

Persamaan 1 *Bandwidth*

- Ukuran data : total data yang berjalan pada suatu jaringan, dengan satuan bit.
- Waktu unduh terbaik : total waktu saat mengunduh suatu data dengan satuan *second* (Yanto. 2014).

- *Delay*

  Delay merupakan waktu yang dibutuhkan suatu paket data untuk sampai dari *source* ke *destination*. Faktor yang dapat mempengarui nilai dari *delay* adalah jarak, media fisik, kongesti, atau proses rentan waktu yang lama. Rumus untuk menghitung rata-rata *delay* dapat dilihat pada persamaan 2 dibawah ini.

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

Persamaan 2 *Delay*

- Total Delay : Total delay pada saat melakukan transfer paket data dari *source* ke *destination*.
- Total paket yang diterima : jumlah paket data yang dapat diterima pada waktu tertentu (Yanto. 2014).

- *Jitter*

Jitter merupakan variasi *delay* dari kedatangan paket data ke destinasi. Faktor yang dapat mempengaruhi *jitter* adalah perubahan pada peningkatan *traffic* pada jaringan sehingga menyebabkan penyempitan pada *bandwidth* dan menyebabkan antrian paket data. Rumus untuk menghitung *jitter* dapat dilihat pada persamaan 3 dibawah ini.

$$Jitter = \frac{\text{Total variasi } delay}{\text{Paket data yang diterima} - 1}$$

Persamaan 3 *Jitter*

- Total variasi *delay* : total dari variasi *delay* yang didapat. Untuk mencari variasi *delay* menggunakan rumus seperti pada persamaan 4 dibawah ini.

$$Variasi\ Delay = (delay\ 2 - delay\ 1) + (delay\ 3 - delay\ 2) + \ldots + (delay\ n - delay\ (n-1))$$

Persamaan 4 Variasi *Delay*

- Paket data yang diterima : jumlah paket data yang dapat diterima pada waktu tertentu (Yanto. 2014).

- *Throughput*

Throughput merupakan *bandwidth* yang diukur dengan satuan waktu tertentu pada saat melakukan transfer data. Rumus yang digunakan untuk menghitung *throughput* dapat dilihat pada persamaan 5 dibawah ini.

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

Persamaan 5 *Throughput*

- Paket data yang diterima : jumlah paket data yang diterima dalam kurun waktu tertentu dengan satuan *bytes*.
- Lama pengamatan : interval waktu yang dilakukan untuk mengamati proses pengiriman paket data (Yanto. 2014).

- *Packets Loss*

Packets Loss merupakan sebuah kondisi atau parameter yang menunjukkan adanya jumlah paket data yang hilang saat dikirim dari *source* ke *destination*. Rumus yang digunakan untuk menghitung *packets loss* dapat dilihat pada persamaan 6 dibawah ini.

$$Packets\ Loss = \frac{(\text{Total paket data yang dikirim} - \text{Total paket data yang diterima})}{\text{Total paket data yang dikirim}} \text{ x } 100\%$$

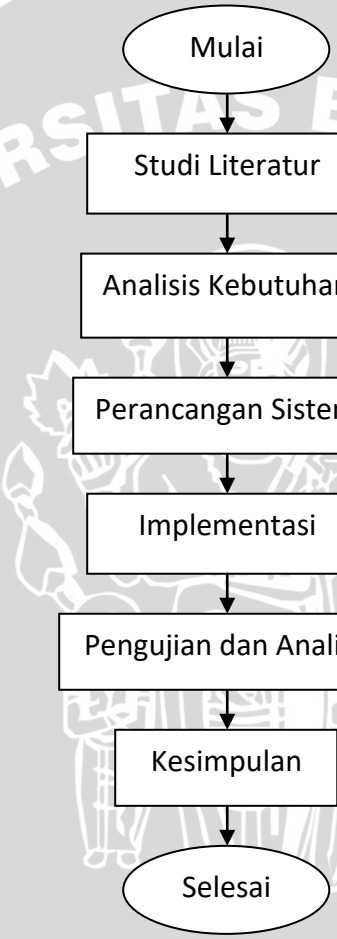
Persamaan 6 *Packets Loss*

- Total paket data yang dikirim : jumlah paket data yang dapat dikirim pada waktu tertentu.
- Total paket data yang diterima : jumlah paket data yang dapat diterima pada waktu tertentu (Yanto. 2014).

BAB 3 METODOLOGI

3.1 Tahapan Penelitian

Konsep metodologi yang digunakan dalam Analisa Kompatibilitas Dan *Quality of Service* Komunikasi *Audio Call* Pada WebRTC dengan Menggunakan Node.js Sebagai *Signaling* terdiri dari Studi *literature*, analisa kebutuhan, perancangan sistem, implementasi, pengujian dan analisa, dan kesimpulan dapat dilihat pada gambar 3.1



Gambar 3.1 Tahapan Penelitian

3.2 Penjelasan Tahapan Penelitian

3.2.1 Studi Literatur

Tahap studi literature dilakukan untuk memperdalam konsep dan teori yang digunakan untuk Analisa Kompatibilitas Dan *Quality of Service* Komunikasi *Audio Call* Pada WebRTC dengan Menggunakan Node.js Sebagai media *Signaling*, terdapat beberapa jurnal maupun buku yang penulis gunakan untuk menunjang penelitian ini.

Beberapa jurnal yang dijadikan referensi dalam penelitian penulis seperti jurnal pada penelitian Muhammad Iqbal C. R., Muchammad Husni dan Hudan Studiawan dengan judul "Implementasi Klien SIP Berbasis Web Menggunakan HTML5 dan Node.js" 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.

3.2.2 Analisa Kebutuhan

Untuk merancang WebRTC *audio call* agar dapat berjalan pada *browser* terdapat beberapa kebutuhan dari perangkat lunak yang penulis gunakan yaitu :

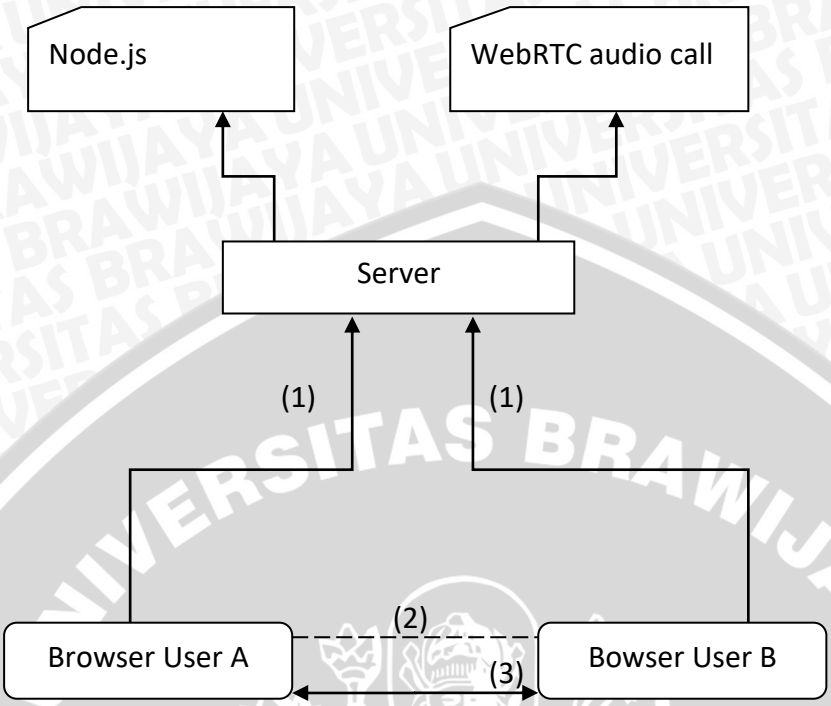
- Ubuntu 14.04 LTS (sebagai OS untuk server).
- Windows 7 (sebagai client 1).
- Windows 8 (sebagai client 2).
- *Framework EasyRTC* (untuk membuat WebRTC *audio call*).
- Node.js (digunakan sebagai proses *signaling*).
- Wireshark v1.12.8 (digunakan untuk men-*capture* paket data yang berjalan pada jaringan lokal).
- Microsoft excel (digunakan untuk menghitung *quality of service*).

Untuk perangkat keras yang digunakan dalam penelitian WebRTC ini yaitu :

- Laptop Acer dengan spesifikasi (core i3, RAM 4GB, NVIDIA GT 540M, *Conexant High Definition Audio*).
- Laptop ASUS dengan spesifikasi (core i3, RAM 2GB, *Realtex Audio*).
- Laptop Samsung dengan spesifikasi (AMD, RAM 2GB)

3.2.3 Perancangan Sistem

Pada perancangan sistem penulis mulai melakukan perancangan sistem WebRTC *audio call* menggunakan *framework easyRTC* dengan node.js sebagai media *signaling*. *Signaling* diperlukan untuk melakukan *call registrasi*, sehingga kedua user yang ingin berkomunikasi dapat terhubung, skema dari proses terjadinya komunikasi *peer-to-peer* yang memanfaatkan server sebagai *signaling* nya dapat dilihat pada gambar 3.2 dibawah ini.



Gambar 3.2 Skema Komunikasi pada WebRTC

1. User A dan user B mengakses halaman WebRTC *audio call* pada server dengan menuliskan ip_server pada port 8181, disini menggunakan metode websocket dari node.js, lalu user A dan user B melakukan *sharing* media mikrofon sehingga user A dan user mendapatkan ID yang berbeda.
2. User A ingin menelepon user B, pada proses ini di perlukannya *signaling* oleh node.js sehingga proses komunikasi dapat terbentuk.
3. User B melakukan *register* pada panggilan dari User A, jika User B menerima panggilan dari User A maka akan terbuat ACK. Setelah ACK terbuat user A dan user B dapat berkomunikasi dengan *direct* atau *peer-to-peer*.

Untuk *client* WebRTC menggunakan *framework easyRTC* sebagai interface *client* pada *browser* dan sebagai media *signaling* nya penulis menggunakan node.js yang sudah terdapat modul socket.io. Untuk codec audio pada WebRTC ini menggunakan codec opus.

3.2.4 Implementasi

Pada tahap implementasi dilakukan setelah tahap perancangan selesai. Implementasi ini dilakukan sesuai dengan rancangan yang telah dibuat pada tahap perancangan, penjelasan tentang implementasi akan dijelaskan lebih detail pada bab Implementasi.

3.2.5 Pengujian dan Analisa

Pada tahap pengujian penulis akan melakukan uji kompatibilitas sistem pada lima *browser*, yaitu chrome v52.0, Mozilla v38.0.5, opera v38.0, safari v5.1.7, dan internet explorer v8.0.

Pada tahap analisa penulis melakukan metode *quality of service*. Penulis mengacu pada standarisasi versi TIPHON, sehingga penulis dapat mengetahui hasil dari monitoring rata-rata *delay*, *jitter*, *throughput*, dan *packets loss* yang nantinya bisa penulis jadikan tolak ukur dari kinerja WebRTC apakah tergolong baik atau jelek. Keempat parameter tersebut yang biasa digunakan untuk mengukur *quality of service* yang berbasis IP yang ada pada suatu jaringan (William S. Bubanto dkk, 2014). Berikut adalah standarisasi dari versi TIPHON.

Nilai	Persentase (%)	Indeks
3.8 – 4	95 – 100	Sangat Baik
3 – 3.79	75 – 94.75	Baik
2 – 2.99	50 – 74.75	Kurang Baik
1 – 1.99	25 – 49.75	Jelek

Tabel 3.1 Indeks Parameter QoS (Sumber TIPHON)

Kategori Delay	Besar Delay	Indeks
Sangat Baik	< 150 ms	4
Baik	150 – 300 ms	3
Sedang	300 – 450 ms	2
Jelek	> 450 ms	1

Tabel 3.2 Standarisasi Delay (Sumber TIPHON)

Kategori Degradasi	Packet loss	Indeks
Sangat Baik	0%	4
Baik	3%	3
Sedang	15%	2
Jelek	25%	1

Tabel 3.3 Standarisasi Packets Loss (Sumber TIPHON)

Kategori Throughput	Throughput	Indeks
Sangat Baik	100%	4
Baik	75%	3
Sedang	50%	2
Jelek	< 25%	1

Tabel 3.4 Standarisasi Throughput (Sumber TIPHON)

Kategori Degradasi	Peak Jitter	Indeks
Sangat Baik	0 ms	4
Baik	0 – 75 ms	3
Sedang	76 – 125 ms	2
Jelek	125 – 225 ms	1

Tabel 3.5 Standarisasi Jitter (Sumber TIPHON)

Tabel 3.1, 3.2, 3.3, 3.4 dan 3.5 digunakan sebagai acuan standarisasi komunikasi yang baik menurut versi TIPHON. Untuk menghitung QoS dengan parameter *delay*, *jitter*, *throughput*, dan *packets loss* menggunakan rumus sebagai berikut :

- *Delay*

Delay merupakan waktu yang dibutuhkan suatu paket data untuk sampai dari *source* ke *destination*. Factor yang dapat mempengarui nilai dari delay adalah jarak, media fisik, kongesti, atau proses rentan waktu yang lama. Rumus untuk menghitung rata-rata delay seperti pada Persamaan 2 *Delay*.

- *Jitter*

Jitter merupakan variasi *delay* dari kedatangan paket data ke destinasi. Faktor yang dapat mempengaruhi *jitter* adalah perubahan pada peningkatan *traffic* pada jaringan sehingga menyebabkan penyempitan pada *bandwidth* dan menyebabkan antrian paket data. Rumus untuk menghitung *jitter* seperti pada Persamaan 3 *Jitter* dan untuk menghitung total variasi *delay* menggunakan Persamaan 4 Variasi *Delay*.

- *Throughput*

Throughput merupakan *bandwidth* yang diukur dengan satuan waktu tertentu pada saat melakukan transfer data. Rumus yang digunakan untuk menghitung *throughput* seperti pada Persamaan 5 *Throughput*.

- *Packets Loss*

Packets Loss merupakan sebuah kondisi atau parameter yang menunjukkan adanya jumlah paket data yang hilang saat dikirim dari *source* ke *destination*. Rumus yang digunakan untuk menghitung *packets loss* seperti pada Persamaan 6 *Packets Loss*.

Yang nantinya akan penulis sesuaikan dengan standarisasi dari versi TIPHON, sehingga dapat mengetahui *Quality of Service* dari *audio call* WebRTC ini.

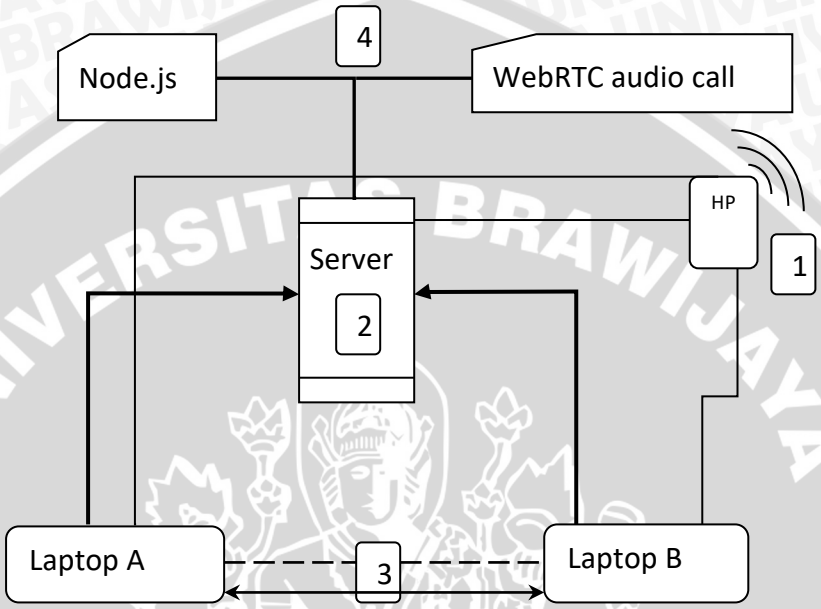
3.2.6 Pengambilan Kesimpulan

Setelah tahap Pengujian dan Analisa selesai penulis akan membuat kesimpulan tentang kompatibilitas WebRTC pada kelima browser tersebut dan *quality of service* WebRTC *audio call* yang dihasilkan dapat memenuhi standarisasi yang baik dari versi TIPHON.

BAB 4 PERANCANGAN SISTEM

Pada bab perancangan sistem, penulis akan menjelaskan rancangan dari sistem yang penulis buat sehingga WebRTC dapat berjalan pada browser.

4.1 Rancangan Arsitektur



Gambar 4.1 Rancangan Arsitektur WebRTC

Pada rancangan arsitektur gambar 4.1 terdapat tiga komponen perangkat dalam arsitektur tersebut, yaitu :

1. *Device Handphone*

   Device handphone (HP) digunakan untuk *tethering*, sehingga *device* yang *connect* ke *hotspot* HP tersebut akan mendapatkan *IP address*, tentunya setiap *device* akan mendapatkan *IP address* yang berbeda.
2. Laptop A

   Pada laptop A menggunakan *Operating System* (OS) windows 7. *IP address* yang didapatkan laptop A 192.168.43.203.
3. Laptop B

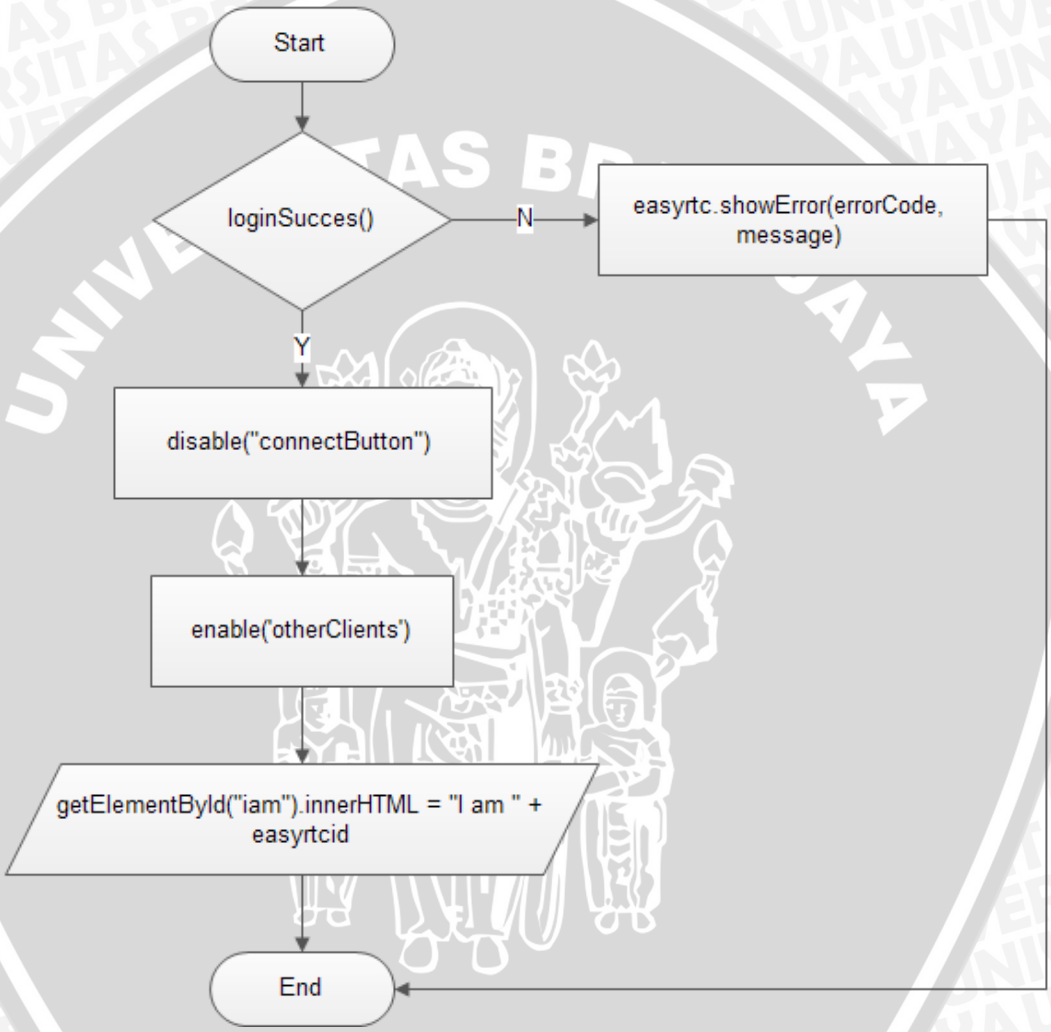
   Pada laptop B hanya terdapat satu OS, yaitu OS windows. Laptop B mendapatkan *IP address* 192.168.43.65.
4. Server

   Pada Server menggunakan OS Ubuntu 14.04. Pada server ini didalamnya terdapat modul node.js, socket.io dan *framework easyRTC*.

Node.js dan socket.io tersebut digunakan untuk proses *signaling* dan proses untuk mengakses WebRTC *audio call* pada port 8181. *IP address* yang digunakan server 192.168.43.215.

Penjelasan tentang empat komponen tersebut berkaitan dengan empat point yang terdapat pada gambar 4.1, berikut adalah flowchart dari proses *login* pada WebRTC, proses terbentuknya komunikasi dan proses *hangup*.

- Flowchart Proses *Login*

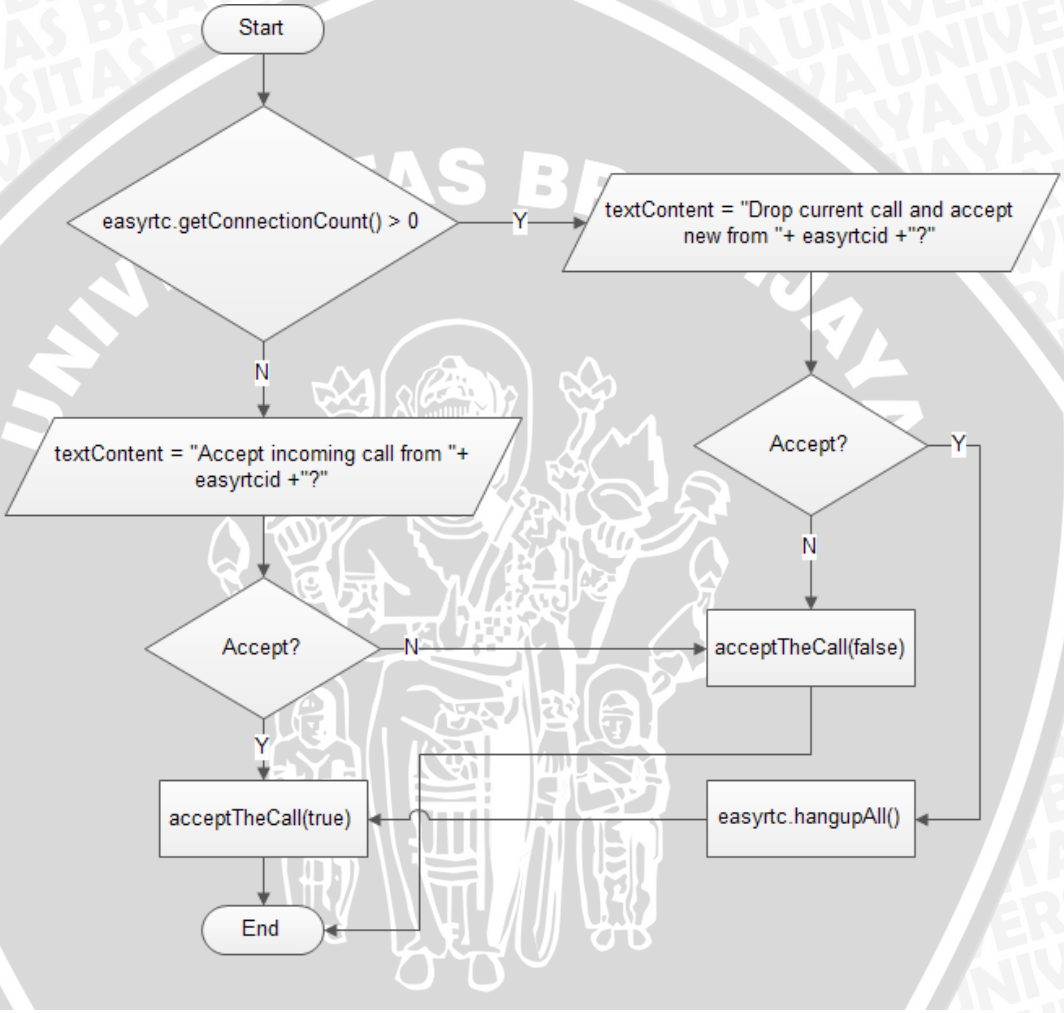


Gambar 4.1 Flowchart Proses Login

Kondisi pada gambar 4.1 adalah dimana *browser* laptop A dan *browser* laptop B telah dapat mengakses WebRTC *audio call* pada server. Pada tahap ini laptop A dan laptop B harus melakukan registrasi terlebih dahulu dengan memilih tombol *connect* pada WebRTC, saat melakukan penggunaka menekan tombol *connect*, method yang dijalankan pada sistem WebRTC adalah loginSuccess(). Node.js akan meminta izin untuk mengakses media mikrofon pada laptop A dan laptop B, apabila *browser* dari pengguna tidak *support* dengan HTML5 dan javascript dari WebRTC maka sistem akan menampilkan

pesan error, jika *browser* dari pengguna *support* dengan HTML5 dan javascript dari WebRTC maka tombol *connect* akan menjadi *disable* dan sistem akan menampilkan pengguna lain yang telah *login* pada sistem serta sistem memberikan sebuah ID pada pengguna, dimana ID tersebut berguna sebagai variabel untuk melakukan komunikasi terhadap pengguna lain, karena setiap pengguna yang *login* pada sistem akan mendapatkan ID yang berbeda.

- Flowchart Proses Terjadinya Komunikasi



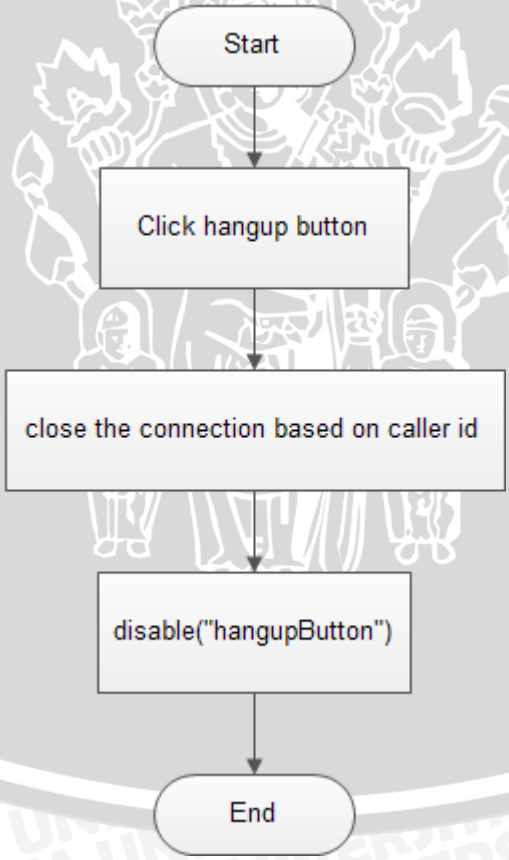
Gambar 4.2 Flowchart Proses Terjadinya Komunikasi

Pada gambar 4.2 adalah proses saat pengguna laptop A ingin menelepon pengguna laptop B, maka laptop A mengirimkan informasi berupa ID nya. Informasi tersebut digunakan untuk proses *signaling* yang dilakukan oleh node.js untuk mencari *node* yang dituju oleh pengguna laptop A. Pada tahap ini sistem akan mendeteksi banyaknya koneksi yang sedang ingin menelepon pengguna laptop B. Apabila method easyRTC.getConnetionCount() yang didapat lebih dari 0, maka pada pengguna laptop B mendapat notifikasi dari sistem, bahwa ada yang ingin menelpon dengan ID A, apakah anda ingin menerima?. Jika pengguna laptop B menerima panggilan dari pengguna laptop A maka proses komunikasi sebelumnya akan di hentikan dengan method hangupAll() dan berganti dengan

komunikasi dengan pengguna laptop A, namun jika pengguna laptop B tidak menerima atau pengguna laptop B me-*reject* panggilan dari pengguna laptop A, maka proses panggilan dari pengguna laptop A ke pengguna laptop B akan dihentikan oleh method acceptTheCall(false).

Jika getConnectionCount kurang dari 0 yang artinya sedang tidak ada aktifitas komunikasi pada pengguna laptop B, maka pada pengguna laptop B mendapatkan notifikasi dari sistem bahwa ada yang meneleponnya dengan ID pengguna A, serta terdapat dua tombol yaitu *accept* dan *reject*. Apabila pengguna laptop B membatalkan panggilan dari pengguna laptop A, maka sistem menjalankan method AcceptTheCall(false), sehingga proses sambungan telepon dari pengguna A ke pengguna B akan terputus. Jika pengguna laptop B menerima panggilan dari pengguna laptop A, maka sistem menjalankan method accpetTheCall(true), sehingga terbentuklah ACK dari kedua pengguna tersebut dan dapat melakukan komunikasi.

- Flowchart Proses Hangup



Gambar 4.3 Flowchart Proses Hangup

Kondisi gambar 4.3 ini adalah saat salah satu dari pengguna laptop A atau pengguna laptop B ingin mengakhiri panggilan teleponnya, dimana mereka telah melakukan proses komunikasi yang telah dijelaskan pada gambar 4.2. Pengguna

tersebut dapat menekan tombol *hangup*, lalu sistem akan menutup koneksi dari *caller* ID yang bersangkutan dan tombol *hangup* akan menjadi *disable*.

BAB 5 IMPLEMENTASI

Pada bab implementasi ini menjelaskan bagaimana mengimplementasikan WebRTC dari konsep yang sudah dirancang pada bab sebelumnya.

5.1 Topologi

Adapun topologi yang digunakan dalam penelitian ini yang nantinya menjadi analisa terhadap hasil nilai dari *Quality of Service* dalam versi TIPHON. Topologi yang digunakan seperti pada Gambar 4.1

Pada topologi tersebut hanya menggunakan dua pengguna WebRTC, dan satu server yang didalamnya sudah terinstall modul node.js untuk proses *signaling*.

5.1.1 Implementasi Modul Node.js dan Framework EasyRTC pada Server

Diperlukannya modul node.js sebagai proses *signaling*, pada penelitian ini dibutuhkan node.js pada server dimana server tersebut berada dalam OS ubuntu 14.04. langkah-langkah untuk menginstall node.js pada server dapat dilihat pada Source Code 1.

```
#curl -sL https://deb.nodesource.com/setup | sudo bash -
#apt-get install -y nodejs
https://github.com/priologic/easyRTC.git
```

Source Code 1 Install Node.js

Setelah node.js selesai terinstal langkah selanjutnya adalah instalasi *framework easyRTC* pada server dengan meletakkan *framework easrtc* pada direktori var/nodes/easyRTC.

```
$sudo su
#svn checkout https://github.com/priologic/easyRTC.git
#cd easyRTC.git/trunk

#mkdir /var/nodes
#mkdir /var/nodes/easyRTC
#cp -a server_example
#cd /var/nodes/easyRTC/server_example
#npm install
#cd node_modules/easyRTC/server_example
#npm install
```

Source Code 2 Install Framework EasyRTC

Setelah *framework easyRTC* selesai terinstal, maka selanjutnya memanfaatkan *javascript* dan css pada *framework easyRTC*, dimana folder tersebut dipindahkan ke direktori /var/nodes/vicky/easyRTC/server_example/node_modules/easyRTC/server_example/static. Langkah-langkahnya untuk memindahkan direktori yang dimaksud dapat dilihat pada Source Code 3.

```
#cd /var/nodes/vicky/easyRTC/server_example/node_modules/easyRTC/demos
#cp -a css
/var/nodes/easyRTC/server_example/node_modules/easyRTC/server_example/static
#cp -a js
/var/nodes/easyRTC/server_example/node_modules/easyRTC/server_example/static
# cp -a images
/var/nodes/easyRTC/server_example/node_modules/easyRTC/server_example/static
```

Source Code 3 Konfigurasi Folder Pada EasyRTC

Sekarang tahap merubah isi *file* yang bernama index.html yang ada pada direktori /var/nodes/vicky/easyRTC/server_example/static, sehingga tampilan WebRTC *audio call* dapat tampil dalam *browser*. Berikut *source code* index.html yang dimodifikasi.

```
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=utf-8" /> <!--skip-->
        <title>WebRTC Audio</title>
        <link rel="stylesheet" type="text/css"
href="/easyRTC/easyRTC.css" />

        <!--hide-->
        <link rel="stylesheet" type="text/css" href="css/landing.css"
/>

        <!-- Prettify Code -->
        <script type="text/javascript"
src="js/prettify/prettify.js"></script>
        <script type="text/javascript"
src="js/prettify/loadAndFilter.js"></script>
        <script type="text/javascript"
src="js/prettify/jquery.min.js"></script>
        <link rel="stylesheet" type="text/css"
href="js/prettify/prettify.css" />

        <!--show-->
        <!--meambahkan library socket.io.js dan easyRTC.js -->
        <script src="/socket.io/socket.io.js"></script>
        <script type="text/javascript"
src="/easyRTC/easyRTC.js"></script>
        <script type="text/javascript"
src="js/demo_audio_only.js"></script>
        <!--hide-->
        <!-- Styles css yang digunakan-->
        <style type="text/css">
            #demoContainer {
                position:relative;
            }
            #connectControls {
                float:left;

```

```
                width:250px;
                text-align:center;
                border: 2px solid black;
            }
            #otherClients {
                height:200px;
                overflow-y:scroll;
            }
            #callerAudio {
                /* display:none; */
                height:10em;
                width:10em;
                margin-left:10px;
            }
            #acceptCallBox {
                display:none;
                z-index:2;
                position:absolute;
                top:100px;
                left:400px;
                border:red solid 2px;
                background-color:pink;
                padding:15px;
            }
        </style>
        <!--show-->
    </head>
    <body>
        <div id="container">
            <!--hide-->
            <div id="header">

            </div>

            <div id="main">
                <!-- Main Content -->

                <!--show-->
                <div id="demoContainer">
                    <div id="connectControls">
                        <button id="connectButton"
onclick="connect()">Connect</button>
                    <button id="hangupButton" disabled="disabled"
onclick="hangup()">Hangup</button>
                    <div id="iam">Not yet connected...</div>
                    <br />
                    <strong>Connected users:</strong>
                    <div id="otherClients"></div>
                </div>

                <div id="videos">
                    <video id="callerAudio"></video>
                    <div id="acceptCallBox"> <!--tombol accept atau
reject di sembunyikan dengan css -->
                        <div id="acceptCallLabel"></div>
                        <button id="callAcceptButton"
>Accept</button> <button id="callRejectButton">Reject</button>
                    </div>
```

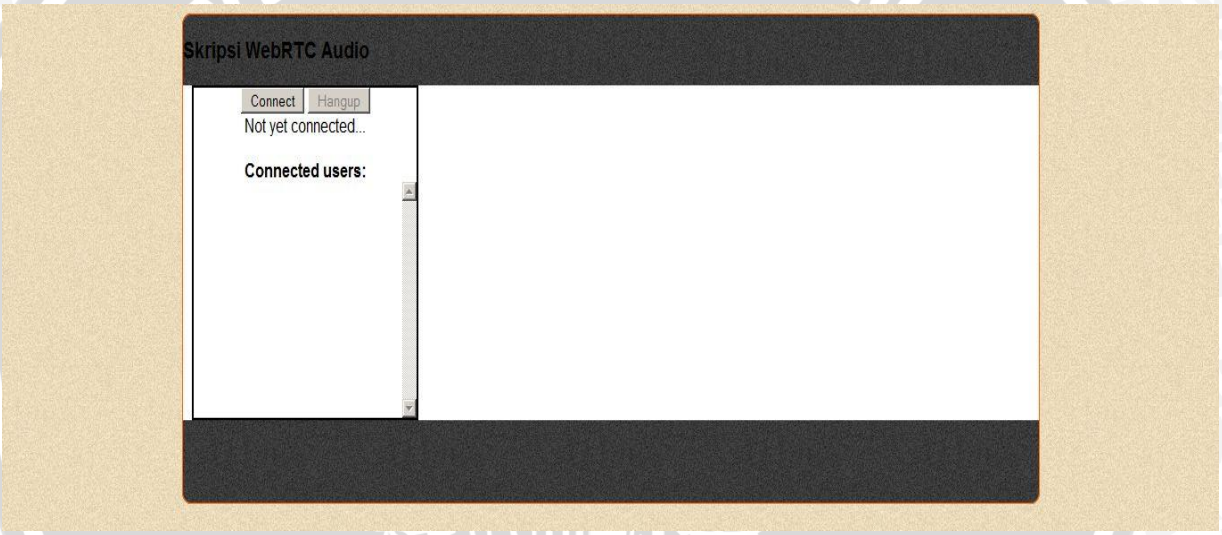
```
                    </div>
                </div>
            </div>

            <div id="footer">

            </div>
        </div>
    </body>
</html>
```

Source Code 4 Konfigurasi File index.html

Inti dari *Source Code* 4 diatas selain memberi tampilan WebRTC pada *website* adalah menintegrasikan javascript dari *framework easyRTC* agar pengguna mendapatkan ID untuk dirinya, ID tersebut digunakan untuk proses *signaling* sehingga antar pengguna dapat melakukan komunikasi. Tampilan dari hasil *Source Code* diatas sebagai berikut



Pengguna akan mendapatkan ID ketika setelah menekan tombol "*Connect*", lalu akan muncul *pop-up* dimana sistem akan meminta izin kepada pengguna untuk mengakses media mikrofonnya.

developer dari *framework easyRTC* ini hanya menggunakan *codec* opus pada *audio*, sehingga pada penelitian ini hanya dapat menggunakan *codec* opus tersebut. Berikut adalah penggalan *source code* yang terdapat pada file "easyRTC_rates.js" yang berada pada direktori /easyRTC/api/labs menandakan bahwa *framework easyRTC* ini menggunakan *codec* opus.

```
function addStereo(sdp){
        var sdpLines = sdp.split('\r\n');
        var opusIndex = findLine(sdpLines, 'a=rtpmap', 'opus/48000');
        var opusPayload;
        if(opusIndex){
           opusPayload = getCodecPayloadType(sdpLines[opusIndex]);
        }
        var fmtpLineIndex = findLine(sdpLines, 'a=fmtp:' +
opusPayload.toString());
```

```
        if(fmtpLineIndex === null){
           return sdp;
        }
        sdpLines[fmtpLineIndex] = sdpLines[fmtpLineIndex].concat('; stereo=1');
        sdp = sdpLines.join('\r\n');
        return sdp;
}
```

Source Code 5 Codec Opus

5.1.2 Distribusi Sistem

Agar pengguna dapat mengakses WebRTC *audio call* maka diperlukannya port untuk alamat WebRTC *audio call*. Pada WebRTC ini memakai port 8181. Berikut adalah konfigurasi pada file server.js pada direktori /var/nodes/easyRTC/server_example/node_modules/easyRTC/server_example.

```
1  // Memanggil modul yang dibutuhkan
2  var http    = require("http");  // http server core module
3  var express = require("express");  // web framework external module
4  var io      = require("socket.io");  // web socket external module
5  var easyRTC = require("../");  // letak direktori modul eksternal
6
7
8  // mensetting dan konfigurasi untuk merujuk ke folder static pada saat
9  mengakses web.
10 var httpApp = express();
11 httpApp.use(express.static(__dirname + "/static/"));
12
13
14 // mensetting Express http server pada port 8181
15 var webServer = http.createServer(httpApp).listen(8181);
16
17 });
18
```

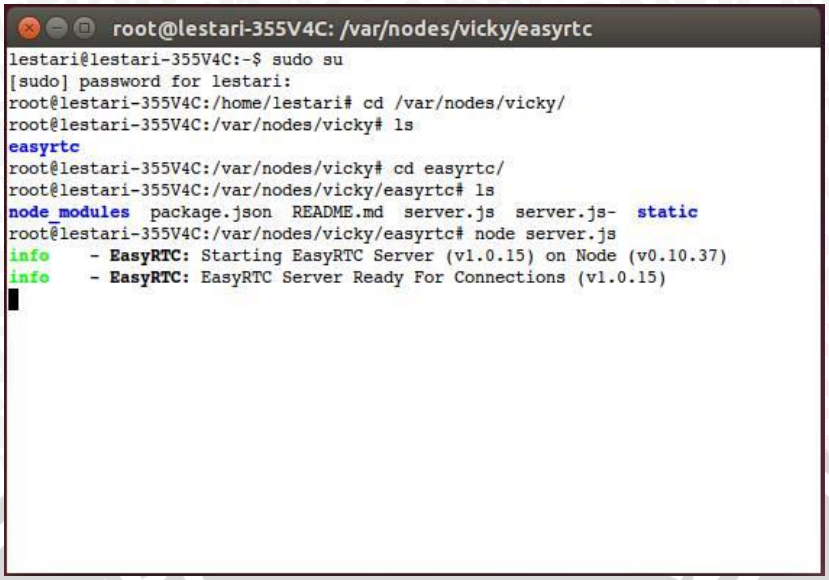
Source Code 6 Konfigurasi File server.js

Setelah semua selesai diseting dan dikonfigurasi, sekarang tahap untuk menghidupkan *signaling* dari server, untuk menjalankan *signaling* agar WebRTC tersebut dapat diakses oleh pengguna sebagai berikut

```
1  #cd /var/nodes/vicky/easyRTC
2  #node server.js
3
```

Source Code 7 Sintak Menjalankan node.js

Maka proses *signaling* akan berjalan dan server menunggu pengguna yang akan mengakses WebRTC *audio call* pada IP address server dengan port 8181, tampilan dari proses *signaling* node.js dapat dilihat pada gambar 5.1



Gambar 5.1 Proses *Signaling*

BAB 6 PENGUJIAN DAN ANALISA

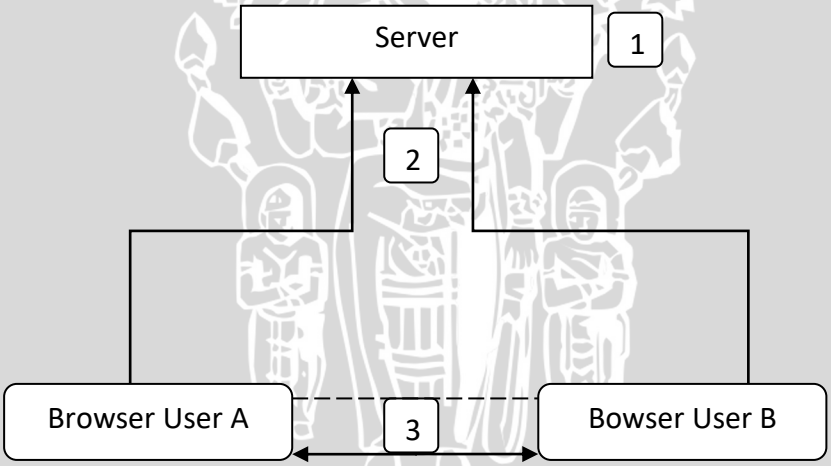
Pada bab pengujian dan analisa ini membahas tentang sistem yang telah implementasikan. Pada tahap ini dilakukan dua pengujian, yaitu menguji kompatibilitas sistem dan menguji *Quality of Service*.

6.1 Skenario Pengujian

Pengujian dilakukan berdasarkan metode pengujian yang telah dirancang sebelumnya. Adapun skenario untuk memastikan bahwa WebRTC *audio call* ini dapat berjalan pada *browser*. Diketahui bahwa IP *address* yang digunakan pada penelitian ini dapat dilihat pada tabel 6.1

IP *Address*	***Device***
192.168.43.215	Server
192.168.43.203	User A
192.168.43.65	User B

Tabel 6.1 IP *Address* Skenario Pengujian



Gambar 6.1 Skenario Pengujian

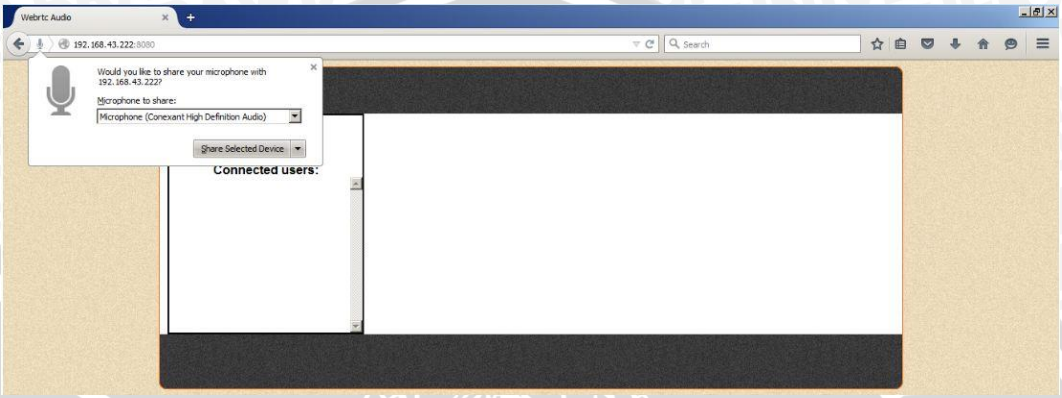
Berikut adalah penjelasan dari skenario pengujian gambar 6.1. Pertama, mengaktifkan proses *signaling* pada server dengan cara masuk pada direktori /var/nodes/vicky/easyRTC dan menuliskan code seperti berikut

```
1   #node server.js
```

Setelah melakukan langkah tersebut maka proses *signaling* akan berjalan seperti pada Gambar 5.1 Proses *Signaling*. Hal ini bertujuan agar pengguna *browser* dapat mengakses alamat WebRTC dengan *IP address* server dengan *port* 8181.

Kedua, mengakses WebRTC *audio call* pada *browser* A dan *browser* B dengan menuliskan *IP address* server dengan *port* 8181.

Ketiga mendaftarkan *browser* A dan *browser* B sehingga kedua *browser* tersebut mempunyai ID yang berbeda, ID tersebut di berikan oleh sistem. Pada tahap ini sistem akan meminta izin untuk mengakses media mikrofon. Tampilan sistem saat meminta izin mengakses media mikrofon pengguna dapat dilihat pada gambar 6.2.



Gambar 6.2 Akses Media Mikrofon

Setelah memberi izin untuk mengakses media mikrofon maka kedua *browser* A dan *browser* B mendapatkan ID yang berbeda, lalu pengguna dari *browser* A dapat melakukan komunikasi dengan menelepon *browser* B.

Pada skenario pengujian ini pengguna dari *browser* A malakukan komunikasi dengan *browser* B dengan empat waktu yang berbeda, empat waktu yang dimaksud adalah lama waktu dalam melakukan komunikasi, empat waktu tersebut adalah 2 menit, 4 menit, 6 menit dan 8 menit. Hal ini bertujuan untuk menganalisis *codec* opus dari *framework easyRTC* menggunakan metode *quality of service* dengan parameter rata-rata *delay, jitter, packets loss* dan *throughput*. Dalam penelitian ini standarisasi *quality of service* menggunakan versi TIPHON.

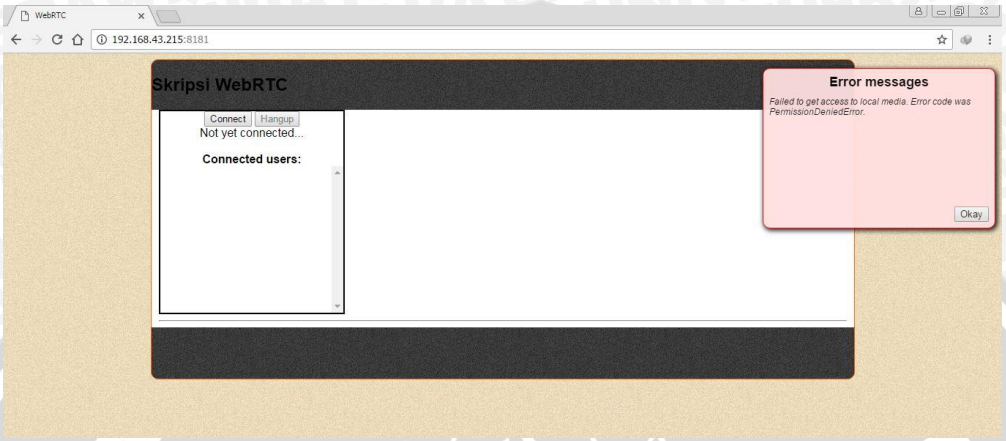
6.2 Pengujian Kompatibilitas Sistem

Pada pengujian kompatibilitas sistem menggunakan lima *browser* untuk pengujian kompatibilitas sistem. Kelima *browser* tersebut adalah chrome v52.0, Mozilla v38.0.5, opera v38.0, safari v5.1.7, dan internet explorer v8.0. Pengujian kompatibilitas ini bertujuan untuk mengetahui apakah *browser* tersebut *support* HTML5 dan apakah javascript dari *framework easyRTC* dapat berjalan pada kelima *browser* tersebut.

6.2.1 Browser Chrome v52.0

Pada pengujian kompatibilitas WebRTC *audio call* menggunakan *browser* chrome versi 52.0 dengan skenario pengujian yang telah dijelaskan pada gambar 6.1 diatas, bahwa *browser* chrome v52.0 dapat mengakses WebRTC *audio call* dengan websocket 192.168.43.215:8181. Namun pada

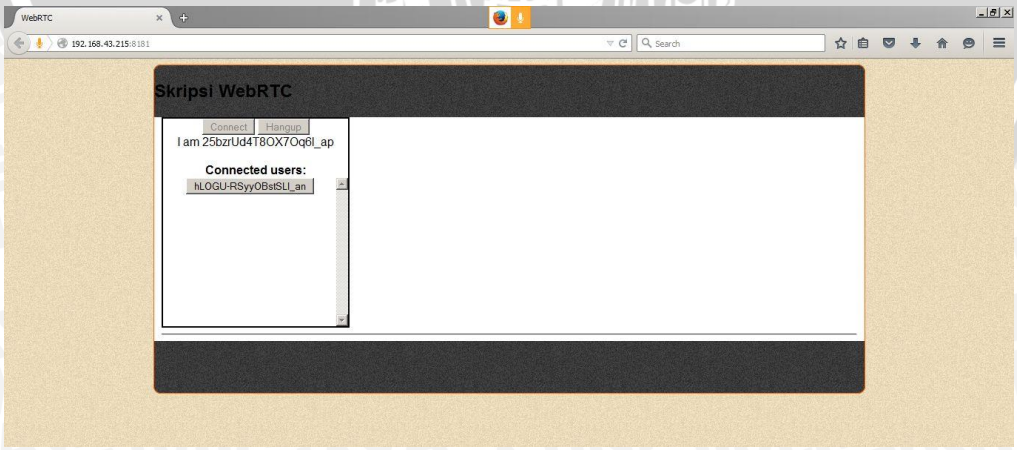
saat sistem meminta izin untuk mengakses media mikrofon, *browser* chrome memberikan notifikasi "*Failed to get access to local media. Error code was PermissionDeniedError*". Hal ini menunjukkan bahwa javascript dari *framework* easyRTC pada API getUserMedia() terjadi *error code*. Tampilan saat terjadi *error* pada saat mengakses media mikrofon dapat dilihat pada gambar 6.3.



Gambar 6.3 Pengujian Kompatibilitas Pada *Browser* Chrome v52.0

6.2.2 Browser Mozilla v38.0.5

Pada pengujian kompatibilitas WebRTC *audio call* menggunakan *browser* mozilla versi 38.0.5 dengan skenario pengujian yang telah dijelaskan pada gambar 6.1 diatas, bahwa *browser* mozilla v38.0.5 dapat mengakses WebRTC *audio call* dengan websocket 192.168.43.215:8181 dan pada saat sistem meminta izin untuk mengakses media mikrofon, *browser* mozilla dapat menampilkan notifikasi bahwa pengguna ingin menyetujui sistem untuk mengakses media mikrofon atau tidak. Tampilan dari proses meminta izin akses media mikrofon pada *browser* Mozilla dapat dilihat pada gambar 6.4.



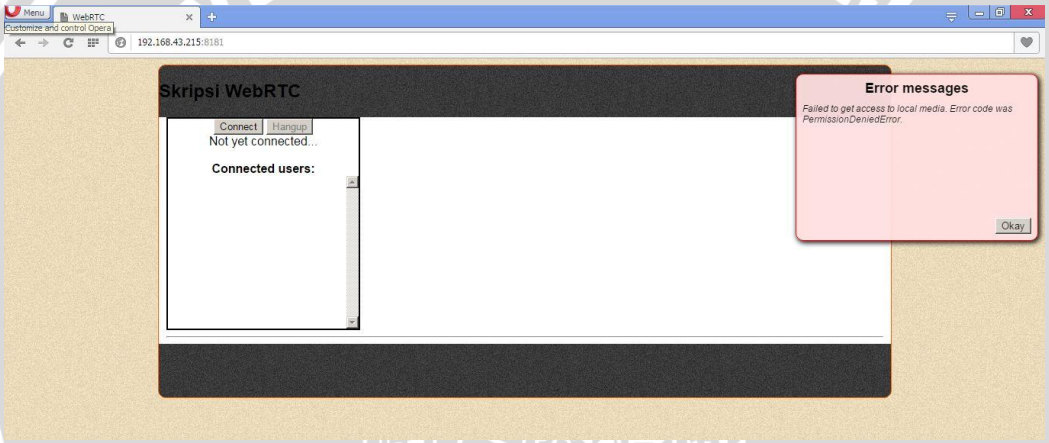
Gambar 6.4 Pengujian Kompatibilitas Pada Browser Mozilla v38.0.5

Hal ini menunjukkan bahwa HTML5 dan javascript dari *framework easyRTC* dapat berjalan sesuai skenario pengujian pada gambar 6.1. Saat

melakukan komunikasi juga berjalan dengan lancar dan tidak ada kendala dalam melakukan komunikasi antar *peers*, suara dari pengguna *browser* A juga dapat terdengar pada pengguna *browser* B dan sebaliknya.

6.2.3 Browser Opera v38.0

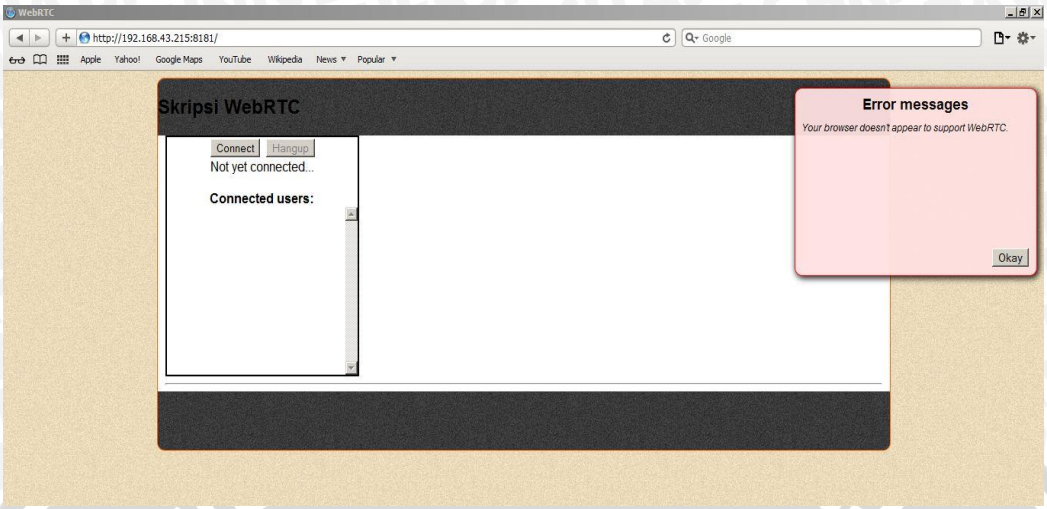
Pada pengujian kompatibilitas WebRTC *audio call* menggunakan *browser* opera versi 38.0 dengan skenario pengujian yang telah dijelaskan pada gambar 6.1 diatas, bahwa *browser* opera v38.0 dapat mengakses WebRTC *audio call* dengan websocket 192.168.43.215:8181. Namun pada saat sistem meminta izin untuk mengakses media mikrofon, *browser* opera memberikan notifikasi "*Failed to get access to local media. Error code was PermissionDeniedError*" sama seperti *error* pada *browser* chrome. Hal ini menunjukkan bahwa javascript dari *framework easyRTC* pada API getUserMedia() terjadi *error code*. Tampilan saat terjadi *error* pada saat mengakses media mikrofon dapat dilihat pada gambar 6.5.



Gambar 6.5 Pengujian Kompatibilitas Pada *Browser* Opera v38.0

6.2.4 Browser Safari v5.1.7

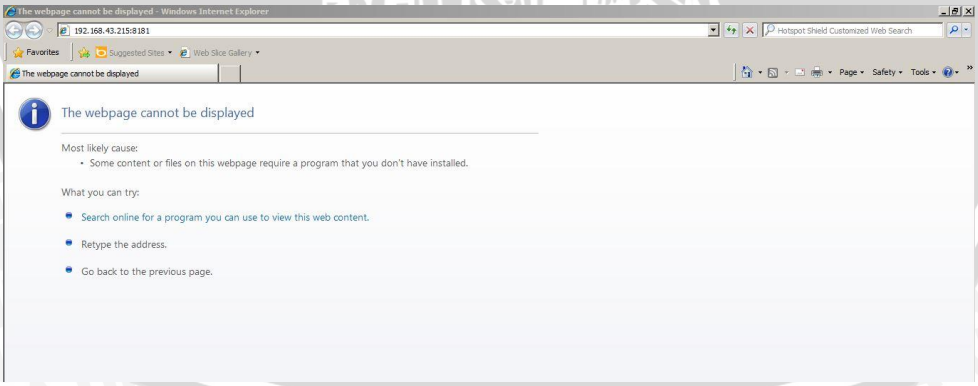
Pada pengujian kompatibilitas WebRTC *audio call* menggunakan *browser* safari versi 5.1.7 dengan skenario pengujian yang telah dijelaskan pada gambar 6.1 diatas, bahwa *browser* safari v5.1.7 dapat mengakses WebRTC *audio call* dengan websocket 192.168.43.215:8181. Namun pada saat sistem meminta izin untuk mengakses media mikrofon, *browser* opera memberikan notifikasi "*Your browser doesn't appear to support WebRTC*". Hal ini menunjukkan bahwa HTML 5 dan javascript dari *framework easyRTC* tidak *support* pada *browser* safari v5.1.7. Tampilan saat terjadi *error* pada saat mengakses media mikrofon dapat dilihat pada gambar 6.6.



Gambar 6.6 Pengujian Kompatibilitas Pada *Browser* Safari v5.1.7

6.2.5 Browser Internet Explorer v8.0

Pada pengujian kompatibilitas WebRTC *audio call* menggunakan *browser* internet explorer versi 8.0 dengan skenario pengujian yang telah dijelaskan pada gambar 6.1 diatas, bahwa *browser* internet explorer v8.0 saat mengakses websocket 192.168.43.215:8181 tidak dapat mengakses halaman WebRTC *audio call* yang terdapat pada server. Pada *browser* internet explorer memberikan notifikasi "*The webpage cannot be displayed*". Hal ini menunjukkan bahwa *browser* internet explorer tidak *support* HTML 5 dan javascript dari *framework easyRTC*. Notifikasi dari *browser* internet explorer dapat dilihat pada gambar 6.7.



Gambar 6.7 Pengujian Kompatibilitas Pada *Browser* Internet Explorer v8.0

6.3 Hasil Analisa Kompatibilitas Sistem

Pada pengujian kompatibilitas WebRTC *audio call* dengan *framework easyRTC* yang telah dilakukan pada kelima *browser* tersebut, dapat menganalisa bahwa *framework easyRTC* hanya dapat berjalan dengan lancar pada *browser* Mozilla, sedangkan pada *browser* lain mempunyai banyak kendala seperti

javascript API getUserMedia() pada *framework easyRTC* mengalami *error* saat dijalankan pada *browser* chrome dan opera, lalu pada *browser* safari tidak *support* dengan WebRTC dikarenkan HTML5 pada *browser* safari masih belum *support* WebRTC, namun masih dapat mengakses WebRTC *audio call* pada *browser*, hal ini menunjukkan bahwa safari masih belum mampu menjalankan *source code* HTML5 pada *framework easyRTC* namun masih dapat menjalankan *source code* javascript pada *framework easyRTC*, dan pada *browser* internet explorer tidak dapat mengakses WebRTC *audio call* dikarenakan HTML5 dan javascript pada *browser* tersebut tidak dapat menjalankan *source code* javascript dan HTML5 pada *framework easyRTC*. Berikut hasil pengujian kompatibilitas yang telah dilakukan dapat dilihat pada tabel 6.2 dibawah ini.

Browser	**HTML5 audio**	**Websocket**	**WebRTC**
Chrome v52.0	X	✓	X
Mozilla v38.0.5	✓	✓	✓
Opera v38.0	X	✓	X
Safari v5.1.7	X	✓	X
Internet Explorer v8.0	X	X	X

Tabel 6.2 Hasil Analisa Kompatibilitas

6.4 Pengujian *Quality of Service*

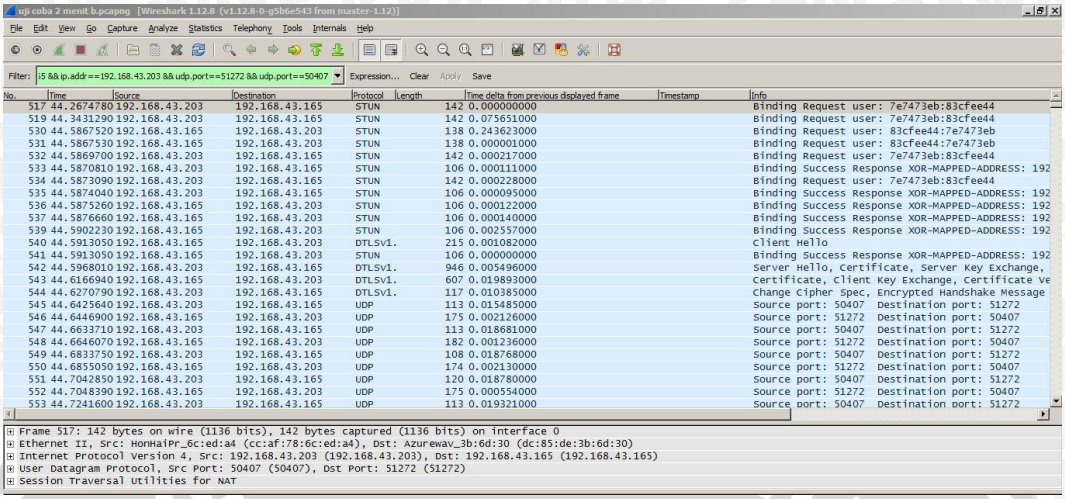
Pada pengujian *quality of service* ini menggunakan wireshark sebagai *capturing data* pada jaringan. Percobaan yang dilakukan dalam komunikasi WebRTC *audio call* menggunakan rentan waktu yang berbeda-beda, yaitu 2 menit, 4 menit, 6 menit dan 8 menit. Pada waktu 2 menit dilakukan tiga kali percobaan, percobaan tersebut berlaku pada menit ke 4 dan seterusnya, selanjutnya mencari rata-rata dari setiap menitnya. Hal tersebut dilakukan bertujuan untuk mengetahui apakah semakin banyaknya paket data yang diterima akan membuat nilai *jitter* semakin naik. Perlu dijelaskan bahwa dalam penelitian ini menggunakan *codec* opus sebagai *codec audio* dan menggunakan bandwidth 128 kbps.

Parameter *quality of service* yang digunakan adalah *delay*, *jitter*, *packets loss* dan *throughput*. Hasil dari perhitungan *quality of service* dari keempat parameter tersebut akan dibandingkan dengan standarisasi komunikasi yang baik versi TIPHON. Untuk detail dari perhitungan *quality of service* dapat dilihat pada file excel yang telah dikerjakan untuk menghitung *quality of service*.

6.4.1 Pengujian 2 menit

Pada pengujian 2 menit ini dilakukan tiga kali percobaan dengan waktu yang kurang lebih sama, yaitu sekitar 2 menit. Detail dari *capture* jaringan

saat melakukan komunikasi menggunakan wireshark dapat dilihat pada gambar 6.8.

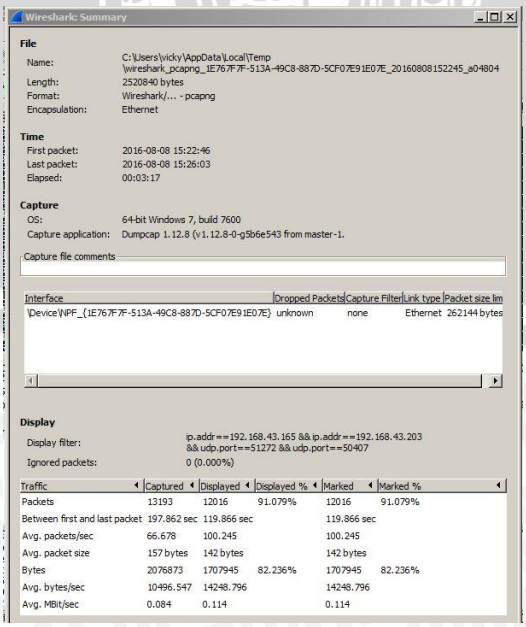


Gambar 6.8 Hasil Capture Data Selama 2 menit

Diperlukannya sintak untuk mem-*filter* paket data, sehingga wireshark akan menampilkan paket data yang penulis inginkan. Berikut sintak yang penulis tuliskan pada kolom filter pada wireshark.

```
ip.addr==192.168.43.65 && ip.addr==192.168.43.203 && udp.port==51272 &&
udp.port==50407
```

Dapat dilihat terdapat beberapa protokol yang berhasil di *capture* oleh wireshark, yaitu STUN, DTLSv1 dan UDP. Pada penelitian ini tidak membahas protokol STUN dan DTLSv1, karena fokus penelitian ini bukan pada protokol tersebut. *Framework easyRTC* menggunakan protokol UDP untuk pertukaran data *mediastream* antar *peers*. Untuk lebih detail tentang paket data yang didapat dalam *filter* tersebut dapat dilihat pada gambar 6.9 dibawah ini, nilai pada *summary* tersebut digunakan untuk menghitung *quality of service*.



Gambar 6.9 Summary Hasil *Filtering* Paket Data Pada Wireshark

Dapat dilihat pada gambar 6.9 bahwa terdapat total paket data yang dikirim maupun yang diterima, ada juga lama waktu antara paket data pertama dikirim sampai paket data terakhir dan seterusnya. Berikut adalah hasil perhitungan yang dilakukan dengan tiga kali percobaan selama waktu kurang lebih 2 menit.

- **Menghitung Rata-rata Delay 2 menit**

  Berikut adalah langkah-langah untuk menghitung rata-rata *delay*. Hal ini bertujuan untuk mengambil rata-rata *delay* dalam tiga kali percobaan, jika percobaan ini dilakukan hanya sekali maka data tersebut tidak *valid* dikarenakan jika terjadi kemungkinan *delay* akan naik atau turun pada saat percobaan 2 menit berikutnya.

Perhitungan Rata-rata *Delay*	
Parameter yang dihitung	Nilai
Total *Delay* (*second*)	120.77068
Total paket yang diterima	12093
Rata-rata *delay* (*second*)	0.009986825
Rata-rata *delay* (ms)	9.986825

Tabel 6.3 Hasil Perhitungan Rata-rata *Delay* Percobaan 1

Perhitungan Rata-rata *Delay*	
Parameter yang dihitung	Nilai
Total *Delay* (*second*)	122.967375
Total paket yang diterima	12315
Rata-rata *delay* (*second*)	0.009985171
Rata-rata *delay* (ms)	9.985171

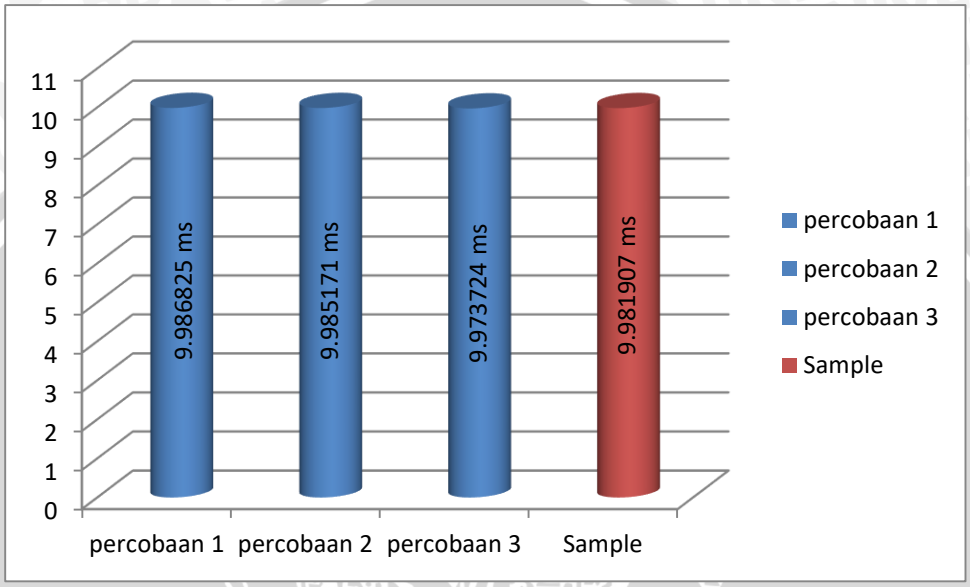
Tabel 6.4 Hasil Perhitungan Rata-rata *Delay* Percobaan 2

Perhitungan Rata-rata *Delay*	
Parameter yang dihitung	Nilai
Total *Delay* (*second*)	120.891511
Total paket yang diterima	12121
Rata-rata *delay* (*second*)	0.009973724
Rata-rata *delay* (ms)	9.973724

Tabel 6.5 Hasil Perhitungan Rata-rata *Delay* Percobaan 3

Total *delay* (*second*) pada tabel 6.3, 6.4 dan 6.5 didapatkan dengan menjumlah semua paket data yang dikirim pada tabel "*time delta from previous displayed frame*" pada wireshark. Total *delay* (*second*) tersebut menunjukkan selisih waktu antara paket dikirim sampai ke tujuan.

Sedangkan total paket data yang diterima didapatkan pada *summary* wireshark pada baris *packets* dan kolom *displayed*. Untuk menghitung rata-rata *delay* dengan satuan *second* menggunakan persamaan 2 *Delay*. Pada tabel 6.3 percobaan 1 menghasilkan nilai rata-rata *delay* 9.986825 ms, pada tabel 6.4 percobaan 2 menghasilkan rata-rata *delay* 9.985171 ms, pada tabel 6.5 percobaan 3 menghasilkan rata-rata *delay* 9.973724 ms. Adapun grafik dari hasil tiga percobaan pada tabel 6.3, 6.4 dan 6.5 diatas, dengan rata-rata *dela*y selama 2 menit.



Gambar 6.10 Grafik Rata-rata *Delay* Dari Tiga Kali Percobaan 2 menit

Dapat dilihat pada gambar 6.10 bahwa untuk rata-rata *delay* selama 2 menit mendapatkan nilai antara 9.973724 ms sampai 9.986825 ms. Sehingga didapatkan nilai rata-rata dari ketiga hasil percobaan tersebut, yang nantinya akan dihitung untuk perhitungan selanjutnya. Rata-rata *delay* dari ketiga percobaan tersebut adalah 9.981907 ms.

- **Menghitung Nilai *Jitter* 2 menit**

Untuk menghitung nilai *jitter* juga melakukan tiga kali percobaan selama 2 menit. Berikut nilai *jitter* yang didapatkan dari tiga kali percobaan.

Perhitungan *Jitter*	
Parameter yang dihitung	Nilai
Total Variasi *Delay* (*second*)	0.005408
Total paket yang diterima -1	12092
Jitter (*second*)	4.47E-07
Jitter (ms)	0.00044724

Tabel 6.6 Hasil Perhitungan Nilai *Jitter* Percobaan 1

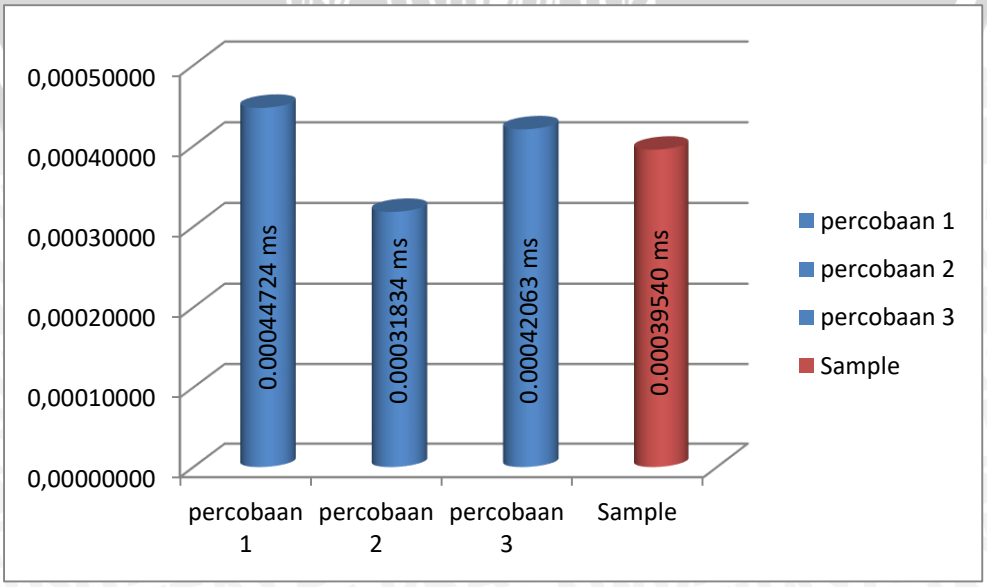
Perhitungan *Jitter*	
Parameter yang dihitung	Nilai
Total Variasi *Delay* (*second*)	0.00392
Total paket yang diterima -1	12314
Jitter (*second*)	3.18E-07
Jitter (ms)	0.00031834

Tabel 6.7 Hasil Perhitungan Nilai *Jitter* Percobaan 2

Perhitungan *Jitter*	
Parameter yang dihitung	Nilai
Total Variasi *Delay* (*second*)	0.005098
Total paket yang diterima -1	12120
Jitter (*second*)	4.21E-07
Jitter (ms)	0.00042063

Tabel 6.8 Hasil Perhitungan Nilai *Jitter* Percobaan 3

Total variasi *delay* (*second*) pada tabel 6.6, 6.7 dan 6.8 didapatkan dengan menggunakan persamaan 4 Variasi *Delay* dengan nilai yang terdapat pada kolom "*time delta from previous displayed frame*" pada wireshark. Sedangkan total paket data yang diterima pada tabel 6.6, 6.7 dan 6.8 didapatkan pada summary wireshark pada baris *packets* dan kolom *displayed*. Untuk mencari nilai *jitter* menggunakan persamaan 3 *Jitter*. Pada tabel 6.6 percobaan 1 menghasilkan nilai *jitter* 0.00044724 ms, pada tabel 6.7 percobaan 2 menghasilkan *jitter* 0.00031834 ms, pada tabel 6.8 percobaan 3 menghasilkan *jitter* 0.00042063 ms. Adapun grafik dari hasil tiga percobaan pada tabel 6.6, 6.7 dan 6.8 diatas, dengan nilai *jitter* selama 2 menit.



Gambar 6.11 Grafik Nilai *Jitter* Dari Tiga Kali Percobaan 2 menit

Dapat dilihat pada gambar 6.11 bahwa untuk nilai *jitter* selama 2 menit mendapatkan nilai antara 0.00031834 ms sampai 0.00044724 ms. Sehingga didapatkan nilai rata-rata dari ketiga hasil percobaan tersebut, yang nantinya akan dihitung untuk perhitungan selanjutnya. Rata-rata *jitter* dari ketiga percobaan tersebut adalah 0.00039540 ms.

- **Menghitung Nilai *Throughput* 2 menit**

Dari hasil tiga kali percobaan dalam waktu kurang lebih 2 menit yang dilakukan mendapatkan nilai *throughput* yang berbeda-beda. Hasil dari nilai *throughput* pada percobaan 2 menit dapat dilihat pada tabel 6.9, 6.10 dan 6.11.

Perhitungan *Throughput*	
Parameter yang dihitung	Nilai
Paket data yang diterima (bytes)	1505539
Lama pengamatan (*second*)	120.771
Throughput (bps)	12466.06387
Throughput (kbps)	12.46606387

Tabel 6.9 Hasil Perhitungan Nilai *Throughput* Percobaan 1

Perhitungan *Throughput*	
Parameter yang dihitung	Nilai
Paket data yang diterima (bytes)	1704774
Lama pengamatan (*second*)	122.967
Throughput (bps)	13863.67074
Throughput (kbps)	13.86367074

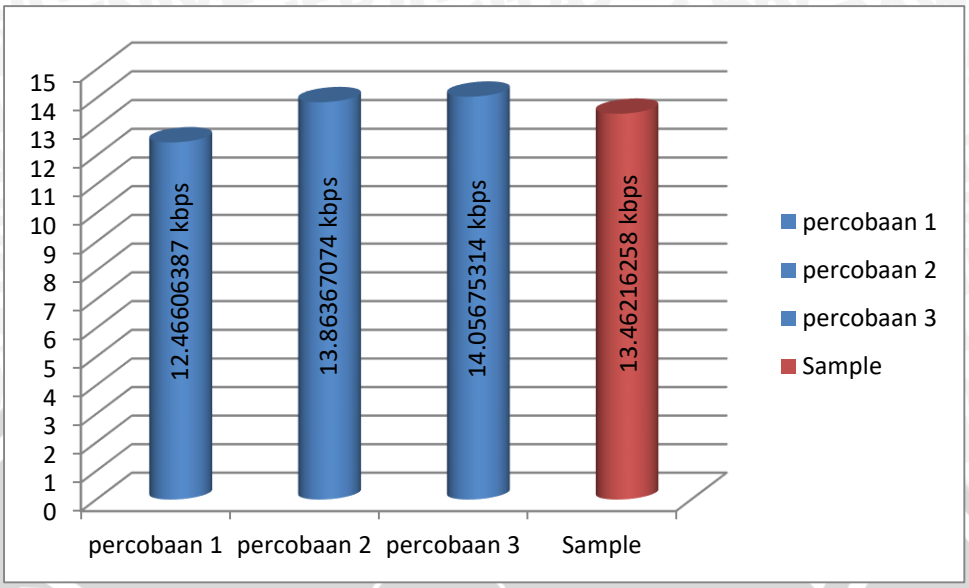
Tabel 6.10 Hasil Perhitungan Nilai *Throughput* Percobaan 2

Perhitungan *Throughput*	
Parameter yang dihitung	Nilai
Paket data yang diterima (bytes)	1699349
Lama pengamatan (*second*)	120.892
Throughput (bps)	14056.75314
Throughput (kbps)	14.05675314

Tabel 6.11 Hasil Perhitungan Nilai *Throughput* Percobaan 3

Nilai *throughput* (bps) pada tabel 6.9, 6.10 dan 6.11 didapatkan dengan menggunakan persamaan 5. Pada tabel 6.9 percobaan 1 menghasilkan nilai *throughput* 12.46606387 kbps, pada tabel 6.10 percobaan 2 menghasilkan *throughput* 13.86367074 kbps, pada tabel 6.11 percobaan 3 menghasilkan *throughput* 14.05675314 kbps. Adapun

grafik dari hasil tiga percobaan diatas, dengan nilai *throughput* selama 2 menit.



Gambar 6.12 Grafik Nilai *Throughput* Dari Tiga Kali Percobaan 2 menit

Dapat dilihat pada gambar 6.12 bahwa untuk nilai *throughput* selama 2 menit mendapatkan nilai antara 12.46606387 kbps sampai 14.05675314 kbps. Sehingga nilai rata-rata dari ketiga hasil percobaan tersebut adalah 13.46216258 kbps, yang nantinya akan dihitung untuk perhitungan selanjutnya

- **Menghitung Nilai *Packets Loss* 2 menit**

Untuk menghitung nilai *packets loss* juga melakukan tiga kali percobaan selama 2 menit.

Perhitungan *Packet loss*	
Parameter yang dihitung	Nilai
Paket data yang dikirim	12941
Paket data yang diterima	12093
Packet loss (%)	6.552816629

Tabel 6.12 Hasil Perhitungan *Packets Loss* Percobaan 1

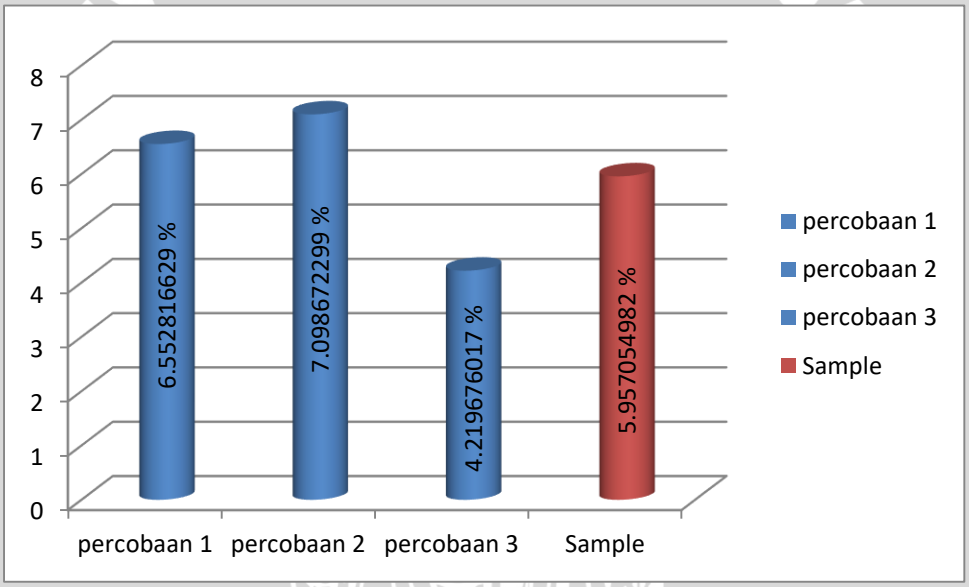
Perhitungan *Packet loss*	
Parameter yang dihitung	Nilai
Paket data yang dikirim	13256
Paket data yang diterima	12315
Packet loss (%)	7.098672299

Tabel 6.13 Hasil Perhitungan *Packets Loss* Percobaan 2

Perhitungan *Packet loss*	
Parameter yang dihitung	Nilai
Paket data yang dikirim	12655
Paket data yang diterima	12121
Packet loss (%)	4.219676017

Tabel 6.14 Hasil Perhitungan *Packets Loss* Percobaan 3

Untuk mendapatkan nilai *packets loss* (%) menggunakan persamaan 6. Pada tabel 6.12 percobaan 1 menghasilkan nilai *packets loss* 6.552816629 %, pada tabel 6.13 percobaan 2 menghasilkan *packets loss* 7.098672299 %, pada tabel 6.14 percobaan 3 menghasilkan *packets loss* 4.219676017 %. Adapun grafik dari hasil tiga percobaan diatas, dengan nilai *packets loss* selama 2 menit.



Gambar 6.13 Grafik Nilai *Packets Loss* Dari Tiga Kali Percobaan 2 menit

Dapat dilihat pada gambar 6.13 bahwa untuk nilai *packets loss* selama 2 menit mendapatkan nilai antara 4.219676017 % sampai 7.098672299 %. Sehingga didapatkan nilai rata-rata dari ketiga hasil percobaan tersebut, yang nantinya akan dihitung untuk perhitungan selanjutnya. Rata-rata dari ketiga percobaan tersebut adalah 5.957054982 %.

- ***Sample* Uji Coba Selama 2 menit**

Berikut adalah *sample* yang didapatkan dari hasil uji coba selama 2 menit. *Sample* ini akan digunakan untuk menghitung *quality of service* dalam rentan waktu 2 s/d 8 menit.

Sample uji coba selama 2 menit	
Kategori	Nilai
Rata-rata Delay (ms)	9.981906667
Jitter (ms)	0.00039540
Throughput (kbps)	13.46216258
Packets Loss (%)	5.957054982

Tabel 6.15 *Sample* Nilai Parameter uji QoS Selama 2 menit

Diketahui pada tabel 6.15 bahwa dalam uji coba selama 2 menit menghasilkan sample nilai rata-rata *delay* 9.981906667 ms, nilai *jitter* 0.00039540 ms, nilai *throughput* 13.46216258 kbps dan nilai *packets loss* 5.957054982 %.

6.4.2 Pengujian 4 menit

Pada pengujian 4 menit ini langkah-langkah yang dilakukan sama dengan pada pengujian 2 menit diatas, yaitu sebanyak 3 kali pengujian dan menggunakan persamaan yang sama untuk menghitung nilai rata-rata *delay*, *jitter*, *throughput* dan *packets loss*.

- **Menghitung Rata-rata *Delay* 4 menit**

Berikut adalah detail dari tabel menghitung rata-rata *delay* 4 dari tiga kali percobaan.

Perhitungan Rata-rata *Delay*	
Parameter yang dihitung	Nilai
Total *Delay* (*second*)	240.596003
Total paket yang diterima	24129
Rata-rata *delay* (*second*)	0.009971238
Rata-rata *delay* (ms)	9.971238

Tabel 6.16 Hasil Perhitungan Rata-rata *Delay* Percobaan 1

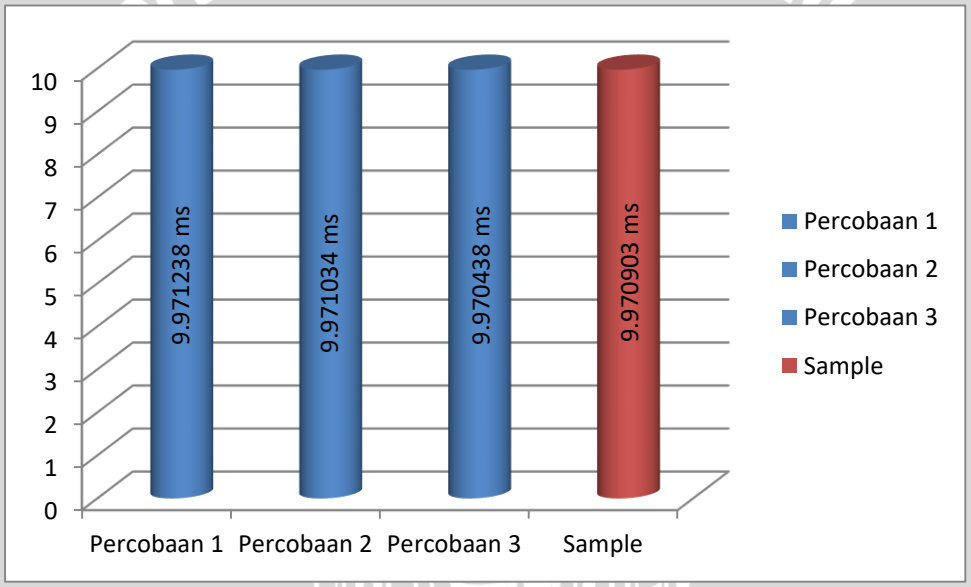
Perhitungan Rata-rata *Delay*	
Parameter yang dihitung	Nilai
Total *Delay* (*second*)	240.800462
Total paket yang diterima	24150
Rata-rata *delay* (*second*)	0.009971034
Rata-rata *delay* (ms)	9.971034

Tabel 6.17 Hasil Perhitungan Rata-rata *Delay* Percobaan 2

Perhitungan Rata-rata *Delay*	
Parameter yang dihitung	Nilai
Total *Delay* (*second*)	241.523887
Total paket yang diterima	24224
Rata-rata *delay* (*second*)	0.009970438
Rata-rata *delay* (ms)	9.970438

Tabel 6.18 Hasil Perhitungan Rata-rata *Delay* Percobaan 3

Dapat dilihat pada tabel 6.16, 6.17 dan 6.18 bahwa hasil perhitungan rata-rata *delay* pada tabel 6.16 percobaan 1 menghasilkan nilai 9.971238 ms, nilai rata-rata *delay* pada tabel 6.17 percobaan 2 adalah 9.971034 ms dan nilai rata-rata *delay* pada 6.18 percobaan 3 adalah 9.970438 ms. Adapun grafik dari hasil tiga percobaan diatas, dengan nilai rata-rata *delay* selama 4 menit.



Gambar 6.14 Grafik Rata-rata *Delay* Dari Tiga Kali Percobaan 4 menit

Dapat dilihat pada grafik 6.14 bahwa untuk nilai rata-rata *delay* selama 4 menit mendapatkan nilai antara 9.970438 ms sampai 9.971238 ms. Sehingga didapatkan nilai rata-rata dari ketiga hasil percobaan tersebut, yang nantinya akan dihitung untuk perhitungan selanjutnya. Rata-rata dari ketiga percobaan tersebut adalah 9.970903 ms.

- **Menghitung Nilai *Jitter* 4 menit**

Berikut adalah detail dari tabel menghitung nilai *jitter* selama 4 menit dari tiga kali percobaan.

Perhitungan *Jitter*	
Parameter yang dihitung	Nilai
Total Variasi *Delay* (*second*)	0.007469
Total paket yang diterima -1	24128
Jitter (*second*)	3.10E-07
Jitter (ms)	0.00030956

Tabel 6.19 Hasil Perhitungan Nilai *Jitter* Percobaan 1

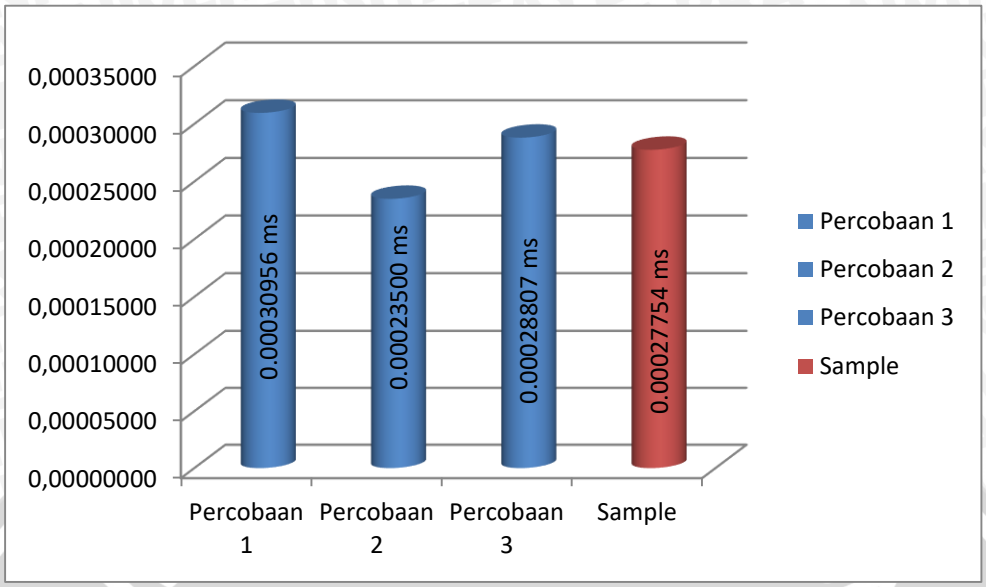
Perhitungan *Jitter*	
Parameter yang dihitung	Nilai
Total Variasi *Delay* (*second*)	5.68E-03
Total paket yang diterima -1	24149
Jitter (*second*)	2.35E-07
Jitter (ms)	0.00023500

Tabel 6.20 Hasil Perhitungan Nilai *Jitter* Percobaan 2

Perhitungan *Jitter*	
Parameter yang dihitung	Nilai
Total Variasi *Delay* (*second*)	6.98E-03
Total paket yang diterima -1	24223
Jitter (*second*)	2.88E-07
Jitter (ms)	0.00028807

Tabel 6.21 Hasil Perhitungan Nilai *Jitter* Percobaan 3

Dapat dilihat bahwa pada tabel 6.19, 6.20 dan 6.21 bahwa hasil perhitungan nilai *jitter* pada tabel 6.19 percobaan 1 menghasilkan nilai 0.00030956 ms, nilai *jitter* pada tabel 6.19 percobaan 1 menghasilkan nilai 0.00023500 ms dan nilai *jitter* pada pada tabel 6.19 percobaan 1 menghasilkan nilai 0.00028807 ms. Adapun grafik dari hasil tiga percobaan diatas, dengan nilai *jitter* selama 4 menit.



Gambar 6.15 Garfik Nilai *Jitter* Dari Tiga Kali Percobaan 4 menit

Dapat dilihat pada gambar 6.15 bahwa untuk nilai rata-rata *jitter* selama 4 menit mendapatkan nilai antara 0.00023500 ms sampai 0.00030956 ms. Sehingga didapatkan nilai rata-rata dari ketiga hasil percobaan tersebut, yang nantinya akan dihitung untuk perhitungan selanjutnya. Rata-rata dari ketiga percobaan tersebut adalah 0.00027754 ms.

- **Menghitung Nilai *Throughput* 4 menit**

Berikut adalah detail dari tabel menghitung nilai *throughput* selama 4 menit dari tiga kali percobaan.

Perhitungan *Throughput*	
Parameter yang dihitung	Nilai
Paket data yang diterima (bytes)	3449220
Lama pengamatan (*second*)	243.381
Throughput (bps)	14172.10053
Throughput (kbps)	14.17210053

Tabel 6.22 Hasil Perhitungan Nilai *Throughput* Percobaan 1

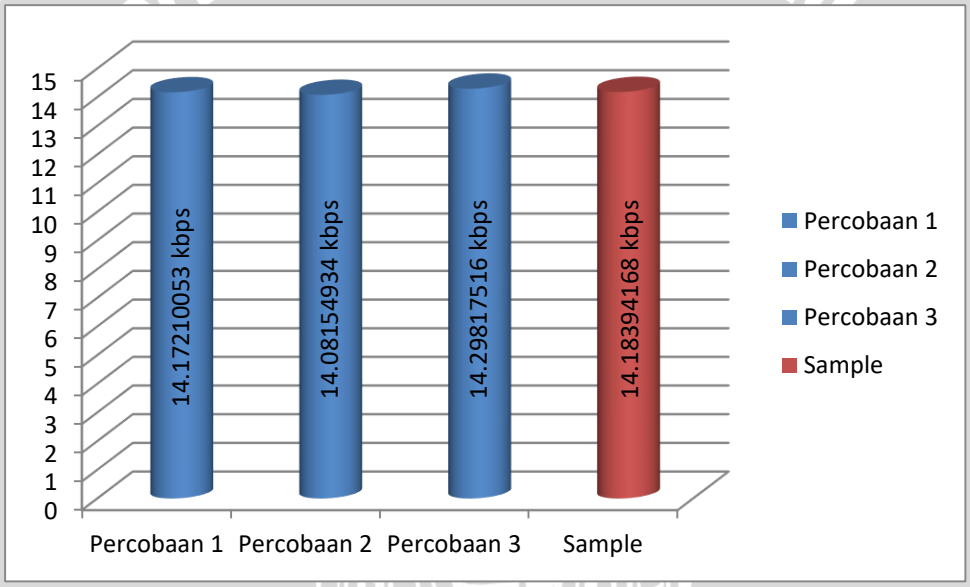
Perhitungan *Throughput*	
Parameter yang dihitung	Nilai
Paket data yang diterima (bytes)	3487521
Lama pengamatan (*second*)	247.666
Throughput (bps)	14081.54934
Throughput (kbps)	14.08154934

Tabel 6.23 Hasil Perhitungan Nilai *Throughput* Percobaan 2

Perhitungan *Throughput*	
Parameter yang dihitung	Nilai
Paket data yang diterima (bytes)	3449878
Lama pengamatan (*second*)	241.281
Throughput (bps)	14298.17516
Throughput (kbps)	14.29817516

Tabel 6.24 Hasil Perhitungan Nilai *Throughput* Percobaan 3

Dapat dilihat bahwa pada tabel 6.22, 6.23 dan 6.24 bahwa hasil perhitungan nilai *throughput* pada tabel 6.22 percobaan 1 menghasilkan nilai 14.17210053 kbps, nilai *throughput* pada tabel 6.23 percobaan 2 adalah 14.08154934 kbps dan nilai *throughput* pada tabel 6.24 percobaan 3 adalah 14.29817516 kbps. Adapun grafik dari hasil tiga percobaan diatas, dengan nilai *throughput* selama 4 menit.



Gambar 6.16 Grafik Nilai *Throughput* Dari Tiga Kali Percobaan 4 menit

Dapat dilihat pada gambar 6.16 bahwa untuk nilai rata-rata *throughput* selama 4 menit mendapatkan nilai antara 14.08154934 sampai 14.29817516 kbps. Sehingga didapatkan nilai rata-rata dari ketiga hasil percobaan tersebut, yang nantinya akan dihitung untuk perhitungan selanjutnya. Rata-rata dari ketiga percobaan tersebut adalah 14.18394168 kbps.

- **Menghitung Nilai *Packets Loss* 4 menit**

Berikut adalah detail dari tabel menghitung nilai *Packets loss* selama 4 menit dari tiga kali percobaan.

Perhitungan *Packet loss*	
Parameter yang dihitung	Nilai
Paket data yang dikirim	25815
Paket data yang diterima	24407
Packet loss (%)	5.454193298

Tabel 6.25 Hasil Perhitungan *Packets Loss* Perocbaan 1

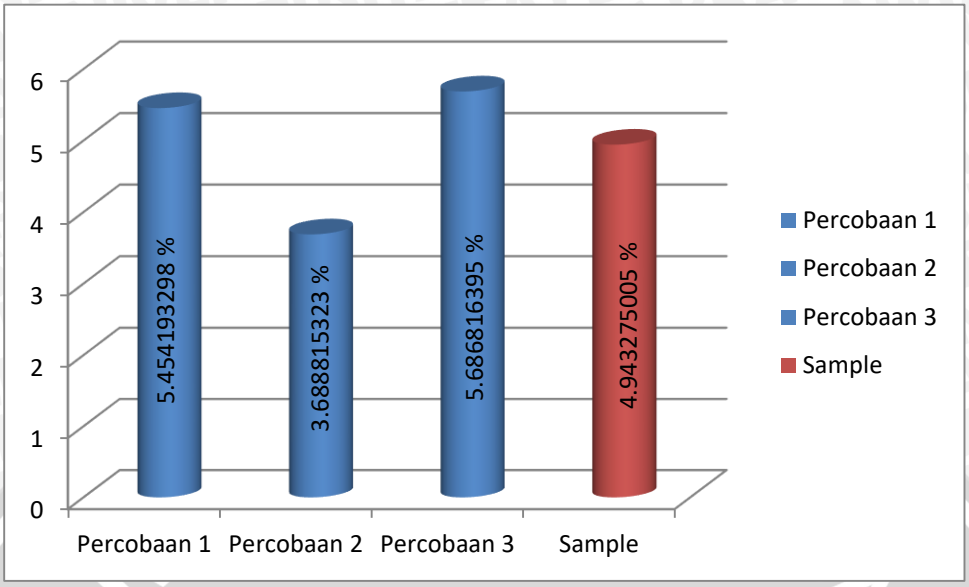
Perhitungan *Packet loss*	
Parameter yang dihitung	Nilai
Paket data yang dikirim	25374
Paket data yang diterima	24438
Packet loss (%)	3.688815323

Tabel 6.26 Hasil Perhitungan *Packets Loss* Percobaan 2

Perhitungan *Packet loss*	
Parameter yang dihitung	Nilai
Paket data yang dikirim	25691
Paket data yang diterima	24230
Packet loss (%)	5.686816395

Tabel 6.27 Hasil Perhitungan *Packets Loss* Percobaan 3

Dapat dilihat bahwa pada tabel 6.25, 6.26 dan 6.27 bahwa hasil perhitungan nilai *packets loss* pada tabel 6.25 percobaan 1 menghasilkan nilai 5.454193298 %, nilai *packets loss* pada tabel 6.26 percobaan 2 adalah 3.688815323 % dan nilai *packets loss* pada tabel 6.27 percobaan 3 adalah 5.686816395 %. Adapun grafik dari hasil tiga percobaan diatas, dengan nilai *packets loss* selama 4 menit.



Gambar 6.17 Grafik Nilai *Packets Loss* Dari Tiga Kali Percobaan 4 menit

Dapat dilihat pada gambar 6.17 bahwa untuk nilai rata-rata *packet loss* selama 4 menit mendapatkan nilai antara 3.68881532 % sampai 5.68681639 %. Sehingga didapatkan nilai rata-rata dari ketiga hasil percobaan tersebut, yang nantinya akan dihitung untuk perhitungan selanjutnya. Rata-rata dari ketiga percobaan tersebut adalah 4.943275005 %.

- ***Sample* Uji Coba Selama 4 menit**

Berikut adalah *sample* yang didapatkan dari hasil uji coba selama 4 menit. *Sample* ini akan digunakan untuk menghitung *quality of servic*e dalam rentan waktu 2 s/d 8 menit.

Sample uji coba selama 4 menit	
Kategori	Nilai
Rata-rata *Delay* (ms)	9.970903333
Jitter (ms)	0.000277543
Throughput (kbps)	14.18394168
Packets Loss (%)	4.943275005

Tabel 6.28 Sample Nilai Parameter uji QoS Selama 4 menit

Diketahui pada tabel 6.28 bahwa dalam uji coba selama 4 menit menghasilkan *sample* nilai rata-rata *delay* 9.970903333 ms, nilai *jitter* 0.000277543 ms, nilai *throughput* 14.18394168 kbps dan nilai *packets loss* 4.943275005 %.

6.4.3 Pengujian 6 menit

Pada pengujian 6 menit ini langkah-langkah yang dilakukan sama dengan pada pengujian 2 menit dan 4 menit sebelumnya, yaitu sebanyak 3 kali pengujian dan menggunakan persamaan yang sama.

- **Menghitung Rata-rata *Delay* 6 menit**

Berikut adalah detail dari tabel menghitung rata-rata *delay* 6 menit dari tiga kali percobaan.

Perhitungan Rata-rata *Delay*	
Parameter yang dihitung	Nilai
Total *Delay* (*second*)	360.874841
Total paket yang diterima	36173
Rata-rata *delay* (*second*)	0.009976359
Rata-rata *delay* (ms)	9.976359

Tabel 6.29 Hasil Perhitungan Rata-rata *Delay* Percobaan 1

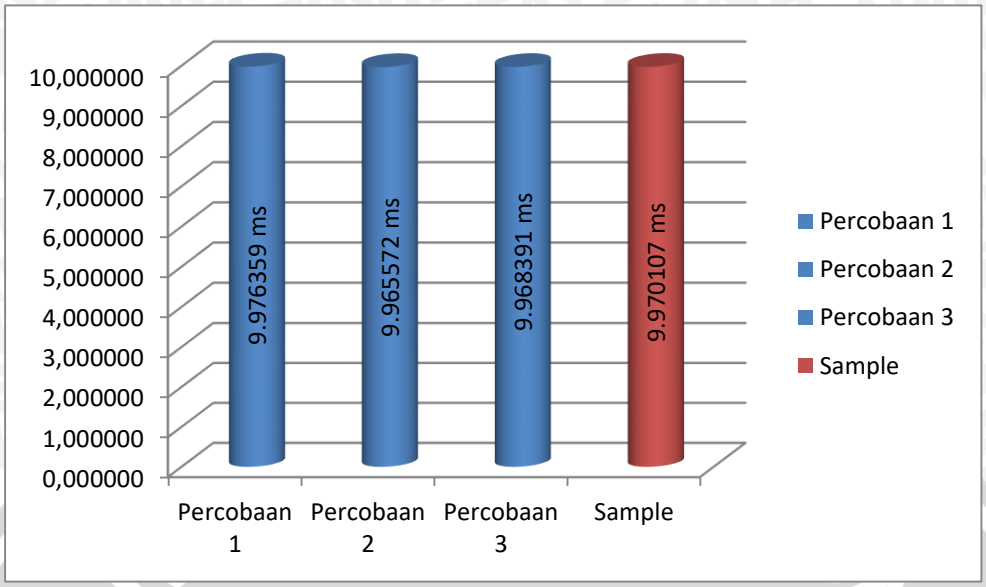
Perhitungan Rata-rata *Delay*	
Parameter yang dihitung	Nilai
Total *Delay* (*second*)	360.285342
Total paket yang diterima	36153
Rata-rata *delay* (*second*)	0.009965572
Rata-rata *delay* (ms)	9.965572

Tabel 6.30 Hasil Perhitungan Rata-rata *Delay* Percobaan 2

Perhitungan Rata-rata *Delay*	
Parameter yang dihitung	Nilai
Total *Delay* (*second*)	360.307484
Total paket yang diterima	36145
Rata-rata *delay* (*second*)	0.009968391
Rata-rata *delay* (ms)	9.968391

Tabel 6.31 Hasil Perhitungan Rata-rata *Delay* Percobaan 3

Dapat dilihat pada tabel 6.29, 6.30 dan 6.31 bahwa hasil perhitungan rata-rata *delay* pada tabel 6.29 percobaan 1 menghasilkan nilai 9.976359 ms, nilai rata-rata *delay* pada tabel 6.30 percobaan 2 adalah 9.965572 ms dan nilai rata-rata *delay* pada tabel 6.31 percobaan 3 adalah 9.968391 ms. Adapun grafik dari hasil tiga percobaan diatas, dengan nilai rata-rata *delay* selama 6 menit.



Gambar 6.18 Grafik Rata-rata *Delay* Dari Tiga Kali Percobaan 6 menit

Dapat dilihat pada gambar 6.18 bahwa untuk nilai rata-rata *delay* selama 6 menit mendapatkan nilai antara 9.965572 ms sampai 9.976359 ms. Sehingga didapatkan nilai rata-rata dari ketiga hasil percobaan tersebut, yang nantinya akan dihitung untuk perhitungan selanjutnya. Rata-rata dari ketiga percobaan tersebut adalah 9.970107 ms.

- **Menghitung Nilai *Jitter* 6 menit**

Berikut adalah detail dari tabel menghitung nilai *jitter* selama 6 menit dari tiga percobaan.

Perhitungan *Jitter*	
Parameter yang dihitung	Nilai
Total Variasi *Delay* (*second*)	0.010702
Total paket yang diterima -1	36172
Jitter (*second*)	2.96E-07
Jitter (ms)	0.000295864

Tabel 6.32 Hasil Perhitungan Nilai *Jitter* Percobaan 1

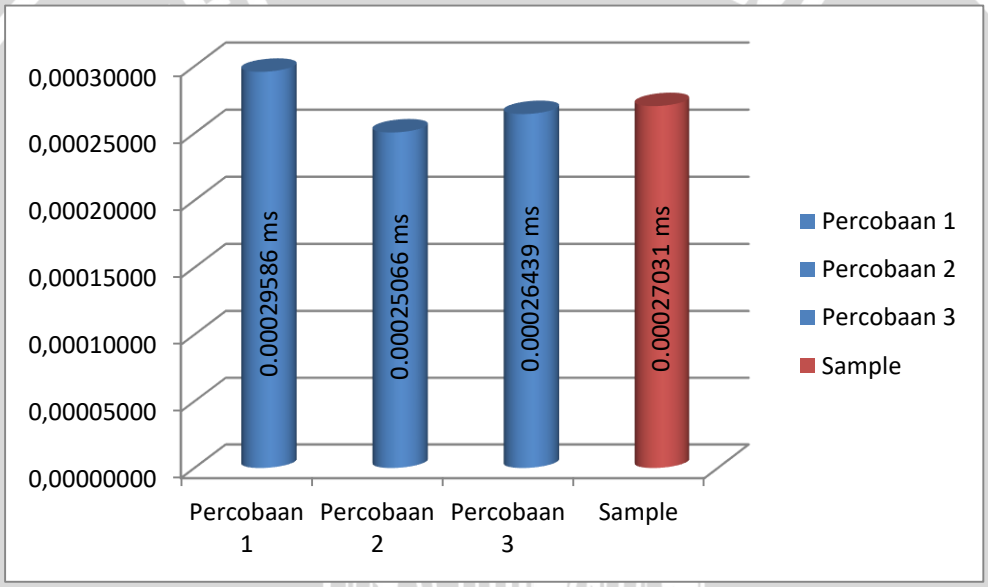
Perhitungan *Jitter*	
Parameter yang dihitung	Nilai
Total Variasi *Delay* (*second*)	0.009062
Total paket yang diterima -1	36152
Jitter (*second*)	2.51E-07
Jitter (ms)	0.000250664

Tabel 6.33 Hasil Perhitungan Nilai *Jitter* Percobaan 2

Perhitungan *Jitter*	
Parameter yang dihitung	Nilai
Total Variasi *Delay* (*second*)	0.009556
Total paket yang diterima -1	36144
Jitter (*second*)	2.64E-07
Jitter (ms)	0.000264387

Tabel 6.34 Hasil Perhitungan Nilai *Jitter* Percobaan 3

Dapat dilihat pada tabel 6.32, 6.33 dan 6.34 bahwa hasil perhitungan nilai *jitter* pada tabel 6.32 percobaan 1 menghasilkan nilai 0.000295864 ms, nilai *jitter* pada tabel 6.33 percobaan 2 adalah 0.000250664 ms dan nilai *jitter* pada tabel 6.34 percobaan 3 adalah 0.000264387 ms. Adapun grafik dari hasil tiga percobaan diatas, dengan nilai *jitter* selama 6 menit.



Gambar 6.19 Grafik Nilai *Jitter* Dari Tiga Kali Percobaan 6 menit

Dapat dilihat pada gambar 6.19 bahwa untuk nilai *jitter* selama 6 menit mendapatkan nilai antara 0.00025066 ms sampai 0.00029586 ms. Sehingga didapatkan nilai rata-rata dari ketiga hasil percobaan tersebut, yang nantinya akan dihitung untuk perhitungan selanjutnya. Rata-rata dari ketiga percobaan tersebut adalah 0.00027031 ms.

- **Menghitung Nilai *Throughput* 6 menit**

Berikut adalah detail dari tabel menghitung nilai *throughput* selama 6 menit dari tiga kali percobaan.

Perhitungan *Throughput*	
Parameter yang dihitung	Nilai
Paket data yang diterima (bytes)	5207509
Lama pengamatan (*second*)	367.009
Throughput (bps)	14189.04986
Throughput (kbps)	14.18904986

Tabel 6.35 Hasil Perhitungan Nilai *Throughput* Percobaan 1

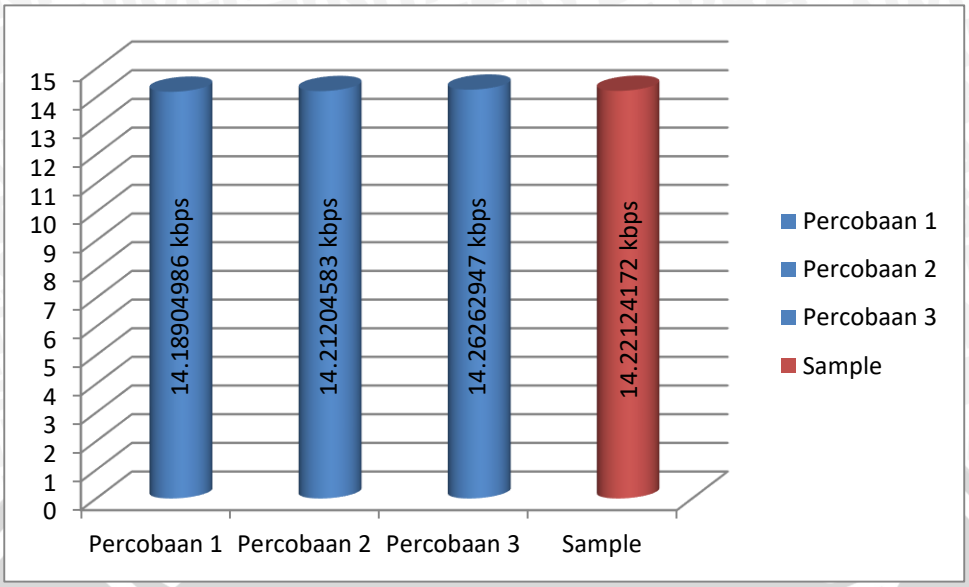
Perhitungan *Throughput*	
Parameter yang dihitung	Nilai
Paket data yang diterima (bytes)	5170726
Lama pengamatan (*second*)	363.827
Throughput (bps)	14212.04583
Throughput (kbps)	14.21204583

Tabel 6.36 Hasil Perhitungan Nilai *Throughput* Percobaan 2

Perhitungan *Throughput*	
Parameter yang dihitung	Nilai
Paket data yang diterima (bytes)	5162373
Lama pengamatan (*second*)	361.951
Throughput (bps)	14262.62947
Throughput (kbps)	14.26262947

Tabel 6.37 Hasil Perhitungan Nilai *Throughput* Percobaan 3

Dapat dilihat pada tabel 6.35, 6.36 dan 6.37 bahwa hasil perhitungan nilai *throughput* pada tabel 6.35 percobaan 1 menghasilkan nilai 14.18904986 kbps, nilai *throughput* pada tabel 6.36 percobaan 2 adalah 14.21204583 kbps dan nilai *throughput* pada tabel 6.37 percobaan 3 adalah 14.26262947 kbps. Adapun grafik dari hasil tiga percobaan diatas, dengan nilai *throughput* selama 6 menit.



Gambar 6.20 Grafik Nilai *Throughput* Dari Tiga Kali Percobaan 6 menit

Dapat dilihat pada gambar 6.20 bahwa untuk nilai throughput selama 6 menit mendapatkan nilai antara 14.18904986 sampai 14.26262947 kbps. Sehingga didapatkan nilai rata-rata dari ketiga hasil percobaan tersebut, yang nantinya akan dihitung untuk perhitungan selanjutnya. Rata-rata dari ketiga percobaan tersebut adalah 14.22124172 kbps.

- **Menghitung Nilai *Packets Loss* 6 menit**

Berikut adalah detail dari tabel menghitung nilai *Packets loss* selama 6 menit dari tiga percobaan.

Perhitungan *Packet loss*	
Parameter yang dihitung	Nilai
Paket data yang dikirim	38011
Paket data yang diterima	36760
Packet loss (%)	3.291152561

Tabel 6.38 Hasil Perhitungan Nilai *Packets Loss* Percobaan 1

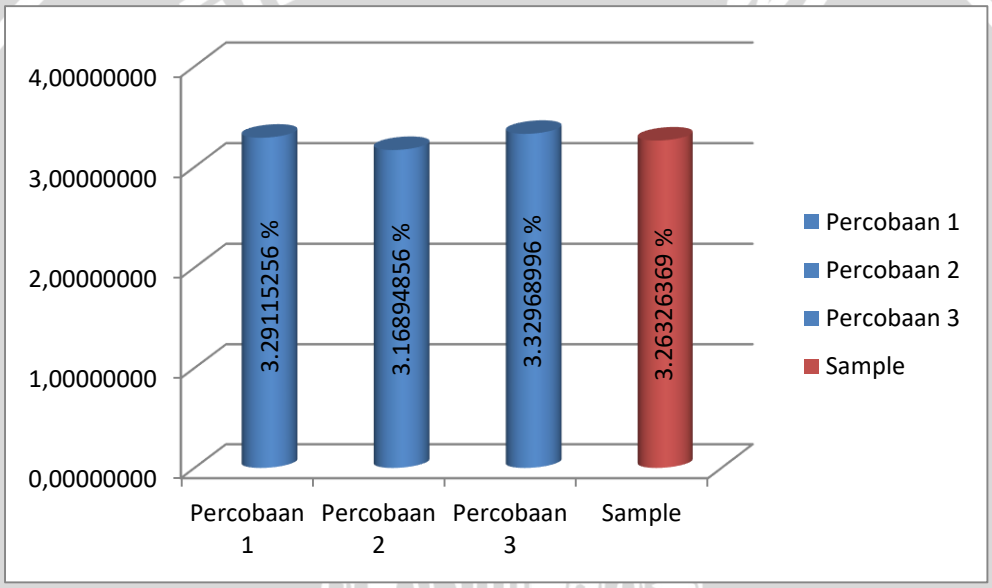
Perhitungan *Packet loss*	
Parameter yang dihitung	Nilai
Paket data yang dikirim	37615
Paket data yang diterima	36423
Packet loss (%)	3.168948558

Tabel 6.39 Hasil Perhitungan Nilai *Packets Loss* Percobaan 2

Perhitungan *Packet loss*	
Parameter yang dihitung	Nilai
Paket data yang dikirim	37511
Paket data yang diterima	36262
Packet loss (%)	3.329689958

Tabel 6.40 Hasil Perhitungan Nilai *Packets Loss* Percobaan 3

Dapat dilihat pada tabel 6.38, 6.39 dan 6.40 bahwa hasil perhitungan nilai *packets loss* pada tabel 6.38 percobaan 1 menghasilkan nilai 3.291152561 %, nilai *packets loss* pada tabel 6.39 percobaan 2 adalah 3.168948558 % dan nilai *packets loss* pada tabel 6.40 percobaan 3 adalah 3.329689958 %. Adapun grafik dari hasil tiga percobaan diatas, dengan nilai *packets loss* selama 6 menit.



Gambar 6.21 Grafik Nilai *Packets Loss* Dari Tiga Kali Percobaan 6 menit

Dapat dilihat pada gambar 6.21 bahwa untuk nilai *packets loss* selama 6 menit mendapatkan nilai antara 3.16894856 % sampai 3.32968996 %. Sehingga didapatkan nilai rata-rata dari ketiga hasil percobaan tersebut, yang nantinya akan dihitung untuk perhitungan selanjutnya. Rata-rata dari ketiga percobaan tersebut adalah 3.26326369 %.

- ***Sample* Uji Coba Selama 6 menit**

Berikut adalah *sample* yang didapatkan dari hasil uji coba selama 6 menit. *Sample* ini akan digunakan untuk menghitung *quality of service* dalam rentan waktu 2 s/d 8 menit.

Sample uji coba selama 6 menit	
Kategori	Nilai
Rata-rata *Delay* (ms)	9.970107333
Jitter (ms)	0.000270305
Throughput (kbps)	14.22124172
Packets Loss (%)	3.263263692

Tabel 6.41 *Sample* Nilai Parameter Uji QoS Selama 6 menit

Diketahui pada tabel 6.41 bahwa dalam uji coba selama 6 menit menghasilkan *sample* nilai rata-rata *delay* 9.970107333 ms, nilai *jitter* 0.000270305 ms, nilai *throughput* 14.22124172 kbps dan nilai *packets loss* 3.263263692 %.

6.4.4 Pengujian 8 menit

Pada pengujian 8 menit ini langkah-langkah yang dilakukan sama dengan pada pengujian 2, 4 dan 6 menit sebelumnya, yaitu sebanyak 3 kali pengujian dan menggunakan persamaan yang sama.

- **Menghitung Rata-rata *Delay* 8 menit**

Berikut adalah detail dari tabel menghitung rata-rata *delay* 8 menit dari tiga percobaan.

Perhitungan Rata-rata *Delay*	
Parameter yang dihitung	Nilai
Total *Delay* (*second*)	483.743111
Total paket yang diterima	48407
Rata-rata *delay* (*second*)	0.009993247
Rata-rata *delay* (ms)	9.993247

Tabel 6.42 Hasil Perhitungan Rata-rata *Delay* Percobaan 1

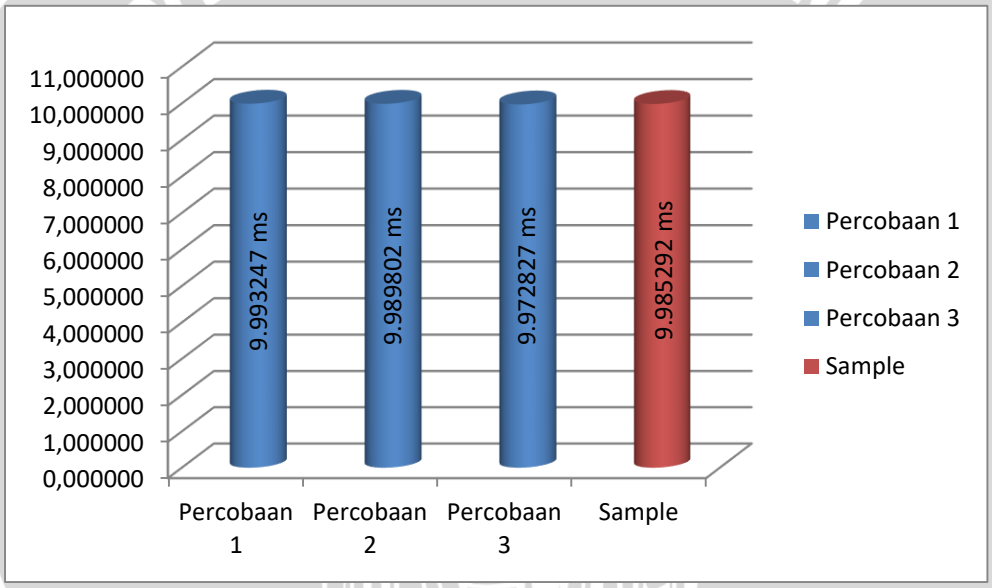
Perhitungan Rata-rata *Delay*	
Parameter yang dihitung	Nilai
Total *Delay* (*second*)	484.105818
Total paket yang diterima	48460
Rata-rata *delay* (*second*)	0.009989802
Rata-rata *delay* (ms)	9.989802

Tabel 6.43 Hasil Perhitungan Rata-rata *Delay* Percobaan 2

Perhitungan Rata-rata *Delay*	
Parameter yang dihitung	Nilai
Total *Delay* (*second*)	482.265945
Total paket yang diterima	48358
Rata-rata *delay* (*second*)	0.009972827
Rata-rata *delay* (ms)	9.972827

Tabel 6.44 Hasil Perhitungan Rata-rata *Delay* Percobaan 3

Dapat dilihat pada tabel 6.42, 6.43 dan 6.44 bahwa hasil perhitungan rata-rata *delay* pada tabel 6.42 percobaan 1 menghasilkan nilai 9.993247 ms, nilai rata-rata *delay* pada tabel 6.43 percobaan 2 adalah 9.989802 ms dan nilai rata-rata *delay* pada tabel 6.44 percobaan 3 adalah 9.972827 ms. Adapun grafik dari hasil tiga percobaan diatas, dengan nilai rata-rata *delay* selama 8 menit.



Gambar 6.22 Grafik Nilai Rata-rata *Delay* Dari Tiga Kali Percobaan 8 menit

Dapat dilihat pada gambar 6.22 bahwa untuk nilai rata-rata *delay* selama 6 menit mendapatkan nilai antara 9.972827 ms sampai 9.993247 ms. Sehingga didapatkan nilai rata-rata dari ketiga hasil percobaan tersebut, yang nantinya akan dihitung untuk perhitungan selanjutnya. Rata-rata dari ketiga percobaan tersebut adalah 9.985292 ms.

- **Menghitung Nilai *Jitter* 8 menit a, b dan c**

Berikut adalah detail dari tabel menghitung nilai *jitter* selama 8 menit dari tiga percobaan.

Perhitungan *Jitter*	
Parameter yang dihitung	Nilai
Total Variasi *Delay* (*second*)	0.012701
Total paket yang diterima -1	48406
Jitter (*second*)	2.62E-07
Jitter (ms)	0.000262385

Tabel 6.45 Hasil Perhitungan Nilai *Jitter* Percobaan 1

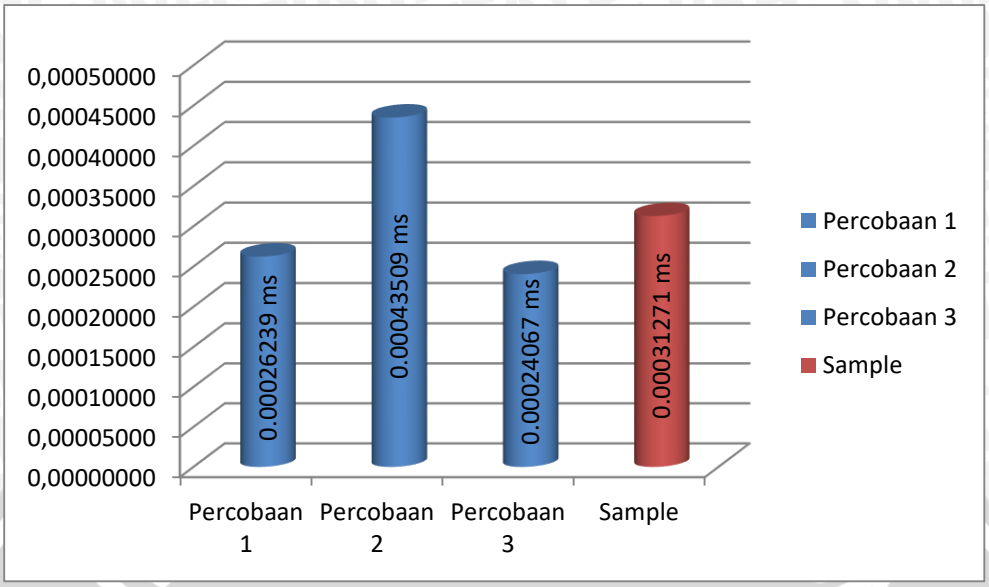
Perhitungan *Jitter*	
Parameter yang dihitung	Nilai
Total Variasi *Delay* (*second*)	0.021084
Total paket yang diterima -1	48459
Jitter (*second*)	4.35E-07
Jitter (ms)	0.000435089

Tabel 6.46 Hasil Perhitungan Nilai *Jitter* Percobaan 2

Perhitungan *Jitter*	
Parameter yang dihitung	Nilai
Total Variasi *Delay* (*second*)	1.16E-02
Total paket yang diterima -1	48357
Jitter (*second*)	2.41E-07
Jitter (ms)	0.00024067

Tabel 6.47 Hasil Perhitungan Nilai *Jitter* Percobaan 3

Dapat dilihat pada tabel 6.45, 6.46 dan 6.47 bahwa hasil perhitungan nilai *jitter* pada tabel 6.45 percobaan 1 menghasilkan nilai 0.000262385 ms, nilai *jitter* pada tabel 6.46 percobaan 2 adalah 0.000435089 ms dan nilai *jitter* pada tabel 6.47 percobaan 3 adalah 0.00024067 ms. Adapun grafik dari hasil tiga percobaan diatas, dengan nilai *jitter* selama 8 menit.



Gambar 6.23 Grafik Nilai *Jitter* Dari Tiga Kali Percobaan 8 menit

Dapat dilihat pada gambar 6.23 bahwa untuk nilai *jitter* selama 8 menit mendapatkan nilai antara 0.00024067 ms sampai 0.00043509 ms. Sehingga didapatkan nilai rata-rata dari ketiga hasil percobaan tersebut, yang nantinya akan dihitung untuk perhitungan selanjutnya. Rata-rata dari ketiga percobaan tersebut adalah 0.00031271 ms.

- **Menghitung Nilai *Throughput* 8 menit**

Berikut adalah detail dari tabel menghitung nilai *throughput* selama 8 menit dari tiga percobaan.

Perhitungan *Throughput*	
Parameter yang dihitung	Nilai
Paket data yang diterima (bytes)	6849533
Lama pengamatan (*second*)	483.743
Throughput (bps)	14159.44623
Throughput (kbps)	14.15944623

Tabel 6.48 Hasil Perhitungan Nilai *Throughput* Percobaan 1

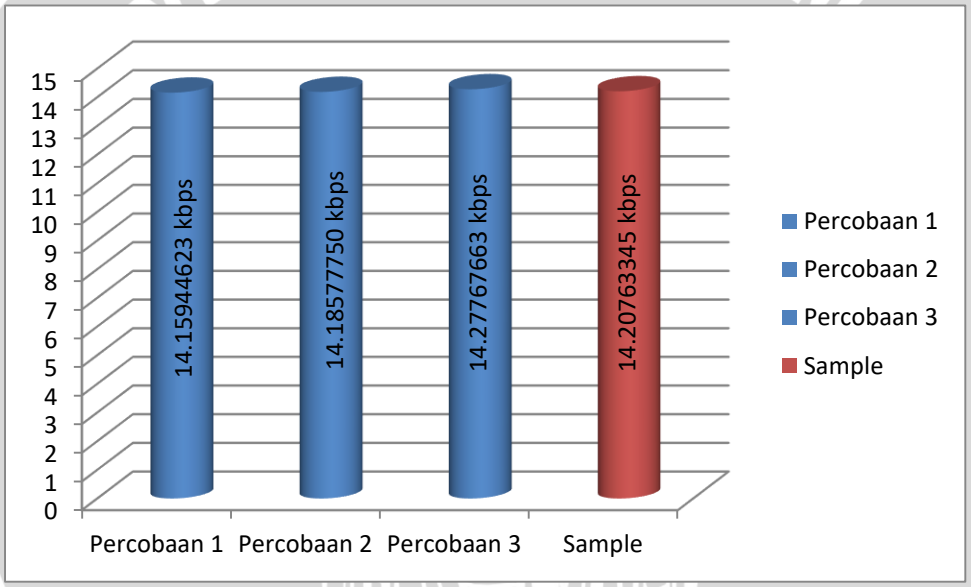
Perhitungan *Throughput*	
Parameter yang dihitung	Nilai
Paket data yang diterima (bytes)	6867420
Lama pengamatan (*second*)	484.106
Throughput (bps)	14185.7775
Throughput (kbps)	14.1857775

Tabel 6.49 Hasil Perhitungan Nilai *Throughput* Percobaan 2

Perhitungan *Throughput*	
Parameter yang dihitung	Nilai
Paket data yang diterima (bytes)	6885638
Lama pengamatan (*second*)	482.266
Throughput (bps)	14277.67663
Throughput (kbps)	14.27767663

Tabel 6.50 Hasil Perhitungan Nilai *Throughput* Percobaan 3

Dapat dilihat pada tabel 6.48, 6.49 dan 6.50 bahwa hasil perhitungan nilai *throughput* pada tabel 6.48 percobaan 1 menghasilkan nilai 14.15944623 kbps, nilai *throughput* pada tabel 6.49 percobaan 2 adalah 14.1857775 kbps dan nilai *throughput* pada tabel 6.50 percobaan 3 adalah 14.27767663 kbps. Adapun grafik dari hasil tiga percobaan diatas, dengan nilai *throughput* selama 8 menit.



Gambar 6.24 Grafik Nilai *Throughput* Dari Tiga Kali Percobaan 8 menit

Dapat dilihat pada gambar 6.24 bahwa untuk nilai *throughput* selama 8 menit mendapatkan nilai antara 14.15944623 kbps sampai 14.27767663 kbps. Sehingga didapatkan nilai rata-rata dari ketiga hasil percobaan tersebut, yang nantinya akan dihitung untuk perhitungan selanjutnya. Rata-rata dari ketiga percobaan tersebut adalah 14.20763345 kbps.

- **Menghitung Nilai *Packets Loss* 8 menit**

Berikut adalah detail dari tabel menghitung nilai *Packets loss* selama 8 menit dari tiga percobaan.

Perhitungan *Packet loss*	
Parameter yang dihitung	Nilai
Paket data yang dikirim	49419
Paket data yang diterima	48407
Packet loss (%)	2.047795382

Tabel 6.51 Hasil Perhitungan Nilai *Packets Loss* Percobaan 1

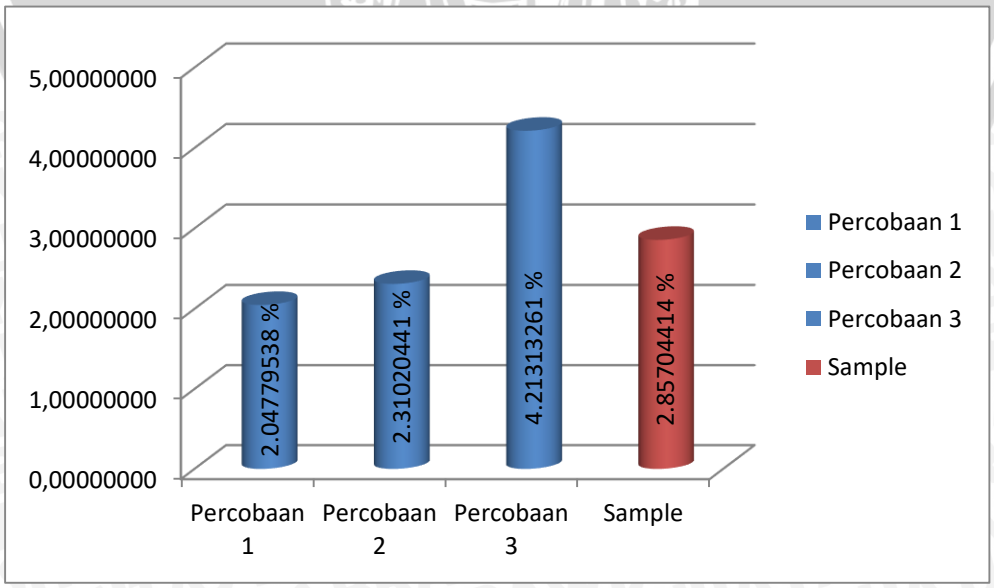
Perhitungan *Packet loss*	
Parameter yang dihitung	Nilai
Paket data yang dikirim	49606
Paket data yang diterima	48460
Packet loss (%)	2.310204411

Tabel 6.52 Hasil Perhitungan Nilai *Packets Loss* Percobaan 2

Perhitungan *Packet loss*	
Parameter yang dihitung	Nilai
Paket data yang dikirim	50485
Paket data yang diterima	48358
Packet loss (%)	4.213132614

Tabel 6.53 Hasil Perhitungan Nilai *Packets Loss* Percobaan 3

Dapat dilihat pada tabel 6.51, 6.52 dan 6.53 bahwa hasil perhitungan nilai *packets loss* pada tabel 6.51 percobaan 1 menghasilkan nilai 2.047795382 %, nilai *packets loss* pada tabel 6.52 percobaan 2 adalah 2.310204411 % dan nilai *packets loss* pada tabel 6.53 percobaan 3 adalah 4.213132614 %. Adapun grafik dari hasil tiga percobaan diatas, dengan nilai *packets loss* selama 8 menit.



Gambar 6.25 Grafik Nilai *Packets Loss* Dari Tiga Kali Percobaan 8 menit

Dapat dilihat pada gambar 6.25 bahwa untuk nilai *packets loss* selama 8 menit mendapatkan nilai antara 2.04779538 % sampai 4.21313261 %. Sehingga didapatkan nilai rata-rata dari ketiga hasil percobaan tersebut, yang nantinya akan dihitung untuk perhitungan selanjutnya. Rata-rata dari ketiga percobaan tersebut adalah 2.85704414 %.

- ***Sample* Uji Coba Selama 8 menit**

Berikut adalah *sample* yang didapatkan dari hasil uji coba selama 6 menit. *Sample* ini akan digunakan untuk menghitung *quality of servic*e dalam rentan waktu 2 s/d 8 menit.

Sample uji coba selama 8 menit	
Kategori	Nilai
Rata-rata *Delay* (ms)	9.985292
Jitter (ms)	0.000312714
Throughput (kbps)	14.20763345
Packets Loss (%)	2.857044136

Tabel 6.54 *Sample* Nilai Parameter Uji QoS Selama 8 menit

Diketahui pada tabel 6.54 bahwa dalam uji coba selama 8 menit menghasilkan *sample* nilai rata-rata *delay* 9.985292 ms, nilai *jitter* 0.000312714 ms, nilai *throughput* 14.20763345 kbps dan nilai *packets loss* 2.857044136 %.

6.4.5 Hasil Pengujian Dari rentan Waktu 2 s/d 8 menit

Disinilah inti dari penelitian ini, menganalisa kompatibilitas dan *quality of service* WebRTC *audio call* menggunakan *framework easyRTC* dengan *codec* opus dan node.js sebagai media signalingnya. Dari beberapa pengujian parameter QoS *delay*, *jitter*, *throughput* dan *packets loss* selama 2 sampai 8 menit yang telah dilakukan, dari *sample* yang didapat pada setiap rentan waktunya, yaitu 2 menit, 4 menit, 6 menit dan 8 menit. Dari nilai *sample* tersebut dapat digunakan untuk menghitung nilai *quality of service* dari rentan waktu 2 sampai 8 menit.

Pada tahap ini akan melakukan perhitungan untuk merata-rata hasil dari setiap parameter pada rentan waktu 2 sampai 8 menit. Hasilnya akan disesuaikan dengan standarisasi komunikasi yang baik versi TIPHON.

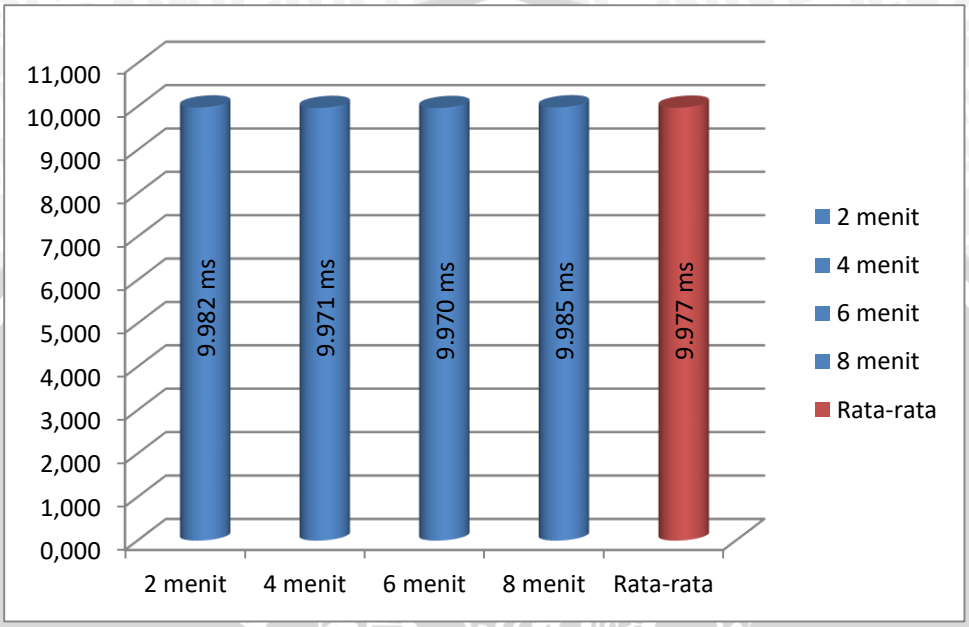
- **Menghitung Rata-rata *Delay* 2 s/d 8 menit**

Setelah mendapatkan *sample* rata-rata *delay* pada setiap rentan waktu. Diketahui bahwa nilai rata-rata *delay* pada setiap rentan waktu 2 s/d 8 menit antara 9.970 ms sampai 9.985 ms. Sehingga didapatkan hasil nilai dari rata-rata *delay* 2 s/d 8 menit dengan rumus

$$rata - rata = \frac{(2\ \text{menit} + 4\ \text{menit} + 6\ \text{menit} + 8\text{menit})}{4}$$

Persamaan 7 Menghitung Rata-rata Nilai dari Keempat Parameter Qos

Dari perhitungan diatas yang menggunakan persamaan 7 menghasilkan nilai rata-rata *delay* 9.977 ms. Adapun grafik yang menunjukkan nilai rata-rata *delay* pada setiap rentan waktu 2 s/d 8 menit.

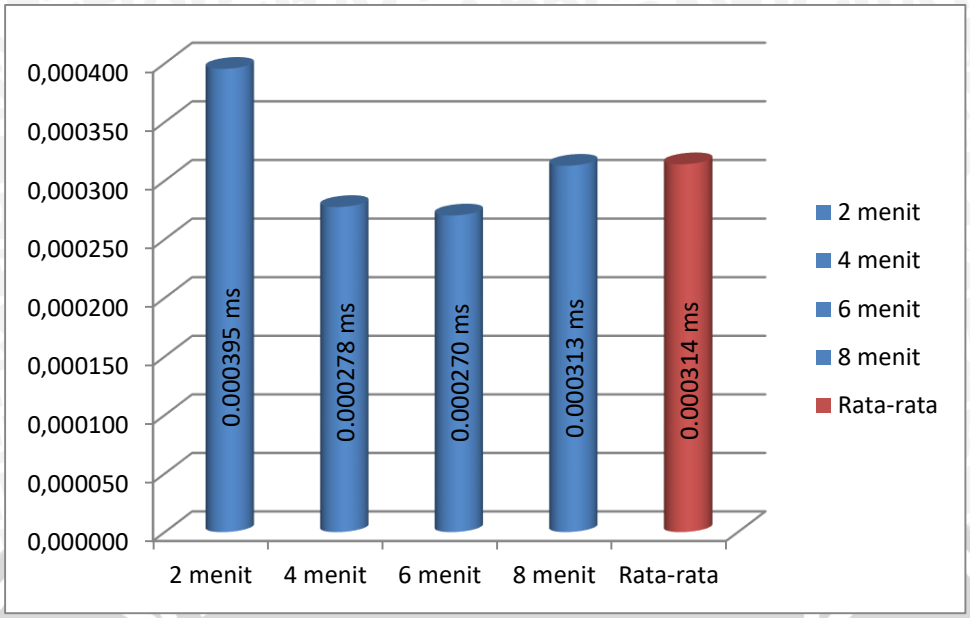


Gambar 6.26 Grafik Nilai Rata-rata *Delay* Pada Rentan Waktu 2 s/d 8 menit

Pada gambar 6.26 grafik yang berwarna merah menunjukkan nilai rata-rata *delay* 9.977 ms, nilai rata-rata *delay* tersebut untuk dibandingkan dengan *quality of service* versi TIPHON.

- **Menghitung Nilai *Jitter* 2 s/d 8 menit**

Pada perhitungan nilai *jitter* yang telah dilakukan di setiap rentan waktunya. Diketahui bahwa nilai rata-rata *jitter* pada setiap rentan waktu 2 s/d 8 menit antara 0.000270 ms sampai 0.000395 ms. Sehingga didapatkan hasil nilai dari rata-rata *jitter* 2 s/d 8 menit dengan persamaan 7. sehingga menghasilkan nilai rata-rata *jitter* 0.000314 ms. Adapun grafik yang menunjukkan nilai rata-rata *jitter* pada setiap rentan waktu 2 s/d 8 menit.

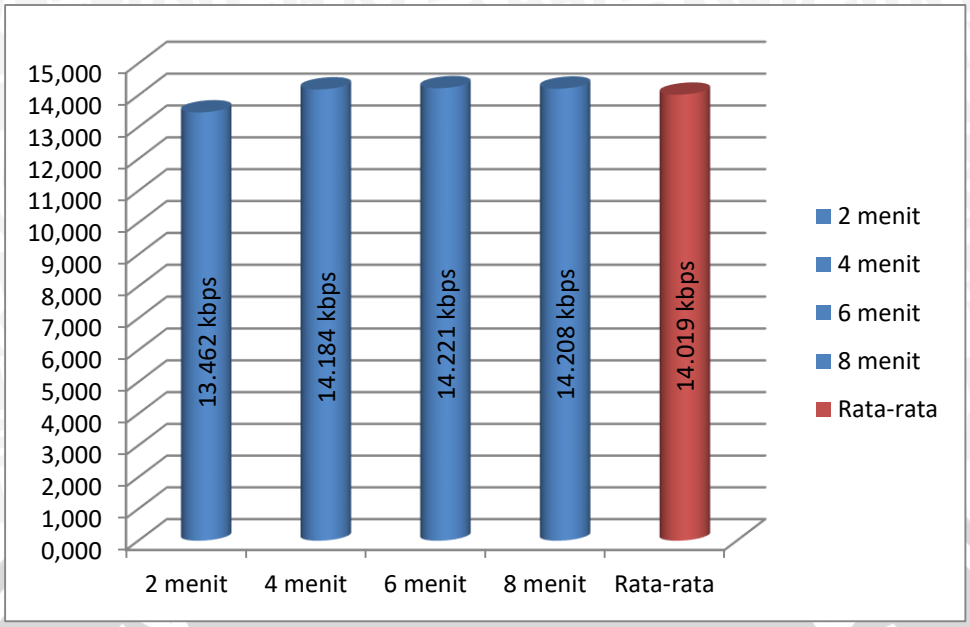


Gambar 6.27 Grafik Nilai Rata-rata *Jitter* Pada Rentan Waktu 2 s/d 8 menit

Pada gambar 6.27 grafik yang berwarna merah menunjukkan nilai rata-rata *jitter* 0.000314 ms, nilai rata-rata *jitter* tersebut untuk dibandingkan dengan *quality of service* versi TIPHON.

- **Menghitung Nilai *Throughput* 2 s/d 8 menit**

Untuk perhitungan nilai *throughput* yang telah dilakukan di setiap rentan waktunya. Diketahui bahwa nilai rata-rata *throughput* pada setiap rentan waktu 2 s/d 8 menit antara 13.462 kbps sampai 14.221 kbps. Sehingga didapatkan hasil nilai dari rata-rata *throughput* 2 s/d 8 menit dengan persamaan 7. sehingga menghasilkan nilai rata-rata *throughput* 14.019 kbps. Adapun grafik yang menunjukkan nilai rata-rata *throughput* pada setiap rentan waktu 2 s/d 8 menit.

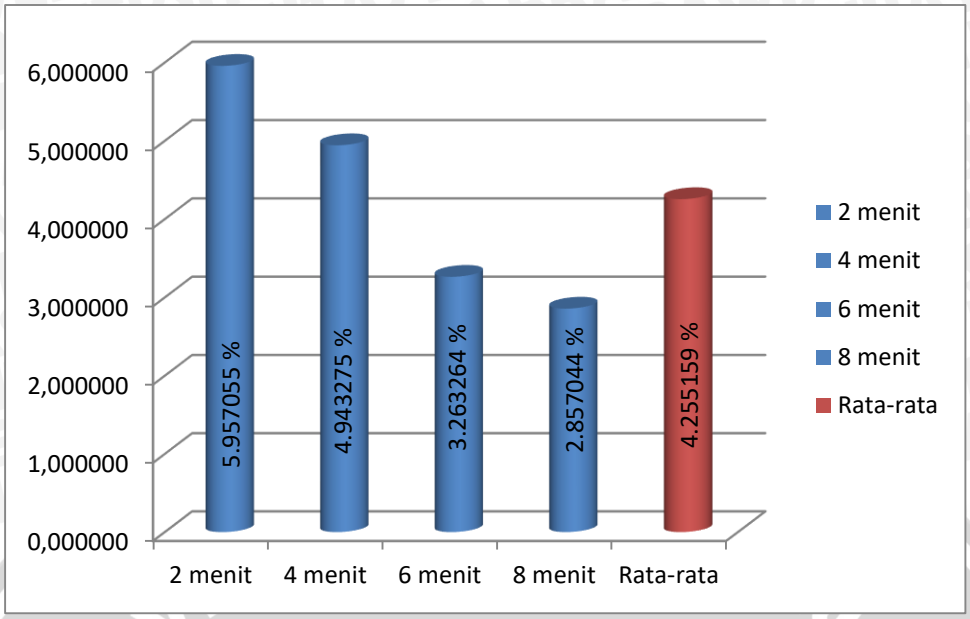


Gambar 6.28 Grafik Nilai Rata-rata *Throughput* Pada Rentan Waktu 2 s/d 8 menit

Pada gambar 6.28 grafik yang berwarna merah menunjukkan nilai rata-rata *throughput* 14.019 kbps, nilai rata-rata *throughput* tersebut untuk dibandingkan dengan *quality of service* versi TIPHON.

- **Menghitung Nilai *Packets Loss* 2 s/d 8 menit**

Pada perhitungan nilai *packets loss* yang telah dilakukan di setiap rentan waktunya. Diketahui bahwa nilai rata-rata *packets loss* pada setiap rentan waktu 2 s/d 8 menit antara 2.857044 %sampai 5.957055 %. Sehingga didapatkan hasil nilai dari rata-rata *packets loss* 2 s/d 8 menit dengan persamaan 7. Sehingga menghasilkan nilai rata-rata *packets loss* 4.255159 %. Adapun grafik yang menunjukkan nilai rata-rata *packets loss* pada setiap rentan waktu 2 s/d 8 menit.



Gambar 6.29 Grafik Nilai Rata-rata *Packets Loss* Pada Rentan Waktu 2 s/d 8 menit

Pada gambar 6.29 grafik yang berwarna merah menunjukkan nilai rata-rata *packets loss* 4.255159 %, nilai rata-rata *packets loss* tersebut untuk dibandingkan dengan *quality of service* versi TIPHON.

- **Hasil Uji Coba Dalam Rentan Waktu 2 s/d 8 menit**

Setelah mendapatkan nilai rata-rata dari setiap parameter *quality of service*, yaitu rata-rata *delay*, *jitter*, *throughput* dan *packets loss*, maka didapatkan hasil dari pengujian quality of service WebRTC audio call dengan *framework easyRTC* yang menggunakan codec opus dengan node.js sebagai media *signaling*-nya pada rentan waktu 2 s/d 8 menit. Hasil tersebut ditampilkan pada tabel 6.55 dibawah ini.

Rata-rata QoS dalam rentan waktu 2 sampai 8 menit	
Kategori	Nilai
Rata-rata *Delay* (ms)	9.977
Jitter (ms)	0.000314
Throughput (kbps)	14.019
Packets Loss (%)	4.255159

Tabel 6.55 Nilai Parameter QoS Yang Dihasilkan Pada Rentan Waktu 2 s/d 8 menit

Setelah mendapatkan data *quality of service*, maka langkah selanjutnya membandingkan nilai dari parameter tersebut dengan standarisasi *quality of service* versi TIPHON. Pembahasan mengenai detail analisa parameter *quality of service* dibahas pada bab 6.5.6.

6.4.6 Analisis Hasil Pengujian Dari rentan Waktu 2 s/d 8 menit

Pada tahap ini melakukan perbandingan nilai dari parameter rata-rata *delay*, *jitter*, *throughput* dan *packets loss* yang didapatkan dari rangkaian percobaan yang telah dilakukan dengan *quality of service* versi TIPHON. Indeks dari standarisasi versi TIPHON dapat dilihat pada tabel 3.1.

- **Perbandingan Nilai Rata-rata Delay dengan versi TIPHON**

Pada *point* ini membandingkan nilai rata-rata delay yang didapatkan dari rangkaian percobaan yang telah dilakukan dengan standarisasi nilai rata-rata delay versi TIPHON. Standarisasi *delay* versi TIPHON dapat dilihat pada tabel 3.2.

Nilai rata-rata *delay* yang dihasilkan adalah 9.977 ms. Jika dibandingkan dengan standarisasi Delay pada versi TIPHON masuk kedalam indeks sangat baik, dikarenakan nilai rata-rata *delay* dari penelitian ini < 150 ms. Dapat disimpulkan bahwa WebRTC *audio call* dengan menggunakan *framework easyRTC* dengan codec opus dan node.js sebagai media *signaling*-nya untuk pertukaran data sangat baik hanya dalam rata-rata *delay* 9.977 ms dari rentan waktu 2 s/d 8 menit.

- **Perbandingan Nilai Jitter dengan versi TIPHON**

Pada point ini membandingkan nilai *jitter* yang dihasilkan dari rangkaian percobaan yang telah dilakukan dengan standarisasi nilai *jitter* versi TIPHON. Standarisasi nilai *jitter* versi TIPHON dapat dilihat pada tabel 3.5.

Nilai *jitter* yang dihasilkan adalah 0.000314 ms. Jika dibandingkan dengan standarisasi *jitter* pada versi TIPHON masuk kedalam indeks baik, dikarenakan nilai rata-rata *jitter* dari penelitian ini adalah 0 - 75 ms. Dapat disimpulkan bahwa WebRTC *audio call* dengan menggunakan *framework easyRTC* dengan codec opus dan node.js sebagai media *signaling*-nya, mampu menangani beban paket data yang bertambah semakin banyak, hal tersebut dapat dilihat dari *record* data yang diterima pada rentan waktu 2 s/d 8 menit. Perlu diketahui faktor meningkatnya nilai *jitter* adalah semakin banyaknya beban paket data yang diterima pada jaringan (Dewi Yolanda S. A. dkk, 2013).

- **Perbandingan Nilai throughput dengan versi TIPHON**

Pada point ini membandingkan nilai *throughput* yang dihasilkan dari rangkaian percobaan yang telah dilakukan dengan standarisasi nilai *throughput* versi TIPHON. Standarisasi *throughput* versi TIPHON dapat dilihat pada tabel 3.4.

Nilai *throughput* yang dihasilkan adalah 14.019 kbps. Perlu diketahui pada penelitian ini hanya menggunakan *bandwidth* 128 kbps. Sehingga dari 128 kbps *throughput* yang dihasilkan adalah 14.128 kbps, maka diperlukannya persamaan untuk mengubah menjadi nilai persentase, dikarenakan standarisasi *throughput* versi TIPHON bernilai persentase.

Persamaan untuk mengubah nilai *throughput* menjadi persentase dapat dilihat pada persamaan 8.

$$Throughput\ (\%) = \frac{Throughput}{Bandwidth} x100$$

Persamaan 8 Persentase Throughput

$$(\%) = \frac{14.019}{128} x100$$

Hasil perhitungan *throughput* dari satuan kbps ke satuan persentase menghasilkan nilai 10.952%. hasil tersebut menunjukkan bahwa dari *bandwidth* 128 kbps hanya menghasilkan *throughput* 14.128 kbps. Sehingga jika persentasekan mendapatkan nilai *throughput* 10.952%

Jika dibandingkan dengan standarisasi *throughput* pada versi TIPHON masuk kedalam indeks jelek, dikarenakan nilai *throughput* dari penelitian ini <25%. Dapat disimpulkan bahwa dengan menggunakan *bandwidth* 128 kbps WebRTC *audio call* dengan menggunakan *framework easyRTC* dengan codec opus dan node.js sebagai media *signaling*-nya, dalam melakukan pengiriman data yang sukses sampai ketujuan masih jelek.

- **Perbandingan Nilai Packets Loss dengan versi TIPHON**

Pada *point* ini membandingkan nilai *packets loss* yang dihasilkan dari rangkaian percobaan yang telah dilakukan dengan standarisasi nilai rata-rata *packets loss* versi TIPHON. Standarisasi *packets loss* versi TIPHON dapat dilihat pada tabel 3.3.

Nilai *packets loss* yang dihasilkan adalah 4.255159%. Jika dibandingkan dengan standarisasi *packets loss* pada versi TIPHON masuk kedalam indeks sedang, dikarenakan nilai packets loss dari penelitian ini > 3%. Hal ini berbanding lurus dengan nilai *throughput* yang dihasilkan, nilai *throughput* yang dihasilkan pada penelitian ini masuk kedalam indeks jelek pada standarisasi versi TIPHON.

BAB 7 PENUTUP

7.1 Kesimpulan

Berdasarkan penelitian WebRTC yang telah dilakukan yaitu menganalisa framework easyRTC dan node.js sebagai media *signaling* serta codec opus yang digunakan untuk mentransmisi suara, dapat disimpulkan bahwa :

1. Dari lima *browser* yang telah dicoba untuk menguji kompatibilitas WebRTC yaitu *browser* chrome v52.0, Mozilla firefox v38.0.5, opera v38.0, safari v5.1.7 dan internet explorer v8.0, WebRTC dapat berjalan dengan lancar hanya pada *browser* Mozilla firefox. Selain *browser* Mozilla firefox terdapat error, error yang ditampilkan oleh sistem adalah sebagai berikut :
   - *Browser* Chrome v52.0

     "*Failed to get access to local media. Error code was PermissionDeniedError*". Hal ini menunjukkan bahwa saat sistem meminta izin untuk mengakses media mikrofon javascript dari *framework easyRTC* pada API getUserMedia() terjadi *error code*.
   - *Browser* Safari v5.1.7

     "*Your browser doesn't appear to support WebRTC*". Hal ini menunjukkan bahwa *browser* Safari v5.1.7 masih belum *support* dengan arsitektur WebRTC, tetapi *browser* safari v5.1.7 mampu untuk mengakses website pada server dengan port 8181, dapat diartikan bahwa HTML5 yang digunakan *browser* safari v5.1.7 masih belum *support* dengan HTML5 yang diterapkan oleh WebRTC.
   - *Browser* Opera v38.0

     "*Failed to get access to local media. Error code was PermissionDeniedError*" sama seperti error pada *browser* chrome.
   - *Browser* Internet Explorer v8.0

     Pada *browser* internet explorer v8.0 memberikan notifikasi *error* "*The webpage cannot be displayed*". Hal ini menunjukkan bahwa *browser* internet explorer tidak *support* untuk diterapkan WebRTC. Untuk mengakses *website* yang ada pada server dengan port 8181 masih belum bisa, dapat diartikan bahwa javascript dari internet explorer v8.0 masih belum *support* dengan javascript yang diterapkan oleh WebRTC.
2. Terdapat empat parameter *quality of service* yang telah diuji pada waktu yang berbeda-beda yaitu 2 menit, 4 menit, 6 menit dan 8 menit parameter *quality of service* yang dimaksud adalah rata-rata *delay*, *jitter*, *throughput* dan *packets loss*.

- Rata-rata *Delay*

  Dapat disimpulkan bahwa nilai *delay* dari pengujian 2 menit, 4 menit, 6 menit dan 8 menit mempunyai nilai yang konstan yaitu antara 9.970 ms sampai 9.985 dan mempunyai rata-rata *delay* 9.977 ms. Dengan *bandwidth* 128 kbps waktu *delay* pengiriman paket data WebRTC *audio call* yang menggunakan *framework easyRTC* dan node.js sebagai media *signaling*-nya serta *codec* opus sebagai transmisi paket data audio memerlukan waktu rata-rata 9.977 ms agar data sampai ketujuan.
- *Jitter*

  Nilai *jitter* yang diperoleh dari pengujian pada rentan waktu 2 menit, 4 menit, 6 menit dan 8 menit adalah berkisar antara 0.000270 ms sampai 0.000395 ms dan mempunyai nilai rata-rata 0.000314 ms. Faktor yang mempengaruhi nilai *jitter* disebabkan karena terdapatnya variasi-variasi dalam panjang antrian dan lama waktu pengolahan paket data. *Jitter* disebut juga dengan variasi *delay* karena pada *latency* terdapat variasi *delay* pada transmisi data dalam jaringan.
- *Throughput*

  Nilai *throughput* yang dihasilkan dari pengujian pada rentan waktu 2 menit, 4 menit, 6 menit dan 8 menit mempunyai nilai antara 13.462 kbps sampai 14.221 kbps dan mempunyai nilai rata-rata 14.019. perlu diketahui pada pengujian ini menggunakan *bandwidth* sebesar 128 kbps namun *throughput* yang dihasilkan hanya 14.019 kbps. Faktor seperti redaman dan kapasitas *bandwidth* dapat mempengaruhi hasil dari pengukuran ini.
- *Packets Loss*

  Nilai *packets loss* yang diperoleh dari pengujian ini pada rentan waktu 2 menit, 4 menit, 6 menit dan 8 menit menghasilkan nilai persentase antara 2.857044% sampai 5.957055% dan mempunyai nilai rata-rata persentase 4.255159%. faktor yang menyebabkan terjadinya *packets loss* karena adanya tumbukan atau tabrakan antara data pada jaringan. Dapat juga *buffer* pada suatu perangkat jaringan kelebihan beban sehingga tidak dapat menampung data baru dan menyebabkan kehilangan data.

3. Dari nilai-nilai yang telah dihasilkan dalam pengujian ini dengan parameter *delay*, *jitter*, *throughput* dan *packets loss* bahwa WebRTC yang menerapkan *framework easyRTC* dan node.js sebagai media *signaling* serta *codec* opus sebagai *codec* untuk transmisi suara dan juga menggunakan *bandwidth* sebesar 128 kbps, dapat disimpulkan bahwa sudah cukup baik untuk melakukan komunikasi VOIP dengan metode WebRTC, karena mempunyai nilai indeks *delay* sangat baik dan nilai indeks *jitter* yang baik menurut versi TIPHON, namun pada nilai indeks *throughput* yang jelek dan nilai indeks *packets loss* yang sedang menurut versi TIPHON.

7.2 Saran

Berdasarkan kesimpulan pada penelitian ini, maka penulis mencoba memberikan beberapa saran agar penelitian ini dapat lebih bermanfaat. Saran yang penulis angkat seperti berikut :

1. Untuk dapat mengetahui hasil analisis *quality of service* yang lebih baik dari WebRTC dengan menggunakan *framework easyRTC* dapat menggunakan *bandwidth* yang lebih variatif.
2. Penelitian WebRTC ini hanya menggunakan codec opus sebagai media transmisi data, sehingga kedepannya dapat diteliti lebih lanjut dengan codec yang berbeda.
3. Parameter uji *quality of service* masih menggunakan *delay*, *jitter*, *throughput* dan *packets loss*. Penulis berharap adanya penelitian lebih lanjut dengan parameter uji *quality of service* yang lain, seperti MOS (*Mean Opinion Score*), pengukuran *CPU usage* atau pengukuran *memory usage*.
4. Penelitian kompatibilitas dari WebRTC masih menggunakan *browser* dengan versi chrome v52.0, Mozilla v38.0.5, opera v38.0, safari v5.1.7, dan internet explorer v8.0, jika kelima *browser* tersebut sudah di perbarui oleh developer, maka dapat dijadikan penelitian lebih lanjut.

DAFTAR PUSTAKA

[1] Brett McLaughlin, 6 Juli 2011. "*What is Node.js?*", publisher O'Reilly Media. http://radar.oreilly.com/2011/07/what-is-node.html.

[2] Muhammad Iqbal C. R., Muchammad Husni dan Hudan Studiawan. Sept, 2012. "*Implementasi Klien SIP Berbasis Web Menggunakan HTML5 dan Node.js*". http://ejurnal.its.ac.id/index.php/teknik/article/view/643.

[3] 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.

[4] Rob Manson, September 2013, "*Getting Started with WebRTC*", www.packtpub.com.

[5] Erick Linask, May 2013, "*Making the Right Signalling Choice*", http://docslide.us/technology/WebRTC-conference-and-expo-may-2013-making-the-right-signalling-choice.html.

[6] Dewi Yolanda S. A., Dr. Ir. Sholeh Hadi Pramono, MS., M. Fauzan Edy Purnomo, ST., MT., 2013, "*Simulasi Kinerja Routing Protokol Open Shortest Path First (OSPF) dan Enhanced Interior Gateway Routing Protocol (EIGRP) Menggunakan Simulator Jaringan Opnet Modeler v. 14.5*" http://elektro.studentjournal.ub.ac.id/index.php/teub/article/view/79.

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

[8] Yanto. 2014, "*ANALISIS QOS (QUALITY OF SERVICE) PADA JARINGAN INTERNET (STUDI KASUS: FAKULTAS TEKNIK UNIVERSITAS TANJUNGPURA)*".

[9] Noviyanti K, St.Rukmini Adikarini, Muhammad Nizwar, Mukarramah Yusuf, 9 Agustus 2014, "*PENGEMBANGAN APLIKASI LIVE CHAT DENGAN MENGGUNAKAN WEBRTC SEBAGAI PEMANFAATAN TEKNOLOGI INFORMASI DAN KOMUNIKASI UNTUK MEDIA PEMBELAJARAN JARAK JAUH (E-LEARNING)*".

[10] Ben Feher, Lior Sidi, Asaf Shabtai, Rami Puzis, 2 Januari 2016, "*The Security of WebRTC*", http://arxiv.org/abs/1601.00184.

[11] Jean-Marc Valin, Gregory Maxwell, Timothy B. Terriberry, and Koen Vos, Oktober 2013, "*High-Quality, Low-Delay Music Coding in the Opus Codec*".

[12] William S. Bubanto, Arie S. M. Lumenta, Xaverius Najoan, 2014, "*Analisis Kualitas Layanan Jaringan Internet (Studi Kasus PT. Kawanua Internetindo Manado)*".

[13] Ansuman Kar, 2014, "Secure Real-Time Communication", https://courses.cs.washington.edu/courses/csep590/06wi/finalprojects/kar.doc.