Analisis QoS pada WebRTC dengan Mekanisme Congestion Control di Jaringan Lokal

Farhan Yuleo Pratama[1], Aji Gautama Putrada [2], Febri Dawani[3],

[1,2,3] Universitas Telkom, Bandung
[1]farhanyuleopratama@students.telkomuniversity.ac.id, [2]ajigps@telkomuniversity.ac.id,
[3]febridwani@telkomuniversity.ac.id

Abstrak

Dengan adanya komunikasi real time video dan audio mengubah cara orang berkomunikasi melalui Internet secara signifikan, hal ini didukung dengan muncul nya Web Komunikasi Real-Time (WebRTC) yang memungkinkan penggunaan browser web untuk transmisi multimedia streaming. Namun terdapat tantangan yaitu bagaimana mengatur kualitas pengiriman saat kondisi *traffi*c yang padat, untuk mengetahui efisiensi implementasi WebRTC di jaringan, parameter QoS yang digunakan adalah *throughput, packet loss, delay and jitter*. Pada tugas akhir ini, hasil aliran video yang di dapat ditransmisikan dengan teknologi WebRTC, dibandingkan yang diperoleh dengan salah satu yang tidak terkontrol *congestion* nya yaitu transmisi UDP dengan menggunakan tools iPerf. Dengan menerapkan skenario *congestion control* diharapkan dapat mengukur sampai mana kualitas layanan multimedia streaming.

Kata kunci : WebRTC, Quality of Service, DSCP, Real-time Communication, UDP, Video Conference

Abstract

With real time video and audio communication changing the way people communicate over the Internet significantly, this is supported by the emergence of Web Real-Time Communication (WebRTC) which allows the use of web browsers for streaming multimedia transmissions. But there are challenges, namely how to regulate the quality of delivery time conditions, traffic crowded to determine the efficiency of the implementation of WebRTC in the network, the QoS parameters used are throughput, packet loss, delay andjitter.In this final project, the results of the video stream that can be transmitted using WebRTC technology, are compared with one that is not controlled by congestion, namely UDP transmission using iPerf tools. By applying the congestion control scenario, it is expected to measure the quality of multimedia streaming services.

Keywords: WebRTC, Quality of Service, DSCP, Real-time Communication, UDP, Video Conference

1. Pendahuluan

1.1 Latar Belakang

Dengan adanya aplikasi video conferences telah mengubah cara orang berkomunikasi melalui Internet secara signifikan, [1] Motivasi berada di balik pengenalan Web Komunikasi Real-Time (WebRTC) adalah untuk memungkinkan penggunaan browser web untuk transmisi multimedia streaming, seperti suara, video, game, dan juga pendukung kolaborasi jarak jauh di antara pengguna.[2]

Namun demikian, jaminan Quality of Service (QoS) yang memadai untuk aplikasi real-time berbasis web seperti percakapan suara dan video tidak diberikan prioritas dalam domain internet. Dengan menggunakan komunikasi real-time berbasis web, dari sudut pandang end-user's pastinya ingin kualitas layanan yang diterima dengan baik untuk audio dan video. Oleh karena itu, peningkatan dukungan QoS End-to-End (E2E) merupakan tantangan berkelanjutan bagi penyedia jaringan. [3]. Tantangan penting dalam jaringan masa depan ialah bagaimana menyediakan konektivitas jaringan dan layanan yang memastikan persyaratan QoS untuk lalu lintas yang berbeda dihasilkan oleh aplikasi[4].

Pada tugas akhir ini, penulis merancang jaringan WebRTC dimana parameter Quality of Service (QoS) yang didapatkan akan dibandingkan dengan yang melalui melalui skenario Lalu lintas UDP dengan menggunakan iperf termasuk faktor time transfer, *bandwidth, delay, jitter* dan *packet loss* [5].

1.2 Topik dan Batasannya

Pada penelitian ini penulis akan menguji performansi WebRTC yang di bangun menggunakan server Node.js, Penelitian ini akan berfokus pada parameter QoS(*throughput,packet loss, delay* dan *jitter*), Pada proses pengujian performansi WebRTC, untuk mendapat nilai dari parameter tersebut digunakan wireshark sebagai analisa paket yang ditransmisikan WebRTC dan UDP pada jaringan, yang terdiri dari *foreground traffic* (dihasilkan dari kamera webcam yang menangkap video) dan *background traffic* (menggunakan Iperf3 sebagai

generic generator *traffic* UDP). Pada penelitian ini terdapat beberapa batasan masalah, yaitu uji performansi mengukur *congestion control* WebRTC, uji coba dilakukan pada jaringan lokal dengan Wireless LAN 802.11. Uji coba dilakukan pada web browser yang mendukung sistem layanan.

1.3 Tujuan

Pada penelitian ini dilakukan dengan tujuan membangun Mendesain dan membangun topologi jaringan sesuai kebutuhan membangun Video Conference WebRTC. Mengukur dan membandingkan hasil kinerja Video Conference WebRTC dengan yang didapat melakukan mekanisme *congestion control* pada WebRTC dan WebRTC dengan *background traffic* UDP.

1.4 Organisasi Tulisan

Bagian selanjutnya pada penelitian ini adalah bagian 2 yang membahas studi terkait dengan penelitian yang telah dilakukan, bagian 3 membahas perancangan sistem, bagian 4 membahas hasil pengujian, dan bagian 5 membahas kesimpulan dari penelitian ini dan saran untuk penelitian selanjutnya. Pada sub-bagian ini dituliskan bagian-bagian selanjutnya (setelah Pendahuluan) pada jurnal TA ini, disertai penjelasan sangat singkat.

2. Studi Terkait

Penelitian yang dilakukan oleh Agnieszka Chodorek,Robert R. Chodorek dan Krzysztof Wajda [17] pada tahun 2020 dengan judul *An Analysis of Sender-driven WebRTC Congestion Control Coexisting with QoS Assurance Applied in IEEE 802.11 Wireless LAN* bahwa pada penelitian ini, melakukan perbandingan vidio yang ditransmisikan melalui WebRtc dibandingkan dengan yang diperloleh melalui UDP dengan tools iPerf

Penelitian yang dilakukan oleh Robert R. Chodorek dan Agnieszka Chodorek [18] pada tahun 2017 dengan *A comparison of QoS parameters of WebRTC video conference with conference bridge placed in private and public cloud* bahwa pada penelitian ini, memfokuskan pada analisis parameter QOS lalu lintas WebRTC, dimana pengujiannya dilakukan 3 jenis bridge in private cloud (the OpenStack cloud), in public cloud (the AWS cloud) and in local network

Penelitian yang dilakukan oleh Micael Gallego, Francisco Gortázar dan Antonia Bertolino [6] pada tahun 2019 dengan *Understanding and estimating quality of experience in WebRTC* applications bahwa pada penelitian ini, memfokuskan pada penilaian QoE aplikasi berbasis WebRTC dan kontribusinya tiga kali lipat. Pertama, disediakan analisis tentang bagaimana topologi WebRTC mempengaruhi kualitas yang dirasakan oleh pengguna

Penelitian yang dilakukan oleh Branislav Sredojev, Dragan Samardzija dan Dragan Posarac [7] pada tahun 2019 dengan *WebRTC role in real-time communication and video conferencing* bahwa pada penelitian ini, Perancangan dan mengimplementasikan mekanisme pensinyalan baru. Bagan urutan pesan yang sesuai dari perilaku komunikasi WebRTC menjelaskan aliran komunikasi antara client dan server. Aplikasi server diimplementasikan sebagai server WebSocket

Penelitian yang dilakukan oleh Branislav Sredojev, Dragan Samardzija dan Dragan Posarac [20] pada tahun 2019 dengan bahwa pada penelitian menjelaskan mekanisme bebas pluginpa dan berbasis browser murni menggunakan *bandwidth* unggahan peer to peer untuk membantu sistem streaming berbasis HTTP pada WebRTC dan karena itu membuatnya lebih mudah untuk diskalakan. Hasil eksperimenmenunjukkan bahwa framework mengurangi lebih dari tiga perempat beban server dengan melayani permintaan segmen dari peer yang berdekatan.

Pada penelitian tugas akhir ini dibangun jaringan lokal dengan menerapkan mekanisme *congestion control* pada WebRTC dan WebRTC dengan *background traffic* UDP, yang ditransmisikan menggunakan generator lalu lintas menggunakan Iperf3. UDP dipilih sebagai *background traffic* dikarenakan kecepatan transfer data yang dimiliki protokol tersebut. Hasil yang didapat dengan menerapkan mekanisme *congestion control*, dapat mempertahankan transfer data dari *background traffic* UDP dan mampu meningkatkan *throughput* WebRTC.

2.1 WebRTC

WebRTC adalah kumpulan standar, protokol, dan API yang memungkinkan audio, video, dan berbagi data berkualitas tinggi P2P yang aman antar browser. Alih-alih mengandalkan plug-in pihak ketiga atau perangkat lunak berpemilik, WebRTC mengubah komunikasi waktu nyata menjadi fitur standar yang dapat dimanfaatkan oleh aplikasi web apapun melalui API JavaScript sederhana, yaitu getUserMedia (yang memperoleh akses ke kamera, mikrofon, atau layar perangkat),
RTCPeerConnection (komunikasi data audio dan video, media encoding dan decoding, media pengiriman melalui jaringan, NAT traversal), dan RTCDataChannel (komunikasi dengan latensi rendah data aplikasi arbitrer antar browser). Untuk mengkoordinasikan sesi WebRTC, diperlukan saluran pensinyalan antara klien. Signaling adalah proses pertukaran pesan untuk mendukung media komunikasi, seperti mengontrol pesan untuk membuka atau mengakhiri sesi media, pesan kesalahan, atau data jaringan. Tidak ada protokol pensinyalan yang ditentukan

untuk WebRTC, membuatnya cocok untuk sejumlah besar kasus penggunaan di mana protokol pensinyalan aktual dipilih oleh pengembang [6]

2.2 Congestion Control

Penyediaan layanan yang ekonomis menjadi penting untuk mengalokasikan sumber daya secara optimal. Untuk menyediakan kemampuan pemrosesan dan penyimpanan, diperlukan juga cadangan *bandwidth* jaringan untuk mengaksesnya. Ini berarti bahwa beberapa jenis sumber daya, seperti kemampuan pemrosesan, *bandwidth* dan penyimpanan, perlu dialokasikan, dan jenis sumber daya individu harus dialokasikan tidak secara independen tetapi secara bersamaan[8],[9]. Jika terjadi lonjakan permintaan terhadap suatu layanan, maka akan timbul persaingan antara permintaan penggunaan sumber daya tersebut. Hal ini dapat menyebabkan terganggunya layanan. Oleh karena itu, perlu dipikirkan suatu tindakan untuk menghindari atau mengurangi kemacetan [10][11].

2.2 NAT Transversal

Untuk mendapatkan media yang mengalir dari satu browser ke browser lainnya, biasanya paket media mungkin perlu melewati firewall dan perangkat NAT (Network Address Translators). NAT adalah mekanisme pemetaan alamat pribadi ke alamat publik yang mengubah informasi alamat dalam paket IP saat sedang transit melalui perangkat untuk perutean. NAT dapat diimplementasikan dengan router Wi-Fi rumah[14].

2.3 Realisasi WebRTC

- Signalling
  Dalam solusi pensinyalan, membuat kontrol. Saat start, Peer1 akan mengirimkan “inisialisasi” ke server, karena perlu mengetahui apakah itu adalah inisiator sesi. Jika demikian, itu akan mengkonfigurasi media pengguna setelah mendapatkan pesan kontrol "mendapatkan media pengguna" dari server. Sekarang Peer1 dalam fase menunggu sampai Peer2 muncul. Setelah aktivasi, Peer2 mengirimkansetara pesan pensinyalan yang, dan ketika mengetahui bahwa itu bukan inisiator, pertama-tama akan mendapatkan "peerChannel" dan kemudian "mendapatkanpengguna media". Setelah itu, kedua belah pihak dapat mulai membuat koneksi peert-peer
- Server Application
  Aplikasi server diimplementasikan sebagaiWebSocket yang serverditulis dalam Node dan dapat digunakan oleh klien atau aplikasi server mana pun. Node.js adalah open source, lingkungan runtime lintas platform untuk sisi server dan aplikasi jaringan. Aplikasi Node.js ditulis dalam JavaScript WebSocket menangani empat jenis pesan kontrol ("inisialisasi", "inisiator", "mendapatkanpengguna media" dan "peerChannel"), dua jenismedia pesan informasi(penawaran SDP dan jawaban SDP) dan satu jenis informasi jaringan pesan (ICE kandidat). Juga mengatur jumlah klien, memutuskan apakah klien adalah inisiator atau harus dihapus ketika sesi selesai.
- Client application
  Aplikasi klien diimplementasikan menggunakan WebRTC API. Peer1 menjalankan metode RTCPeerConnection createOffer(). Setelah itu, ia menetapkan deskripsi lokal menggunakan metode setLocalDescription() dan kemudian mengirimkan deskripsi sesi ini ke Peer2 melalui server pensinyalan. Kemudian, Peer2 menyetel deskripsi yang dikirim Peer1 sebagai deskripsi jarak jauh menggunakan metode setRemoteDescription(). Peer2 menjalankan metode RTCPeerConnection createAnswer() . Di createAnswer(), ia menetapkan deskripsi lokal dan mengirimkannya ke Peer1. Ketika Peer1 mendapatkan deskripsi sesi Peer2, itu menetapkannya sebagai deskripsi jarak jauh dengan metode setRemoteDescription()[14].

\

2.4 API WebRTC

Diterbitkan sebagai teknologi open source oleh Google pada Mei 2011 dan mencakup komponen dasar untuk komunikasi waktu nyata di web. Dengan komponen-komponen tersebut, yang dapat diakses melalui JavaScript API (Application Programming Interface), pengembang memiliki kesempatan untuk membuat aplikasi web yang mengesankan. Google, Mozilla dan Opera mendukung WebRTC dan mereka akan berpartisipasi dalam pengembangan lebih lanjut dari teknologi ini. Saat ini, masih ada platform dan browser yang mendukung sebagian atau tidak mendukung WebRTC[12] . Komponen utama dari API WebRTC adalah:

Komponen utama dari API WebRTC adalah:

- MediaStream - memungkinkan browser web mengakses kamera dan mikrofon;
- RTCPeerConnection - mengatur panggilan audio atau video;
- RTCDataChannel - memungkinkan browser mengirim data melalui koneksi peer-to-peer[14].

2.5 WebSocket

WebSocket adalah fiture yang relatif baru untuk browser, dan distandarisasi pada tahun 2011. Hal terpenting tentang WebSocket adalah membuka sesi dari peer to server, terbuka sampai sesi ditutup. Pesan dapat berupa teks atau biner. Bagan urutan pesan komunikasi WebRTC menjelaskan aliran komunikasi antara peer to server[12].

Protokol WebSocket memiliki dua bagian. handshake terdiri dari pesan dari klien dan tanggapan handshake dari server. Bagian kedua adalah transfer data. Implementasi WebSocket API Jetty sepenuhnya terintegrasi ke dalam server HTTP Jetty dan kontainer servlet (lihat http://jetty.codehaus.org/jetty). Jadi, servlet Jetty dapat memproses dan menerima permintaan untuk memutakhirkan koneksi HTTP ke koneksi WebSocket. Rincian lebih lanjut tentang proses komunikasi WebSocket tersedia di pekerjaan kami sebelumnya[13].

2.6 Node.js

Node.js - juga disebut Node - adalah contoh JavaScript di sisi server (lihat http: // nodejs.org). Ini didasarkan pada implementasi runtime Google - mesin "V8". V8 dan Node sebagian besar diimplementasikan dalam C dan C ++, dengan fokus pada kinerja dan konsumsi memori yang rendah. Namun, sementara V8 mendukung sebagian besar JavaScript di browser (terutama Google Chrome), Node bertujuan untuk mendukung proses server yang berjalan lama[15].

Proses Node tidak bergantung pada multithreading untuk mendukung eksekusi logika bisnis secara bersamaan; itu didasarkan pada model eventing I / O asynchronous. Bayangkan proses server Node sebagai utas tunggal yang menyematkan mesin JavaScript untuk mendukung penyesuaian. Ini berbeda dari kebanyakan sistem eventing untuk bahasa pemrograman lain, yang datang dalam bentuk pustaka: Node mendukung model eventing di tingkat bahasa[15].

2.7 Iperf

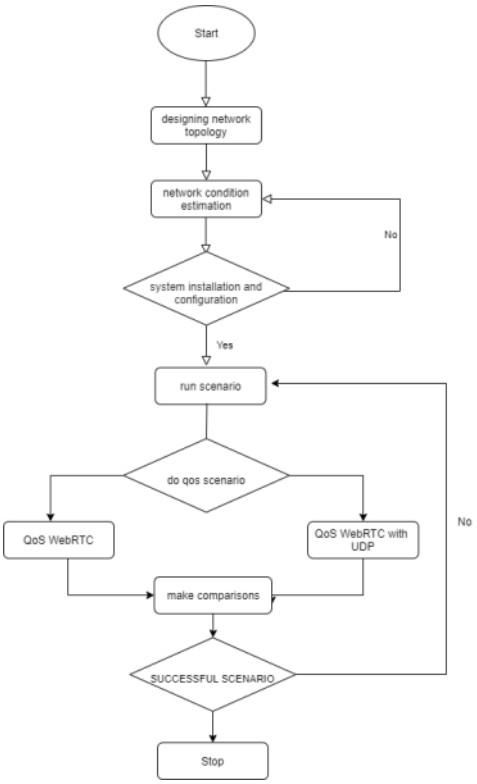
Iperf - Ini adalah alat yang digunakan untuk mengukur *bandwidth* TCP maksimum, memungkinkan penyetelan berbagai parameter dan karakteristik UDP. Metrik Iperf dalam TCP adalah: transfer waktu dan kecepatan transfer sedangkan metrik UDP adalah: transfer waktu, kecepatan transfer, *bandwidth*, jitter dan packet loss [16].

2.8 UDP

UDP adalah protokol transfer sederhana, dan juga merupakan protokol tanpa koneksi yang tidak dapat diandalkan. Dalam transmisi informasi data, selama saling mengetahui alamat IP dan nomor port UDP, transfer data dapat dilakukan, sehingga efisiensi UDP sangat tinggi. Dalam proses komunikasi jaringan P2P, teknologi holing punching UDP dapat membuat pihak yang berkomunikasi menetapkan proses sesi UDP antara titik asal dan tujuan dengan bantuan server perantara untuk mencapai transmisi sumber daya informasi. Ide utamanya adalah: dalam proses transfer informasi, server perantara diperlukan untuk berkomunikasi dengan klien; pada saat yang sama, alamat titik akhir jaringan publik UDP harus disimpan sebelumnya di server untuk memelihara klien. Ketika komunikasi antara dua pihak diperlukan, pihak mana pun dapat memperoleh alamat pihak lain melalui server perantara untuk membuat koneksi session[22].

3. Perancangan Sistem

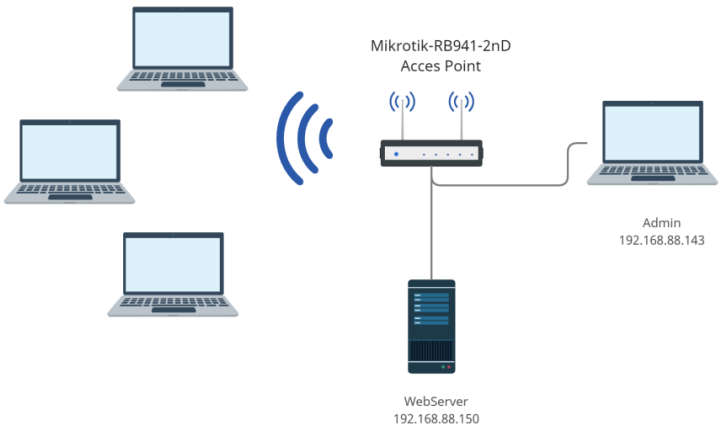
Pada penelitian ini akan merancang WebRTC dan WebRTC dengan *background traffic* UDP menggunakan tools iperf, Berikut merupakan alur sistem yang akan dibangun.



Gambar 1. Application Flow Overview

3.1 Topologi Jaringan

Topologi jaringan yang akan diimplementasikan sebagai berikut:



Gambar 2. Network Topology

Pada topologi jaringan lokal tersebut terdapat beberapa bagian yaitu admin untuk meremote dan melihat *traffic* pada jaringan dan client akan mengakses web server melalui jaringan wifi yang terkoneksi

3.2 Konfigurasi Sistem

Pada tahap implementasi pada topologi jaringan pada penelitian ini, menggunakan dua jenis perangkat, yaitu perangkat keras dan perangkat keras, berikut merupakan perangkat perangkat yang akan digunakan dalam implementasi tugas akhir ini sebagai berikut :

Tabel 1. The use of hardware in this research

1	1 buah Laptop sebagai Web Server, Operating System : Windows 10 64-bit,: Intel(R) Core(TM) i7-8750H,8 GB RAM Harddisk 1000 GB,Slot 100 Mbps / Fast Ethernet
2	1 buah Laptop Admin Remote router . Operating System : Windows 8.1 64-bit, Processor : Intel(R) Core(TM) i3-5750H. 4 GB RAM Harddisk 1000 GB. Slot 100 Mbps / Fast Ethernet
3	1 buah wireless router Mikrotik RB941-2Nd-TC 4 LAN port 1000 Mbps
4	Laptop Pendukung Sebagai Client

Tabel 2. The use of software in this research

1	WINBOX	Untuk konfigurasi router Mikrotik
2	Visual Studio Code (Node.js)	Untuk membangun WebServer untuk WebRtc
3	Google Chrome	Google Chrome merupakan Web Browser yang mendungkung WebRTC
4	Wireshark	Wireshark berfungsi untuk menganalisis dan melihat lalu lintas dan paket paket pada interface jaringan
5	Iperf3	Sebagai Uji Bandwith pada jaringan

3.2 Membangun Koneksi

Untuk dapat menghubungkan web server ke client pada jaringan lokal, adapun langkah tahapan pada penelitian ini sebagai berikut:

Tabel 3. Connected ports

Ethernet	Koneksi
Ether 1	WEB SERVER
Ether 2	Remote

Pada router yang digunakan memiliki, ether 1 dan 2 digunakan sebagai jaringan lokal, untuk client bisa mengakses web server yaitu dengan koneksi titik akses melalui *WiFi*

3.3 Koneksi *Client*

Ketika client saling terhubung, client akan mengakses IP public server untuk bisa terhubung ke web, client melakukan pengiriman data berupa websocket, lalu pada dilakukan pengecekan websocket disisi server, setelah terkoneksi lalu WebRTC membuat komunikasi UDP menggunakan STUN Server, sehingga koneksi client-server terhubung, dan TRUN Server sebagai media penerus data.

3.4 Pengiriman dan Penerimaan Data

Ketika client terhubung dengan server, terjadi pengiriman dan penerimaan data baik antar client maupun server, Data yang dikirim berupa audio dan video hasil capture dari perangkat multimedia

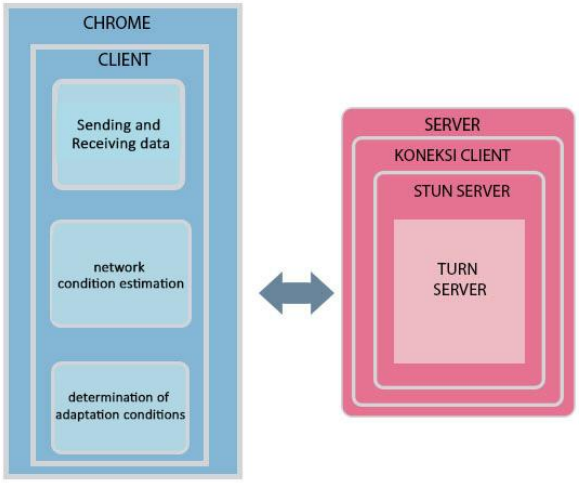
3.5 Estimasi Kondisi Jaringan

Ketika client terhubung ke server, server melakukan estimasi kondisi jaringan dari pada client Estimasi dilakukan dengan menggunakan API dari WebRTC.

3.6 Penentuan Kondisi Adaptasi

Penentuan kondisi untuk mengukur QoS(*delay, jitter, throughput,* dan *packet loss*). Adaptasi dilakukan dengan 3 tahapan *No Resource Allocation,Resource Allocation: the QoS Assurance State*,*Resource Allocation*: the *Congestion State*

3.7 Arsitektur Sistem Aplikasi



Gambar 3. Architecture Application System

Arsitektur pada WebRTC yang dibangun terdiri dari modul antarmuka client (membantu koneksi ke server) dan modul server, setelah client terhubung ke server dilakukan estimasi kondisi jaringan agar pengguna saling terhubung, setelah itu dapat dilakukan penentuan kondisi adaptasi

Pada modul server yang berisi Node.js, digunakan untuk menjalankan modul client sebagai penghubung JavaScript, untuk menjalankan modul server (WebRTC server to client), yaitu STUN Server yang menjalankan fitur TURN Server untuk melakukan penghubungan antara client

4. Perancangan Sistem

Pada bab ini akan dibahas tentang bagaimana hasil pengujian implementasi, serta menganalisis performansinya. Analisis yang dilakukan meliputi parameter QoS nya seperti *delay*, *jitter*, *throughput* dan *packet loss*. Dari analisis ini akan diketahui bagaimana performansi layanan WebRTC pada server aplikasi yang telah dibuat menggunakan server Node.js pada jaringan lokal.

4.1 Pengujian Quality

4.1.1 Tujuan Pengujian

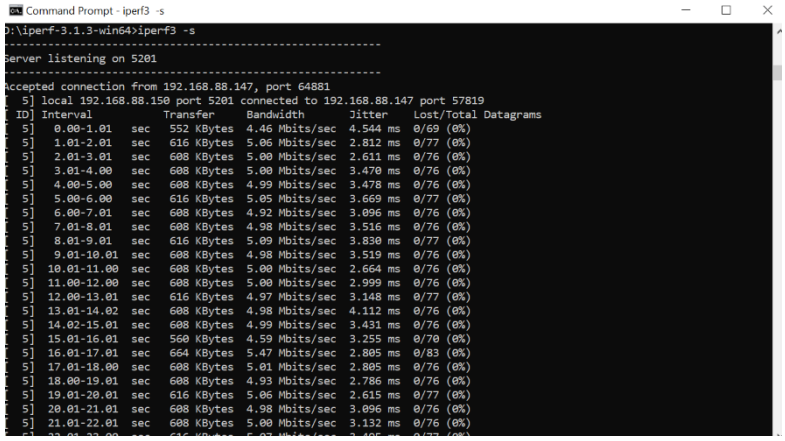
Pengujian QoS WebRTC untuk tugas akhir ini mengukur kualitas layanan WebRTC yang telah dibangun menggunakan Node.js sebagai server aplikasi nya. Parameter pengujian QoS meliputi *throughput, packet loss, delay* dan *jitter*. Untuk mendapat kan nilai dari parameter tersebut digunakan wireshark sebagai analisa paket yang ditransmisikan pada WebRTC yang terdiri dari *foreground traffic* (dihasilkan dari kamera webcam yang menangkap video) dan *background traffic* (menggunakan Iperf3 sebagai generator *traffic* UDP, sebagai *traffic* latar belakang webrtc), maka dari itu perbandingan pada setiap *throughput* nya adalah hasil dari *congestion control atau packet loss*, dengan mengikuti acuan QoS sesuai standar ITU-T G.114.

4.1.2 Sistematik Pengujian

Selama proses pengujian *traffic* jaringan menggunakan router mikrotik rb941-2nd-tc (Wireless 802.1) dengan kecepatan maksimal 32 Mbps dan menggunakan Frekuensi 2.4 Hz sebagai acces point. Jumlah aliran video akan bertambah selama pengujian sampai titik akses penuh yaitu diuji sampai Dengan 7 aliran video yang dikirim secara bersamaan. Pada aplikasi ini menggunakan dua transmisi *traffic*, yaitu sebagai berikut :

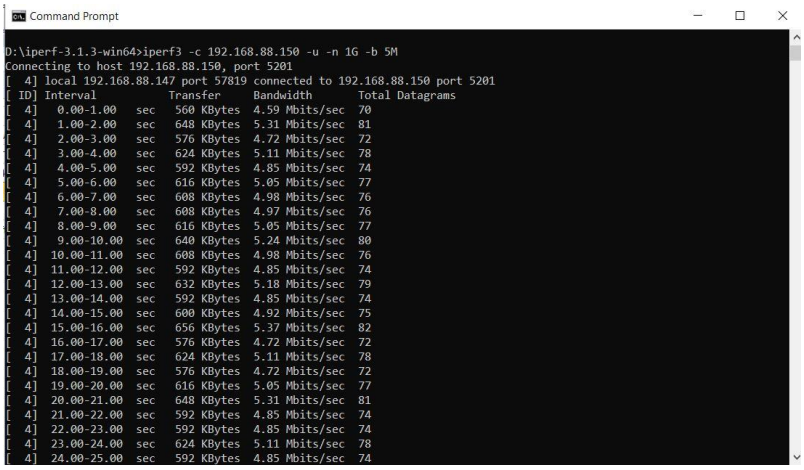
1. WebRTC dengan *Background Traffic* UDP

Pengujian performansi background *traffic* dengan mentransmisikan paket UDP menggunakan iperf3 yang digunakan sebagai *traffic* generator yang ditransmisikan melalui titik akses, dengan pengujian mengirim 1Gb data dengan kecepatan 5 Mbps, Pengujian Ini bertujuan untuk melihat apakah



WebRTC dapat mempertahankan transmisi nya setela dikirim *background traffic* tertentu dan melihat kinerja protokol UDP'

Gambar 4. UDP Syntax Output On Server

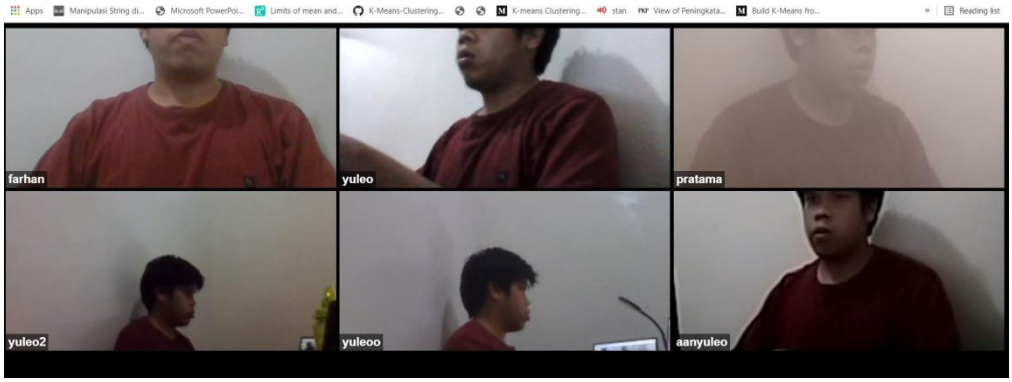


Gambar 5. UDP Syntax Output On Client

Pada Gambar 4 & 5 menunjukan hasil output dari sintaks yang telah dijalankan dengan konfigurasi pada server " Iperf3 –s " perintah ini menjalankan iperf di mode server dengan default port 5201 yang berfungsi sebagai kontrol channel lalu di sisi server dengan konfigurasi "iperf3 –c IP –u –n 1G –b 5M" perintah ini berfungsi menyambungkan client ke IP server dengan menggunakan protokol UDP lalu mengirimkan 1Gb data dengan kecepatan 5Mbps

2. WebRTC

Pengujian performansi *foreground traffic* yang dihasilkan langsung dari kamera WebRTC



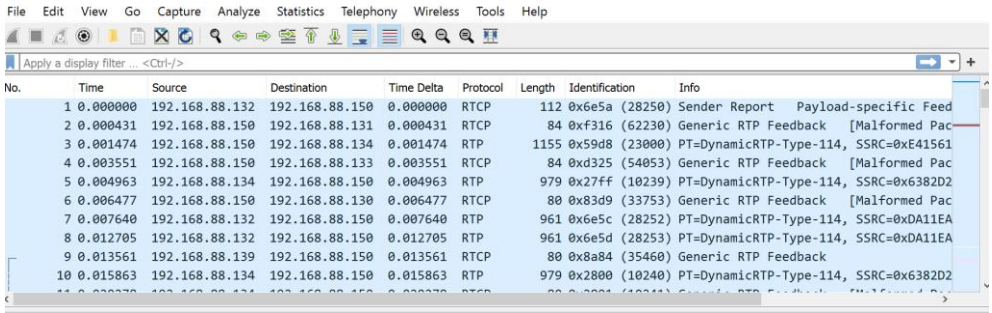
Gambar 6. WebRTC testing process on local network

Pada Gambar 6 merupakan contoh proses testing pada WebRTC pada jaringan lokal, percobaan dilakukan hingga 8 aliran video secara bersamaan dengan menggunakan kamera bawaan dari laptop.

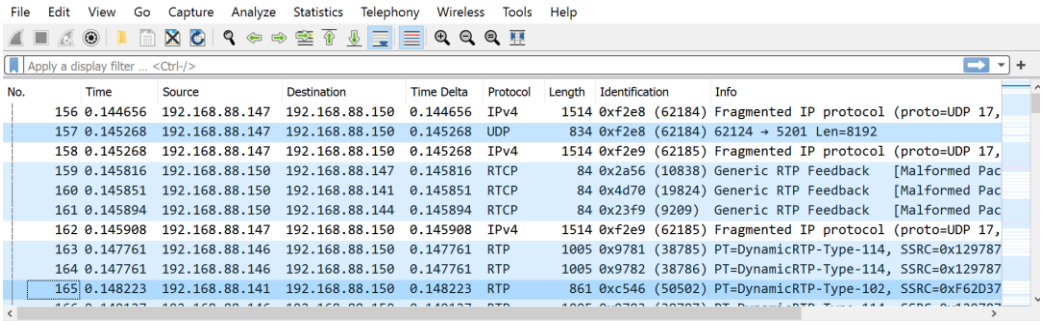
4.2 Hasil Pengujian

Pengujian yang dilakukan untuk pada Wireless 802.11 dengan melakukan perbandingan dalam *throughput* pada saat WebRTC dalam kondisi macet, yaitu hasil yang didapat berupa *Congestion Control* dan *packet loss*. Pengujian ini dilakukan menggunakan wireshark sebagai analisa paket yang tertangkap pada jaringan sampai dengan 10 aliran video secara bersamaan saat mentransmisikan paket dengan menggunakan standar acuan sebagai berikut :

1. Packet Loss
   Standar ITU-T G.1010 adalah di bawah 3%.
2. Delay
   Standar ITU-T G.114 adalah kurang dari 150ms
3. Jitter
   Standar ITU-T G.144 adalah kurang dari 50ms



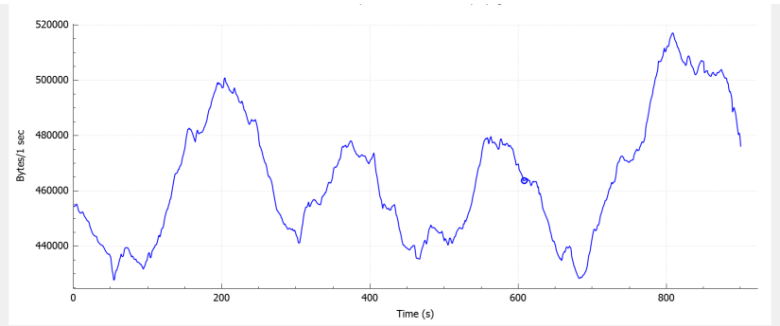
Gambar 7. Wireshark WebRTC Display



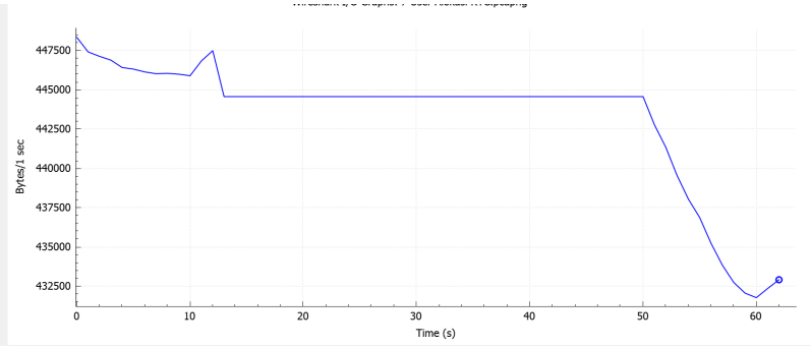
Gambar 8. Wireshark WebRTC Display with UDP Background Traffic

Protokol *transport real-time*. RTP menyediakan fungsi transportasi jaringan end-to-end yang cocok untuk aplikasi yang mentransmisikan data secara *real time* seperti data audio, video atau simulasi, melalui layanan jaringan *multicast* atau *unicast*. [21].

Pada Gambar 7 & Gambar 8 terlihat protokol apa saja yang difilter oleh Wireshark, dengan menggunakan fitur wireshark, protokol yang terenkripsi dapat dilihat. Sehingga protokol yang digunakan dalam pengujian ini adalah RTP dan UDP dengan port iperf 5201, selanjut nya menganalisa paket dari protokol tersebut seperti *throughput, packet loss, delay* dan *jitter.*



Gambar 9. WebRTC Throughput In 15 Minutes



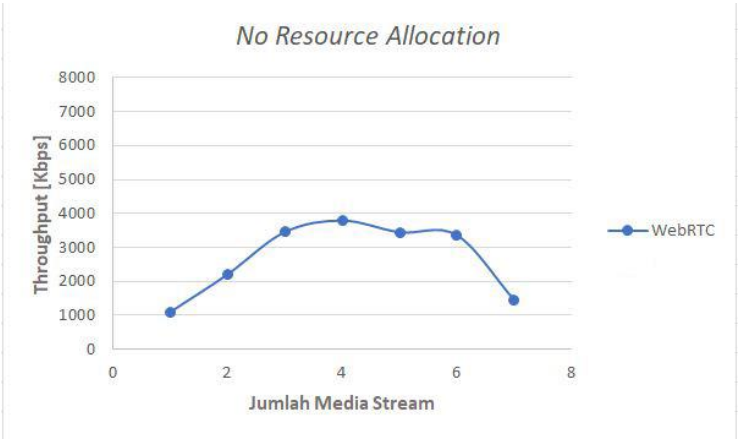
Gambar 10. WebRTC Throughput In 1 Minute

Pada Gambar 9 & 10 merupakan grafik yang didapat dengan melakukan pengujian WebRTC selama 1 menit dan 15 menit, nilai rata rata *throughput* WebRTC yang didapat sama yaitu 3560 Kbps, sehingga dalam pengujian yang dilakukan menggunakan *sample* waktu 1 menit setiap jumlah media stream nya.

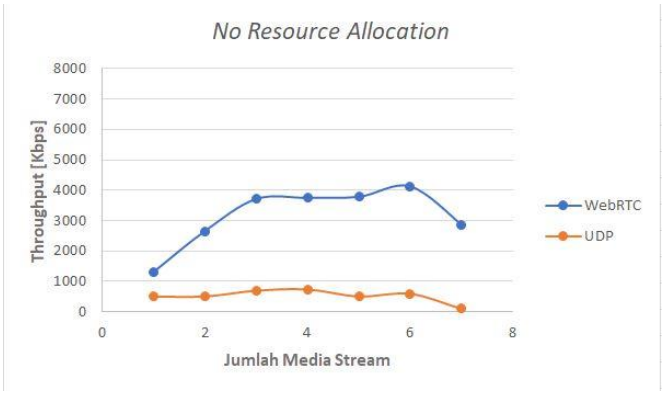
4.1.1 Hasil Pengujian Tanpa Alokasi Bandwidth

1. Throughput

   Pada pengujian ini bertujuan untuk mengetahui jumlah *throughput* pada transmisi WebRTC dan WebRTC dengan *background traffic* UDP tanpa alokasi *bandwidth*, yang artinya titik akses tidak menjamin QoS untuk perangkat yang melakukan panggilan video.



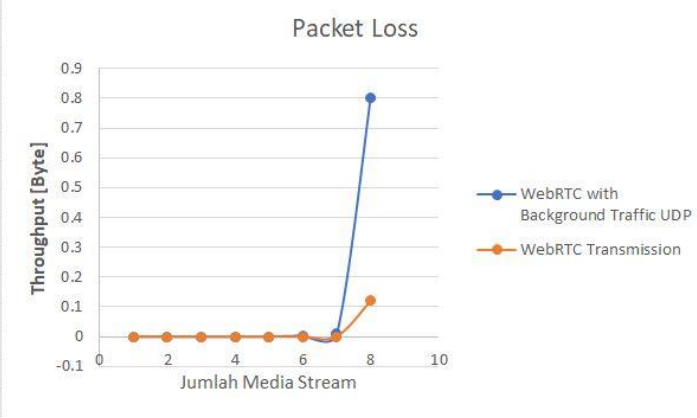
Gambar 11. Throughput Test Results Without Bandwidth Allocation on WebRTC



Gambar 12. Throughput Test Results Without Bandwidth Allocation with WebRTC with UDP

Pada Gambar 11 dan 12 dapat dilihat total *throughput* pada WebRTC dan WebRTC dengan *background traffic* UDP, ditransmisikan sampai dengan 7 aliran video secara bersamaan. Protokol UDP memiliki kecepatan transfer yang lebih cepat dalam transfer data,namun data yang dikirim tidak menjamin sampai ke tujuan. Pada saat pengiriman 5 Mbps paket UDP dan pengiriman 7 aliran video secara bersamaan (kondisi *network congestion*), terlihat transmisi UDP mengalami penurunan dari 504 Kbps menjadi 96 Kbps. Pada WebRTC juga mengalami penurunan *traffic*, namun tetap dapat mempertahankan aliran videonya, hal ini bisa terjadi karena kemampuan WebRTC mampu menggantikan paket yang hilang secara acak dan pesan dikirim langsung dan membantu mengurangi *bandwidth* pada server pensinyalan langsung menuju ke client tanpa routing proxy pada server. Pengujian berhenti saat ditransmisikan aliran hingga 8 client secara bersamaan dengan *background traffic* UDP, perangkat mengalami *disconnect* pada jaringan, dikarenakan router tidak dapat menampung *traffic* yang ditransferkan pada titik akses.

2. Packet Loss

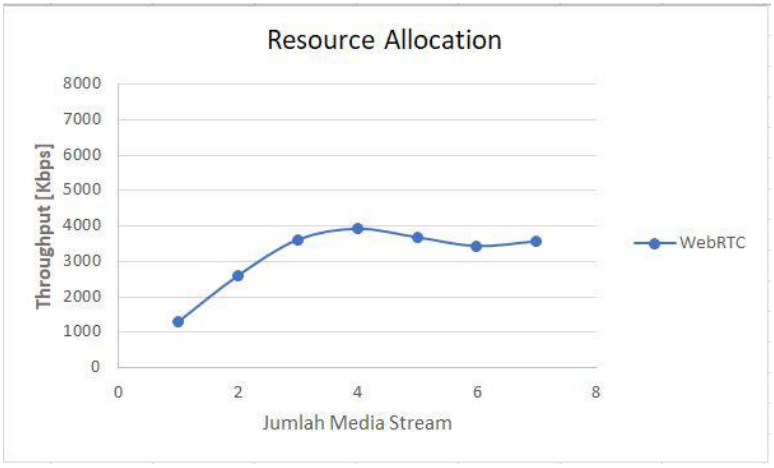


Gambar 13. Packet Loss Results Without Allocation

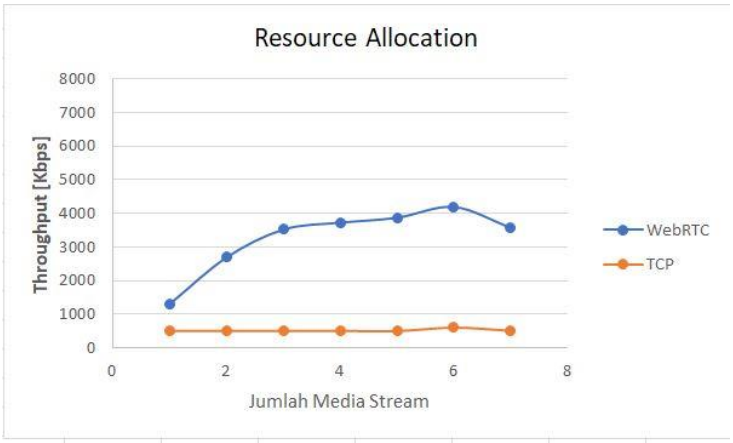
Pada Gambar 13 terdapat paket loss saat dilakukan transmisi WebRTC dan WebRTC dengan *background traffic* UDP, ini bisa terjadi karena kesalahan bit saat dilakukan pengujian. Yaitu dalam pengujian ini mengirim UDP dengan kecepatan 5 Mbps. Dari data yang didapat menunjukan terdapat 0.8% *packet loss* pada WebRTC dengan *background traffic* dan 0.15% untuk WebRTC.

4.1.2 Hasil Pengujian Dengan Alokasi Bandwidth

Pada pengujian ini bertujuan untuk mengetahui jumlah *throughput* pada transmisi WebRTC dan WebRTC dengan *background traffic* UDP, dengan alokasi *bandwidth*, yang artinya titik akses menjamin QoS untuk perangkat yang melakukan panggilan video, alokasi *bandwidth* disini menggunakan limitasi *bandwidth* pada perangkat yang terhubung, dengan melakukan alokasi yang tepat untuk jaringan ini menggunakan limitasi *bandwidth* sebesar 512 Kbps untuk kecepatan upload dan download 1024, alokasi *bandwidth* menggunakan router mikrotik dengan menggunakan fitur queue yang tersedia di mikrotik.



Gambar 14. Throughput Test Results With Bandwidth Allocation on WebRTC



Gambar 15. Throughput Test Results With WebRTC Bandwidth Allocation with UDP

Perbandingan ini dilakukan untuk mencari *bandwidth* minimal dan bandwith yang tepat untuk meningkatkan layanan Webrtc, Pada Gambar 14 dan 11 terlihat peningkatan *throughput* setelah dilakukan alokasi *bandwidth* pada WebRTC, hasil yang didapat mampu meningkatkan *throughput* WebRTC hingga 15%. Pada Gambar 15 dan 12 terjadi peningkatan sebesar 10% pada *throughput* WebRTC dan juga dapat mempertahankan protokol UDP yang sebelumnya turun hingga 96Kbps tetap mengirimkan UDP hingga 504Kbps.

4.1.3 Hasil Pengujian Dengan Alokasi (Congestion State)

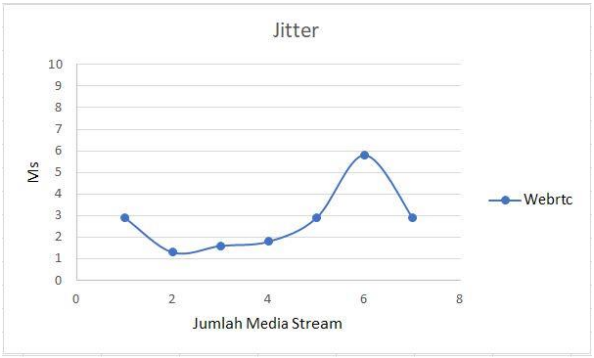
Secara default titik akses 802.11 tidak menjamin kontrol akses, jadi jika volume *traffic* yang disediakan untuk transmisi melalui titik akses tertentu melebihi kemampuan teknis titik akses, maka terjadi *traffic* padat yang menyebabkan kemacetan pada jaringan, *delay* dan *jitter* sangat berpengaruh pada aplikasi yang menggunakan video conference dan VoIP.

1. *Delay*

Gambar 16. WebRTC Time Parameters (Delay)

Tujuan pengujian ini adalah menghitung berapa lama dibutuhkan paket data agar diteruskan ke penerima waktu yang dibutuhkan paket data agar diteruskan ke penerima. Dari pengiriman sampai 7 aliran video secara bersamaan didapat rata rata *delay* sebesar 2.08 Ms menunjukan nilai yang didapat masih sesuai standar ITU-T G.114.

2. Jitter



Gambar 17. WebRTC Time Parameters (Jitter)

Tujuan pengujian ini adalah untuk mendapatkan variasi dari *delay* untuk mendapatkan deviasi rata rata *delay*. Dari pengiriman sampai 7 aliran video secara bersamaan didapat rata rata *jitter* sebesar 2.7 Ms, hasil menunjukan nilai yang didapat sesuai standar ITU-T G.114.

Tabel 4. Comparison Results of Congestion Control

Hasil Data Uji Performa WebRTC *Transmision* & *Background Traffic* 7 aliran video					
Bandwidth	Protocol	WebRtc Transmision		WebRTC dengan *Background Traffic* UDP	
		Throughput	Packet Loss	*Throughput*	Packet Loss
No Allocation	UDP	-	0,12%	504 Kbps	0,8 %
	WebRTC	1452 Kbps		7629 Byte	
Allocation	UDP	-	0%	4001 Byte	0,12 %
	WebRTC	3576 Kbps		7740 Byte	

Tabel 5. Testing Results of Comparison of Allocation Tests

Hasil Data Uji Performa WebRTC *Transmision* & *Background Traffic* 7 aliran video			
Bandwidth	Protocol	WebRTC Transmision	WebRTC dengan *Background Traffic* UDP
		Throughput	*Throughput*
Allocation 258Kbps	UDP	-	488 Kbps
	WebRTC	2528 Kbps	2528 Kbps
Allocation 512 Kbps	UDP	-	488 Kbps
	WebRTC	2792 Kbps	2792 Kbps
Allocation 1024 Kbos	UDP	-	504 Kbps
	WebRTC	3048 Kbps	3576 Kbps

5. Kesimpulan

Dari hasil pengujian dan analisis QoS WebRTC dengan mekanisme *congestion control* pada jaringan lokal didapat kesimpulan sebagai berikut :

1. Pada penelitian ini berhasil dibangun jaringan lokal dengan menggunakan menggunakan Wireless 802.11 sebagai titik akses, dan menerapkan mekanisme *congestion control* pada WebRTC
2. WebRTC dapat menyesuaikan real-time transfer untuk mengirim audio dan video yang ditransmisikan meskipun titik akses tidak menjamin QoS, dikarenakan kemampuan WebRTC membantu mengurangi *bandwidth* pensinyalan langsung menuju ke client tanpa routing proxy pada server.
3. Dengan alokasi *bandwidth*, nilai *throughput* WebRTC dapat ditingkatkan hingga 10-15%, dan dapat mempertahankan protocol UDP yang sebelumnya mengalami penurunan hingga 80%.

Saran untuk penelitian selanjutnya Agar kenyamanan pengguna saat melakukan video conference lebih lagi disarankan agar :

1. Alokasi *bandwidth* diatur sebaik mungkin. Cara lain melakukan alokasi *bandwidth* dapat dilakukan seperti menggunakan Grouping(Saat ada bandwith tersedia dapat ke pengguna lain), Burst(selama tidak menggunakan *bandwidth* tertentu terus menerus dapat naik diatas limit nya), Priority (Memprioritaskan pengguna tertentu).
2. Lalu Diperlukan bandwith yang lebih besar dan router yang memadai agar performance (*throughput,packet loss, delay dan jtter*), pada server agar meningkatkan kualitas pengguna dalam melakukan video conference dengan lebih baik.

REFERENSI

[1] IETF, "rtcweb" .[Online]. Available: https://datatracker.ietf.org/wg/rtcweb/charter/ Accessed March 29, 2019.

[2] Atwah, R., Iqbal, R., Shirmohammadi, S., & Javadtalab, A. (2015). A Dynamic Alpha Congestion Controller for WebRTC. 2015 IEEE International Symposium on Multimedia (ISM). doi:10.1109/ism.2015.63

[3] Haensge, K., & Maruschke, M. (2015. 2015 18th International Conference on Intelligence in Next Generation Networks. doi:10.1109/icin.2015.7073800.

[4] Ayadi, I., Diaz, G., & Simoni, N. (2013). QoS-based network virtualization to future networks: An approach based on network constraints. 2013 Fourth International Conference on the Network of the Future (NoF). doi:10.1109/nof.2013.6724515.

[5] Barayuga, V. J. D., & Yu, W. E. S. (2014). Study of Packet Level UDP Performance of NAT44, NAT64 and IPv6 Using Iperf in the Context of IPv6 Migration. 2014 International Conference on IT Convergence and Security (ICITCS). doi:10.1109/icitcs.2014.7021814.

[6] García, B., Gallego, M., Gortázar, F., & Bertolino, A. (2018). Understanding and estimating quality of experience in WebRTC applications. Computing. doi:10.1007/s00607-018-0669-7

[7] Suciu, G., Stefanescu, S., Beceanu, C., & Ceaparu, M. (2020). WebRTC role in real-time communication and video conferencing. 2020 Global Internet of Things Summit (GIoTS). doi:10.1109/giots49054.2020.91196

[8] S.Kuribayashi, "Optimal Joint Multiple Resource Allocation Method for Cloud Computing Environments," International Journal of Research and Reviews in Computer Science (IJRRCS), Vol. 2, No.1, pp.1-8, Feb. 2011.

[9] S.Tsumura and S.Kuribayashi: "Simultaneous allocation of multiple resources for computer communications networks," In Proceeding of 12th Asia-Pacific Conference on Communications (APCC2006), 2F-4, Aug. 2006

[10] M.Gusat, R.Birke and C.Minkenberg, "Delay-based cloud congestion control," In Proceeding of GLOBECOM'2009, Nov. 2009.

[11] B.Raghavan, K.Vishwanath, S.Ramabhadran, K.Yocum and A.C.Snoeren, "Cloud control with distributed rate limiting," In Proceeding of SIGCOMM'07, Aug. 2007.

[12] A. Sergiienko, Sergiienko, "WebRTC Blueprints, Develop your very b own media applications and services using WebRTC," in Birmingham, Packt Publishing, ISBN 978-1-78398-310-0, May 2014.A. Zeidan, A. Lehmann, and U. Trick, "WebRTC enabled multimedia conferencing and collaboration solution," WTC 2014, in Berlin, Germany, Print ISBN 978-3-8007-3602-7, June 2014.

[13] Pimentel, V., & Nickerson, B. G. (2012). Communicating and Displaying Real-Time Data with WebSocket. IEEE Internet Computing, 16(4), 45–53. doi:10.1109/mic.2012.64

[14] Sredojev, B., Samardzija, D., & Posarac, D. (2015). WebRTC technology overview and signaling solution design and implementation. 2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO). doi:10.1109/mipro.2015.7160422

[15] Tilkov, S., & Vinoski, S. (2010). Node.js: Using JavaScript to Build High-Performance Network Programs. IEEE Internet Computing, 14(6), 80–83. doi:10.1109/mic.2010.145

[16] IPERF, "Networking with iperf" .[Online]. Available: http://openmaniak.com/iperf.php. Accessed December 29, 2013

[17] Chodorek, A., Chodorek, R. R., & Wajda, K. (2019). An Analysis of Sender-driven WebRTC Congestion Control Coexisting with QoS Assurance Applied in IEEE 802.11 Wireless LAN. 2019 International Conference on Software, Telecommunications and Computer Networks (SoftCOM). doi:10.23919/softcom.2019.8903749

[18] Chodorek, R. R., Chodorek, A., Rzym, G., & Wajda, K. (2017). A Comparison of QoS Parameters of WebRTC Videoconference with Conference Bridge Placed in Private and Public Cloud. 2017 IEEE 26th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises

[19] Alimudin, A., & Muhammad, A. F. (2018). Online Video Conference System Using WebRTC Technology for Distance Learning Support. 2018 International Electronics Symposium on Knowledge Creation and Intelligent Computing (IES-KCIC). doi:10.1109/kcic.2018.8628568

[20] Varma, M., Yarnagula, H. K., & Tamarapalli, V. (2017). WebRTC-based peer assisted framework for HTTP live streaming. 2017 9th International Conference on Communication Systems and Networks (COMSNETS). doi:10.1109/comsnets.2017.7945420

[21] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A transport protocol for real-time applications," RFC 3550, 2003.

[22] Zhang Yamei, & Cai Pengfei. (2010). Research on using UDP to traverse NAT under P2P network environment. 2010 3rd International Conference on Advanced Computer Theory and Engineering(ICACTE). doi:10.1109/icacte.2010.5579841