A COMPARATIVE STUDY OF WEBRTC AND WEBSOCKET PERFORMANCE IN REAL-TIME VOICE COMMUNICATION

PERBANDINGAN PERFORMA WEBRTC DAN WEBSOCKET UNTUK INTERAKSI SUARA REAL-TIME

Alwin Hartono Limaran[1], Agung Wicaksono[2], Patah Herwanto[3]
[1,2,3]Informatics Engineering, Faculty of Information Technology, STMIK Indonesia Mandiri
Jl. Belitung No.7, Merdeka, Kec. Sumur Bandung, Kota Bandung, Jawa Barat 40113
alwin.hartono@protonmail.com[1], agung.wicaksono.362389010@gmail.com[2], pherwanto@stmik-im.ac.id[3]

Abstract *- This study compares the performance of Web Real-Time Communication (WebRTC) and WebSocket for bidirectional voice interaction with the ChatGPT API on an Orange Pi 3B edge device over a 50 Mbps Ethernet connection. The objective is to determine the most efficient and reliable protocol for real-time interaction on resource-constrained devices. The study evaluates latency, bandwidth, CPU utilization, and jitter metrics automatically collected to assess the efficiency and stability of both protocols. Results show that WebRTC achieves approximately 150 ms latency with lower network and CPU consumption and near-zero jitter, while WebSocket exhibits around 200 ms latency with higher bandwidth usage and CPU load. Practically, WebRTC is more efficient and stable for client-side production due to its resource-efficient and responsive characteristics, whereas WebSocket is better suited for custom data streaming or high-throughput server-to-server communication. These findings provide a quantitative foundation for developers in selecting the optimal protocol for real-time voice applications in edge environments.*

Keywords *- WebRTC, WebSocket, Real-Time Voice, Openai API, voice to voice.*

Abstrak - Penelitian ini membandingkan performa Web Real-Time Communication (WebRTC) dan WebSocket untuk komunikasi suara dua arah dengan ChatGPT API pada perangkat edge Orange Pi 3B melalui koneksi Ethernet 50 Mbps. Tujuannya menentukan protokol paling efisien dan andal untuk interaksi *real-time* di perangkat berdaya terbatas. Studi ini mengevaluasi metrik latensi, bandwidth, CPU, dan jitter yang dikumpulkan otomatis untuk menilai efisiensi dan kestabilan kedua protokol. Hasil menunjukkan WebRTC memiliki latensi sekitar 150 ms dengan konsumsi jaringan dan CPU lebih rendah serta jitter nyaris nol, sedangkan WebSocket menunjukkan latensi sekitar 200 ms dengan bandwidth dan beban CPU lebih tinggi. Secara praktis, WebRTC lebih efisien dan stabil untuk produksi di sisi klien karena karakteristiknya yang hemat sumber daya dan responsif, sementara WebSocket lebih sesuai untuk skenario aliran data kustom atau komunikasi antarserver ber-throughput tinggi. Temuan ini memberikan dasar kuantitatif bagi pengembang dalam memilih protokol optimal untuk aplikasi suara *real-time* di lingkungan edge.

Kata Kunci - WebRTC, WebSocket, Suara Real-Time, Openai API, suara ke suara

I. PENDAHULUAN

Percakapan suara *real-time* menuntut latensi rendah, efisiensi bandwidth, dan kestabilan koneksi agar pengalaman pengguna (Quality of Experience/QoE) tetap natural[1], [2],[3]. Dalam konteks ini, terdapat dua pendekatan arsitektural yang umum digunakan: WebRTC, yang sejak awal dirancang untuk media *real-time* dengan dukungan jitter buffer, congestion control, dan acoustic echo cancellation; serta WebSocket, kanal TCP full-duplex untuk data interaktif yang tidak memiliki optimasi media setingkat WebRTC [4], [5]. Secara praktik, WebRTC lebih sesuai untuk aplikasi klien karena tumpukan medianya secara native menangani encoding/decoding, noise suppression, pengendalian jitter/kongesti, dan NAT traversal. WebSocket [6],[7] tetap relevan untuk skenario

server-to-server atau ketika logika aplikasi berada di sisi server, namun memerlukan pengelolaan manual terhadap buffering, backpressure, *heartbeat*, ukuran bingkai, dan reconnect, yang meningkatkan risiko instabilitas bila tidak direkayasa dengan cermat.

Meskipun riset tentang WebRTC (telekonferensi/VoIP)[8],[9] dan WebSocket[10] sudah luas, hampir tidak ada evaluasi *head-to-head* pada agen suara berbasis model bahasa di perangkat edge yang menggunakan telemetri otomatis per detik dan pencatatan insiden konektivitas seperti *timeout, reconnect,* dan *close code* secara terstruktur. Kesenjangan ini penting karena kondisi edge menambah kendala daya komputasi dan jaringan yang tidak tercermin pada studi berbasis desktop atau cloud. Studi ini menutup celah tersebut melalui eksperimen terkontrol di perangkat *edge* dengan kondisi uji identik untuk WebRTC dan WebSocket, logging per detik, serta analisis metrik kuantitatif yang dapat direplikasi. Tujuan penelitian adalah mengidentifikasi protokol yang paling efisien dan andal untuk percakapan suara dua arah berkelanjutan pada sumber daya terbatas; hipotesis kerja menyatakan bahwa WebRTC akan unggul pada latensi, efisiensi bandwidth, dan kestabilan koneksi berkat mekanisme media *real-time* terintegrasi, sedangkan WebSocket cenderung menawarkan throughput lebih tinggi dengan konsekuensi beban sumber daya dan variabilitas yang lebih besar.

II. SIGNIFIKANSI STUDI

Pemilihan WebRTC atau WebSocket merupakan keputusan arsitektural yang memengaruhi latensi *end-to-end*, efisiensi bandwidth, ketahanan koneksi, model keamanan kunci API (alur token *ephemeral* di klien versus kunci jangka panjang di server), serta beban rekayasa untuk pengelolaan media *real-time*. Perkembangan platform *real-time* saat ini mendorong penggunaan WebRTC secara native di sisi klien, sementara WebSocket tetap relevan utamanya untuk komunikasi server-to-server dan pengembangan prototipe. Literatur mengenai WebRTC (telekonferensi/VoIP) dan WebSocket (aplikasi web interaktif) memang luas, namun evaluasi *head-to-head* pada agen suara berbasis model bahasa besar di perangkat edge berdaya rendah dengan pencatatan metrik otomatis per detik dan log insiden konektivitas masih jarang. Kekosongan ini menyulitkan pengambil keputusan ketika menskalakan agen suara ke lingkungan nyata dengan keterbatasan sumber daya jaringan dan komputasi. Studi ini menyajikan perbandingan kuantitatif latensi, *jitter*, *throughput*, dan penggunaan CPU pada perangkat hemat daya (Orange Pi 3B); mengevaluasi kestabilan konektivitas melalui pencatatan kejadian putus-sambung, *timeout heartbeat* (ping/pong), serta kode penutupan dan dampaknya terhadap kelancaran percakapan; dan menyediakan metodologi replikasi dengan metrik yang dihasilkan otomatis per detik beserta prosedur pengujian, parameter jaringan, dan konfigurasi perangkat yang terdokumentasi ringkas.

A. Kontribusi Studi

Studi ini (i) menyajikan evidensi kuantitatif *head-to-head* atas latensi ujung-ke-ujung (*end to end*), *jitter*, *throughput*, dan penggunaan CPU pada perangkat hemat daya; (ii) mendokumentasikan kestabilan konektivitas termasuk putus-sambung, *timeout* ping/pong, dan kode penutupan beserta dampaknya; (iii) menyediakan metodologi replikasi dengan metrik otomatis per detik, prosedur uji, parameter jaringan, dan konfigurasi perangkat yang ringkas; serta (iv) merumuskan *best-practice* WebSocket berbasis temuan yang meliputi pengiriman audio sebagai *binary frames* dengan durasi bingkai pendek (~10–20 ms), *heartbeat* (ping/pong) periodik, *rate limiting/backpressure*, *exponential backoff* dan *circuit breaker*, serta pencatatan per detik yang dirujuk kembali pada bagian diskusi/lampiran. Implikasi praktis dibahas pada bagian Diskusi; secara ringkas, WebRTC lebih sesuai untuk klien (latensi lebih rendah, efisiensi bandwidth lebih baik, koneksi lebih stabil), sementara WebSocket tepat untuk skenario server-to-server atau prototipe cepat.

B. Studi Literatur

Percakapan suara *real-time* yang andal bergantung pada desain protokol media dan mekanisme transport [11]. WebRTC dikembangkan untuk komunikasi *real-time* di peramban (*browser*) dengan tumpukan media bawaan (SRTP [12]/DTLS [13]), *jitter buffer*, kontrol kongesti, *acoustic echo cancellation*, serta NAT traversal melalui ICE yang mengoordinasikan STUN/TURN. Di sisi akuisisi audio, API getUserMedia() dan *constraints* modern memungkinkan pengaturan peranti dan fitur seperti echoCancellation, autoGainControl, dan noiseSuppression. WebSocket, sebagai kanal TCP *full-duplex*, unggul dalam fleksibilitas pertukaran data, namun tidak menyertakan optimasi media setara WebRTC; pengembang perlu menangani *buffering*, *backpressure*, ukuran bingkai, serta *heartbeat* (ping/pong) untuk menjaga koneksi tetap hidup. Ekosistem *real-time* terkini mempercepat adopsi WebRTC untuk agen suara produksi. Layanan model bahasa modern menyediakan *Realtime API* yang dapat diakses melalui WebRTC dan WebSocket, bahkan SIP untuk telepon. Arsitektur yang umum dipakai: klien peramban terhubung via WebRTC dengan token sementara (*ephemeral*), sementara logika kerja (mis. *tool calling*, *guardrail*) dikendalikan di server melalui kanal pengendali (*server-side controls*/sideband). Dengan demikian, WebRTC memperkecil kerumitan *pipeline* audio di klien, sedangkan WebSocket tetap relevan untuk skenario server-to-server atau integrasi khusus yang membutuhkan kontrol penuh atas aliran data.

C. Metode Penelitian

1. B.1 Perangkat & Lingkungan

Pengujian pada Orange Pi 3B (Gambar 1) (ARM [14] Cortex-A55 quad-core, RAM 4 GB, SSD 256 GB) dengan Ubuntu 22.04.5 LTS. Koneksi Ethernet 50 Mbps; kondisi jaringan dikendalikan agar seragam.

Gambar 1. Orange Pi 3B

D. B.2 Implementasi

1) WebSocket

Pada jalur WebSocket (Python Command Line Interface - CLI [15]), implementasi berfungsi sebagai bridge tanpa antarmuka grafis antara mikrofon, speaker, dan layanan *voice-to-voice* ChatGPT OpenAI berbasis Application Programming Interface (API) melalui WebSocket Secure (WSS). Audio direkam dari perangkat melalui Advanced Linux Sound Architecture (ALSA) pada sistem operasi sebagai Pulse Code Modulation 16bit, dipecah menjadi chunk biner, lalu dikirim ke API; respons audio diterima sebagai delta biner dan diputar ke perangkat keluaran. Aplikasi menangani event buka/pesan/penutupan serta *heartbeat* ping/pong untuk menjaga "kesehatan" koneksi bila batas waktu terlampaui, sesi ditutup otomatis dan tercatat ke Comma Separated Values. Layanan berjalan di latar (mis. sebagai unit systemd), sehingga mudah diawasi dan diotomasi pada perangkat edge. Alasan pemilihan Python: ekosistem input/output asinkron yang matang untuk WebSocket, akses langsung ke perangkat audio dan proses OS, kemudahan pengelolaan dan koordinasi otomatis tugas latar/logging, serta fleksibilitas merakit pipeline audio kustom dalam lingkungan *headless*.

2) WebRTC

Pada jalur WebRTC dengan JavaScript/HTML di peramban, implementasi memanfaatkan media stack native getUserMedia() untuk akses mikrofon dengan constraints, Real-Time Communication Peer Connection untuk negosiasi Session Description Protocol, Interactive Connectivity

Establishment dan Session Traversal Utilities for Network Address Translation untuk melewati NAT[16], serta jalur Secure Real-time Transport Protocol untuk pengiriman audio. Server Node.js menerbitkan token sementara bagi klien, meneruskan permintaan koneksi (*offer*) dan respons (*answer*) ke layanan Realtime OpenAI, lalu mengembalikan jawaban tersebut untuk menyelesaikan proses negosiasi. Setelah itu, track audio dari lawan bicara diputar otomatis, dan saluran data digunakan untuk mengirimkan peristiwa atau informasi debug. Statistik operasional dikumpulkan melalui getStats() di sisi klien, lalu dikirim ke server untuk disimpan dalam file CSV, dengan penambahan data sistem yang diperlukan CPU, RAM, dll). Alasan pemilihan JavaScript dan menggunakan *browser* karena seluruh komponen media waktu nyata, keamanan, dan penanganan NAT tersedia secara native; distribusi klien bersifat *zero*-install; serta kontrol kualitas audio dapat dikonfigurasi langsung melalui API peramban menjadikannya ideal untuk aplikasi klien *voice-to-voice* yang menuntut respons *real-time* dan pengalaman pengguna mulus.

E. B.3 Prosedur dan Logging

Untuk menjaga kesetaraan skenario, pertanyaan yang diajukan ke model dibuat identik pada kedua protokol; namun panjang jawaban dari model bahasa dapat bervariasi sehingga durasi percakapan dan jumlah log berbeda. Pada jalur WebSocket, logging berjalan paralel dengan aliran audio: modul pencatat menghitung throughput *uplink*/*downlink* per interval dari akumulasi byte, menurunkan Round-Trip Time (RTT)[17] dari mekanisme *heartbeat* ping/pong, menghitung jitter sebagai variasi antar-interval, serta mengambil metrik dari Operating System. Seluruh metrik dan peristiwa koneksi termasuk timeout ping/pong, reconnect, dan close code ditulis periodik ke berkas CSV dengan stempel waktu yang konsisten. Pada jalur WebRTC dengan HTML dan JavaScript di peramban, logging memanfaatkan telemetri bawaan RTCPeerConnection melalui getStats() pada interval tetap untuk memperoleh byte in/out, RTT, dan *jitter*, yang dikirim ke server Node.js melalui *endpoint* "/events" dan diperkaya dengan metrik proses sebelum disimpan ke CSV. Peristiwa negosiasi SDP, pembentukan jalur ICE/STUN, serta aktivitas data channel juga dicatat untuk penelusuran.

Latensi WebRTC didefinisikan sebagai turn-taking latency
Peristiwa user.speaking.stop dan ai.speaking.start ditandai di klien menggunakan jam monotonik performance.now() dan nilai selisihnya (L_{turn}) dikirim ke server melalui /events untuk direkam ke CSV. Pengujian dilakukan minimal lima sesi independen dengan ≥ 10 giliran percakapan per sesi; sampel 2 detik pertama setiap sesi (handshake dan negosiasi) dikeluarkan dari perhitungan. Nilai latensi kemudian diringkas sebagai p50, p95, dan rentang (min–max). Pendekatan ini memastikan pengukuran latensi bersifat *reproducible*, berbasis event nyata, dan tidak bergantung pada entri sentinel fase awal. Seluruh skrip implementasi (HTML/JavaScript dan Node.js), contoh log CSV, serta skema tokenisasi sementara yang digunakan dalam eksperimen ini telah tersedia secara terbuka di repositori Zenodo dan dapat diakses melalui tautan [18] untuk mendukung replikasi hasil penelitian.

III. HASIL DAN PEMBAHASAN

A. Hasil

Perbedaan yang konsisten antarprotokol dapat kita lihat di Tabel 1. Dari sisi throughput, WebSocket jauh lebih tinggi dibanding WebRTC baik pada unduh maupun unggah: median down-kbps 384 vs 61,8 kbps dan median up-kbps 668 vs 28,9 kbps. Pada ekor distribusi, kuantil 95% WebSocket juga lebih besar, menunjukkan kapasitas puncak yang signifikan pada WebSocket. Kestabilan laten (diwakili *jitter*) cenderung lebih baik pada WebRTC: median *jitter* 0 ms dengan p95 ≈ 1 ms, sedangkan WebSocket menampilkan median ≈ 8,38 ms dan p95 ≈ 8,38 ms. Ini mengindikasikan variasi antar-interval yang lebih besar pada WebSocket meskipun *throughputnya* tinggi. Nilai p50 dan p95 jitter WebSocket yang identik (≈8,38 ms) menunjukkan bahwa data jitter bersifat diskret dan

tidak kontinu. Hal ini disebabkan oleh resolusi timer dan interval sampling pada jalur Python CLI yang menggunakan mekanisme event loop asinkron. Pada implementasi tersebut, time.perf_counter() dan scheduler asinkron memiliki resolusi efektif sekitar 8–10 ms di sistem ARM dengan beban I/O tinggi, sehingga variasi kecil antar-paket tidak terekam. Selain itu, pengambilan sampel per-interval yang relatif kasar (≈0,24 s median) membulatkan nilai jitter ke satuan milidetik dominan. Akibatnya, distribusi jitter WebSocket tampak "terkuantisasi" di nilai 8,38 ms dan tidak menampilkan rentang halus sebagaimana WebRTC yang memiliki pengukuran berbasis getStats() dengan resolusi sub-milidetik. Secara konseptual, temuan jitter yang rendah pada WebRTC dan nilai tetap pada WebSocket berkaitan langsung dengan pertanyaan riset utama tentang efisiensi dan kestabilan interaksi suara waktu nyata di perangkat edge. Literatur QoE menunjukkan bahwa stabilitas jitter berperan penting terhadap persepsi kelancaran percakapan; variasi yang besar dapat menurunkan skor MOS dan metrik objektif seperti PESQ maupun POLQA. Dengan demikian, meskipun kedua protokol mampu mempertahankan latensi di bawah ambang percakapan alami, performa jitter yang lebih stabil pada WebRTC menandakan potensi kualitas QoE yang lebih baik bagi pengguna.

Fenomena jitter WebRTC yang tercatat sering kali nol bukanlah anomali, melainkan konsekuensi teknis dari mekanisme protokol yang memang dirancang untuk menstabilkan aliran media. WebRTC memiliki jitter buffer di sisi penerima yang meratakan variasi waktu kedatangan paket, dilengkapi dengan kontrol kongesti dan adaptasi *bitrate* yang memastikan ritme pengiriman tetap konsisten. Selain itu, setiap frame audio diberi timestamp RTP dan dikendalikan melalui RTCP sehingga sinkronisasi *clock* menjaga keteraturan meskipun ada variasi kecil dalam jalur transmisi. Pada kondisi eksperimen yang stabil melalui Ethernet 50 Mbps tanpa loss mengindikasikan buffer ini tidak pernah "dipaksa" mengompensasi fluktuasi sehingga jitter terukur sering kali benar-benar nol. Kontras dengan itu, WebSocket tidak memiliki jitter buffer atau mekanisme penstabil media; setiap variasi kecil akibat TCP, *event loop*, atau *scheduling* kernel langsung terbaca sebagai *jitter* non-nol. Oleh karena itu, catatan jitter WebRTC yang rendah hingga nol justru mengonfirmasi bahwa protokol berfungsi sempurna sesuai tujuan desainnya.

Beban CPU memperlihatkan pola terpisah antara proses uji dan sistem. Beban CPU proses lebih tinggi pada WebSocket (mean 10,65% dan p95 19,67%) dibanding WebRTC (mean 3,78% dan p95 8,16%), sedangkan CPU sistem median justru lebih rendah pada WebSocket (2,5% vs 3,78%). Nilai maksimum CPU pada WebSocket pun lebih tinggi, menandakan lonjakan beban sesekali. Perlu dicatat, *missing rate* untuk metrik jaringan cukup besar pada kedua protokol. Karena itu, interpretasi sebaiknya menitikberatkan pada median dan kuantil, bukan sekadar rerata. Missing rate tinggi terutama berasal dari interval tanpa sampel valid saat aliran audio jeda sesaat atau pembaruan getStats() terhadap byte/paket tertunda. Situasi ini membuat mean bias ke rendah karena interval nol terbaca sebagai trafik rendah, sedangkan median (p50) dan kuantil (p95) lebih mewakili segmen komunikasi yang aktif. Karena itu, analisis difokuskan pada p50/p95 sebagai ringkasan utama performa selama sesi aktif. Secara keseluruhan, hasil menunjukkan *trade-off*: WebSocket memberikan throughput lebih tinggi dengan jitter dan beban proses yang lebih besar, sementara WebRTC cenderung lebih stabil namun *throughput* lebih rendah.

Tabel I
Ringkasan Statistik Kinerja WebRTC vs WebSocket

Metrik	CPU_PROC_PCT_Mean	CPU_SYS_PCT_Mean	Down_kbps	Jitter_Ms	Up_kbps
mean__WebRTC	3.782	4.251	51.534	0.531	28.852
mean__WebSocket	10.650	3.622	409.837	7.984	924.250
p50__WebRTC	3.330	3.780	61.800	0.000	28.900
p50__WebSocket	10.000	2.500	384.000	8.380	668.160
p95__WebRTC	8.160	8.840	64.500	1.000	31.965
p95__WebSocket	19.670	5.100	480.000	8.380	2673.408
min__WebRTC	0.010	0.000	0.000	0.000	0.000
min__WebSocket	9.500	2.400	38.400	7.470	38.400
max__WebRTC	20.670	22.170	64.900	3.000	32.500

max__WebSocket	39.800	43.600	537.600	8.840	3870.720
missing_rate__WebRTC	0.000	0.000	0.923	0.923	0.923
missing_rate__WebSocket	0.665	0.657	0.788	0.799	0.956

Rumus berikut mendefinisikan asal-usul dan satuan tiap metrik yang diringkas pada Tabel 1. Seluruh nilai pada Tabel 1 (mean, p50/p90/p95, serta min-maks) merupakan agregasi dari deret waktu per detik yang dihitung dengan rumus tersebut; *missing rate* menyatakan fraksi interval tanpa sampel valid. Missing yang tinggi dapat membias nilai rerata ke rendah karena interval kosong terbaca sebagai trafik rendah; oleh karena itu kami menitikberatkan ringkasan kuantil (p50/p95). Secara interpretatif, jitter yang lebih kecil menandakan kedatangan paket yang lebih stabil sesuai definisi RTP, throughput yang lebih besar menunjukkan kapasitas kanal yang lebih tinggi, sedangkan CPU yang lebih kecil mencerminkan beban komputasi aplikasi/sistem yang lebih ringan. Dengan demikian, pembaca dapat menautkan ringkasan statistik pada Tabel 1 ke mekanisme pengukuran yang mendasarinya.

Notasi & Satuan Umum

- Jendela sampel tetap: $\Delta t = 1$ s(per detik).
- Gunakan SI: 1 kbps = 1000 bit/s; 1 ms = 0.001 s.
- Indeks waktu: $t = 1,2, \dots, T$.
- Dua protokol: $P \in$ {WebRTC,WebSocket}. Semua metrik dihitung per-t, lalu diringkas (mean/p50/p95/min/max/missing_rate) per protokol.

1. Jitter_Ms -Inter-arrival jitter

- $D_t = [(R_t - R_{t-1}) - (S_t - S_{t-1}),$ $\quad J_t = J_{t-1} + \frac{|D_t| - J_{t-1}}{16}$, $J_0 = 0$ (1)

Keterangan:

- D_t selisih variasi waktu antar-kedatangan paket
- S_t= timestamp pengirim paket ke-t (detik).
- R_t= waktu terima paket ke-t (detik).
- Laporkan dalam ms: Jitter_Ms$(t) = 1000 \cdot J_t$.
- Interpretasi: makin kecil → makin stabil antar-kedatangan paket.
- Catatan sinkronisasi: definisi $D_t$di atas tidak memerlukan jam pengirim/penerima tersinkron absolut.

2. Up_kbps - *Throughput uplink* per detik

$$T_{up}(t) = \frac{8 \times byte_{sent}(t)}{\Delta t} \; ; \; Up\_kbps(t) = \frac{T_{up}(t)}{1000} \quad (2)$$

Keterangan:

- $T_{up}(t)$ dalam satuan bit/s dikonversi menjadi U$p_kbps(t)$ dalam satuan kbps
- Karena Δt = 1 s maka penyederhanaan menjadi Up_kbps (t) = 8 x bytes_sent(t)/1000
- bytes_sent(t)= total byte keluar pada detik t
- Interpretasi: makin besar - kapasitas *uplink* tercapai makin tinggi.

3. Down_kbps — *Throughput downlink* per detik

$$T_{down}(t) = \frac{8 \times byte_{recv}(t)}{\Delta t} \; ; \; Down\_kbps(t) = \frac{T_{down}(t)}{1000} \quad (3)$$

Keterangan:

- $T_{down}(t)$ dalam satuan bit/s dikonversi menjadi $Down\_kbps(t)$ dalam satuan kbps
- Karena Δt = 1 s maka penyederhanaan menjadi $Down\_kbps$ (t) = 8 x bytes_recv(t)/1000
- bytes_recv(t)= total byte masuk pada detik t
- Interpretasi: makin besar - kapasitas *downlink* tercapai makin tinggi.

4. **CPU_PROC_PCT_Mean -Utilisasi CPU proses (dinormalisasi inti)**

$$CPU\%_{proc}(t) = \frac{\Delta U_{proc}(t)}{\Delta t \,.\, N_{cores}} x\ 100\% \quad (4)$$

Keterangan:

- $\Delta Uproc(t) = Uproc(t) - Uproc(t-1)$
- $U_{\text{proc}} = U_{\text{user}} + U_{\text{system}}$ (waktu CPU proses, satuan detik CPU)
- $\Delta t = 1\text{s}$
- N_{cores}= jumlah inti logis

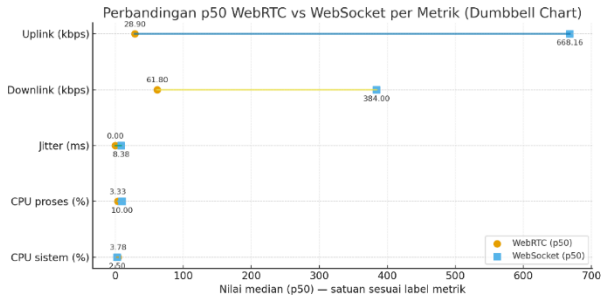
5. **CPU_SYS_PCT_Mean - Utilisasi CPU sistem (kernel-mode, skala mesin)**

$$CPU\%_{sys}(t) = \frac{\Delta U_{kernel}(t)}{\Delta\, U_{total}(t)} x\ 100\% \quad (5)$$

Keterangan:

- $\Delta U_{\text{kernel}}(t)$ = kenaikan waktu CPU kernel total (semua inti) selama Δt.
- $\Delta U_{\text{total}}(t)$ = total waktu CPU (user+kernel+idle+…) semua inti selama Δt.
- Interpretasi**:** indikasi overhead OS/driver; makin kecil - makin ringan.

Gambar 2 merangkum perbedaan median. Pada throughput, penanda WebSocket berada lebih ke kanan, menunjukkan throughput median lebih tinggi. Untuk jitter, penanda WebRTC berada di 0 ms sedangkan WebSocket di beberapa milidetik, menandakan kestabilan interval lebih baik pada WebRTC. Untuk beban CPU, WebSocket lebih tinggi pada CPU proses, sementara CPU sistem cenderung lebih rendah dibanding WebRTC. Urutan baris menegaskan dominasi selisih pada throughput, diikuti jitter, lalu CPU Fokus pada median/kuantil dipilih karena lebih robust terhadap interval hilang dibanding mean. Visual ini ringkasan cepat; detail tetap merujuk Tabel 1 dan 2.



Gambar 2. Perbandingan Median (P50) Lima Metrik: WebRTC (Titik) vs WebSocket (Persegi)

Untuk melengkapi analisa pada Tabel 1 dan Gambar 2 maka Tabel 2 menyajikan kualitas data dan kontinuitas sampel untuk WebRTC dan WebSocket. Komponen *missing rate* pada Tabel 1 diurai menjadi proporsi nilai nol (*zero-rate*) dan positif: pada downlink, WebSocket memiliki *nonzero-rate* lebih tinggi (≈0,212) dibanding WebRTC (≈0,076); pada uplink, WebRTC lebih tinggi (≈0,076) daripada WebSocket (≈0,044). Nilai nol murni sangat kecil (≈0–0,0007), sehingga besarnya *missing* terutama berasal dari interval tanpa rekaman. Implikasinya, mean dapat bias ke rendah; interpretasi utama memakai p50/p95. Perhitungan 1 − zero − nonzero menghasilkan *missing* yang konsisten dengan Tabel 1 (mis. uplink WebSocket ≈0,956; downlink WebSocket ≈0,788; uplink/downlink WebRTC ≈0,923).

Dari sisi kontinuitas waktu, pengambilan sampel WebRTC lebih rapat secara tipikal (median interval ≈0,017 s) dibanding WebSocket (≈0,241 s). Walau *p95 interval* WebRTC sedikit lebih besar (≈0,599 s vs 0,439 s), frekuensi jeda panjang tetap lebih kecil pada WebRTC (gap > 2 s ≈0,81%; > 5 s ≈0,034%) dibanding WebSocket (≈2,45% dan ≈0,79%). Sebaliknya, jeda maksimum menunjukkan ekor distribusi yang ekstrem pada WebRTC (≈2039 s) dan tetap cukup besar pada WebSocket (≈231 s), mengindikasikan segmen log yang terputus. Untuk RTT, hanya jalur WebSocket yang menghasilkan deret pengukuran (n = 248) dengan rtt-timeout-rate (>10 s) ≈ 0,512. Pada jalur

WebRTC, metrik RTT berada pada statistik transport dan tidak termasuk dalam subset metrik yang diambil; karena itu deret RTT tidak tersedia di log dan tidak dievaluasi. Pada keduanya, approx-dropout-rate = 0. Kekosongan pada sebagian timestamp merefleksikan interval tanpa observasi misalnya perbedaan frekuensi sampling, fase negosiasi, atau segmentasi sesi bukan indikasi dropout pada lapisan aplikasi.

Tabel II
Kualitas Data dan Kontinuitas Sampel

Pengukuran	WebSocket	WebRTC
up_kbps_zero_rate	0.000	0.001
up_kbps_nonzero_rate	0.044	0.076
down_kbps_zero_rate	0.000	0.001
down_kbps_nonzero_rate	0.212	0.076
both_zero_rate	0.000	0.001
median_interval_s	0.241	0.017
p95_interval_s	0.439	0.599
max_interval_s	231.265	2039.495
gaps_gt_2s_rate	0.024	0.008
gaps_gt_5s_rate	0.008	0.000
approx_dropout_rate	0.000	0.000
rtt_timeout_rate_(>10s)	0.512	0.000
rtt_nonnull	248.000	0.000

Secara keseluruhan, Tabel 2 menjelaskan WebSocket memberi proporsi sampel *downlink* bernilai positif lebih besar [19] tetapi lebih sering mengalami jeda menengah, sedangkan WebRTC lebih kontinu secara tipikal namun menyisakan ekor jeda yang sangat panjang. selaras dengan temuan *trade-off throughput* vs stabilitas pada hasil utama.

B. Pembahasan

1. Latensi

Rujukan pengukuran interaktif menunjukkan WebRTC ≈150 ms (stabil 120 - 180 ms), sedangkan log menunjukkan WebSocket ≈205 ms. Nilai ini menempatkan WebRTC lebih sesuai untuk percakapan (~150 - 200 ms) dengan margin yang lebih aman saat terjadi fluktuasi jaringan. Kolom latensi WebRTC pada log mengandung nilai sentinel, sehingga interpretasi berbasis pengukuran interaktif lebih tepercaya.

2. Efisiensi Bandwidth

Median throughput mengindikasikan WebSocket jauh lebih tinggi (uplink/downlink) dibanding WebRTC (tertera pada Gambar 2). Rentang WebSocket juga lebar hingga ~3,87 Mbps. Secara fungsional, ini berarti biaya jaringan WebSocket lebih besar dan berpotensi membebani link terbatas; WebRTC lebih hemat berkat kompresi dan kontrol aliran bawaan protokol. Tabel 2 menunjukkan proporsi sampel bernilai positif (*non-zero rate*) yang lebih besar pada downlink WebSocket, selaras dengan temuan *throughputnya*.

3. Beban CPU

Beban CPU proses lebih tinggi pada WebSocket (≈10,65%) dibanding WebRTC (≈3,78%). Pada perangkat *edge*, selisih ini mengurangi ruang untuk komputasi lain (mis. ASR/TTS, tool calling). Walau WebRTC melakukan enkode *audio*, beban yang lebih rendah karena dipengaruhi oleh jalur data yang lebih matang/efisien dan kontrol pengiriman yang adaptif.

4. Kestabilan Konektivitas

Log WebSocket memperlihatkan putus-sambung dan *timeout* ping/pong; rtt-timeout-rate > 10 s mencapai ~0,51 (Tabel 2). Selain itu, interval antarsampel WebSocket lebih renggang dan frekuensi jeda > 2 s dan > 5 s lebih tinggi dibanding WebRTC. Kondisi ini menjelaskan mengapa statistik nilai pada Tabel 1 cenderung bias: missing rate besar dan jeda panjang mengurangi representativitas data saat beban jaringan berubah.

5. Implikasi Praktis

Untuk penerapan produksi di sisi klien, WebRTC menjadi pilihan utama karena profil latensi rendah, pemakaian bandwidth yang hemat, dan stabilitas koneksi yang lebih baik. WebSocket tetap relevan untuk komunikasi server-to-server atau pengembangan prototipe; agar reliabilitasnya mendekati WebRTC diperlukan konfigurasi ketat yang mencakup pengaturan interval dan batas waktu *heartbeat* (ping–pong), aktivasi serta penalaan TCP keepalive pada tingkat sistem operasi guna mempercepat deteksi koneksi mati, penggunaan I/O non-blocking untuk menghindari head-of-line blocking pada alur audio, pembatasan ukuran bingkai dan pemecahan audio menjadi potongan yang konsisten, mekanisme rekoneksi tangguh berbasis exponential backoff dengan pemulihan state, serta penerapan *watchdog* untuk memicu ulang koneksi ketika terdeteksi anomali.

6. Harapan dan Arah Pengembangan

- *Eksperimen Terkontrol Jaringan*, Uji parameter *delay/loss/jitter* untuk memetakan zona operasi optimal kedua protokol.
- *Qoe Terukur*, Tambahkan indikator pengalaman dan turn taking latency agar hasil teknis langsung berelasi ke persepsi pengguna.
- *Best-Practice* WebSocket**,** Eksplor codec (OPUS/OGG) di atas WebSocket, ukuran bingkai 10 - 60 ms, rate limiting/backpressure, serta kebijakan *heartbeat* yang presisi.
- *Replikasi dan Interval Kepercayaan*, Jalankan beberapa sesi independen dan laporkan median bersama 95% CI (bootstrap) serta ukuran efek.
- *Profiling Sumber Daya*, Pisahkan kontribusi CPU per komponen (I/O, enkode/dekode, jaringan) termasuk uji pada perangkat mobile kelas menengah.
- *Pembanding Modern*, Tambahkan WebTransport/QUIC sebagai kandidat transport generasi baru.

Ringkasnya, hasil mendukung WebRTC sebagai pilihan utama untuk percakapan *real-time* di klien, sementara WebSocket menyediakan jalur yang fleksibel untuk kebutuhan aliran data kustom dan eksperimen tingkat aplikasi keduanya saling melengkapi dalam peta solusi produksi.

KESIMPULAN

Studi ini membandingkan WebRTC dan WebSocket untuk percakapan suara *real-time* pada perangkat edge Orange Pi 3B melalui koneksi Ethernet 50 Mbps. Hasil menunjukkan bahwa WebRTC lebih tepat untuk klien (*browser/mobile*): latensi stabil ≈150 ms (tipikal 120–180 ms), *jitter* sangat kecil (p50 = 0 ms; p95 ≈ 1 ms), serta kebutuhan bandwidth dan beban proses lebih rendah (median uplink ≈ 28,9 kbps; median downlink ≈ 61,8 kbps; rata-rata beban proses ≈ 3,78%). Sementara itu, WebSocket memberikan median throughput lebih tinggi, namun jitter lebih bervariasi, beban proses lebih besar, dan sesekali terjadi reconnect/timeout sehingga kurang ideal untuk menjaga kelancaran percakapan. Eksperimen ini berlangsung tanpa rekayasa kondisi jaringan (tidak ada injeksi loss/delay/jitter) dan hanya dievaluasi pada satu platform (perangkat yang sama); pada jalur WebSocket, kuantisasi timer JavaScript serta periodisasi interval sampling di aplikasi turut membentuk karakterisasi jitter yang terukur. Missing rate pada sebagian metrik dapat membias *mean*, sehingga interpretasi utama menitikberatkan pada kuantil (p50/p95). Ke depan, pengujian terkontrol dengan tc/netem, evaluasi QoE serta turn-taking latency, perluasan pembanding ke WebTransport, dan profiling sumber daya yang lebih granular di berbagai perangkat akan memperkuat generalisasi temuan dan menghubungkan metrik teknis dengan persepsi pengguna. Secara praktis, untuk produksi di sisi klien yang menuntut respons natural (≤ ~200 ms), stabilitas, dan efisiensi sumber daya, WebRTC direkomendasikan sebagai pilihan utama. WebSocket tetap relevan pada skenario server-to-server atau aliran data kustom dengan *payload* kecil atau *high-event-rate streams*, dengan prasyarat melakukan rekayasa reliabilitas agar kinerjanya mendekati kebutuhan percakapan real-time.

REFERENSI

[1] Y. L. Oktavianus, I. Elfitri, and O. W. Purbo, "Perancangan dan Analisis Jaringan FTTB Berbasis Teknologi GPON Pada Bangunan Hotel," *INOVTEK Polbeng - Seri Inform.*, vol. 8, no. 1, p. 88, Jun. 2023, doi: 10.35314/isi.v8i1.3213.

[2] S. Thangam, M. Gurupriya, A. S. Revanth, D. Arun Joel, C. M. Shankar, and I. S. Koushik, "VoIP QoS Refinement through Call Sequencing using Adaptive Jitter Buffer Algorithm," in *2024 15th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, IEEE, Jun. 2024, pp. 1–7. doi: 10.1109/ICCCNT61001.2024.10725456.

[3] Y. Liang, Y.-C. Lee, and A. Teng, "Real-Time Communication," in *Multimedia over IP and Wireless Networks*, Elsevier, 2007, pp. 503–525. doi: 10.1016/B978-012088480-3/50016-3.

[4] S. Turkar, R. Singh, A. Patil, P. Kumar, and S. L. Tambe, "Web RTC (Real Time Communication)," *Int. J. Res. Eng. Appl. \& Manag.*, vol. 10, no. 1, pp. 128–131, Apr. 2024, doi: 10.35291/2454-9150.2024.0178.

[5] H. Mahmoud and R. Abozariba, "A systematic review on WebRTC for potential applications and challenges beyond audio video streaming," *Multimed. Tools Appl.*, vol. 84, no. 6, pp. 2909–2946, Nov. 2024, doi: 10.1007/s11042-024-20448-9.

[6] A. Olaru and M. Pricope, "Multi-Modal Decentralized Interaction in Multi-Entity Systems," *Sensors*, vol. 23, no. 6, p. 3139, Mar. 2023, doi: 10.3390/s23063139.

[7] Q. Zhang, "A Web-Based Synchronized Architecture for Collaborative Dynamic Diagnosis and Therapy Planning," *IEEE Access*, vol. 11, pp. 421–437, 2023, doi: 10.1109/ACCESS.2022.3232275.

[8] G. Bingöl, S. Porcu, A. Floris, and L. Atzori, "QoE Estimation of WebRTC-based Audio-visual Conversations from Facial and Speech Features," *ACM Trans. Multimed. Comput. Commun. Appl.*, vol. 20, no. 5, pp. 1–23, May 2024, doi: 10.1145/3638251.

[9] S. Islam, M. Welzl, and T. Fladby, "Real-Life Implementation and Evaluation of Coupled Congestion Control for WebRTC Media and Data Flows," *IEEE Access*, vol. 10, pp. 95046–95066, 2022, doi: 10.1109/ACCESS.2022.3206041.

[10] Y. Wu, R. Chang, J. M. Hellerstein, A. Satyanarayan, and E. Wu, "DIEL: Interactive Visualization Beyond the Here and Now," *IEEE Trans. Vis. Comput. Graph.*, vol. 28, no. 1, pp. 737–746, Jan. 2022, doi: 10.1109/TVCG.2021.3114796.

[11] G. Carofiglio, G. Grassi, L. Muscariello, M. Papalini, and J. Samain, "ROBUST: A Reliable and Flexible Media Transport for Real-Time Services," *IEEE Trans. Netw. Serv. Manag.*, vol. 20, no. 3, pp. 2475–2488, Sep. 2023, doi: 10.1109/TNSM.2023.3293871.

[12] G. Perna *et al.*, "Real-Time Classification of Real-Time Communications," *IEEE Trans. Netw. Serv. Manag.*, vol. 19, no. 4, pp. 4676–4690, Dec. 2022, doi: 10.1109/TNSM.2022.3189628.

[13] K. Rzepka, P. Szary, K. Cabaj, and W. Mazurczyk, "Performance evaluation of Raspberry Pi 4 and STM32 Nucleo boards for security-related operations in IoT environments," *Comput. Networks*, vol. 242, p. 110252, Apr. 2024, doi: 10.1016/j.comnet.2024.110252.

[14] C. Xie, Z. Zhang, R. Jin, F. Chai, S. Wang, and P. Yang, "Design and computational optimization of a laser communication adaptive optics wavefront processor based on ARM architecture," *Opt. Express*, vol. 33, no. 14, p. 30252, Jul. 2025, doi: 10.1364/OE.560515.

[15] X. Ren, Y. Shao, Y. Zhang, Y. Ni, Y. Bi, and R. Li, "Primerdiffer: a python command-line module for large-scale primer design in haplotype genotyping," *Bioinformatics*, vol. 39, no. 4, Apr. 2023, doi: 10.1093/bioinformatics/btad188.

[16] A. Pashamokhtari, N. Okui, Y. Miyake, M. Nakahara, and H. H. Gharakheili, "Combining Stochastic and Deterministic Modeling of IPFIX Records to Infer Connected IoT Devices in Residential ISP Networks," *IEEE Internet Things J.*, vol. 10, no. 6, pp. 5128–5145, Mar. 2023, doi: 10.1109/JIOT.2022.3222116.

[17] M. Orfanos *et al.*, "Testing and Evaluation of Wi-Fi RTT Ranging Technology for Personal Mobility Applications," *Sensors*, vol. 23, no. 5, p. 2829, Mar. 2023, doi: 10.3390/s23052829.

[18] A. H. Limaran and A. Wicaksono, "Supplementary Dataset for 'A Comparative Study of WebRTC and WebSocket Performance in Real-Time Voice Communication,'" Oct. 2025, *Zenodo*. doi: 10.5281/zenodo.17272914.

[19] A. Chodorek and R. R. Chodorek, "Web Real-Time Communications-Based Unmanned-Aerial-Vehicle-Borne Internet of Things and Stringent Time Sensitivity: A Case Study," *Sensors*, vol. 25, no. 2, p. 524, Jan. 2025, doi: 10.3390/s25020524.