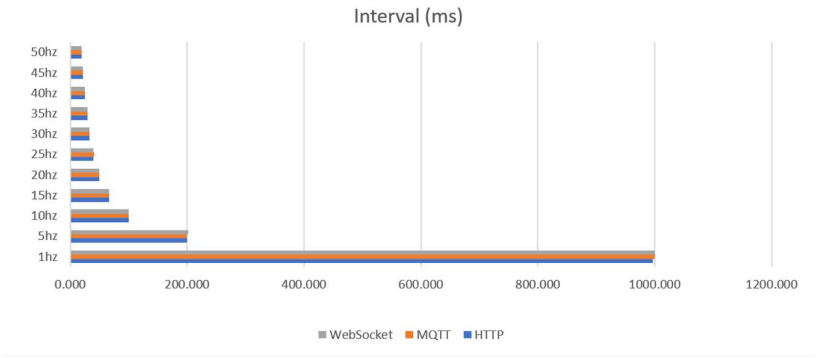
BAB III

A. Hasil Interval dan Packet Loss

Pengujian awal difokuskan pada verifikasi apakah ketiga protokol mampu mempertahankan interval pengiriman sesuai frekuensi target. Untuk setiap protokol (HTTP, MQTT, dan WebSocket), ESP8266 mengirimkan paket selama 120 detik pada sebelas frekuensi yang berbeda, yaitu 1, 5, 10, 15, 20, 25, 30, 35, 40, 45, dan 50hz. Rekapitulasi interval rata-rata yang dihitung dari selisih waktu penerimaan ts_server antar paket, bersama dengan jumlah paket yang diterima dan packet loss.



Gambar. 2. Grafik interval pada setiap protocol

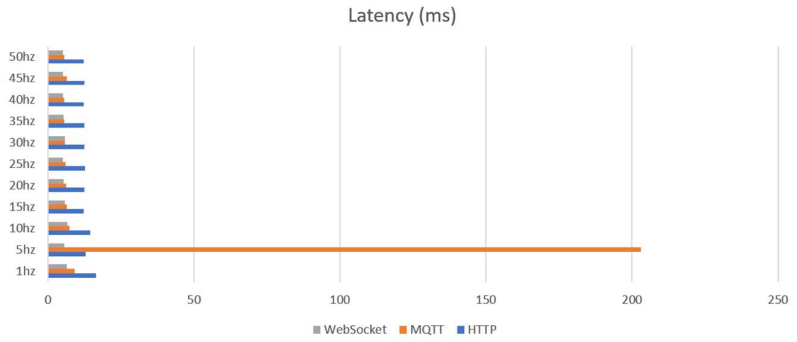
Terlihat pada Gambar 2 untuk seluruh protokol, interval rata-rata sangat dekat dengan nilai teoritis sekitar 1000ms untuk 1hz, 200ms untuk 5hz, dan sekitar 20ms untuk 50hz, dengan Deviasi interval rata-rata untuk setiap protokol pada seluruh frekuensi disajikan pada Tabel II.

TABEL II
HASIL UJI INTERVAL (MS)

Interval	HTTP	MQTT	WebSocket
1hz	996,704	999,983	1000,000
5hz	199,995	200,010	201,653
10hz	99,997	99,986	99,990
15hz	66,998	66,998	66,999
20hz	49,999	50,000	50,004
25hz	39,999	40,789	39,997
30hz	32,999	33,011	32,999
35hz	28,999	29,010	29,003
40hz	24,999	24,994	24,997
45hz	21,999	22,000	22,002
50hz	20,008	19,994	19,999

Dari sisi keandalan, analisis terhadap kolom seq pada ketiga berkas CSV memperlihatkan bahwa nilai minimum dan maksimum seq konsisten dengan jumlah paket unik yang tercatat berada pada kisaran 0% untuk seluruh kombinasi protokol dan frekuensi. Dengan kata lain, selama durasi 120 detik per skenario, tidak terdeteksi kehilangan paket yang signifikan pada jaringan Wi-Fi lokal yang digunakan.

B. Hasil Latency



Gambar. 3. Grafik latency pada setiap protokol

Pada Gambar 3 terlihat pola bahwa WebSocket secara konsisten memiliki latency rata-rata paling rendah, diikuti oleh MQTT, sedangkan HTTP memiliki nilai RTT sedikit lebih tinggi namun tetap berada dalam orde belasan milidetik. Pola ini

selaras dengan hasil beberapa penelitian terbaru yang melaporkan bahwa protokol berbasis koneksi persisten seperti MQTT dan WebSocket cenderung menghasilkan delay yang lebih kecil dibandingkan HTTP berbasis request response tradisional. Studi menemukan bahwa MQTT over WebSocket mampu mempertahankan delay rendah pada aplikasi IoT, mempertegas keunggulan pendekatan koneksi persisten untuk komunikasi real-time[20].

TABEL III
HASIL UJI LATENCY

Interval	HTTP	MQTT	Web-Socket
1hz	16,4492	9,0763	6,4153
5hz	12,8781	203,1018	5,6694
10hz	14,4145	7,4012	6,7089
15hz	12,1323	6,4969	5,7331
20hz	12,4539	6,3005	5,3397
25hz	12,6679	5,9010	5,1747
30hz	12,5052	5,7079	5,6950
35hz	12,5509	5,5709	5,3254
40hz	12,3155	5,5922	5,0706
45hz	12,3715	6,5554	5,1397
50hz	12,2354	5,6013	5,0610

Hasil pengukuran pada Tabel III menunjukkan bahwa rata-rata RTT berada pada kisaran belasan milidetik untuk semua frekuensi, misalnya sekitar 16ms pada 1hz dan menurun menjadi sekitar 12ms pada frekuensi yang lebih tinggi, dengan nilai minimum sekitar 9-11ms dan maksimum sesekali mencapai ratusan milidetik akibat spike jaringan.

Spike latency yang muncul secara sporadis dalam pengujian ini merupakan manifestasi dari sifat non-deterministik pada jaringan nirkabel berbasis CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance). Secara teknis, lonjakan ini dipicu oleh mekanisme random backoff dan retransmisi pada lapisan Link Layer ketika terjadi kontensi kanal atau interferensi sesaat, yang menyebabkan RTT meningkat tajam pada interval waktu tertentu. Selain faktor akses medium, fenomena bufferbloat pada access point juga berkontribusi terhadap lonjakan latensi di mana penumpukan antrean paket pada memori buffer router dapat

memperbesar end-to-end delay meskipun throughput rata-rata terlihat stabil[21]. Dari perspektif lapisan Transport, penggunaan protokol TCP turut memperkuat efek ini melalui mekanisme Delayed ACK dan retransmission timeout (RTO). Ketika kondisi saluran nirkabel mengalami fluktuasi kualitas sinyal (fading), TCP akan melakukan retransmisi yang mengakibatkan nilai latensi maksimum melonjak jauh di atas nilai median, menciptakan profil distribusi data yang memiliki long-tail.

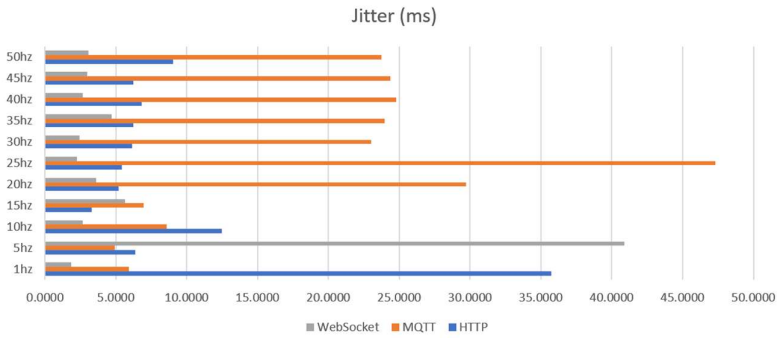
C. Hasil Jitter

Nilai jitter RTT pada setiap protokol dan frekuensi disajikan pada Tabel IV

TABEL IV
HASIL UJI JITTER

Interval	HTTP	MQTT	Web-Socket
1hz	35,7175	5,9095	1,8536
5hz	6,3862	4,9096	40,8768
10hz	12,4952	8,5953	2,6579
15hz	3,2942	6,9493	5,6425
20hz	5,1967	29,6967	3,5918
25hz	5,4342	47,3132	2,2506
30hz	6,1568	23,0113	2,4343
35hz	6,2435	23,9901	4,6777
40hz	6,8269	24,7635	2,6420
45hz	6,2354	24,3530	2,9984
50hz	9,0448	23,7257	3,0579

sedangkan Perbandingan jitter antar protokol divisualisasikan pada Gambar 4



Gambar. 4. Grafik jitter pada setiap protokol

Terdapat tren yang menarik, yaitu pada frekuensi rendah (1-10hz), jitter ketiga protokol relatif serupa, tetapi pada frekuensi menengah hingga tinggi (≥20hz) jitter salah satu protokol, misalnya MQTT, cenderung meningkat dibandingkan HTTP dan WebSocket. Hal ini menunjukkan bahwa meskipun rata-rata latency masih di

kisaran belasan milidetik, variasi delay antar paket pada frekuensi tinggi menjadi lebih besar untuk protokol tertentu, yang bisa berdampak pada aplikasi IoT yang sensitif terhadap jitter.

Hasil pengujian pada Tabel IV menunjukkan bahwa MQTT mengalami lonjakan jitter pada frekuensi menengah-tinggi (≥20hz) hingga mencapai kisaran 23,0-47,3ms, yang secara signifikan melampaui performa HTTP dan WebSocket. Tingginya jitter ini berimplikasi pada penurunan prediktabilitas temporal komunikasi data. Pada aplikasi closed-loop control (seperti sistem kemudi otomatis atau sinkronisasi mesin industri), jitter yang tinggi menyebabkan interval sampling menjadi tidak seragam (non-uniform sampling). Secara teknis, hal ini dapat mengakibatkan fluktuasi pada estimasi variabel keadaan (state estimation) dan memicu instabilitas pada aksi kontrol[22].

Selain itu, pada sensor kritis yang bekerja dengan batasan waktu ketat (hard real-time), variabilitas delay yang lebar dapat menyebabkan pesan sampai melewati ambang batas (missed deadline), yang berisiko pada kegagalan koordinasi antara sensor dan aktuator. Sebagai konsekuensi praktis, jika MQTT tetap digunakan karena keunggulan arsitektur publish/subscribe-nya, maka diperlukan strategi mitigasi seperti penyertaan timestamp presisi tinggi pada payload untuk rekonstruksi urutan data, serta optimalisasi konfigurasi buffer di sisi broker guna meminimalisir penumpukan antrean pada load tinggi.

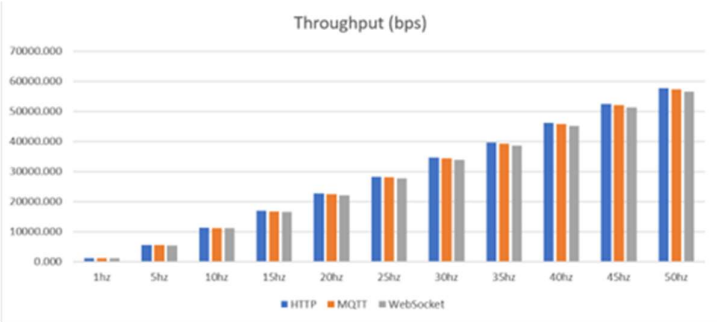
D. Hasil Throughput

Hasil pengukuran menunjukkan bahwa throughput pesan per detik mendekati nilai frekuensi target untuk semua protokol; misalnya pada frekuensi 50hz diperoleh sekitar 50 pesan per detik selama durasi 120 detik, sejalan dengan hasil analisis jumlah paket dan durasi pada bagian sebelumnya. Sementara itu, throughput meningkat hampir linier terhadap frekuensi, dengan nilai terendah pada 1hz dan tertinggi pada 50hz. Perbedaan kecil antar protokol pada throughput terutama disebabkan oleh perbedaan ukuran payload efektif dan overhead framing masing-masing protokol, bukan karena kegagalan jaringan dalam mengirimkan

data. Pola throughput ini konsisten dengan berbagai penelitian yang membandingkan protokol IoT berbasis pesan, di mana peningkatan frekuensi sampling akan berbanding lurus dengan jumlah data yang dikirim per satuan waktu selama jaringan belum mencapai titik jenuh[23]. Rekapitulasi throughput untuk setiap protokol dan frekuensi disajikan pada Tabel V, sedangkan visualisasi perbandingannya ditunjukkan pada Gambar 5.

TABEL V
HASIL UJI THROUGHPUT (BITS PER SECOND)

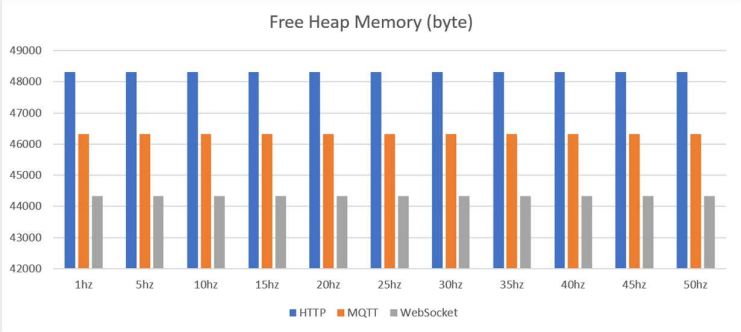
Interval	HTTP	MQTT	WebSocket
1hz	1100,809	1085,512	1069,001
5hz	5569,554	5514,156	5410,848
10hz	11253,451	11177,148	11016,577
15hz	16847,881	16734,053	16490,991
20hz	22570,951	22415,613	22096,741
25hz	28210,434	28016,645	27619,192
30hz	34557,526	34322,748	33847,278
35hz	39458,867	39190,402	38639,566
40hz	46018,735	45692,771	45066,311
45hz	52375,879	52053,123	51299,994
50hz	57610,585	57226,158	56423,401



Gambar. 5. Hasil uji throughput

E. Hasil Free Heap Memory pada ESP8266

Nilai free heap memory rata-rata untuk setiap protokol pada seluruh frekuensi disajikan pada Tabel VI, sedangkan visualisasi perbandingannya ditunjukkan pada Gambar 6



TABEL VI
HASIL UJI FREE MEMORI PADA ESP8266

Interval	HTTP	MQTT	Web-Socket
1hz	48320	46328	44328
5hz	48320	46328	44328
10hz	48320	46328	44328
15hz	48320	46328	44328
20hz	48320	46328	44328
25hz	48320	46328	44328
30hz	48320	46328	44328
35hz	48320	46328	44328
40hz	48320	46328	44328
45hz	48320	46328	44328
50hz	48320	46328	44328

Perbedaan yang tampak justru berasal dari perbedaan overhead library masing-masing protokol. Misalnya, protokol HTTP, yang menggunakan ESP8266HTTPClient, menunjukkan nilai free heap rata-rata sedikit lebih tinggi dibandingkan MQTT dan WebSocket, sedangkan protokol yang membutuhkan stack lebih kompleks seperti WebSocket client dan stack MQTT cenderung memiliki nilai heap sedikit lebih rendah namun tetap dalam batas aman.

Secara umum, hasil pengukuran menunjukkan bahwa nilai free memori pada ESP8266 relatif stabil di seluruh frekuensi pengiriman, tanpa tren penurunan yang signifikan dari waktu ke waktu. Hal ini mengindikasikan tidak adanya kebocoran memori yang berarti selama pengujian multi-frekuensi dua menit per stage.