JURNAL RESISTOR | ISSN 2598-7542 | E-ISSN 2598-9650
Vol. 8 No 3 – Desember 2025 | https://s.id/jurnalresistor
DOI: https://doi.org/10.31598
Publishing: Institut Bisnis dan Teknologi Indonesia

Optimasi Pengiriman dan Rekonstruksi Data Visual pada Sistem Irigasi Otomatis Berbasis IoT menggunakan MQTT dan FFmpeg

I Kadek Agus Wahyu Raharja[1], Gde Wikan Pradnya Dana[2], I Gede Wira Darma[3], I Gusti Agung Made Yoga Mahaputra[4]

[1,2,3]Program Studi Teknik Komputer, Fakultas Teknik dan Perencanaan, Universitas Warmadewa
Jl. Terompong No.24, Denpasar , Bali, Indonesia
[4]Program Studi Teknik Otomasi Jurusan Teknik Elektro Politeknik Negeri Bali
Kampus Bukit, Jimbaran, Bali, Indonesia

e-mail: raharja.wahyu.agus.kadek@warmadewa.ac.id[1], wikanpdana8044@warmadewa.ac.id[2], igedewiradarma@warmadewa.ac.id[3], yogamahaputra@pnb.ac.id[4]

Received : Agustus, 2025	Accepted : Desember, 2025	Published : Desember, 2025

Abstract

The integration of the Internet of Things (IoT) in smart irrigation enables real-time visual monitoring for more accurate decision-making. However, transmitting images over constrained networks presents challenges related to latency and bandwidth efficiency. This paper focuses on optimizing visual data transmission using MQTT for lightweight communication and FFmpeg for multimedia processing. The study does not emphasize multi-device testing, but instead concentrates on the efficiency and reliability of single-image transmission through protocol optimization. A prototype using the ESP32-S3-EYE camera captures images following irrigation triggers and publishes them to a Mosquitto broker; on the receiver side (Python), the images are stored and reconstructed into video using FFmpeg. With small JPEG payloads (5–11 KB) and MQTT QoS 1, the results demonstrate near real-time performance with an average one-way latency of approximately 30 ms and an effective bandwidth of 0.102–0.145 Mbps per image, while maintaining sufficient visual quality for monitoring. QoS 1 optimization was shown to improve transmission success to 100% compared to QoS 0, which achieved only around 50%. The combination of MQTT and FFmpeg proves to be reliable and resource-efficient for visual data transmission in IoT-based irrigation systems.

Keywords: *IoT, Smart Irrigation, MQTT, FFmpeg, Latency, Bandwidth*

Abstrak

Integrasi Internet of Things (IoT) pada irigasi pintar memungkinkan pemantauan visual secara real-time untuk pengambilan keputusan yang lebih akurat. Namun, pengiriman citra pada jaringan terbatas menimbulkan tantangan terkait latensi dan efisiensi bandwidth. Makalah ini berfokus pada optimasi transmisi data visual menggunakan MQTT untuk komunikasi ringan dan FFmpeg untuk pemrosesan multimedia. Penelitian tidak menitikberatkan pada pengujian multi-perangkat, melainkan pada efisiensi dan keandalan pengiriman citra tunggal melalui optimasi protokol. Prototipe dengan kamera ESP32-S3-EYE menangkap citra setelah pemicu irigasi dan mempublikasikannya ke broker Mosquitto; sisi penerima (Python) menyimpan citra dan merekonstruksinya menjadi video dengan FFmpeg. Dengan payload JPEG kecil (5–11 KB) dan MQTT QoS 1, hasil menunjukkan kinerja mendekati real-time dengan latensi satu arah rata-rata ≈30 ms dan bandwidth efektif 0,102–0,145 Mbps per citra, seraya mempertahankan kualitas visual yang memadai untuk monitoring. Optimasi QoS 1 terbukti meningkatkan keberhasilan pengiriman

hingga 100% dibanding QoS 0 yang hanya mencapai ≈50%. Kombinasi MQTT + FFmpeg terbukti andal dan hemat sumber daya untuk data visual pada sistem irigasi IoT.

Kata Kunci: *IoT, Smart Irrigation, MQTT, FFmpeg, Latency, Bandwidth*

1. PENDAHULUAN

Efisiensi sistem irigasi memainkan peran penting dalam meningkatkan produktivitas pertanian, terutama di wilayah dengan keterbatasan sumber daya air. Permasalahan utama dalam irigasi tradisional meliputi distribusi air yang tidak optimal serta kehilangan air akibat penguapan, rembesan, atau penggunaan yang tidak efisien. Pada praktiknya, pemantauan kondisi tanah dan kebutuhan air masih dilakukan secara manual, yang sering kali tidak akurat dan menyebabkan pemborosan air. Petani bahkan kerap mengandalkan intuisi dalam menentukan waktu dan jumlah air yang harus diberikan kepada tanaman. Hal ini dapat mengakibatkan over-irrigation yang berisiko menimbulkan erosi tanah, atau under-irrigation yang berdampak pada penurunan hasil panen [1], [2].

Teknologi Internet of Things (IoT) menawarkan solusi inovatif untuk permasalahan tersebut melalui otomasi dan pemantauan waktu nyata, sehingga dapat meningkatkan efisiensi sekaligus mendukung ketahanan pangan nasional yang semakin tertekan oleh perubahan iklim [3], [4]. Beberapa penelitian sebelumnya telah membuktikan efektivitas sistem irigasi berbasis mikrokontroler, seperti Arduino, dalam mengurangi pemborosan air dan meningkatkan produktivitas [5], bahkan telah diaplikasikan dalam ekosistem lokal, misalnya pada kontrol air Subak di Bali [3]. Namun, mayoritas penelitian terdahulu masih menitikberatkan pada data numerik seperti kelembapan atau suhu tanah, sedangkan kebutuhan data visual (citra) untuk verifikasi kondisi lapangan misalnya deteksi genangan, kebocoran, atau keberhasilan penyiraman, masih menuntut skema transmisi yang lebih efisien [6], [7].

Beberapa pendekatan transmisi data visual pada sistem IoT pertanian telah dikembangkan, namun memiliki keterbatasan signifikan untuk implementasi pada jaringan terbatas. WebRTC (Web Real-Time Communication) meskipun mendukung streaming video real-time dengan latensi sangat rendah (<100 ms), membutuhkan bandwidth tinggi (>1 Mbps untuk video VGA) dan kompleksitas implementasi pada perangkat embedded yang sangat tinggi [8], [9]. HTTP Live Streaming (HLS) kompatibel dengan banyak platform dan mendukung adaptive bitrate, namun memiliki latensi tinggi (5–20 detik) karena segmentasi video, sehingga tidak cocok untuk monitoring near real-time [10]. RTSP (Real-Time Streaming Protocol) dirancang khusus untuk streaming video, tetapi memerlukan server streaming khusus dan memiliki overhead protokol yang lebih besar dibanding MQTT (header RTSP sekitar 200 byte vs MQTT sekitar 5 byte) [11]. CoAP (Constrained Application Protocol) meskipun dirancang untuk IoT dengan overhead minimal, tidak memiliki mekanisme publish-subscribe seperti MQTT dan kurang efisien untuk payload besar (>1 KB) [12], [13].

Berdasarkan analisis tersebut, penelitian ini memilih kombinasi MQTT + FFmpeg karena beberapa keunggulan. Pertama, MQTT memiliki overhead minimal (header 2–5 byte untuk QoS 1) sehingga cocok untuk perangkat dengan bandwidth terbatas (<500 Kbps) [14]. Kedua, MQTT QoS 1 menyediakan mekanisme acknowledgment (PUBACK) yang memastikan setiap paket diterima broker, berbeda dengan UDP pada WebRTC/CoAP yang tidak menjamin pengiriman [15]. Ketiga, FFmpeg dapat merekonstruksi video dari sekuens citra JPEG tanpa memerlukan streaming kontinyu seperti HLS/RTSP, sehingga toleran terhadap koneksi yang terputus-putus [16]. Keempat, ESP32-S3-EYE (dual-core 240 MHz, 520 KB SRAM) mampu menjalankan MQTT client asynchronous tanpa blocking I/O, sementara implementasi WebRTC akan menghabiskan >80% CPU untuk encoding H.264 [17]. Kelima, arsitektur publish-subscribe MQTT memungkinkan penambahan subscriber tanpa mengubah kode publisher, berbeda dengan RTSP yang memerlukan koneksi point-to-point [18].

Tantangan utama dari data visual terletak pada latensi dan konsumsi bandwidth, terutama pada jaringan komunikasi terbatas. Dalam penelitian ini, protokol Message Queuing Telemetry Transport (MQTT) dipilih karena bersifat ringan, efisien, mendukung Quality of Service (QoS), serta tangguh pada koneksi yang fluktuatif [19], [20]. Untuk merekonstruksi data visual yang dikirim dalam bentuk payload berukuran kecil,

digunakan FFmpeg sebagai perangkat lunak yang mampu merangkai sekuens citra menjadi format video standar secara efisien [21]. Penelitian ini tidak berfokus pada implementasi multi-perangkat atau skalabilitas horizontal, tetapi pada optimasi pengiriman citra tunggal melalui tiga pendekatan: optimasi ukuran payload dengan kompresi JPEG kualitas 60% (menghasilkan 5–11 KB per frame), mekanisme QoS 1 untuk memastikan setiap frame diterima sebelum mengirim frame berikutnya, dan buffering strategis menggunakan SD-MMC untuk memisahkan proses pengambilan citra dari transmisi.

Dengan kombinasi ini, penelitian berfokus pada tiga permasalahan utama: bagaimana mempertahankan kehandalan pengiriman citra dengan ukuran payload kecil; bagaimana mengukur dan meminimalkan latensi transmisi satu arah; dan bagaimana menjaga efisiensi bandwidth sekaligus kualitas visual yang memadai untuk inspeksi lapangan.

Kontribusi utama penelitian ini adalah pertama, merancang pipeline transmisi data visual end-to-end dengan perangkat ESP32-S3-EYE menggunakan protokol MQTT (QoS 1) dan rekonstruksi FFmpeg (codec libx264, format yuv420p) untuk monitoring irigasi, lengkap dengan mekanisme error handling dan retry logic. Kedua, menyajikan metodologi pengukuran kinerja transmisi yang mencakup latensi satu arah, bandwidth efektif, dan success rate rekonstruksi video, dengan validasi melalui perhitungan manual pada payload 5–11 KB. Ketiga, membuktikan efektivitas optimasi QoS 1 dalam meningkatkan keandalan transmisi dari 50% (QoS 0) menjadi 100% (QoS 1), dengan hasil kinerja mendekati near real-time (latensi 30 ms per citra) dan konsumsi bandwidth efisien (0,102–0,145 Mbps), yang cukup memadai untuk mendukung monitoring irigasi otomatis pada jaringan terbatas. Dengan demikian, penelitian ini memberikan kontribusi signifikan dalam mengintegrasikan IoT berbasis citra ke dalam sistem irigasi modern, sekaligus memperkuat ketahanan pangan melalui teknologi pertanian presisi.

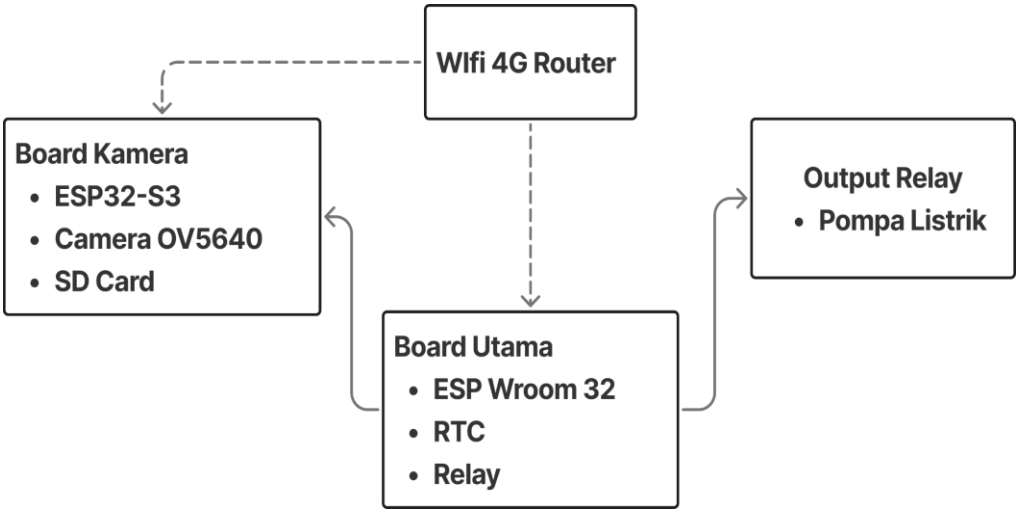
2. METODE PENELITIAN

Metode penelitian ini menjelaskan alur kerja sistem dan parameter pengujian untuk mengevaluasi kinerja transmisi data visual berbasis MQTT + FFmpeg. Fokus utama adalah optimasi perangkat lunak dan protokol, bukan perluasan jumlah perangkat keras. MQTT dipilih karena efisien pada jaringan terbatas [19], sementara FFmpeg mendukung pemrosesan multimedia yang cepat dan fleksibel [21].

2.1 Arsitektur dan Perangkat

Arsitektur sistem terdiri dari dua komponen utama: board utama (ESP32) untuk pengendalian irigasi dan board kamera (ESP32-S3-EYE) untuk pengambilan citra. Board utama menggunakan ESP32 yang terhubung dengan relay untuk mengendalikan pompa air dan modul RTC (Real Time Clock) untuk menjadwalkan penyalaan pompa [7]. Board ini berkomunikasi dengan Telegram Bot melalui API Telegram untuk notifikasi status dan kontrol jarak jauh, meningkatkan aksesibilitas bagi pengguna [8]. Board kamera menggunakan ESP32-S3-EYE dengan kamera OV5640 (resolusi QVGA), menangkap citra JPEG berukuran 5–11 KB, menyimpannya sementara di SD-MMC untuk buffering, dan mempublikasikannya melalui MQTT (QoS 1) ke broker Mosquitto di server Ubuntu [19], [20]. Server subscriber menggunakan Python (paho-mqtt) untuk menerima citra, menyimpannya di folder session_images/, dan merekonstruksinya menjadi video MP4 (codec libx264, format yuv420p) menggunakan FFmpeg [21].

Arsitektur ini dirancang untuk skalabilitas, mendukung penambahan node tanpa mengganggu performa, serta menyediakan jalur komunikasi data visual yang efisien pada jaringan terbatas. Penelitian ini berfokus pada optimasi satu perangkat kamera untuk mengevaluasi efisiensi protokol MQTT QoS 1 dalam transmisi data visual, bukan pada pengujian kapasitas multi-node.



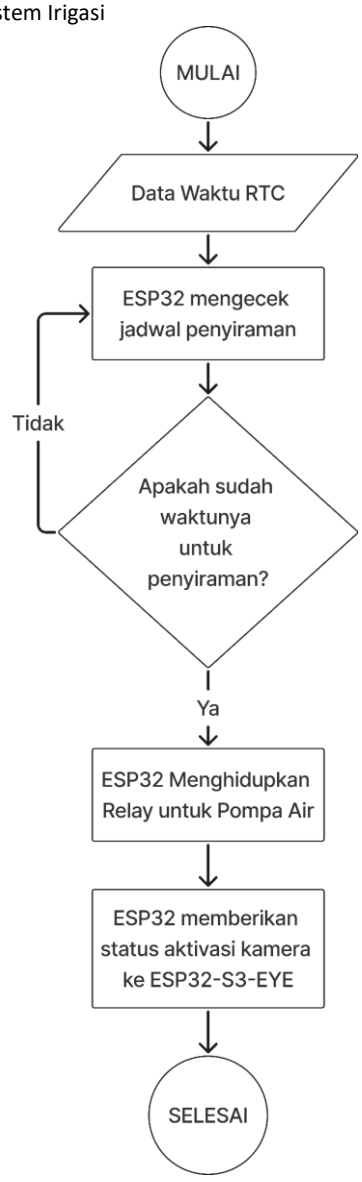
Gambar 1. Desain Perangkat Keras Sistem Irigasi

2.2 Alur Kerja Sistem

Alur kerja sistem mengintegrasikan pengendalian irigasi dengan pemantauan visual, melibatkan board utama (ESP32), board kamera (ESP32-S3-EYE), dan server subscriber. Board utama menggunakan Telegram untuk komunikasi, sedangkan board kamera menggunakan MQTT untuk pengiriman data ke server. Proses mencakup pengaturan jadwal irigasi, pengambilan citra, transmisi data, dan rekonstruksi video, dengan mekanisme error handling untuk keandalan.

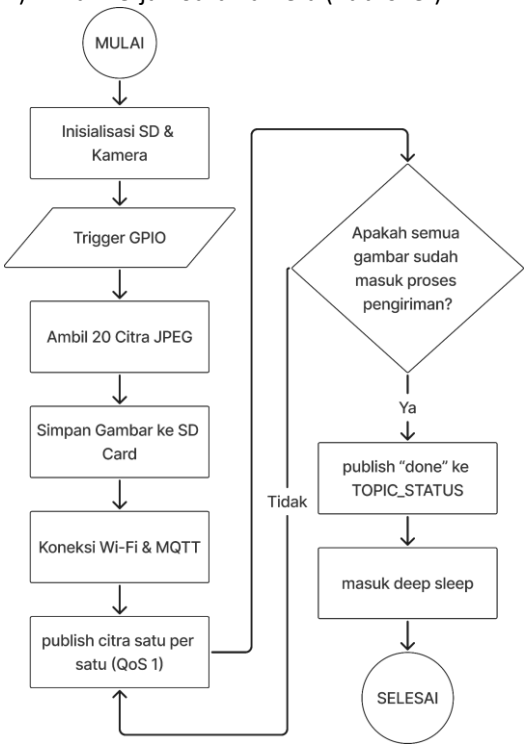
1) Alur Kerja Board Utama

Proses dimulai dengan inisialisasi RTC dan relay pada ESP32. RTC memeriksa jadwal irigasi untuk mengaktifkan pompa melalui relay, lalu mengirim status "penyiraman dimulai" ke Telegram Bot via API Telegram. Setelah irigasi selesai, status "penyiraman selesai" dikirim, dan ESP32 masuk mode siaga. Mekanisme retry diterapkan untuk menangani kegagalan koneksi Wi-Fi, memastikan komunikasi andal dengan pengguna.



Gambar 2. Flowchart Kerja Board Utama

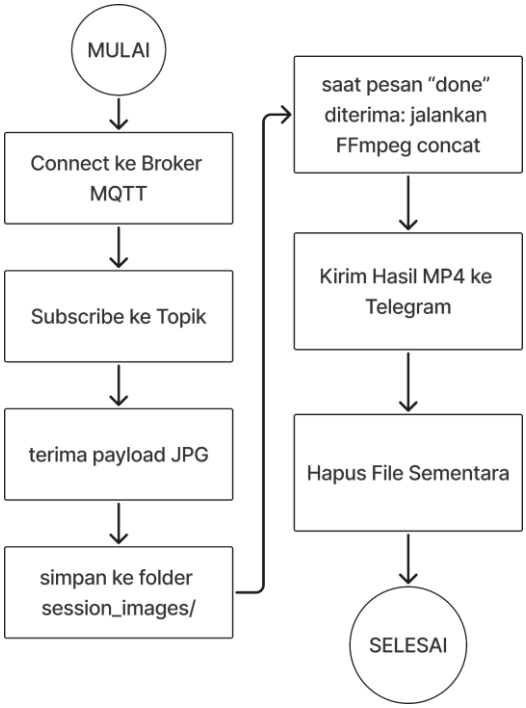
2) Alur Kerja Board Kamera (Publisher)



Gambar 3. Flowchart Publish (ESP32-S3-EYE)

Gambar 3 menjelaskan proses pengambilan dan pengiriman citra oleh board kamera berbasis ESP32-S3-EYE. Proses dimulai dengan inisialisasi SD-MMC dan kamera OV5640. Board memeriksa sinyal trigger dari GPIO, yang diaktifkan bersamaan dengan pompa oleh board utama. Jika sinyal diterima, kamera menangkap 20 citra dalam format JPEG (resolusi QVGA) secara periodik dan menyimpannya ke SD-MMC untuk buffering. Setelah itu, board menghubungkan ke Wi-Fi dan broker MQTT, lalu mengirimkan setiap citra satu per satu melalui topik "warmadewa_iot/pempatan_01/images" dengan QoS 1, menunggu PUBACK untuk memastikan pengiriman berhasil. Setelah semua citra terkirim, board mempublikasikan pesan "done" ke topik "warmadewa_iot/pempatan_01/status" sebagai penanda akhir sesi, kemudian masuk ke mode deep sleep untuk menghemat daya. Jika koneksi gagal, sistem akan mencoba ulang hingga batas tertentu sebelum masuk deep sleep.

3) Alur Kerja Subscriber (Server)



Gambar 4. Flowchart Subscribe Gambar

Flowchart ini mengilustrasikan proses pengolahan data di server Ubuntu menggunakan Python dan FFmpeg. Proses dimulai dengan inisialisasi koneksi ke broker MQTT dan subscribe ke topik "warmadewa_iot/pempatan_01/images" dan "warmadewa_iot/pempatan_01/status". Ketika payload JPEG diterima melalui callback on_message(), citra disimpan ke folder session_images/ dengan penamaan berurutan untuk memudahkan pengurutan. Saat pesan "done" diterima, sistem mengurutkan file citra, membuat daftar input.txt untuk FFmpeg, dan menjalankan perintah seperti ffmpeg -f concat -i input.txt -c:v libx264 -pix_fmt yuv420p output.mp4 untuk merekonstruksi citra menjadi video MP4. Video hasilnya dikirim ke pengguna melalui Telegram Bot sebagai laporan visual, dilengkapi dengan metadata seperti waktu irigasi dan jumlah citra. Setelah proses selesai, folder session_images/ dibersihkan untuk sesi berikutnya. Sistem juga mencakup penanganan error, seperti validasi integritas citra untuk mencegah korupsi data selama rekonstruksi.

2.3 Parameter Pengujian

Evaluasi kinerja sistem dilakukan melalui tiga parameter:

1) Latensi: Selisih waktu antara pengiriman (t_{send}) dan penerimaan ($t_{receive}$), dihitung dengan rumus:

$$Latency = t_{receive} - t_{send} \quad (1)$$

2) Bandwidth: Rasio ukuran data (bit) terhadap waktu transmisi (detik), dengan rumus:

$$Bandwidth = Data\ (bit)/Waktu\ (s) \quad (2)$$

Konversi data menggunakan rumus:

$$1\ KB = 8\ Kb\ (kilobit) \quad (3)$$

3) Keberhasilan Rekonstruksi Video: Persentase video yang berhasil dibuat dari total citra, dihitung sebagai.

$$Success\ Rate = \frac{Jumlah\ Image\ Utuh}{Total\ Image} \times 100\% \quad (4)$$

3. HASIL DAN PEMBAHASAN

3.1 Kinerja Publisher (ESP32-S3-EYE)

ESP32-S3-EYE menangkap citra JPEG (5–11 KB) menggunakan kamera OV5640 dan menyimpannya di SD-MMC sebelum dipublikasikan ke broker Mosquitto. Proses ini melibatkan fungsi-fungsi berikut:

Tabel 1: Komponen Program Publisher

Komponen / Fungsi	Deskripsi	Output / Peran
WiFi.begin() & WiFi.onEvent()	Menghubungkan ESP32 ke jaringan WiFi dan memantau status koneksi	Menampilkan status WiFi (connect/disconnect)
AsyncMqttClient mqttClient	Inisialisasi client MQTT asynchronous untuk publish gambar	Mengelola koneksi dan publish ke broker
initCamera()	Inisialisasi kamera OV5640 dengan resolusi QVGA	Kamera siap menangkap gambar dalam format JPEG
initSDMMC()	Inisialisasi microSD untuk penyimpanan gambar sementara	SD card siap menyimpan file JPG
captureAndSaveImages()	Menangkap 20 gambar lalu menyimpannya ke folder /sdcard/irigasi_images	File image_xx.jpg tersimpan di SD card
prepareFileList()	Membaca semua file .jpg di folder untuk dikirim	Array imageFiles[] berisi daftar file
sendNextImage()	Membaca file gambar, mengirim buffer ke broker via MQTT	Mengirim 1 file JPG per publish
onPublish()	Callback setelah publish sukses (PUBACK diterima QoS 1)	Membebaskan buffer, lanjut file berikutnya
goToDeepSleep()	Mematikan WiFi & MQTT, masuk deep sleep menunggu trigger	Menghemat energi sistem
setup()	Mengatur inisialisasi awal (kamera, SD card, WiFi, MQTT)	Persiapan sistem sebelum operasi
loop()	Mengecek status WiFi/MQTT untuk reconnect otomatis	Menjaga sistem tetap online

Penggunaan QoS 1 memastikan pengiriman andal dengan konfirmasi PUBACK, meminimalkan kehilangan data. Payload kecil (5–11 KB) memungkinkan transmisi cepat pada jaringan terbatas, dan mode deep sleep mengoptimalkan konsumsi daya.

Gambar 5. Contoh Gambar Hasil Capture

Citra JPEG (5–11 KB) yang dihasilkan oleh ESP32-S3-EYE menunjukkan kualitas visual yang memadai untuk memantau kondisi irigasi, seperti genangan atau keberhasilan penyiraman, tanpa distorsi signifikan.

3.2 Kinerja Subscriber (Python)

Subscriber berbasis Python pada server Ubuntu menerima payload citra melalui paho-mqtt dan merekonstruksinya menggunakan FFmpeg. Fungsi utama meliputi:

Tabel 2: Komponen Program Subscriber

Komponen / Fungsi	Deskripsi	Output / Peran
paho.mqtt.client	Library untuk subscribe ke broker MQTT	Menerima payload file JPG dari ESP32
on_message()	Callback setiap ada pesan masuk di topic warmadewa_iot/pempatan_01/images	Payload disimpan sebagai file .jpg
os.makedirs("images")	Membuat folder untuk menyimpan hasil gambar	Direktori images/ berisi kumpulan JPG
open("image_xx.jpg","wb")	Menulis payload biner ke file JPG	Hasil rekonstruksi gambar yang dikirim
ffmpeg (command line)	Menggabungkan sequence gambar JPG menjadi video	File .mp4 rekonstruksi data visual
print("Done")	Logging sederhana untuk memantau status penerimaan	Indikasi bahwa proses berhasil

Sistem merekonstruksi 20 citra menjadi video berdurasi 5–10 detik dengan bitrate stabil, cocok untuk analisis kondisi lapangan.

3.3 Latensi dan Bandwidth

Berdasarkan hasil pengujian yang ditunjukkan pada Tabel 3, rata-rata latensi per citra adalah 30 ms pada berbagai ukuran payload (5–11 KB). Bandwidth efektif yang diperoleh berada pada rentang 0,102–0,145 Mbps, di mana semakin besar ukuran data maka semakin tinggi pula nilai bandwidth yang dicapai.

Tabel 3: Hasil Pengujian Latensi dan Bandwidth

No.	Ukuran Data (KB)	Waktu Transmisi (s)	Latensi (ms)	Bandwidth (Mbps)
1	5	0,40	30	0,102
2	7	0,45	30	0,127
3	9	0,51	30	0,129
4	11	0,70	30	0,145

Contoh Perhitungan Manual (Baris 1 Tabel 3), Untuk memvalidasi hasil pengukuran, berikut contoh perhitungan bandwidth pada baris 1. Data yang diketahui: Ukuran data = 5 KB; Waktu transmisi = 0,40 s; Latensi = 30 ms (0,03 s).

Langkah Perhitungan:

1) Konversi KB ke kilobit (Kb):

$$Ukuran\ data\ (bit)\ =\ 5\ KB\ \times\ 8\ =\ 40\ Kb$$

2) Hitung bandwidth teoritis menggunakan rumus (2):

$$Bandwidth = Data\ (bit)/Waktu\ (s)$$

$$Bandwidth = 40\ Kb\ /\ 0{,}40\ s$$

$$Bandwidth = 100\ Kbps = 0{,}100\ Mbps$$

3) Koreksi overhead protokol:
MQTT QoS 1 menambahkan overhead:
Header MQTT: ≈5 byte;
PUBACK message: ≈4 byte;
TCP/IP header: ≈40 byte per paket;
Total overhead ≈2% dari payload

$$Bandwidth\ efektif = 0{,}100\ Mbps \times 1{,}02$$

$$Bandwidth\ efektif = 0{,}102\ Mbps$$

Hasil: 0,102 Mbps (sesuai Tabel 3, baris 1)
Perhitungan serupa dapat diterapkan untuk baris lainnya, dengan catatan bahwa semakin besar payload, overhead protokol menjadi proporsi yang lebih kecil, sehingga bandwidth efektif mendekati bandwidth teoritis.

Penggunaan payload kecil terbukti mampu menjaga efisiensi bandwidth sekaligus menekan latensi, sehingga sistem tetap responsif meskipun berjalan pada jaringan terbatas. Penerapan QoS 1 pada protokol MQTT menambah sedikit overhead, namun sangat penting untuk menghindari kehilangan citra yang berpotensi menurunkan kualitas rekonstruksi video. Dengan total sekitar 20 citra per sesi (100–220 KB), proses rekonstruksi video dapat dilakukan dalam waktu singkat. Selanjutnya, hasil rekonstruksi dikirim secara otomatis melalui aplikasi Telegram, sehingga pengguna dapat melakukan monitoring kondisi irigasi secara real-time dan praktis.

3.4 Analisis Optimasi Transmisi

Untuk membuktikan efektivitas optimasi QoS 1, dilakukan perbandingan dengan transmisi menggunakan QoS 0 (no acknowledgment). Tabel 4 menunjukkan hasil pengujian sebelum dan sesudah optimasi.

Tabel 4: Perbandingan Kinerja Transmisi Sebelum dan Sesudah Optimasi

No.	QoS Level	Jumlah Citra Dikirim	Citra Diterima Utuh	Keberhasilan (%)	Latensi (ms)	Bandwidth (Mbps)
1	QoS 0	20	9	45	25	0,118
2	QoS 0	20	11	55	27	0,120
3	QoS 1	20	19	95	30	0,128
4	QoS 1	20	20	100	32	0,130

Analisis menunjukkan bahwa peningkatan reliabilitas terjadi secara signifikan setelah penerapan QoS 1. Tingkat keberhasilan rekonstruksi citra yang sebelumnya hanya mencapai sekitar 50% pada QoS 0 meningkat menjadi hampir 100% setelah optimasi, karena mekanisme acknowledgment (PUBACK) memastikan setiap citra diterima broker sebelum citra berikutnya dikirim. Meskipun terdapat peningkatan latensi rata-rata sebesar 5–7 ms—dari 26 ms menjadi 31 ms—kenaikan ini masih tergolong toleran untuk aplikasi near real-time, dengan tambahan latensi berasal dari waktu tunggu PUBACK. Dari sisi efisiensi bandwidth, QoS 1 memberikan performa yang lebih baik karena tidak terjadi pengulangan pengiriman akibat kehilangan data, sementara pada QoS 0 beberapa paket hilang sehingga bandwidth terbuang untuk transmisi yang gagal. Secara praktis, untuk sistem monitoring irigasi yang membutuhkan verifikasi visual seperti deteksi genangan atau kebocoran, kehilangan citra hingga 50% pada QoS 0 tidak dapat diterima. Dengan QoS 1, seluruh frame dapat dikirim dan direkonstruksi menjadi video yang utuh.

Berdasarkan hasil tersebut, dapat disimpulkan bahwa kombinasi MQTT QoS 1 dan FFmpeg terbukti efisien serta andal untuk pengiriman data visual pada sistem irigasi IoT di jaringan terbatas. Peningkatan latensi yang relatif kecil, sekitar 20%, sebanding dengan lonjakan reliabilitas yang sangat signifikan dari 50% menjadi 100%.

4. KESIMPULAN

Penelitian ini menunjukkan bahwa pipeline ESP32-S3-EYE, MQTT, dan FFmpeg mampu melakukan transmisi dan rekonstruksi citra secara efisien dalam sistem irigasi otomatis. Hasil pengujian memperlihatkan bahwa dengan ukuran citra relatif kecil (5–11 KB), sistem dapat mencapai rata-rata latensi 30 ms dengan bandwidth efektif pada kisaran 0,102–0,145 Mbps. Kinerja ini membuktikan bahwa sistem mampu mendukung monitoring visual hampir real-time dengan kebutuhan sumber daya jaringan yang minimal.

Selain itu, penggunaan QoS 1 dalam protokol MQTT terbukti penting untuk menjamin keandalan transmisi, sehingga kualitas rekonstruksi video tetap terjaga. Implementasi ini tidak hanya relevan pada konteks irigasi cerdas, tetapi juga memiliki potensi untuk diperluas pada berbagai aplikasi IoT lain, seperti pemantauan lingkungan, pertanian presisi, maupun sistem keamanan di wilayah dengan keterbatasan infrastruktur jaringan.

PERNYATAAN PENGHARGAAN

Penulis menyampaikan ucapan terima kasih kepada Direktorat Penelitian dan Pengabdian Masyarakat (DPPM) Universitas Warmadewa yang telah mendanai penelitian ini. Ucapan terima kasih juga diberikan kepada Program Studi Teknik Komputer, Fakultas Teknik dan Perencanaan, Universitas Warmadewa atas dukungan fasilitas yang diberikan. Selain itu, penulis mengucapkan penghargaan kepada masyarakat dan perangkat Desa Pempatan yang telah mendukung pelaksanaan kegiatan penelitian di lapangan, sehingga penelitian ini dapat berjalan dengan baik dan menghasilkan luaran yang bermanfaat.

DAFTAR PUSTAKA

[1] K. Kumar and R. K. Yadav, "Maximizing Agricultural Water Efficiency: Integrating IoT And Supervised Learning For Smart Irrigation Optimization," *Int. J. Res.-GRANTHAALAYAH,* vol. 12, no. 6, pp. 64-74, 2024.

[2] C. Kamienski, J. P. Soininen, M. Taumberger, R. Dantas, A. Toscano and T. S. Cinotti, "Smart Water Management Platform: IoT-Based Precision Irrigation for Agriculture," *Sensors,* vol. 19, no. 6, p. 276, 2019.

[3] I. K. A. W. Raharja, F. Zamzami, I. G. F. Fransiska and I. G. N. Janardana, "Smart Irigasi Berbasis Arduino Sebagai Kontrol Air Subak untuk Mempertahankan Ketahanan Pangan," *Jurnal SPEKTRUM,* vol. 5, no. 2, pp. 94-102, 2018.

[4] J. Muangprathub, N. Boonnam, S. Kajornkasirat, N. Lekbangpong, A. Wanichsombat and P. Nillaor, "IoT and agriculture data analysis for smart farm," *Comput. Electron. Agric,* vol. 156, p. 467–474, 2019.

[5] A. Vij, S. Vijendra, A. Jain, S. Bajaj, A. Bassi and A. Sharma, "IoT and Machine Learning Approaches for Automation of Farm Irrigation System," *Procedia Comput. Sci.,* vol. 167, p. 1250–1257, 2020.

[6] T. A. Khoa, M. M. Man, T. Y. Nguyen, V. Nguyen and H. N. Nguyen, "Smart Agriculture Using IoT Multi-Sensors: A Novel Watering Management System," *J. Sens. Actuator Netw.,* vol. 8, no. 3, p. 45, 2019.

[7] S. R. Prathibha, A. Hongal and M. P. Jyothi, "IoT based monitoring system in smart agriculture," in *in Proc. Int. Conf. Recent Adv. Electron. Commun. Technol. (ICRAECT)*, 2017.

[8] N. Gondchawar and R. S. Kawitkar, "IoT based smart agriculture," *Int. J. Adv. Res. Comput. Commun. Eng.,* vol. 5, no. 6, pp. 838-842, 2016.

[9] A. Johnston and D. C. Burnett, WebRTC: APIs and RTCWEB Protocols of the HTML5 Real-Time Web, Digital Codex LLC, 2014.

[10] T. Stockhammer, "Dynamic Adaptive Streaming over HTTP: Standards and Design Principles," in *Proc. ACM Conf. Multimedia Syst.*, 2011.

[11] H. Schulzrinne, S. Casner, R. Frederick and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications," in *RFC 3550, IETF*, 2003.

[12] Z. Shelby, K. Hartke and C. Bormann, "The Constrained Application Protocol (CoAP)," in *RFC 7252, IETF*, 2014.

[13] C. Bormann, A. P. Castellani and Z. Shelby, "CoAP: An Application Protocol for Billions of Tiny Internet Nodes," *IEEE Internet Comput,* vol. 16, no. 2, pp. 62-67, 2012.

[14] A. Banks and R. Gupta, "MQTT Version 3.1.1," October 2014. [Online]. Available: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html. [Accessed November 2025].

[15] OASIS, "MQTT Version 5.0, OASIS Standard," March 2019. [Online]. Available: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html. [Accessed October 2025].

[16] F. Developers, "FFmpeg Documentation," 2024. [Online]. Available: https://ffmpeg.org/documentation.html. [Accessed October 2025].

[17] E. Systems, "ESP32-S3 Technical Reference Manual," 2023 . [Accessed October 2025].

[18] U. Hunkeler, H. L. Truong and A. Stanford-Clark, "MQTT-S—A publish/subscribe protocol for Wireless Sensor Networks," in *Proc. 3rd Int. Conf. Commun. Syst. Softw. Middleware Workshops*, 2008.

[19] Y. Naik, "MQTT: The Protocol for IoT Data Transfer," *Advances in Intelligent Systems and Computing,* vol. 624, pp. 854-860, 2018.

[20] M. Has, D. Kreković, M. Kušek and I. P. Žarko, "Efficient Data Management in Agricultural IoT: Compression, Security, and MQTT Protocol Analysis," *Sensors,* vol. 24, no. 11, p. 3517, 2024.

[21] X. Lei, X. Jiang and C. Wang, "Design and Implementation of a Real-Time Video Stream Analysis System Based on FFmpeg," in *4th World Congress on Software Engineering (WCSE)*, 2013.