JURNAL MEDIA INFORMASI TEKNOLOGI
Volume 3, No. 1, Februari 2026
DOI: https://doi.org/10.69616/mit.v3i1.267

Implementation of Event-Driven Architecture in Soil Moisture Sensor Data Transmission Based on Master–Slave

Cut Chairunnisa Maulidia[1*], Hafiz Muhardi[1], Uray Ristian[1], Nirsal[2], Idawati[3]
[1]Universitas Tanjungpura, Indonesia.
[2]Universitas Cokroaminoto Palopo, Indonesia.
[3]Universitas Andi Jemma Palopo, Indonesia.
***** Corresponding Author. E-mail: h1051201071@student.untan.ac.id

Article History
Received:
Feb 13th, 2026

Revised:
Jul 31th, 2026

Accepted:
Aug 1st, 2026

Keywords
Event-Driven,
ESP32,
IoT,
Master-Slave,
QoS,
Soil Moisture.

ABSTRACT
The decline in chili crop productivity is partly caused by Soil Moisture that is not properly monitored. Manual monitoring is considered less effective, while IoT systems with periodic data transmission tend to be inefficient. This study develops a Soil Moisture monitoring system based on an Event-Driven master–slave architecture using four ESP32 devices. Three act as slaves connected to Soil Moisture sensors, and one serves as the master responsible for data transmission. The Event-Driven method applies two mechanisms: Batch Update every 5 minutes under normal conditions and Immediate Alert when moisture values fall outside the ideal range of 50–70%. System performance is compared with a non-Event-Driven method based on QoS parameters: delay, jitter, payload, and the number of data packets. Testing is conducted through four scenarios, each lasting one hour. Results show that the Event-Driven system is more efficient. The average delay and jitter are 547.35 ms and 557.05 ms, lower than the non-Event-Driven system (3,340.73 ms and 1,755.71 ms). In addition, fewer packets are sent with larger payloads. Although it does not yet meet the TIPHON standard, this method can transmit data immediately during significant changes while reducing the number of transmitted packets.

This is an open access article under the CC-BY-SA license.

PENDAHULUAN

Di Indonesia, tanaman cabai merupakan salah satu komoditas hortikultura unggulan yang banyak dibudidayakan oleh petani karena memiliki nilai ekonomi yang tinggi serta permintaan pasar yang terus meningkat. Meskipun demikian, produktivitas cabai nasional masih tergolong rendah dengan rata-rata produksi hanya mencapai 6,7 ton per hektar [1]. Kondisi tersebut dipengaruhi oleh berbagai faktor, salah satunya adalah perubahan cuaca yang berdampak langsung terhadap tingkat kelembapan tanah sebagai media tumbuh tanaman [2]. Kelembapan tanah yang tidak sesuai dapat menghambat pertumbuhan dan menurunkan hasil panen, padahal kisaran kelembapan yang ideal untuk budidaya cabai berada pada rentang 50%–70% [3]. Oleh karena itu, pemantauan kelembapan tanah menjadi aspek penting dalam mendukung pertumbuhan tanaman secara optimal. Namun, proses monitoring suhu dan kelembapan secara manual masih kurang efektif karena memerlukan waktu, tenaga, serta tidak dapat memberikan informasi kondisi lahan secara real-time. Untuk mengatasi permasalahan tersebut, dikembangkan sistem monitoring berbasis website yang memungkinkan pengguna memantau kondisi kelembapan tanah dengan lebih mudah dan efisien [4].

Pengembangan sistem monitoring tersebut didukung oleh teknologi komunikasi nirkabel (*wireless*) yang memungkinkan proses pengiriman informasi berlangsung dengan cepat tanpa memerlukan kabel, memiliki jangkauan komunikasi yang luas, serta dapat memanfaatkan koneksi internet [5]. Pada penelitian ini, mekanisme komunikasi data menerapkan metode *Master-Slave*, yaitu satu node *master* bertugas mengatur proses penerimaan data yang dikirimkan oleh beberapa node *slave*. Pendekatan ini relatif sederhana karena hanya menggunakan satu *master* sebagai pusat komunikasi sehingga proses pengiriman dan pengelolaan data menjadi lebih terstruktur [6].

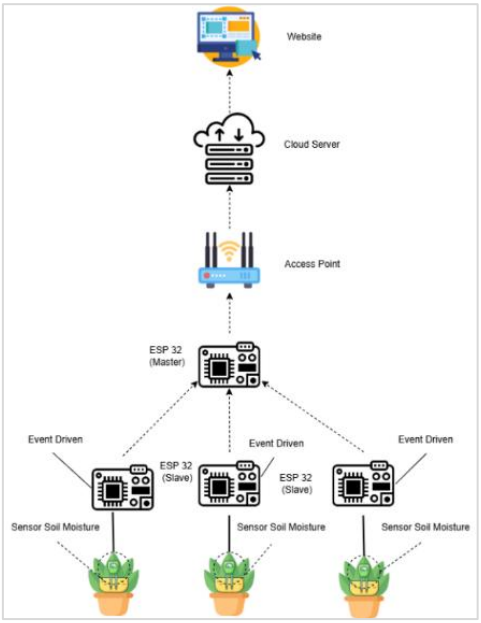
Metode *Event-Driven Architecture (EDA)* [7], [8] merupakan arsitektur terdistribusi yang dirancang untuk memenuhi kebutuhan aplikasi dengan skalabilitas dan performa tinggi. Dalam arsitektur ini, sistem akan menjalankan proses hanya setelah menerima satu atau beberapa *Event* dari sistem lain. Arsitektur *Event-Driven* (EDA) merupakan suatu pendekatan dalam perancangan sistem di mana setiap kejadian atau peristiwa tertentu berfungsi sebagai pemicu bagi pelaksanaan suatu proses [9], [10]. Arsitektur ini telah banyak diterapkan dalam pengembangan sistem informasi, khususnya di bidang industri, karena kemampuannya yang lebih adaptif dan efisien dalam mengendalikan jalannya aplikasi dibandingkan model arsitektur konvensional [11].

Berdasarkan penelitian terdahulu, maka diangkat penelitian berjudul "Penerapan *Event-Driven* Arsitektur Dalam Transmisi Pengiriman Data Sensor ESP-32 Berbasis *Master-Slave*". Penelitian ini bertujuan menerapkan sistem *Event-Driven* dalam pengiriman data sensor dengan mengirimkan data hanya saat terjadi perubahan kondisi kelembapan tanah. Dengan menggabungkan arsitektur *Master-Slave* dan *Event-Driven*, sistem diharapkan mampu merespons perubahan kelembapan tanah lebih cepat serta menghemat penyimpanan data dan penggunaan bandwidth.

METODE

Perancangan Sistem

Analisis kebutuhan sistem meliputi penentuan perangkat keras dan perangkat lunak yang digunakan. Perangkat keras yang digunakan terdiri dari ESP32*, Expansion Board ESP32*, sensor *Soil Moisture*, dan adaptor 12V 1A. Sementara itu, perangkat lunak yang digunakan meliputi *Arduino IDE* untuk pemrograman mikrokontroler, *Visual Studio Code* sebagai editor.



Gambar 1. Arsitektur Sistem Monitoring Tanaman

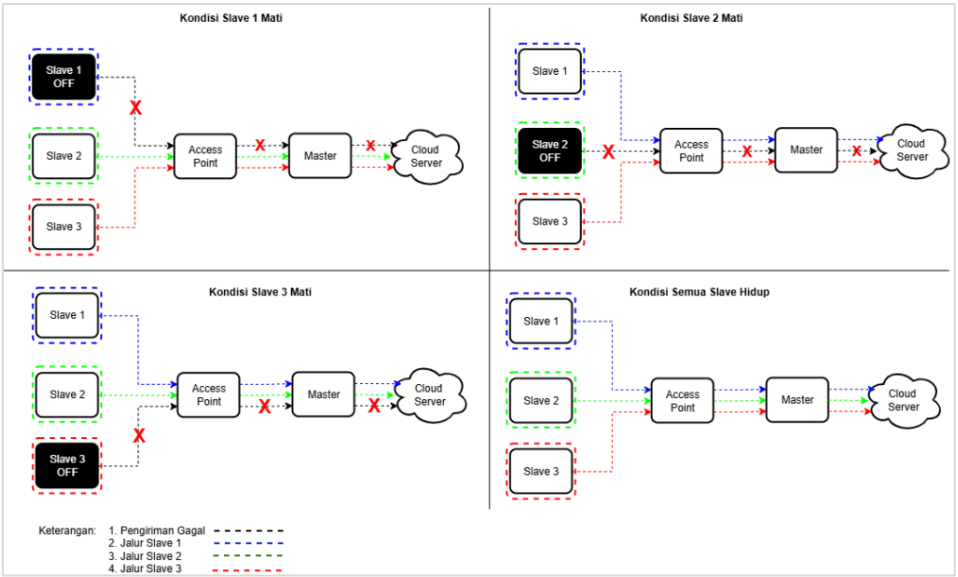
Penelitian ini mengembangkan sistem monitoring kelembapan tanah berbasis arsitektur *Event-Driven* dengan penerapan sistem *Master-Slave*. *Slave* membaca data sensor *Soil Moisture*, kemudian mengirimkan data ke *master* melalui mekanisme *Immediate Alert* apabila terdeteksi

kondisi kelembapan ekstrem, sedangkan data pada kondisi normal dikirim melalui *Batch Update*. Data yang diterima *master* selanjutnya diolah untuk perhitungan parameter *Quality Of Service* (*QoS*) [12], [13], meliputi delay, jitter, dan *payload*, [14], [15] kemudian dikirim ke *cloud server* untuk disimpan di *database* dan ditampilkan pada dashboard monitoring. Arsitektur Sistem Monitoring Tanaman dilihat pada Gambar 1.

Perancangan perangkat keras terdiri atas satu node *master* dan tiga node *slave* berbasis ESP32. Setiap *slave* dilengkapi tiga sensor *Soil Moisture* pada pin IO33, IO34, dan IO35 untuk membaca kelembapan tanah pada beberapa titik pengukuran. Node *master* bertugas menerima dan mengumpulkan data dari seluruh *slave*, menambahkan informasi waktu sinkronisasi, kemudian meneruskannya ke server.

Komunikasi antara node *slave* dan *master* dilakukan melalui jaringan Wi-Fi menggunakan protokol HTTP (*POST*). Setiap *slave* mengirimkan data sensor dalam format JSON melalui mekanisme *Immediate Alert* ketika terjadi kondisi ekstrem dan *Batch Update* pada kondisi normal. Selanjutnya, node *master* menerima data tersebut dan meneruskannya ke server untuk proses penyimpanan dan monitoring.

Implementasi metode *Event-Driven* dilakukan dengan membagi mekanisme pengiriman data menjadi dua skema, yaitu *Immediate Alert* dan *Batch Update*. Node *slave* membaca data kelembapan tanah kemudian mengirimkan data secara langsung melalui *Immediate Alert* apabila nilai berada pada kondisi ekstrem atau terjadi perubahan yang signifikan. Pengujian dilakukan menggunakan empat skenario, yaitu: (1) *slave* 1 dan *slave* 2 aktif sementara *slave* 3 tidak aktif; (2) *slave* 1 dan *slave* 3 aktif sementara *slave* 2 tidak aktif; (3) *slave* 2 dan *slave* 3 aktif sementara *slave* 1 tidak aktif; dan (4) seluruh *slave* aktif. Keempat skenario metode *Event-Driven* ditunjukkan pada Gambar 2.



Gambar 2. Empat Skenario Metode *Event-Driven*

Perancangan basis data menggunakan MySQL sebagai sistem manajemen basis data yang mengelola hasil pembacaan sensor *Soil Moisture* dari setiap *slave* serta parameter QoS yang meliputi *delay*, *jitter*, dan *payload*. Basis data dirancang dalam bentuk relasi antar tabel sehingga mampu mendukung penyimpanan dan pengelolaan data monitoring secara terstruktur.

Antarmuka website berupa Dashboard Monitoring dirancang sebagai halaman utama sistem SmartGarden yang menampilkan kondisi kelembapan tanah dari setiap *slave*. Dashboard menyajikan informasi berupa rata-rata *delay*, rata-rata *jitter*, rata-rata ukuran *payload*, serta tabel pembacaan sensor *Soil Moisture* secara *real-time*.

Metode Pengujian

Pengujian sistem dilakukan dengan membandingkan dua pendekatan, yaitu tanpa metode *Event-Driven* dan menggunakan metode *Event-Driven*. Masing-masing pendekatan diuji selama satu jam pada setiap skenario pengujian, dengan setiap *slave* mengirimkan data ke server setiap lima menit. Pada pengujian menggunakan metode *Event-Driven*, sistem juga mencatat riwayat kejadian (*event history*) serta parameter *Quality of Service* (QoS) yang meliputi *delay*, *jitter*, dan *payload*.

HASIL DAN PEMBAHASAN

Pada sistem monitoring, Implementasi sistem monitoring pada perangkat *Slave* dilakukan untuk memastikan setiap perangkat dapat membaca, mengolah, dan mengirim data kelembapan tanah ke *master*. Setiap *slave* menggunakan mikrokontroler ESP32 yang terhubung dengan sensor *Soil Moisture* untuk membaca tingkat kelembapan tanah secara berkala. Hasil pembacaan sensor kemudian dikonversi ke dalam bentuk persentase dan dikategorikan ke dalam beberapa kondisi seperti kering, lembab, dan basah. Implementasi Sistem *Monitoring* pada Node *Slave* dapat dilihat pada Gambar 3.

Gambar 3. Implementasi Sistem *Monitoring* pada *Slave*

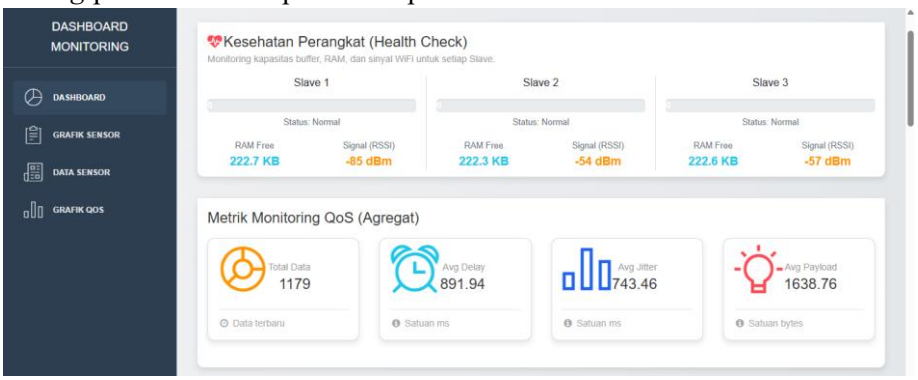
Pada sistem *monitoring master*, Komunikasi antar pada sistem ini dilakukan melalui koneksi WiFi dengan protokol *HTTP*, di mana ESP32 *Master* berfungsi sebagai penerima dan pengelola data, sedangkan ESP32 *Slave* mengirimkan hasil pembacaan sensor kelembapan tanah. Implementasi Sistem *Monitoring* pada *Master* dapat dilihat pada Gambar 4.

Gambar 4. Implementasi Sistem Monitoring pada *Master*

Pada Implementasi pada node *master* dilakukan dengan menjadikan ESP32 sebagai pusat komunikasi yang menerima data dari seluruh node *slave* melalui protokol *HTTP*. Data yang diterima terdiri dari hasil pembacaan sensor yang dikirim melalui mekanisme *Batch Update* maupun *Immediate Alert*, kemudian *master* menambahkan informasi waktu, menyimpan data secara lokal, serta meneruskannya ke server untuk proses penyimpanan dan monitoring melalui dashboard sistem. Implementasi sistem komunikasi antara *master* dan *slave* dilihat pada Gambar 5.

Gambar 5. Implementasi Komunikasi *Master* dan *Slave*

Implementasi komunikasi antara *Master* dan *Slave* pada sistem ini dilakukan dengan memanfaatkan jaringan WiFi lokal sebagai media pertukaran data. Setiap *Slave* membaca nilai kelembapan tanah dari sensor *Soil Moisture*, kemudian mengirimkan data tersebut ke *Master* melalui protokol *HTTP* menggunakan metode POST. *Master* berfungsi sebagai pusat pengendali yang menerima seluruh data dari *Slave*, menambahkan timestamp, serta disimpan ke server untuk ditampilkan pada halaman monitoring. Komunikasi ini berjalan dua arah, di mana *Master* tidak hanya menerima data, tetapi juga dapat memantau status koneksi masing-masing *Slave*. Tampilan *website* sistem *monitoring* penelitian ini dapat dilihat pada Gambar 6.



Gambar 6. Tampilan *Website* Sistem *Monitoring* Kelembapan Tanah Tanaman

Hasil Implementasi

Pengujian sistem dilakukan dengan membandingkan dua skenario, yaitu pengujian sistem dengan penerapan arsitektur *Event-Driven* dan pengujian sistem tanpa *Event-Driven*, untuk menilai pengaruh mekanisme *Immediate Alert* dan *Batch Update* terhadap proses pengiriman data sensor pada komunikasi *master–slave*. Selain itu, dilakukan pula pengujian black box untuk memastikan bahwa seluruh fitur sistem, mulai dari pengiriman data sensor hingga tampilan dashboard monitoring, telah berjalan sesuai dengan rancangan yang ditetapkan.

Pengujian Sistem Pemantauan Tanaman Berbasis Arsitektur *Event-Driven*

Pengujian sistem dilakukan dengan menerapkan mekanisme *Batch Update*, di mana setiap *Slave* membaca nilai kelembapan tanah setiap 10 detik dan menyimpan data ke dalam buffer untuk kemudian dikirim secara berkala setiap 5 menit dalam satu paket data ke *Master*. Pengujian berlangsung selama 1 jam untuk setiap kombinasi skenario, yaitu *Slave* 1–2, *Slave* 1–3, *Slave* 2–3, dan seluruh *Slave* aktif secara bersamaan. Selama pengujian, *Master* mencatat parameter *Quality Of Service* (*QoS*), meliputi delay, jitter, ukuran *payload*, dan jumlah paket yang diterima pada setiap proses pengiriman batch, kemudian hasilnya disimpan ke server dan ditampilkan melalui dashboard SmartGarden. Pengujian menggunakan metode *Event-Driven* dapat dilihat pada Tabel 1.

Tabel 1. Pengujian Menggunakan Metode *Event-Driven*

Pengujian	Status *Slave*	*Delay (ms)*	*Jitter (ms)*	*Payload (byte)*	Pengiriman Paket (kali)
1	*Slave* 1 & *Slave* 2 hidup, *Slave* 3 mati.	340,91	498,25	1494,88	23
2	*Slave* 1 & *Slave* 3 hidup, *Slave* 2 mati.	503,28	437,08	1522,22	23
3	*Slave* 2 & *Slave* 3 hidup, *Slave* 1 mati.	750,23	482,91	1530,26	23
4	Semua *Slave* 1, *Slave* 2, dan *Slave* 3 hidup.	594,96	809,95	1515,05	34
Rata-rata Pengiriman		547,35	557,05	1515,60	25,75

Berdasarkan Tabel 1, hasil pengujian menunjukkan bahwa banyaknya *slave* yang aktif berpengaruh terhadap nilai *delay, jitter*, dan ukuran *payload* pada proses pengiriman data. Pada skenario dua, nilai delay dan jitter cenderung lebih tinggi karena pengiriman data lebih sering terjadi akibat pemicu *Immediate Alert*. Sementara itu, ketika semua *slave* aktif, sistem cenderung lebih stabil dengan jumlah paket lebih banyak dan ukuran *payload* yang lebih besar pada setiap pengiriman karena lebih banyak data yang dikirim melalui mekanisme *Batch Update*. Secara umum, hasil ini menunjukkan bahwa metode *Event-Driven* dapat mengurangi frekuensi pengiriman data dengan cara menggabungkan beberapa data dalam satu paket, meskipun nilai delay dan jitter masih dipengaruhi oleh kondisi lalu lintas data pada setiap skenario pengujian. Pengujian tanpa menggunakan metode *Event-Driven* dapat dilihat pada tabel Tabel 2.

Tabel 2. Pengujian Tanpa Menggunakan Metode *Event-Driven*

Pengujian	Status *Slave*	*Delay* (ms)	*Jitter* (ms)	*Payload* (byte)	Pengiriman Paket (kali)
1	*Slave* 1 & *Slave* 2 hidup, *Slave* 3 mati.	3072,16	1816,75	103,00	868
2	*Slave* 1 & *Slave* 3 hidup, *Slave* 2 mati.	3922,90	1667,09	102,99	895
3	*Slave* 2 & *Slave* 3 hidup, *Slave* 1 mati.	3758,65	1683,24	102,99	945
4	Semua *Slave* 1, *Slave* 2, dan *Slave* 3 hidup.	2609,22	1855,77	102,98	1339
Rata-rata Pengiriman		3340,73	1755,71	102,99	1,001

Berdasarkan Tabel 2, hasil pengujian tanpa metode *Event-Driven* menunjukkan bahwa jumlah paket data yang dikirim menjadi lebih banyak dengan ukuran *payload* yang lebih kecil pada setiap pengiriman, karena data dikirim secara periodik tanpa mempertimbangkan kondisi perubahan nilai sensor, sementara nilai delay dan jitter cenderung lebih stabil namun kurang efisien dibandingkan skenario *Event-Driven*. Hasil pengujian menunjukkan bahwa sistem tanpa *Event-Driven* kurang efisien dalam penggunaan jaringan karena mengirim lebih banyak paket data selama proses monitoring. Namun, nilai delay dan jitter cenderung lebih stabil, sehingga metode ini masih dapat digunakan pada sistem yang sederhana dan tidak membutuhkan pengiriman data yang terlalu efisien, meskipun tetap kalah efektif dibandingkan metode *Event-Driven* dalam mengurangi jumlah pengiriman data. Tabel perbandingan dapat dilihat pada table Tabel 3.

Tabel 3. Tabel Perbandingan

Pengujian	Metode *Event-Driven*	Metode Non-*Event-Driven*	Selisih
1	23	868	845
2	23	895	872
3	23	945	922
4	34	1339	1305
Rata-rata Pengiriman	**25,75**	**1,001**	**986**

Berdasarkan Tabel 3, dapat dilihat bahwa jumlah pengiriman paket pada metode *Event-Driven* jauh lebih sedikit dibandingkan metode Non-*Event-Driven* pada seluruh skenario pengujian. Pada Pengujian 1, 2, dan 3, metode *Event-Driven* masing-masing hanya melakukan 23 kali pengiriman paket, sedangkan pada Pengujian 4 jumlah pengiriman meningkat menjadi 34 kali karena seluruh slave aktif secara bersamaan. Sebaliknya, metode Non-*Event-Driven* menghasilkan jumlah pengiriman paket yang jauh lebih besar, yaitu mulai dari 868 hingga 1.339 kali pada setiap skenario pengujian. Secara rata-rata, metode *Event-Driven* hanya menghasilkan 25,75 kali pengiriman, sedangkan metode Non-*Event-Driven* mencapai sekitar 1.001 kali pengiriman paket sehingga, metode Non-*Event-Driven* melakukan pengiriman data secara periodik tanpa mempertimbangkan perubahan kondisi, sehingga jumlah paket yang terkirim menjadi lebih banyak.

Pembahasan

Berdasarkan hasil pengujian, metode *Event-Driven* menghasilkan total pengiriman data sebanyak 103 kali pengiriman paket, sedangkan metode *Non-Event-Driven* mengirimkan paket yang lebih banyak, yaitu 4.047 kali pengiriman paket. Pada setiap skenario pengujian, metode *Event-Driven* selalu menunjukkan jumlah paket yang lebih sedikit dibandingkan metode *Non-Event-Driven*. Hal ini terjadi karena pada metode *Event-Driven*, data tidak dikirim secara terus-menerus, melainkan hanya pada waktu tertentu melalui *Batch Update* serta saat terjadi perubahan kondisi yang signifikan melalui *Immediate Alert.* Sementara itu, pada metode Non-*Event-Driven*, data dikirim secara periodik tanpa memperhatikan ada atau tidaknya perubahan nilai sensor, sehingga jumlah paket yang terkirim menjadi lebih banyak. Dengan berkurangnya total paket yang dikirim, metode *Event-Driven* dinilai lebih efisien karena dapat mengurangi pengiriman data yang tidak diperlukan. Hasil ini menunjukkan bahwa penerapan metode *Event-Driven* lebih cocok digunakan pada sistem monitoring karena mampu menekan beban jaringan dan membuat proses pengiriman data menjadi lebih efisien.

SIMPULAN

Berdasarkan hasil pengujian, Penerapan arsitektur *Event-Driven* pada sistem monitoring kelembapan tanah berbasis *master–slave* menggunakan ESP32 terbukti mampu meningkatkan efisiensi pengiriman data sensor. Sistem *Batch Update* setiap 5 menit pada kondisi normal dan *Immediate Alert* saat kelembapan berada di luar rentang ideal 50–70% membuat sistem hanya mengirimkan data ketika diperlukan. Cara ini memungkinkan beberapa data digabung dalam satu payload sehingga frekuensi pengiriman paket menjadi lebih sedikit dibandingkan metode non *Event-Driven.* Hasil dari empat skenario pengujian, sistem *Event-Driven* menghasilkan nilai *delay* dan *jitter* yang lebih rendah serta jumlah pengiriman paket data yang jauh lebih sedikit dibandingkan sistem non *Event-Driven*. Walaupun nilai *QoS* yang diperoleh belum sepenuhnya memenuhi standar TIPHON, sistem tetap mampu menjaga kestabilan komunikasi saat beberapa slave aktif secara bersamaan. Secara keseluruhan, arsitektur *Event-Driven* dinilai lebih efektif untuk diterapkan pada sistem monitoring IoT karena mampu mengurangi frekuensi pengiriman, meningkatkan efisiensi transmisi data, dan tetap cepat tanggap merespons perubahan kondisi kelembapan tanah.

DAFTAR PUSTAKA

[1] E. E. Jeksen and D. Sari, “Analisis Prospek Peningkatan Produksi Cabai Rawit (Capsicum frutescens L.) di Indonesia,” Universitas Tribhuwana Tunggadewi, 2022. [Online]. Available: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4285742

[2] A. K. Nalendra and M. Mujiono, “Perancangan IoT (Internet of Things) pada Sistem Irigasi

Tanaman Cabai," *Gener. J.*, vol. 4, no. 2, pp. 61–68, 2020, doi: 10.29407/gj.v4i2.14187.

[3] A. A. Aulia, L. D. Samsumar, and E. Suryadi, "Sistem monitoring kelembaban dan otomatisasi penyiraman tanaman cabai berbasis internet of things (iot)," *J. Rekayasa Sist. Inf. dan Teknol.*, vol. 2, no. 2, pp. 681–695, 2024.

[4] R. Eka Budiani, J. Dedy Irawan, and D. Rudhistiar, "Sistem Monitoring Dan Penyiraman Otomatis Pada Tanaman Cabai Berbasis Internet of Things (Iot)," *JATI (Jurnal Mhs. Tek. Inform.*, vol. 8, no. 2, pp. 1331–1338, 2024, doi: 10.36040/jati.v8i2.9149.

[5] R. Sahbani and A. Hamid, "Pengiriman Data Sensor Suhu dan Asap Menggunakan Longe Range (LoRa)," *ABEC Indones.*, pp. 1063–1080, 2021.

[6] A. Yoshua, R. Primananda, and A. S. Budi, "Implementasi Pengiriman Data Multi-Node Sensor Menggunakan Metode Master-slave pada Komunikasi LoRa," *J. Pengemb. Teknol. Inf. dan Ilmu Komput.*, vol. 4, no. 10, pp. 3445–3454, 2020.

[7] H. Cabane and K. Farias, "On the impact of event-driven architecture on performance: An exploratory study," *Futur. Gener. Comput. Syst.*, vol. 153, pp. 52–69, 2024.

[8] S. Khriji, Y. Benbelgacem, R. Chéour, D. El Houssaini, and O. Kanoun, "Design and implementation of a cloud-based event-driven architecture for real-time data processing in wireless sensor networks," *J. Supercomput.*, vol. 78, no. 3, pp. 3374–3401, 2022.

[9] M. Rafli, J. R. Wibowo, and Y. Saragih, "Evaluasi Arsitektur Serverless dalam Meningkatkan Efisiensi Eksekusi Aplikasi Berbasis Event-Driven," 2026.

[10] R. Adila, R. J. Akbar, and H. Studiawan, "Rancang Bangun Modul Portofolio Pegawai pada Aplikasi MyITS Human Capital Management dengan Arsitektur Event Driven," *J. Tek. ITS*, vol. 11, no. 1, pp. A29–A34, 2022.

[11] A. Fauzi, E. Harli, and Y. Haryanto, "Implementasi Arsitektur Event-Driven dan Microservices untuk Mesin Vending," *Format J. Ilm. Tek. Inform.*, vol. 10, no. 2, p. 135, 2021, doi: 10.22441/format.2021.v10.i2.004.

[12] A. Stevens, "Quality of Service (QoS) in ARM Systems: An Overview," *ARM White Pap.*, vol. 134, 2014.

[13] B. Mehra and A. Datar, "Evaluation of the quality of service parameters for IoT in outdoor environment," in *2024 5th International Conference on Smart Electronics and Communication (ICOSEC)*, 2024, pp. 839–844.

[14] A. K. Saleh, H. P. A. Tjahyaningtijas, N. Nurhayati, and L. Rakhmawati, "Quality of service (QOS) comparative analysis of wireless network," *Ina. (Indonesian J. Electr. Electron. Eng.*, vol. 5, no. 2, pp. 30–37, 2022.

[15] M. R. Kamil, F. Arzalega, R. Rosalinda, and A. Sani, "Analisis Kualitas Layanan Jaringan Internet Wifi PT. XYZ dengan Metode QoS (Quality of Service)," *J. Bid. Penelit. Inform.*, vol. 1, no. 1, pp. 45–56, 2023.