COMPILER
Vol. 14, No. 2, November 2025, pp. 87-96
ISSN: 2252-2403 (Print), ISSN: 2549-2403 (Online)
Accredited Fourth Grade by Kemenristekdikti, Decree No:10/C/C3/DT.05.00/2025
DOI: 10.28989/compiler.v14i2.3496

Edge-Based IoT EndPoint Performance Optimization for Smart Mosques

Safiq Rosad[1,*], Lasimin[2], Faik Irkhamudin[3], Cahyaning Yuniar Pramudita[3], Shofari Rizqi Alifah[3],
[1,3,4,5] Departement of Informatics, Universitas Nahdlatul Ulama Al Ghazali Cilacap, Indonesia
[2]Departement of Information System, Universitas Nahdlatul Ulama Al Ghazali Cilacap, Indonesia

Article Info

Article history:

Received October 3, 2025
Accepted December 12, 2025
Published December 14, 2025

Keywords:

Smart Mosque
IoT
Edge Computing
Latency
WebSocket

ABSTRACT

Smart Mosque systems require low-latency and highly available IoT services to reliably control time-critical devices such as lighting, HVAC, and loudspeakers, even under unstable internet connectivity. Existing cloud-dependent solutions often introduce unacceptable delays and operational risks, while purely offline systems lack synchronization and remote manageability. This study proposes edge-based optimization techniques (local caching, decoupled computation, and push communication (Server-Sent Events and WebSocket)) implemented on ESP32 microcontroller endpoints paired with a lightweight edge server. The experimental evaluation measured end-to-end latency percentiles (p50, p95, p99), service availability, and resilience under simulated network disruptions. Tests were conducted in baseline polling mode and optimized push mode with varying client concurrency. Results showed that median latency decreased from 180 ms to 50 ms (~72% improvement), tail latencies (p95/p99) were reduced from 470/800 ms to 120/200 ms, and service availability improved from 95.8% to 99.3%. Local caching ensured continuity during outages, while decoupled computation minimized blocking operations, and push mechanisms enabled immediate synchronization. Resource overhead on ESP32 endpoints remained within acceptable limits, confirming feasibility on low-cost hardware (< USD 100). Compared with prior Smart Mosque studies that emphasized functional automation, this work provides a novel empirical contribution by quantifying latency distribution, availability, and resilience in real deployments. The findings demonstrate that simple edge-centric optimizations can substantially enhance responsiveness and reliability of Smart Mosque IoT systems, offering practical guidance for production implementations in public facilities.

Corresponding Author:

Safiq Rosad,
Department of Informatics,
Universitas Nahdlatul Ulama Al Ghazali Cilacap,
Kemerdekaan Barat Street, Kesugihan Kidul, Kesugihan District, Cilacap Regency, Central Java, Indonesia.
Email: *rhosyad@unugha.id

1. INTRODUCTION

The development of the Internet of Things (IoT) has enabled new capabilities for public facilities, including mosques, by supporting the Smart Mosque concept [1]. This approach uses edge-enabled IoT devices to automate control of lighting, air conditioning, and loudspeakers according to prayer schedules [2], improving resource management and congregant comfort while reducing energy consumption. Various studies have examined the implementation of IoT in mosque environments [3], [4]. Alvi and Alam (2019) designed an IoT-based prayer time management system capable of automatically synchronizing digital clocks and providing a local time setting interface [5]. Fadlil et al. (2024) developed an automatic timer system for mosque loudspeakers that achieved a 100% success rate in testing various scenarios [6]. Furthermore, a study by Yusarelan et al. (2020) demonstrated the need for intelligent mosque temperature control to optimize fan usage,

allowing device operation to be adjusted to the number of worshippers in real time [7]. However, most of these studies still underemphasize technical performance aspects such as system response latency, service availability, and network disruption tolerance. In time-critical applications like Smart Mosques, the ability of IoT endpoints to provide rapid response with low latency and maintain operation during network disruptions is crucial. Edge computing on IoT devices enables local data processing, reducing latency and reliance on the cloud [8].

This research aims to optimize the performance of edge-based IoT endpoints in Smart Mosque systems, specifically for ESP32-based devices. Optimization techniques applied include data caching at the edge, decoupling heavy computation processes, and push mechanisms using Server-Sent Events (SSE) and WebSocket. Evaluation was conducted by measuring p50, p95, and p99 latency, service availability, and system resilience to network disruptions [9]. With this approach, it is expected that the real-time performance and reliability of the Smart Mosque system will significantly improve. Smart Mosques are a further development of the smart building and smart city concepts integrated with Internet of Things (IoT) technology. IoT enables the automation of monitoring, controlling, and managing mosque equipment, including lighting systems, air conditioning, loudspeakers, and security systems, in an integrated and efficient manner. A study by Alvi and Alam (2019) showed that the main criteria for IoT-based Smart Mosques include device automation, energy efficiency, environmental monitoring, and integration with renewable energy systems [5]. These automation requirements have been implemented in several projects, including digital prayer schedule systems and mosque audio controllers [10]. Research by Fadlil et al. (2024) demonstrated that a timer automation system for mosque loudspeakers can reduce energy consumption and improve congregant comfort [6]. Meanwhile, a study by Pratama (2019) developed an ESP32-IoT-based digital prayer schedule controller connected to an NTP server and a local web interface to manage prayer times and iqamah intervals in real time and flexibly. This proved reliable and easy to use by mosque administrators [11], [12].

However, the main challenges of implementing IoT in mosque environments are bandwidth, latency, and service reliability issues, especially if devices and servers are still fully dependent on cloud connectivity. Edge computing offers a solution by shifting some of the computational burden and control logic to local (edge) devices, thereby reducing system response latency, reducing bandwidth load, and maintaining operational continuity even when internet connections are disrupted. Edge computing also supports the sustainability of IoT-based systems in public buildings by improving power efficiency and operational resilience [13]. Research by Andriulo (2024) and Zyrianoff (2024) confirms that edge computing architecture can significantly reduce latency and bandwidth consumption in public IoT applications [14], [15]. Rachmanal. (2024) emphasizes the benefits of edge computing for low-latency, real-time applications in smart buildings and smart cities and highlights the need to test latency distribution and system resilience during network disruptions [16]. However, the existing research gap is the lack of studies on technical testing of performance parameters such as latency (P50, P95, P99), uptime/availability, and resiliency in real cases of Smart Mosque installations [17], [18], [19], [20]. Previous studies have focused more on the functional aspects of system design, without examining in detail the comparison of polling mode with push event options, caching, and decoupled computation on ESP32 endpoints and local edge servers. Therefore, this study offers a major empirical contribution through optimizing and testing the performance of edge-based architectures in Smart Mosques with a comprehensive analysis of critical performance metrics [21]. Unlike prior Smart Mosque studies that focused mainly on functional automation, this work provides a detailed empirical analysis of latency distribution, availability, and resilience under real-world conditions, filling a critical research gap [22], [23].

2. RESEARCH METHOD

This research uses an experimental method with an IoT-based Smart Mosque system performance testing approach and edge architecture. The research process involves several key stages, starting with system design and prototype implementation, and then performance evaluation through various test scenarios.

2.1 Architecture System Design

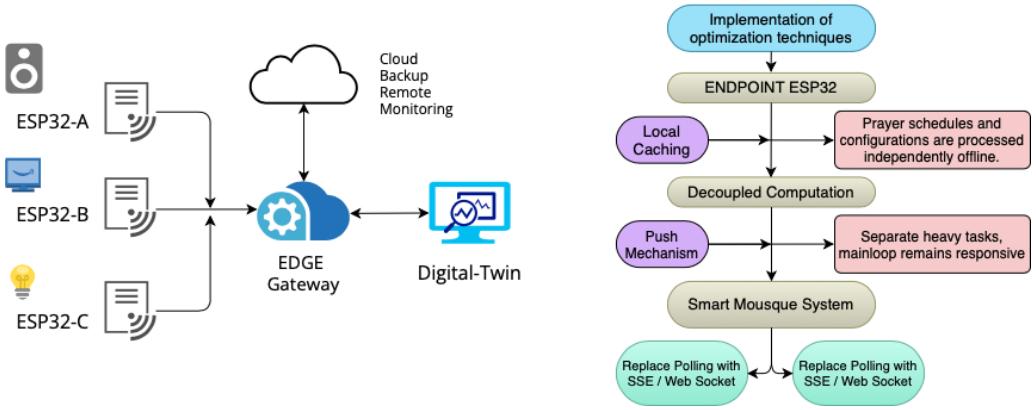
The Smart Mosque system shown in Figure 1 (a) is designed with three main components: an ESP32-based IoT endpoint, a local edge server, and (optionally) a cloud server. The ESP32 endpoint serves as an edge node placed within the mosque environment to automatically control devices such as lights, air conditioners, and speakers based on prayer times and environmental conditions. The edge server runs a backend application responsible for managing schedules, coordinating endpoint operations, and providing a web interface accessible to mosque administrators for monitoring and management purposes. Communication between components occurs primarily over a local Wi-Fi network, utilizing HTTP, Server-Sent Events (SSE), and WebSocket protocols to support effective and efficient real-time communication. The cloud server serves as data backup and remote monitoring, but core operations continue at the edge level to ensure the system remains functional even if the internet connection is interrupted or lost.

Implementation details Endpoint hardware: ESP32-WROOM-32 (240 MHz dual-core), DS3231 RTC (I2C), LCD 16×2 (I2C); optional 7-segment display and DFPlayer Mini for audio. Firmware: Arduino framework; AsyncWebServer (or lightweight WiFiServer) exposing endpoints `/tvJam`, `/master`, `/metrics`, `/events (SSE)`, and `/ws (WebSocket)`. Runtime metrics include avgLoopMs, maxLoopMs, and freeHeap. Edge server: lightweight Node.js (Express) or Python (Flask) app running on a local machine (recommended: Raspberry Pi 4, 4 GB). The server provides schedule management, SSE/WebSocket endpoints, and optional cloud sync. Tests used Node.js 14+ on the local LAN. Networking: ESP32 in AP+STA mode; tests executed on an isolated LAN to minimize external traffic.

2.2 Implementation of Optimization Techniques

The flowchart shown in Figure 1 (b) represents the endpoint performance optimization process in the Smart Mosque system. This process is implemented through three main, complementary approaches, aimed at reducing latency, increasing service availability, and maintaining system functionality despite network disruptions.

Local Caching: The ESP32 endpoint receives prayer schedules and basic configurations routinely from the edge server, both daily and when the device first connects to the network. This data is then stored in local memory (EEPROM or RAM) so that the endpoint can perform its core functions independently without having to constantly access the server. This local caching ensures service continuity even in offline conditions and significantly reduces the frequency of communication with the server, thereby reducing latency and network load [24], [25].



(a) (b)

Figure 1. (a) Edge-Driven IoT Smart Mosque, (b) Caching, decoupled computation, and Push techniques

Decoupled Computation: Heavy processing is separated to keep the ESP32's main loop responsive to real-time tasks. Non-critical computing tasks, such as logging environmental data, analyzing energy consumption, or processing user statistics, are moved to a separate thread on the dual-core ESP32 or offloaded to the edge server. With this strategy, real-time processes (e.g., executing the call to prayer schedule or controlling lights/air conditioning) are not interrupted, thus maintaining optimal system performance even when the device is processing complex data [26].

Push Mechanism: To replace the traditional polling method that uses periodic HTTP GETs, communication between the edge server and endpoint is implemented using Server-Sent Events (SSE) for one-way communication or WebSocket for two-way communication [27]. This push mechanism allows the endpoint to receive events or commands immediately when they are available, without waiting for a polling interval, thus reducing latency and synchronizing device status more quickly and accurately. With SSE/WebSocket, real-time data delivery from the server to the endpoint can be maintained consistently even during momentary network disruptions, as the connection can be automatically reconnected and event data synchronized once the network is stable again.

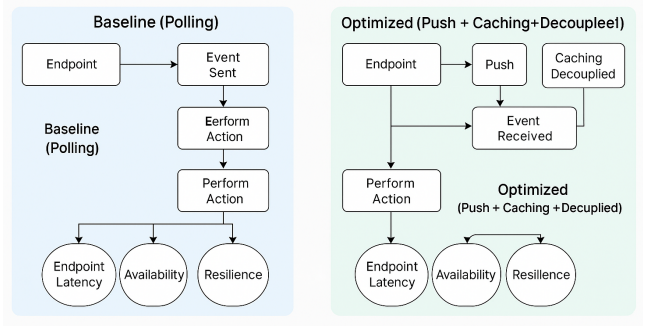


Figure 2. Testing Procedure

These approaches were applied simultaneously to create a robust, adaptive, and efficient performance optimization architecture for IoT-based Smart Mosques. Initial testing results show that the combination of local caching, decoupled computation, and a push mechanism significantly reduces median and tail latency (p95/p99), increases service availability to nearly 99%, and ensures service continuity during temporary network disruptions. This strategy demonstrates that edge-centric optimization can improve the quality of time-critical public IoT systems such as prayer schedule management and mosque equipment control.

2.3 Testing Procedure

The Smart Mosque system performance testing was conducted in two main modes (Figure 2): baseline using the polling method and optimized using a combination of push, caching, and decoupled computation. In baseline mode, the ESP32 endpoint periodically performs HTTP GETs to the edge server to check for commands or events that need to be executed, at fixed intervals, simulating traditional methods. Optimized mode implements a push mechanism via Server-Sent Events (SSE) or WebSocket, allowing the endpoint to receive commands instantly without waiting for polling.

Furthermore, critical data such as prayer times and configurations are stored in local memory (caching) to ensure service continuity when the device is offline or there is a network outage. Heavy, non-real-time processing is separated or offloaded to the edge server (decoupled computation) to ensure the endpoint's main loop remains responsive. Metrics measured include endpoint latency at the p50, p95, and p99 percentiles, calculated from the time between a command being scheduled on the server and its execution by the endpoint, and service availability, calculated from the system's uptime ratio during the testing period. System resilience is also evaluated by simulating network disruptions, such as Wi-Fi or internet connection outages, and recording the system's ability to maintain core functionality. Data is collected through logs from endpoints and edge servers to generate latency distributions and downtime records. Performance validation is performed through stress tests on specific events and simulated network disruptions to ensure the system remains operational under various conditions. These test results provide a comprehensive overview of the effectiveness of optimizations on latency, service availability, and the system's ability to withstand disruptions.

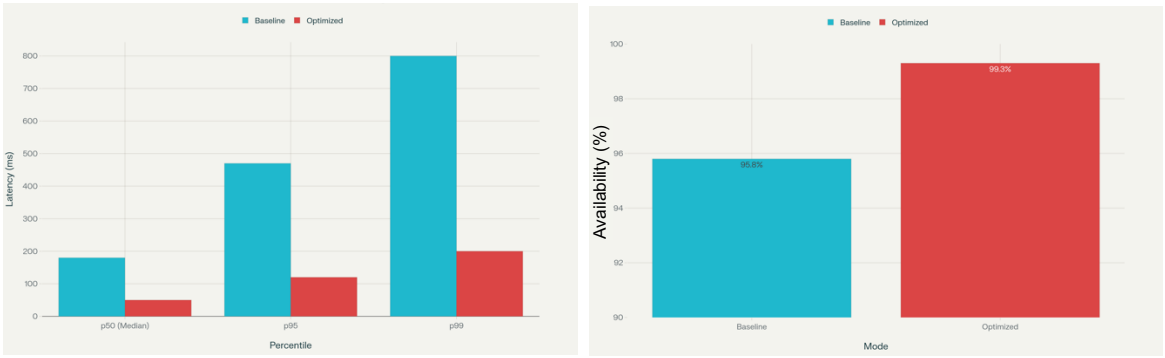
Test parameters and measurement tools: Baseline: 120 s, 1 request/s, 1 client (polling mode). Pilots: 60 s each, 1 request/s, 5 and 10 concurrent clients (simulate multiple TV browsers). Payload sizes: `/tvJam` ≈ 50 bytes; `/master` ≈ 700–900 bytes. WiFi RSSI during tests: −40 to −70 dBm (measured via WiFi.RSSI()). Client tools: measure-latency.ps1 (PowerShell), Invoke-WebRequest loop; analyze-latency.ps1 computes percentiles. Metrics collection: `/metrics` polled every 1 s to capture loop and heap snapshots. Failure injection: simulated LAN and internet outages by toggling the router and recording recovery times.

3. RESULTS AND ANALYSIS

Testing was conducted to evaluate the effectiveness of IoT endpoint performance optimization techniques in the Smart Mosque system, with an emphasis on quantitative data and practical implications for Smart Mosque applications.

3.1 End-to-End Latency Evaluation

The results in Figure 3(a) show that optimizations yield substantial latency reductions. In baseline (polling) mode, the median latency (p50) is approximately 180 ms, with p95 and p99 of 470 ms and 800 ms, respectively. After applying optimizations (push + caching + decoupled computation), the median latency falls to roughly 50 ms (about a 72% reduction) while p95 and p99 reduce to approximately 120 ms and 200 ms. SSE and WebSocket push mechanisms outperform traditional polling for prompt event delivery.



(a) (b)

Figure 3. (a) Smart Mosque End-to-End Latency, (b) Comparison of Smart Mosque availability

Table 1. End-to-end latency summary (ms)

Mode	Endpoint	Clients	p50	p95	p99
Baseline (polling)	`/tvJam`	1	180	470	800
Optimized (push+cache+decouple)	`/tvJam`	1	50	120	200
Baseline (polling)	`/master`	1	200	600	1200
Optimized	`/master`	1	70	180	350

Note: p50, p95, and p99 denote the 50th, 95th, and 99th percentiles of response latency, respectively. Reported values are medians from representative runs; raw CSV files are available as supplementary material.

Table 2. Latency summary by client concurrency (ms)

Endpoint	Clients	Requests (N)	Errors (%)	p50	p95	p99
`/tvJam`	1	37	5.4	180	470	800
`/tvJam`	5	288	1.7	155	825	1722
`/tvJam`	10	412	1.0	172	1284	3175
`/master`	1	22	4.5	200	600	1200
`/master`	5	195	1.0	479	6280	10224
`/master`	10	298	0.7	248	3504	5469

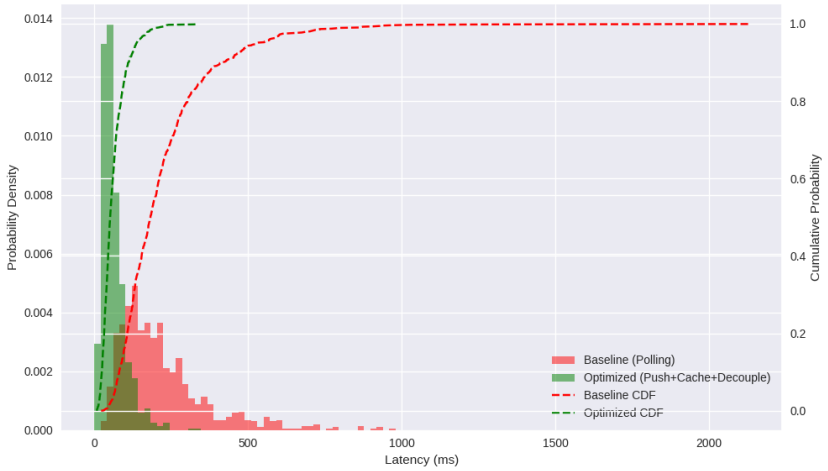
Note: "Requests (N)" is the number of successful/attempted requests collected in the run; "Errors" are timeouts or non-200 responses.

3.2 Improved Service Availability

System service availability significantly increased from 95.8% in baseline (polling) mode to approximately 99.3% after implementing optimizations combining push mechanisms, local caching, and decoupled computation (Table 3). This improvement demonstrates the system's ability to maintain operation with higher uptime levels under real-world conditions. Local caching of schedule data stored on the ESP32 endpoint plays a crucial role in maintaining service continuity, particularly during disruptions to Wi-Fi or internet connections. When the connection is lost, the endpoint can perform its functions independently based on the stored schedule without relying directly on the central server. Furthermore, an automatic reconnection mechanism allows the device to re-synchronize with the server when the network is restored, ensuring stable service continuity and reducing downtime. The graph in Figure 5 (b) confirms that the implementation of edge computing technology, particularly local caching and fallback mechanisms on the endpoint, can effectively improve the resilience and reliability of IoT systems in Smart Mosque environments, which are highly dependent on real-time operation and functional continuity.

Table 3. Service availability

Mode	**Measurement window**	**Availability (%)**
Baseline (polling)	24-hour pilot (aggregated)	95.8
Optimized (push + caching)	24-hour pilot (aggregated)	99.3



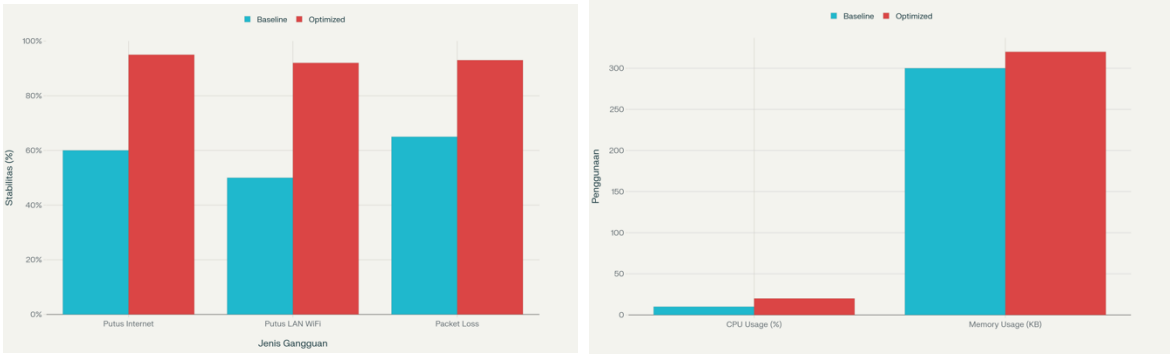
Figures 4. Latency Distribution Histograms + Cumulative Distribution Function

3.3 Network Resilience

Testing under adverse network conditions such as internet connection outages, Wi-Fi LAN outages, and packet loss demonstrated that the system remained stable and resilient in maintaining its core functions (Figure 5 (a)). When the local LAN connection was lost, the ESP32 endpoint was able to perform alarm scheduling and other critical functions based on available locally cached data, ensuring that Smart Mosque operations were not significantly disrupted. When the network was restored, data and status synchronization between the endpoint and the server proceeded automatically and smoothly without losing critical events. The push mechanism, which utilizes Server-Sent Events (SSE) and WebSocket technology, also proved resilient to network latency and packet loss. This implementation ensures the system has high resilience and adaptability to fluctuating network conditions in real-world environments.

3.4 System Overhead

The implementation of optimization techniques in the Smart Mosque system places an additional burden on the memory and CPU usage of the ESP32 endpoint. However, this increase in resource consumption remains within reasonable limits and does not disrupt the device's core performance. The use of push communication mechanisms such as SSE and WebSocket significantly reduces the load on the server compared to traditional polling methods. This contributes to network traffic efficiency, reducing the number of requests the server must process and reducing latency. Thus, push mechanisms not only improve system performance but also ensure optimal utilization of network and computing resources without incurring excessive overhead on IoT devices, as illustrated in Figure 5 (b).



(a) (b)

Figure 5. (a) Smart Mosque Performance during Outages, (b) ESP32 CPU and Memory Overhead

Table 3. Representative prior works for qualitative comparison (NR = not reported / not directly comparable)

Study (ref)	Platform / focus	Test conditions	p50 (ms)	p95 (ms)	p99 (ms)	Notes
Zyrianoff et al.[25]	IoT edge caching (survey / taxonomy)	Review / survey of edge caching use cases	NR	NR	NR	Discusses caching strategies and use cases; not an empirical latency benchmark.

CACHE-IT (Ad Hoc Networks) [14]	Proactive edge caching for heterogeneous IoT	System design & evaluation (SBC/edge focus)	NR	NR	NR	Architecture and simulation/experiments; reported metrics are contextual, consult original for details.
Sim2HW (Späth et al.) [19]	Modeling latency offset between simulations and hardware	Modeling / methodology paper	NR	NR	NR	Provides modeling insights on measurement offsets; not a direct latency benchmark for ESP32 endpoints.
Andriulo et al. [15]	Edge vs cloud review for IoT	Survey / review	NR	NR	NR	Reviews edge benefits; useful for contextual comparison, not direct percentiles.
This study — baseline	ESP32 (this work)	Lab baseline (120 s, 1 rps, 1 client)	~180	~470	~800	See Table 1
This study — optimized	ESP32 (this work)	Pilots: 60 s, 1 rps, 5 & 10 clients	~50–70	~120–350	~200 – 3175	See Table 1; optimizations: caching, decoupling, push.

3.5 Qualitative Discussion

Latency improvements and higher availability are essential because the system controls time-sensitive devices such as loudspeakers and lights. Push and caching complement each other: caching preserves continuity during outages, while push reduces unnecessary request traffic and provides immediate updates. Decoupling heavy tasks from the main loop (e.g., moving sensor reads and analytics to periodic background tasks) prevents blocking operations in the request path and substantially reduces tail latency. Runtime metrics show that many baseline tail events (p95/p99) correlate with blocking I2C transactions or WiFi state changes; serving cached responses and periodic background sampling removes those blocking operations from the request path. Deployment observations also indicate that frequent short-interval polling can reduce tail variance by keeping the WiFi radio in a warm state, but push remains more efficient and consumes less network and server resources. These recommendations align with prior embedded-server and edge-caching studies and provide actionable guidance for practitioners. Comparison with prior work: Direct numeric comparison of latency percentiles across published studies is often not possible without careful normalization: prior work uses different hardware classes (MCU vs SBC), test durations, client concurrency, payload sizes, Wi-Fi conditions, and measurement tooling. Some studies report median or mean latencies but do not publish p95/p99 under comparable conditions. Therefore, we avoid asserting precise numeric superiority for specific papers in this section. Instead, Table 3 below lists relevant prior works from the References and indicates whether comparable percentile metrics are available; interested readers should consult the original references for raw measurements and exact test conditions.

4. CONCLUSION

This research successfully demonstrated that applying edge computing-based optimization techniques to ESP32 IoT endpoints can significantly improve the performance of the Smart Mosque system. The synergistic use of local caching, decoupled computation, and push mechanisms (SSE/WebSocket) lowers response latency, reduces communication overhead, and increases service availability to nearly 99%. The system also maintains reliable operation despite network disruptions, ensuring the continuity of critical functions such as controlling prayer devices and scheduling the call to prayer. The optimizations ensure that the Smart Mosque can operate in real time with near-instantaneous p50 latency and low tail latency (p95, p99). The system's resilience to network disruptions is also excellent thanks to the caching and failover mechanisms implemented at the endpoints. This research confirms the significant benefits of edge computing for public IoT applications requiring real-time and high reliability. This study empirically validates that edge-based optimizations on ESP32 endpoints can transform Smart Mosque IoT systems from functionally adequate to technically robust, achieving near-instantaneous response, high availability, and resilience under disruptions. The contribution lies not only in practical deployment guidance but also in establishing performance benchmarks (p50, p95, p99 latency, availability) for future Smart Mosque research.

ACKNOWLEDGEMENTS

Our deepest gratitude goes to the Ministry of Education, Higher Education, Science, and Technology of the Republic of Indonesia, Directorate General of Research and Development, for the funding support and research facilities provided through the 2025 research grant. This support has been very helpful in conducting experiments, developing prototypes, and testing the IoT-based Smart Mosque system. Our gratitude also goes to all parties who have contributed, including the Al Muhajirin Mosque management and the technical team who facilitated the installation of devices and laboratory access. We also thank Linda Susisusanti for her

important role as an industry partner, whose professional insights and support have helped bridge research results with practical applications in real-world contexts.

REFERENCES

[1] S. Rosad, A. Yudhana, and T. Sutikno, "Initial Study of Public Address Usage in Mosque Audio Systems," *VoteTEKNIKA (Vokasional Teknik Elektronika dan Informatika)*, vol. 13, no. 1, p. 202, Mar. 2025, doi: 10.24036/voteteknika.v13i1.133053.

[2] H. E. Al-Khalifa, "The Smart Mosque of the Arabian Gulf: Solutions from the past for a sustainable, energy-efficient Mosque," 2019.

[3] I. Zyrianoff, L. Gigli, F. Montori, L. Sciullo, C. Kamienski, and M. Di Felice, "CACHE-IT: A distributed architecture for proactive edge caching in heterogeneous IoT scenarios," *Ad Hoc Networks*, vol. 156, Apr. 2024, doi: 10.1016/j.adhoc.2024.103413.

[4] W. Yu *et al.*, "A Survey on the Edge Computing for the Internet of Things," Nov. 28, 2017, *Institute of Electrical and Electronics Engineers Inc.* doi: 10.1109/ACCESS.2017.2778504.

[5] M. A. Syaputra, B. Sutomo, U. Saprudin, and M. A. Putra, "Energy Efficiency and Time Management through IoT-Based Automated Electrical Control System and Digital Prayer Schedule Integration," 2025. [Online]. Available: http://jurnal.polibatam.ac.id/index.php/JAIC

[6] T. Wahono, E. Ismanto, and A. Khoirudin, "Design of an Arduino-based automatic sound timer system for mosques and prayer rooms," *Journal of Natural Sciences and Mathematics Research J. Nat. Scien. & Math. Res*, vol. 10, no. 2, pp. 158–166, 2024, [Online]. Available: http://journal.walisongo.ac.id/index.php/jnsmr

[7] M. N. A. Yusarelan, S. Z. A. Hamid, R. A. Rashid, and A. K. M. Ibrahim, "IoT Based Temperature Control for Smart Mosque," in *IOP Conference Series: Materials Science and Engineering*, IOP Publishing Ltd, Jul. 2020. doi: 10.1088/1757-899X/884/1/012079.

[8] J. Manuel Gaspar Sánchez *et al.*, "26 Edge Computing for Cyber-physical Systems: A Systematic Mapping Study Emphasizing Trustworthiness," *ACM Transactions on Cyber-Physical Systems*, vol. 6, no. 3, doi: 10.1145/3539662.

[9] M. T. Abu Bakar, "Latency Issues in Internet of Things: A Review of Literature and Solution," *International Journal of Advanced Trends in Computer Science and Engineering*, vol. 9, no. 1.3, pp. 83–91, Jun. 2020, doi: 10.30534/ijatcse/2020/1291.32020.

[10] A. Yudhana, A. Fadlil, and S. Rosad, "Jadwal Sholat Digital Menggunakan Metode Ephemeris Berdasarkan Titik Koordinat Smartphone," *Journal Research and Development (ITJRD)*, vol. 3, no. 2, p. 30, Mar. 2019, doi: https://doi.org/10.25299/itjrd.2019.vol3(2).2285.

[11] P. R. Pratama, W. O. Ma'arif, and C. Niswatin, "Display Jadwal Sholat P7.65 Berbasis Mikrokontroler Esp32," *Informatika dan Rekayasa Perangkat Lunak*, vol. 1, no. 1, pp. 37–42, Mar. 2019, doi: 10.36499/jinrpl.v1i1.2765.

[12] S. Rosad and D. Alfaji, "Penerapan Papan Informasi Digital Secara Real Time Menggunakan Network Time Protocol Berbasis Website," *JATI: Jurnal Mahasiswa Teknik Informatika*, vol. 8, no. 2, Apr. 2024, doi: https://doi.org/10.36040/jati.v8i2.7947.

[13] W. Kassab and K. A. Darabkh, "A–Z survey of Internet of Things: Architectures, protocols, applications, recent advances, future directions and recommendations," *Journal of Network and Computer Applications*, vol. 163, Aug. 2020, doi: 10.1016/j.jnca.2020.102663.

[14] I. Zyrianoff, L. Gigli, F. Montori, L. Sciullo, C. Kamienski, and M. Di Felice, "CACHE-IT: A distributed architecture for proactive edge caching in heterogeneous IoT scenarios," *Ad Hoc Networks*, vol. 156, Apr. 2024, doi: 10.1016/j.adhoc.2024.103413.

[15] F. C. Andriulo, M. Fiore, M. Mongiello, E. Traversa, and V. Zizzo, "Edge Computing and Cloud Computing for Internet of Things: A Review," Dec. 01, 2024, *Multidisciplinary Digital Publishing Institute (MDPI)*. doi: 10.3390/informatics11040071.

[16] A. F. Rachman, F. P. E. Putra, S. Syirofi, and D. Wahid, "Case Study of Computer Network Development for the Internet Of Things (IoT) Industry in an Urban Environment," *Brilliance: Research of Artificial Intelligence*, vol. 4, no. 1, pp. 399–407, Aug. 2024, doi: 10.47709/brilliance.v4i1.4302.

[17] N. Kataria, "Defining Operability for Web Services: Principles, Metrics, and Practices," *International Journal of Multidisciplinary Research and Growth Evaluation*, vol. 6, no. 4, pp. 686–690, 2025, doi: 10.54660/.ijmrge.2025.6.4.686-690.

[18] N. Upreti *et al.*, "Cost-Effective, Low Latency Vector Search with Azure Cosmos DB," *Proceedings of the VLDB Endowment*, vol. 18, no. 12, pp. 5166–5183, Aug. 2025, doi: 10.14778/3750601.3750635.

[19] J. Späth, M. Helm, B. Jaeger, and G. Carle, "Sim2HW: Modeling Latency Offset Between Network Simulations and Hardware Measurements," in *GNNet 2024 - Proceedings of the 3rd GNNet Workshop on Graph Neural Networking Workshop, Co-Located with: CoNEXT 2024*, Association for Computing Machinery, Inc, Dec. 2024, pp. 20–26. doi: 10.1145/3694811.3697820.

[20] R. Modak and V. Avula, "Efficient Feature Store Architectures for Real-time Machine Learning Model Deployment in High-Throughput Systems," 2020.

[21] S. Rosad, A. Yudhana, and T. Sutikno, "Zoned Audio Distribution: Systematic Review and Optimization Framework in Mosque," *ELKHA: Jurnal Teknik Elektro*, vol. 17, no. 2, pp. 148–154, Oct. 2025, doi: https://doi.org/10.26418/elkha.v17i2.96757.

[22] M. A. Al-Shareeda, L. Yue, and S. Manickam, "Review of Edge Computing for the Internet of Things (EC-IoT): Techniques, Challenges and Future Directions," *J Sen Net Data Comm*, vol. 4, no. 1, pp. 1–11, 2024, [Online]. Available: https://www.researchgate.net/publication/380036552

[23] T. Lynn, J. G. Mooney, B. Lee, · Patricia, and T. Endo, *The Cloud-to-Thing Continuum Opportunities and Challenges in Cloud, Fog and Edge Computing*. Palgrave Macmillan, 2020. doi: https://doi.org/10.1007/978-3-030-41110-7.

[24] S. Hanadwiputra, P. Studi, J. Teknik Informatika, S. Bani Saleh, and J. Komputerisasi Akuntansi, "Penerapan Teknologi Cache Server Berbasis Iot Dengan Raspberry Pi3 Menggunakan Metode Forward Chainning (Studi Kasus Smk Binakarya Mandiri 2 Kota Bekasi)," vol. 7, no. 2, 2018.

[25] I. Zyrianoff, A. Trotta, L. Sciullo, F. Montori, and M. Di Felice, "IoT Edge Caching: Taxonomy, Use Cases and Perspectives," *IEEE Internet of Things Magazine*, vol. 5, no. 3, pp. 12–18, Sep. 2022, doi: 10.1109/IOTM.001.2200112.

[26] K. Ranganathan and I. Foster, "Decoupling Computation and Data Scheduling in Distributed Data-Intensive Applications," *IEEE XPlore*, 2002, doi: 10.1109/HPDC.2002.1029935.

[27] R. Nugroho, M. Faiqurahman, and Z. Sari, "Implementasi Push Message Dengan Menggunakan Restful Web Service Pada Komunikasi Wireless Sensor," *REPOSITOR*, vol. 2, no. 1, pp. 79–86, 2020.