Energy Consumption Analysis of Application Layer Protocols in Realistic IoT Uplink Scenarios: An Empirical Study on an ESP32 Microcontroller

Daniele Quartinieri
Informatics Institute Amsterdam
University of Amsterdam,
Amsterdam, the Netherlands
daniele.quartinieri@student.uva.nl

Marios Avgeris
Informatics Institute Amsterdam
University of Amsterdam,
Amsterdam, the Netherlands
m.avgeris@uva.nl

Adam Belloum
Informatics Institute Amsterdam
University of Amsterdam,
Amsterdam, the Netherlands
a.s.z.belloum@uva.nl

***Abstract*—The rise of the Internet of Things (IoT) has enabled applications ranging from consumer devices to industrial systems. Yet the success of these deployments depends on hardware and software design choices made under tight resource and energy constraints. This paper assesses the energy consumption of application-layer protocols in representative IoT uplink scenarios. We built a measurement-driven testbed with an ESP32 as the sender, an INA226 for power sensing, and an Arduino Pro Micro for data logging. Using two scenario-inspired benchmarks, we evaluated four communication protocols. The results provide practical guidance for constraint-driven design: at low uplink rates, protocol choice is largely dictated by deployment requirements, as energy differences are rarely practically significant; at higher rates, protocols can materially affect energy consumption.**

***Index Terms*—IoT, TCP/IP, network, protocols, OSI, RTOS, energy, power, benchmark, MQTT, HTTP, CoAP, WebSockets.**

I. Introduction

THE Internet of Things (IoT) is widely regarded as one of the most influential technological developments of the past two decades. It has enabled applications spanning home automation [1], agriculture [2], and broader Industry 4.0 deployments [3]. This breadth of use cases has, in turn, driven a proliferation of hardware platforms and communication protocols designed to accommodate diverse operating conditions and constraints.

However, this variety also confronts developers with non-trivial design choices: selecting the hardware platform, deciding whether to adopt a real-time operating system (RTOS) and which one, and choosing an appropriate network stack and supporting libraries. Each decision can materially affect energy consumption and resource utilization, two of the dominant constraints in IoT systems. Energy efficiency is repeatedly identified as a central requirement, since many devices operate under tight CPU/memory budgets and battery limitations [4]. As a result, the software stack must be engineered to respect these constraints and minimize avoidable overhead [5].

At the application layer, protocols such as MQTT, CoAP, and AMQP must be matched to application requirements while remaining sufficiently lightweight for constrained devices [6]. Efficiency considerations extend to lower layers as well, where encapsulation and routing mechanisms influence transmitter energy usage and overall power draw [7]. Consequently, lightweight IP stacks and effective low-power modes are commonly required [8]. Against this backdrop, energy consumption has become a key metric when evaluating and comparing IoT protocol solutions, including at the application layer [9]–[16].

Motivated by these constraints and the practical need to choose among competing application-layer protocols, we evaluate whether protocol choice significantly affects energy consumption on the widely adopted ESP32 microcontroller platform [17] in two real-world-inspired IoT scenarios, and whether the most energy-efficient protocols remains consistent across benchmarks. In doing so, we extend prior practical studies with multi-benchmark evidence on constrained hardware. Our contribution is twofold:

- We build an end-to-end, sensor-in-the-loop energy measurement testbed (using an ESP32, an INA226 and an Arduino logger) and evaluated four application-layer protocols over two realistic, increasing-load benchmarks.
- We quantify whether application-layer protocol choice affects ESP32 energy consumption and whether protocol energy-efficiency rankings vary across workloads.

The rest of the paper is structured as follows: Section II presents background. Section III reviews related work. Section IV describes the methodology. Section V details the implementation and execution workflow. Section VI reports and discusses the results. Section VII concludes the paper.

II. Background

A. Common communication protocols in IoT

To select protocols for benchmarking, we reviewed IoT protocol survey and overview papers published between 2019 and 2025 and identified the most frequently cited protocols across them. We retained protocols appearing in at least 50% of the works (Table I), excluded closed-source options, and

979-8-3315-5204-6/26/$31.00 © 2026 IEEE

removed protocols that cannot run on the ESP32 microcontroller without additional hardware. After standardizing on a common RTOS and software stack (ESP-IDF via the Arduino core, Fig. 1), the final set included HTTP/HTTPS, MQTT, and CoAP, with WebSocket added for completeness.

TABLE I
APPLICATION-LAYER PROTOCOLS REFERENCED IN AT LEAST 50% OF THE SURVEYED OVERVIEW PAPERS

Protocol	[18]	[19]	[20]	[4]	[21]	[22]	[23]	[16]	[24]
HTTP/HTTPS	Y				Y	Y		Y	Y
MQTT	Y		Y	Y	Y	Y	Y	Y	Y
AMQP	Y			Y	Y	Y	Y	Y	Y
CoAP	Y		Y	Y	Y	Y	Y	Y	Y
Zigbee		Y	Y	Y	Y		Y		
6LoWPAN	Y	Y	Y	Y	Y	Y	Y		Y
DDS	Y			Y	Y		Y	Y	Y
XMPP	Y				Y	Y	Y	Y	Y
Zwave		Y	Y		Y		Y		Y
802.15.4		Y	Y	Y	Y	Y	Y		Y

Fig. 1. Simplified software stack for running Arduino sketches on the ESP32, showing the Arduino core as a compatibility layer on top of ESP-IDF, which includes FreeRTOS, and the underlying hardware.

B. Real-Time Operating System (RTOS)

A real-time operating system (RTOS) is an operating system whose correctness depends not only on what it computes, but also on when the results are produced [25]. RTOSs are widely used in embedded systems, where bare-metal superloop with interrupts becomes increasingly difficult to manage as system complexity grows [26]. An RTOS provides multitasking abstractions such as scheduling, task prioritization, timing guarantees, and preemption to support deterministic execution; many RTOSs also include services like filesystems, communication stacks, and security mechanisms.

III. RELATED WORK

Several works examine the energy consumption of application-layer protocols [9]–[16] and, more broadly, their performance. A recurring finding is that CoAP tends to be more energy-efficient than MQTT, and MQTT more energy-efficient than HTTP [9], [10], [13]–[16], consistent with protocol overhead being a major driver of energy usage [11]. CoAP's low overhead is due in part to its use of UDP at the transport layer [14]. Communication patterns also matter: [13] found publish-subscribe (pub/sub) less demanding than request-response (req/res). Moreover, energy consumption depends on protocol options and versions, for example, MQTT energy use generally increases with higher QoS levels, and HTTP/2 can improve energy efficiency relative to HTTP/1.1 [9], [13]. Unless stated otherwise, we refer to HTTP/1.1 in this paper. Despite these trends, results are not fully consistent across studies, as illustrated by conflicting observations on AMQP energy usage between [9] and [14].

Across the reviewed literature, three gaps emerge. First, protocol coverage is often limited under consistent experimental conditions, as noted by [15] and reflected in studies that evaluate only two to three protocols [11]–[13]. Second, benchmark design is frequently arbitrary rather than scenario-driven; payload sizes are often omitted, or are not justified, and similar issues arise for transmission rates [9]–[11]. This matters because payload size and transmission frequency can induce different behaviors across protocols [9]. Third, the chosen hardware and software stacks are often not representative of constrained IoT deployments, relying on Raspberry Pi platforms [9], [11] or even desktop computers [13], rather than on microcontrollers with software stacks tailored to constrained environments.

This paper targets these gaps by benchmarking representative application-layer protocols on a constrained ESP32-based platform using scenario-driven workloads, with the goal of providing a more realistic and reliable characterization of protocol energy performance.

IV. METHODOLOGY

A. Testbed Design

Benchmarks were executed on an ESP32-WROOM-32, with power monitored using an INA226 (Fig. 3). INA226 samples were read over I2C by an Arduino Pro Micro and streamed via USB serial to a laptop for storage and post-processing. An unmonitored receiver node was used (second ESP32 or laptop for MQTT), as summarized in Fig. 2. The monitored ESP32 was powered through a USB-C power-delivery trigger board. Given the advantages of RTOS-based development over bare-metal, we adopted an RTOS implementation. We evaluated several modern RTOS options against three criteria: i) active maintenance, ii) out-of-the-box ESP32 support, and iii) compatibility with the application-layer protocol libraries under test. Based on ecosystem maturity, native ESP32 support, and the broad adoption of FreeRTOS in its vanilla form [27], we selected ESP-IDF FreeRTOS. We used it via the Arduino core to leverage widely adopted protocol libraries (Fig. 1).

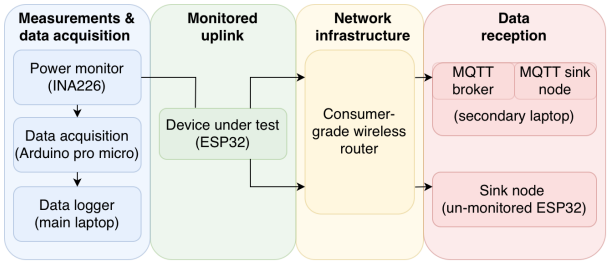

Fig. 2. Testbed Overview

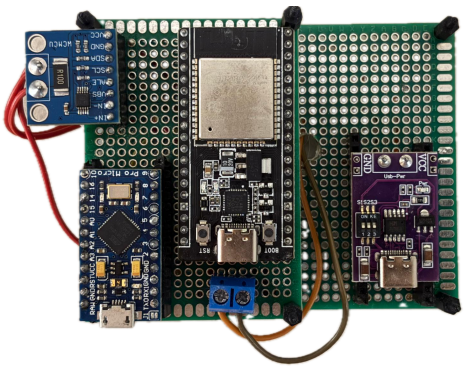

Fig. 3. From left to right: INA226, Arduino Pro Micro, ESP-32 board, USB-C power-delivery trigger board.

B. Benchmarking Tools, Metrics and Criteria

We collect power-consumption metrics from the monitored ESP32 while it executes each benchmark. Power is measured using an INA226, which samples the bus voltage and shunt voltage via its internal ADC [28]. We then compute energy (J) by integrating power over time; because power is sampled at discrete time instants, we approximate the integral using the rectangular rule. It is worth noting that the minimum amount of detectable current of the INA226 depends on the shunt resistor sizing. Following prior IoT energy studies [9], [12], we use energy (J) as the primary metric for comparison.

To ensure the relevance of our measurements, we designed two benchmarks inspired by real-world scenarios. The selected scenarios have markedly different data-rate requirements, yet share a common concern for energy efficiency:

1) Light: This models periodic sensing data from a humidity and temperature sensor (e.g., DHT11). Motivated by [29] which reports meteorological payload sizes in the range of 2–12 bytes, we selected a 4-byte raw payload (temperature and humidity), as a reasonable lower bound. When formatted in JSON for transmission, e.g., {"t": 24.5, "h": 60.2} this becomes 21 bytes. For the transmission interval, we use a consumer weather-station setting of one update every 5 minutes [30], corresponding to a data rate of 0.56 bps.

2) Heavy: This models bird-movement tracking. While [31] specifies GPS sampling every 3 seconds (high-frequency case), the payload fields can be inferred (e.g., id, timestamp, latitude, longitude, altitude, speed, battery, satellite number). A plausible JSON-formatted payload, such as {"id": 53100, "timestamp":"2025-08-07T12:30:45Z", "latit": 43.47126774300904, "longit": 11.28529038302559, "altit": 578, "speed": 3.2, "batt": 5, "satnumb": 3}, is 160 bytes. At one transmission every 3 seconds, this corresponds to an expected data rate of 0.4267 kbps.

C. Data Collection

1) Data Acquisition and Logging: Measurements were collected on a laptop via the Arduino Pro Micro over USB serial. The Arduino formatted the serial stream as CSV, which was stored on the laptop upon reception. The collected data and complete replication package are available at [32]. The Arduino logged INA226 measurements acquired over I2C, together with the run identifier and the active benchmark, which were provided by the ESP32 via GPIO. More details follow in Section V.

TABLE II
BENCHMARK SUMMARY.

Benchmark	Payload	Frequency	Datarate
Light	21 bytes	5 mins	0.56 bps
Heavy	160 bytes	3 seconds	0.43 kbps

2) Sampling Rate: The INA226 was configured in continuous mode to sustain the target update rate. Readouts are block-averaged over non-overlapping windows: each register read returns the average of the most recent N samples, where the per-sample interval is determined by the configured conversion times (bus and shunt). For instance, with conversion times set to 1.1 ms for both bus and shunt and an averaging setting of 16, the effective averaging window is $(1.1 + 1.1) \times 16 = 35.2$ ms, i.e., 16 samples spaced by 2.2 ms.

To select appropriate settings, we ran the heavy HTTP benchmark and evaluated multiple configurations. Motivated by the 10 ms sampling rate used in [9], we compared 18 ms and 8 ms averaging windows and observed similar medians estimates. We therefore selected the 8 ms configuration (1.1 ms conversions and an averaging setting of 4), which remains within the 10 ms target while reducing noise through longer conversion times [28]. The configuration options follow [28] and were implemented using [33].

D. Statistical Analysis

Data analysis combined descriptive visualization, statistical testing, and practical significance assessment. We used box plots to summarize the energy-consumption distributions per protocol and benchmark, highlighting the median, interquartile range (IQR), and overall spread.

For the statistical inference, the box plots suggested non-normal distributions; therefore, we used non-parametric tests. We first applied a Kruskal–Wallis test to evaluate whether protocols differed within each benchmark. When the test was significant, we performed pairwise Mann–Whitney U tests as post hoc analyses, applying a Bonferroni correction to mitigate the risk of familywise error rate (FWER).

To assess practical significance, we evaluated whether pairwise differences between protocols were meaningful relative to measurement variability. Specifically, we compared the absolute difference between protocol medians (chosen for robustness to outliers) against a noise threshold derived from the 12 repeated runs, shown in Fig. 4. The threshold was computed by scaling the median absolute deviation (MAD) of the 12 runs to its standard-deviation-equivalent value (MAD $\times$ 1.4826) [34]. Differences not exceeding this threshold were treated as not practically significant.

V. Implementation

A. Experimental Setup

Based on the protocol selection in Section II, we evaluated MQTT, HTTP, CoAP, and WebSocket. MQTT was tested at all three QoS levels, treated as distinct variants, yielding six protocol configurations in total. Each configuration was evaluated under two network-traffic intensity levels (light and heavy). With five replicas per condition the experiment comprised 60 runs in total. The design supports analysis of main effects across benchmarks and of protocol–benchmark interactions for the light and heavy levels.

To assess potential ordering effects in repeated runs of the same protocol, we executed the heavy HTTP benchmark for 12 consecutive runs while recording energy consumption, using a shortened run duration of 3 minutes. We plotted energy versus run order and applied a Spearman correlation test to evaluate monotonic trends. With no cooldown interval, a significant ordering effect was observed. Increasing the cooldown to 1 minute eliminated the correlation. The 1-minute cooldown remained effective at the full run duration (6 minutes), as shown in Fig. 4, where no significant correlation was found (Spearman's $\rho = -0.2098$, $p = 0.5128$).

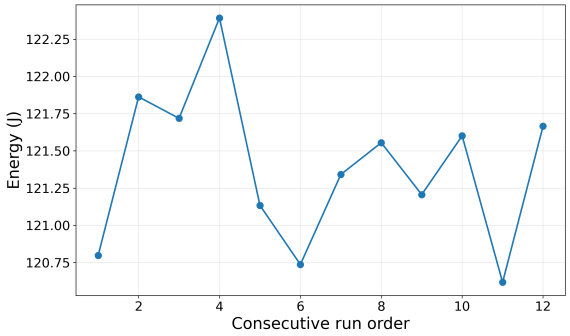

Fig. 4. Energy used (J) over consecutive runs of heavy HTTP benchmark, with 60 seconds cooldown between runs, and runs of 6 minutes

B. Execution Workflow

Boards were mounted on a perforated prototyping board and socketed to allow replacement; signal and power wiring was kept as short as possible, and cable cross-sections were kept consistent where cables were used. All tests were conducted through a consumer-grade wireless router configured with DHCP reservations (outside the DHCP pool) to assign fixed IP addresses to the boards and, when applicable, the broker. Measurements followed a fixed startup/shutdown procedure. First, the laptop logging script was started. Next the Arduino was powered via USB, the broker (if applicable) was started, and then the receiver/subscriber was brought up. Finally, the ESP32 sender was powered through the USB-C PD trigger. Data acquisition was stopped by interrupting the logger, which finalized and stored the measurements as CSV files; the boards and broker (if used) were then powered down. Post-processing was performed via a sequential Python pipeline driven by a single orchestration script and a JSON configuration. Measurement intervals for INA226 were gated using monitored ESP32 GPIO status change as a lock/unlock signal.

VI. Results & Discussion

In this section we present and discuss the results of evaluating the testbed under the two benchmarks. In Fig. 5 we report the results for the light benchmark. All protocols exhibit very similar energy consumption, with values (excluding outliers) clustering at approximately 111–$112J$ per run. CoAP and MQTT (QoS0, QoS1, QoS2) form the lower cluster, with HTTP slightly higher. WebSocket is the only protocol that appears distinct, with a median roughly 0.5 J above HTTP.

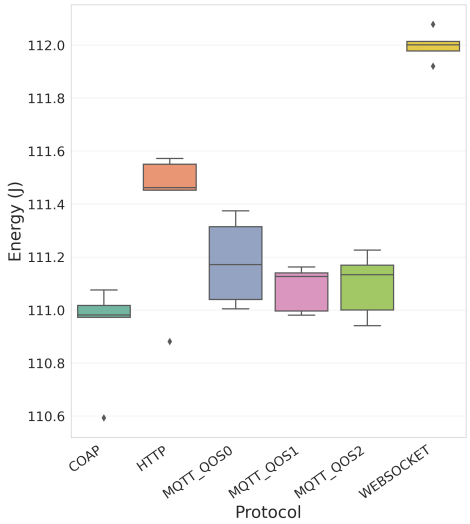

Fig. 5. Energy (J) used by a single run during the *light* benchmark.

TABLE III
PAIRWISE SIGNIFICANCE OF PROTOCOLS IN LIGHT BENCHMARK

Protocol A	**Protocol B**	**Statistically different ($p<0.05$)**	**Practically different ($\Delta>0.4333$)**
CoAP	HTTP	N	Y
CoAP	MQTT QoS0	N	N
CoAP	MQTT QoS1	N	N
CoAP	MQTT QoS2	N	N
CoAP	Websocket	Y	Y
HTTP	MQTT QoS0	N	N
HTTP	MQTT QoS1	N	N
HTTP	MQTT QoS2	N	N
HTTP	Websocket	Y	Y
MQTT QoS0	MQTT QoS1	N	N
MQTT QoS0	MQTT QoS2	N	N
MQTT QoS0	Websocket	Y	Y
MQTT QoS1	MQTT QoS2	N	N
MQTT QoS1	Websocket	Y	Y
MQTT QoS2	Websocket	Y	Y

A Kruskal-Wallis test indicates statistically significant differences among protocols ($H(5) = 17.975$, $p = 0.003$). Post-hoc Mann–Whitney U test with Bonferroni correction identify significant differences only in comparisons involving Websocket ($p = 0.0079$). Notably, the CoAP–MQTT QoS0 comparison is close to significance ($p = 0.0556$). Using the MAD-based practical-significance threshold of

$0.4333J$, the following pairs exceed the threshold and are thus considered practically significant: CoAP–HTTP (0.481), CoAP–WebSocket (1.020), HTTP–WebSocket (0.539), MQTT QoS0–WebSocket (0.829), MQTT QoS1–WebSocket (0.875), and MQTT QoS2–WebSocket (0.867). Overall, the practical-significance assessment aligns with the Bonferroni-corrected post hoc results, except for CoAP–HTTP, which is not statistically significant under Mann–Whitney but exceeds the practical threshold (Table III).

In Fig. 6 we present the results for the heavy benchmark. CoAP exhibits the lowest median energy consumption, followed by WebSocket and MQTT QoS0. A gap of roughly $2J$ separates this lower-energy cluster from MQTT QoS1, MQTT QoS2, and HTTP, which form a higher-energy cluster. Compared to the light benchmark Fig. 5, the distributions are less tightly clustered, indicating stronger protocol-dependent differences at this load level.

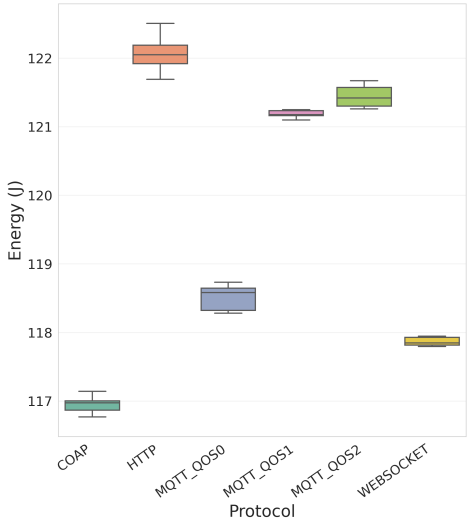

Fig. 6. Energy (J) used by a single run during the *heavy* benchmark.

TABLE IV
PAIRWISE SIGNIFICANCE OF PROTOCOLS IN HEAVY BENCHMARK

Protocol A	**Protocol B**	**Statistically different ($p<0.05$)**	**Practically different ($\Delta>0.4333$)**
CoAP	HTTP	Y	Y
CoAP	MQTT QoS0	Y	Y
CoAP	MQTT QoS1	Y	Y
CoAP	MQTT QoS2	Y	Y
CoAP	Websocket	Y	Y
HTTP	MQTT QoS0	Y	Y
HTTP	MQTT QoS1	Y	Y
HTTP	MQTT QoS2	Y	Y
HTTP	Websocket	Y	Y
MQTT QoS0	MQTT QoS1	Y	Y
MQTT QoS0	MQTT QoS2	Y	Y
MQTT QoS0	Websocket	Y	Y
MQTT QoS1	MQTT QoS2	Y	N
MQTT QoS1	Websocket	Y	Y
MQTT QoS2	Websocket	Y	Y

A Kruskal-Wallis test confirms statistically significant differences among protocols ($H(5) = 28.2258$, $p \approx 0.00$). Post-hoc Mann–Whitney U tests with Bonferroni correction indicate significant differences across all pairwise comparisons. Using the MAD-based practical-significance threshold of $0.4333\ J$, all pairwise differences are practically significant except MQTT QoS1 versus MQTT QoS2 ($\Delta = 0.243$J). Overall, the practical-significance assessment corroborates the post hoc statistical results, with the caveat that the QoS1–QoS2 difference, while statistically significant, is not practically significant under our noise-based criterion (Table IV).

The results indicate that protocol choice matters primarily once communication becomes frequent enough for per-message overheads to dominate. In the light benchmark, WebSocket is consistently more energy-hungry than the other protocols, while the remaining protocols are effectively indistinguishable within measurement variability; therefore, no meaningful energy-based ranking can be established among CoAP, HTTP, and MQTT variants at this load level. This suggests that, for low-rate sensing workloads, idle costs and platform effects largely mask protocol differences.

In the heavy benchmark, protocol effects become clear and a practical ranking emerges: CoAP is the most energy-efficient, followed by WebSocket and MQTT QoS0, with MQTT QoS1/2 and HTTP consuming more energy. This ordering is broadly consistent with the literature for the overlapping protocols (Section III), where CoAP typically outperforms MQTT and HTTP in energy terms, and higher MQTT QoS levels incur additional overhead. In our measurements, the QoS1–QoS2 difference is statistically detectable but not practically meaningful, so neither can be confidently preferred.

WebSocket's behavior highlights the interaction between workload characteristics and protocol mechanics. Prior work notes that once the initial handshake is complete, WebSocket avoids repeating HTTP headers, reducing per-message overhead [16]. This likely contributes to its strong performance at the higher transmission frequency of the heavy benchmark, whereas in the light benchmark the long idle intervals make fixed and idle energy costs more salient, and the handshake/connection maintenance overhead can disproportionately affect total energy per run.

VII. Conclusion

This paper assessed the energy cost of application-layer protocols for representative IoT uplink scenarios. We built a measurement-driven testbed with an ESP32 sender, an INA226 for power sensing, and an Arduino Pro Micro for logging, and evaluated four protocols across two increasingly demanding, scenario-inspired benchmarks. Overall, protocol choice can influence energy consumption, but the effect is workload dependent. In the light benchmark, CoAP, HTTP, and MQTT are broadly comparable, while WebSocket is consistently higher. In the heavy benchmark, CoAP is the most energy-efficient, followed by WebSocket and MQTT QoS0, then MQTT QoS1 and QoS2, with HTTP being the least efficient. These shifts show that protocol energy does not scale uniformly with workload; WebSocket is a clear example, performing poorly at low rates yet competitively at higher rates. Across conditions, CoAP and MQTT QoS0 remain among the best-performing

options. This work additionally contributes two motivated benchmarks that can serve as reusable reference workloads for future evaluations. The results also suggest that the INA226 offers a practical, low-cost approach for non-simulated energy measurement without specialized lab equipment.

Future work includes extending the analysis to downlink traffic, adding additional benchmarks for finer-grained workload coverage, evaluating implementations without the Arduino core layer, and repeating the study across RTOSes (e.g., ESP-IDF FreeRTOS vs. Zephyr).

ACKNOWLEDGMENT

This work was supported by the DIGITAfrica project, which has received funding from the Horizon Europe research and innovation programme under grant agreement No. 101187966.

REFERENCES

[1] M. Al-Kuwari, A. Ramadan, Y. Ismael, L. Al-Sughair, A. Gastli, and M. Benammar, "Smart-home automation using iot-based sensing and monitoring platform," in *2018 IEEE 12th International Conference on Compatibility, Power Electronics and Power Engineering (CPE-POWERENG 2018)*, pp. 1–6, IEEE, 2018.

[2] P. Jayashankar, S. Nilakanta, W. J. Johnston, P. Gill, and R. Burres, "Iot adoption in agriculture: the role of trust, perceived value and risk," *Journal of Business & Industrial Marketing*, vol. 33, no. 6, pp. 804–821, 2018.

[3] McKinsey&Company, "What are industry 4.0, the fourth industrial revolution, and 4ir?," Aug 2022. [Accessed 25-10-2024].

[4] C. Sobin, "A survey on architecture, protocols and challenges in iot," *Wireless Personal Communications*, vol. 112, no. 3, pp. 1383–1429, 2020.

[5] S. Sinche, D. Raposo, N. Armando, A. Rodrigues, F. Boavida, V. Pereira, and J. S. Silva, "A survey of iot management protocols and frameworks," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 2, pp. 1168–1190, 2019.

[6] T. Salman and R. Jain, "Networking protocols and standards for internet of things," *Internet of things and data analytics handbook*, pp. 215–238, 2017.

[7] A. Triantafyllou, P. Sarigiannidis, and T. D. Lagkas, "Network protocols, schemes, and mechanisms for internet of things (iot): Features, open challenges, and trends," *Wireless communications and mobile computing*, vol. 2018, no. 1, p. 5349894, 2018.

[8] O. Ali, M. K. Ishak, M. K. L. Bhatti, I. Khan, and K.-I. Kim, "A comprehensive review of internet of things: Technology stack, middlewares, and fog/edge computing interface," *Sensors*, vol. 22, no. 3, 2022.

[9] T. Stefanec and M. Kusek, "Comparing energy consumption of application layer protocols on iot devices," in *2021 16th International Conference on Telecommunications (ConTEL)*, pp. 23–28, 2021.

[10] A. Shahrokhi and M. Ahmadi, "Power evaluation of iot application layer protocols," in *2023 7th International Conference on Internet of Things and Applications (IoT)*, pp. 1–7, 2023.

[11] J. Hofer and S. Pawaskar, "Impact of the application layer protocol on energy consumption, 4g utilization and performance," in *2018 3rd Cloudification of the Internet of Things (CIoT)*, pp. 1–7, 2018.

[12] S. Krug, I. Shallari, and M. O'Nils, "A case study on energy overhead of different iot network stacks," in *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*, pp. 528–529, 2019.

[13] R. Canek, P. Borges, C. Taconet, S. Voulgaris, and D. Eyers, "Analysis of the impact of interaction patterns and iot protocols on energy consumption of iot consumer applications," in *Distributed Applications and Interoperable Systems*, vol. 13272 of *Lecture Notes in Computer Science*, pp. 131–147, Switzerland: Springer International Publishing AG, 2022.

[14] N. Naik, "Choice of effective messaging protocols for iot systems: Mqtt, coap, amqp and http," in *2017 IEEE International Systems Engineering Symposium (ISSE)*, pp. 1–7, 2017.

[15] J. Dizdarević, F. Carpio, A. Jukan, and X. Masip-Bruin, "A survey of communication protocols for internet of things and related challenges of fog and cloud computing integration," *ACM computing surveys*, vol. 51, no. 6, pp. 1–29, 2019.

[16] D. Glaroudis, A. Iossifides, and P. Chatzimisios, "Survey, comparison and research challenges of iot application protocols for smart farming," *Computer networks (Amsterdam, Netherlands : 1999)*, vol. 168, pp. 107037–, 2020.

[17] M. Babiuch, P. Foltýnek, and P. Smutný, "Using the esp32 microcontroller for data processing," in *2019 20th International Carpathian Control Conference (ICCC)*, pp. 1–6, 2019.

[18] L. Tightiz and H. Yang, "A comprehensive review on iot protocols' features in smart grid communication," *Energies*, vol. 13, no. 11, p. 2762, 2020.

[19] K. Hofer-Schmitz and B. Stojanović, "Towards formal verification of iot protocols: A review," *Computer Networks*, vol. 174, p. 107233, 2020.

[20] J. Tournier, F. Lesueur, F. Le Mouël, L. Guyon, and H. Ben-Hassine, "A survey of iot protocols and their security issues through the lens of a generic iot stack," *Internet of Things*, vol. 16, p. 100264, 2021.

[21] E. Al-Masri, K. R. Kalyanam, J. Batts, J. Kim, S. Singh, T. Vo, and C. Yan, "Investigating messaging protocols for the internet of things (iot)," *IEEE Access*, vol. 8, pp. 94880–94911, 2020.

[22] S. Jaloudi, "Communication protocols of an industrial internet of things environment: A comparative study," *Future Internet*, vol. 11, no. 3, p. 66, 2019.

[23] M. Lombardi, F. Pascale, and D. Santaniello, "Internet of things: A general overview between architectures, protocols and applications," *Information*, vol. 12, no. 2, p. 87, 2021.

[24] M. N. Bhuiyan, M. M. Rahman, M. M. Billah, and D. Saha, "Internet of things (iot): A review of its enabling technologies in healthcare applications, standards protocols, security, and market opportunities," *IEEE Internet of Things Journal*, vol. 8, no. 13, pp. 10474–10498, 2021.

[25] R. Luna and S. A. Islam, "Security and reliability of safety-critical rtos," *SN Computer Science*, vol. 2, p. 356, 2021.

[26] A. Eliasz, *Zephyr RTOS Embedded C Programming: Using Embedded RTOS POSIX API*. Springer, 2024.

[27] D. Dublensky, E. Kanunnikov, and T. Makhmudov, "Comparison of time characteristics of real-time operating systems (rtos) on microcontrollers with core architectures arm and risc-v," in *2025 International Russian Smart Industry Conference (SmartIndustryCon)*, pp. 630–635, 2025.

[28] T. Instruments, "Ina226 36v, 16-bit, ultra-precise i2c output current, voltage, and power monitor with alert," Sep 2024. [Accessed 16-06-2025].

[29] S. K. Chakravarty, A. Batra, N. Singh, and G. Kumar, "Low power consumption analysis of communication protocol for the meteorological internet of things," in *2024 12th International Conference on Intelligent Systems and Embedded Design (ISED)*, pp. 01–06, 2024.

[30] netatmo, "Smart home weather station." [Accessed 6-08-2025].

[31] T. Schaub, A. Millon, C. De Zutter, R. Buij, J. Chadœuf, S. Lee, A. Mionnet, and R. H. G. Klaassen, "How to improve the accuracy of height data from bird tracking devices? an assessment of high-frequency gps tracking and barometric altimetry in field conditions," *Animal Biotelemetry*, vol. 11, no. 1, p. 31, 2023.

[32] Quartinocci, "Repository for "energy consumption analysis of application layer protocols in realistic iot uplink scenarios"." https://github.com/Quartinocci/ECAALPRIUS, 2026. GitHub repository, accessed 2026-03-10.

[33] R. Tillaart, "Ina226: Arduino library for the ina226 current/power monitor." https://github.com/RobTillaart/INA226, 2026. GitHub repository, accessed 2026-01-13.

[34] P. Rousseeuw and M. Hubert, "Anomaly detection by robust statistics," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 8, 03 2018.