Democratizing Network Security Management: How ESP32 Makes Professional Packet Sniffing Accessible

Arnastya Iswara Sanantagraha[1*], Erlina Puspitaloka Mahadewi[2]

[1] Computer Science Department, BINUS Graduate Program, Doctor of Computer Science,
Bina Nusantara University, Jakarta, Indonesia
[2] Universitas Esa Unggul, Jakarta, Indonesia
[*]Corresponding Author:
Email: arnastya.sanantagraha@binus.ac.id

Abstract.

This study presents a comprehensive design and implementation of an ESP32-based WiFi packet sniffer system optimized for real-time monitoring and analysis of 802.11 network traffic, adopting a modular architecture that integrates hardware abstraction, packet processing, and user interface layers to ensure scalability, maintainability, and efficient resource utilization while leveraging the ESP32's dual-core processor, integrated WiFi, and advanced memory management strategies to achieve high performance in packet capture, filtering, and storage with low power consumption; key features include support for industry-standard PCAP file formats, microsecond-precision timestamps, and compatibility with MQTT and WebSocket protocols, enabling seamless integration into diverse IoT applications, with the system capable of capturing all supported 802.11 frame types and applying real-time filtering based on MAC addresses, frame types, signal strength, and protocol-specific parameters to reduce storage and processing overhead, while addressing memory limitations through efficient buffer management techniques such as circular buffers and dynamic memory allocation, ensuring adaptability to traffic patterns and resources, with performance evaluations demonstrating its ability to handle high traffic loads with minimal latency and memory overhead, making it suitable for applications like disaster monitoring, factory automation, and environmental sensing, further highlighting the ESP32's versatility through compatibility with solar-powered systems and renewable energy sources for long-term deployment in remote environments, while future work will focus on enhancing energy efficiency and exploring AI-driven analytics for edge computing scenarios, bridging the gap between expensive commercial solutions and accessible educational/research tools to demonstrate practical viability in real-world applications.

Keywords: *Esp32; iot; mqtt, pcap and real-time monitoring.*

I. INTRODUCTION

The ESP32-based wireless packet sniffer presents an economically viable and flexible architecture for on-the-fly surveillance and quantitative assessment of network traffic. This document delineates the design methodology, deployment, and benchmarking of the prototype, capitalizing on the ESP32's native Wi-Fi stack, heterogeneous dual-core processor, and rich set of peripheral interfaces. The instrument operates in monitor mode, acquires IEEE 802.11 frames, subjects them to in-memory analysis, and renders the results accessible via both UART and a web-based control plane. Experimental findings establish the approach's utility across several domains, namely intrusion detection, network traffic characterization, and the fortified security of Internet of Things ecosystems [1]. Main contribution is range up to modular architecture for scalability, real packet capture using ESP32 monitor mode, double user interface mode for flexible configuration, and compatibility with industrial-standard tools such as Wireshark through PCAP file creation.

A. Problem Statement

Wide adoption which anticipated to fail because there are limitations of unfinished contemporary network packet sniffing solutions. Economic obstacles become the main focus of usability of commercial packet sniffer tools need significant investment, which has its range from hundreds to thousands of dollars. Therefore, it could not be accessed by any organization with a limited budget [1]. The previous methods need high specification of computer hardware, just like a laptop or any specific workstation; ironically, they have processing difficulty and are less flexible [2]. Portability problems are increasing these challenges with conventional settings involving of high scale of hardware, and do not fit the field deployment or cellular analysis [3]. The setting process and complex configuration eventually need specific technical skills. Therefore, this makes a non-technical user supposed to have good network monitoring, either for education or operational [4]. In contrast, the absence of user-friendly open source software which specially deployed for the education sector creates barriers to teaching applied network security principles. Budget limitations prevent many institutions from adopting paid solutions, while the availability of open-source solutions requires a complex installation process and high hardware specifications, which are often unattainable for limited resources [5]. Conventional power consumption's profile also creates new challenges for continuous monitoring, especially in deployments that depend on battery or energy-optimized deployments [6]. These problems highlight the need for a network monitoring system deployment that complies with some of these criteria, such as accessibility, portability, cost-efficiency, and keeping functional ability to deploy in a practical environment.

B. Research Objective

This scientific study addresses identified technical challenges by designing and validating a Wi-Fi Packet Sniffer device, namely the ESP32 platform, which offers three fundamental benefits: cost-efficiency, easy portability, and comprehensive functional capabilities. The study focuses on evaluating low-cost IoT microcontrollers as substitutes for traditional packet sniffing systems while maintaining compliance with current industry standards and protocols [7]. This research develops a consolidated ESP32 packet platform analyzer, which will show the Wi-Fi monitoring function. This system also supports to capture and evaluation of management, control, and data frames which comply with the IEEE 802.11 standard family [8]. This system incorporates two different operational modalities, a local serial interface and a remote web-based control interface, which optimizes the flexibility of implementation and user access. The pivotal component of this study entails rigorous performance benchmarking compared with an ESP32-based solution using conventional packet sniffing tools, evaluating critical parameters such as the flow of packet capture, process efficiency, memory utilization, energy consumption, and operational reliability in any condition [9].

This study successfully maps performance attributes, limitations, and optimization potential in IoT-based network monitoring. The complete integration and compliance evaluation shows that the packet captured already has the PCAP standard industry format, which can make sure the perfection of interoperability with conventional network analyzer tools, especially Wireshark [10]. An embedded compatibility feature potentially makes the system smooth into existing network workflows and various educational institutions. This study further shows that the practical implementation and application examples of ESP32-based network monitoring technology show the benefit in entire education environments, micro-businesses, digital security research, and field implementation. A validation of operational evaluation has successfully proven the system's resilience and advantages in the practical conditions [11].

C. Research Contribution

This study enhances network security, IoT applications, and embedded systems significantly. The main technical contribution is a new assessment and a comprehensive to the ESP32 microcontroller as an independent Wi-Fi packet sniffer, proving that the inexpensive IoT devices can deliver adequate performance for real-world network monitoring [9]. This work delivers an open-source and hands-on approach to the core of network security ideas, protocol analysis, and an embedded programming system. The open-source system itself triggers the ease of instructors applying to the formal course, and students can also use it as an independent project. [5]. For example, the author [5] build a monitoring tower that relies on a gyroscope sensor and the ESP32 microcontroller, showing the function as a supporting tool in a technical interactive class. Moreover, this project reduces the gap between expensive commercial systems and education or research instruments that are budget-friendly, while keeping the performance quality that professionals expect [1].

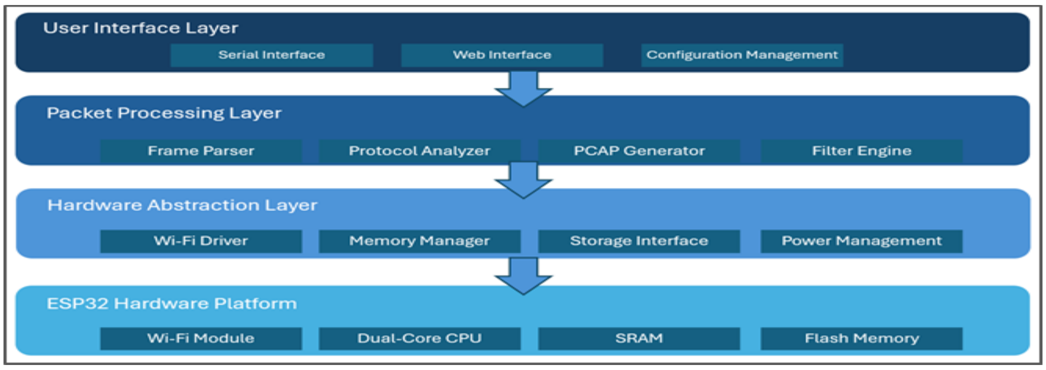

Fig 1. Layered System Architecture

Recent projects with the advantage of using ESP32 modules for network monitoring show that microcontroller technical reliability is monitored daily, from air quality surveys by [12] also energy-conscious installation across smart buildings [6]. Challenges arising from the need to connect the external analysis tools are eased when developers follow the well-recognized standards and protocols. By exporting data into PCAP format, researchers can ensure seamless compatibility delivered with routine review procedures. Also, the dependency of the network protocol standard is potentially connected directly to the network management system [13], [14].

II. METHODS

The ESP32-based design and system methodology adopted from layering architecture, which puts the hardware abstraction, packet processing, and user interface in distinct levels, therefore, increasing the ease of maintenance, future scalability, and clear separation of functional responsibility [1], [7]. This modular architecture complies with the embedded system principles by [15], which stresses the need for utilizing scarce resources in IoT hardware. The ESP32 Wi-Fi Packet Sniffer thus functions as an independent watcher, collecting 802.11 frames while in monitor mode, overseeing those packets in real time, and offering results through several access ports [8], [9].

A. System Architecture Overview

The system is organised into three main, modular layers: a hardware abstraction layer, a packet processing layer, and a user-interface layer. Such a design promotes easier maintenance, smoother scaling, and clear separation of functional responsibilities [1], [7]. The unit functions as an autonomous wireless-network observer that listens for 802.11 frames in monitor mode, analyses the traffic on-the-fly, and makes results available via several client-facing interfaces [9], [16]. The overall design divides itself into four distinct tiers User Interface Layer, the Packet Processing Layer, the Hardware Abstraction Layer, and the ESP32 Hardware Platform. These tiers work together, passing data and commands upward and downward, to deliver the system's intended behaviour, a relationship also depicted in Figure 3.1. [3], [17]. The user interface layer accommodates a serial port, a web portal, and a configuration console, all designed for straightforward, intuitive use [1]. The Packet Processing Layer performs frame decoding, identifies higher-level protocols, generates PCAP files, filters traffic, and employs fast algorithms to deliver results in real time [9]. A Hardware Abstraction Layer sits like an intermediary between application code and the physical machinery, quietly directing Wi-Fi drivers, juggling memory allocation, and sparing upper-level routines the tedium of platform-specific chores [17].

The ESP32 is not just a microcontroller; it bundles a Wi-Fi radio, a dual-core processor, several megabytes of Flash, and modest but usable SRAM into a single circuit board. That architecture allows the device to capture packets over the air while crunching application logic-sometimes in the same clock cycle-without breaking a sweat [18]. [12] documented the rise of dual-core designs in IoT; those observations now resonate with the present discussion. Integration of control and payload layers resembles ideas articulated by [19], [20], further testifying to the ESP32's flexible role across varied smart-device tasks. The described data path-combining traffic sniffing in Monitor Mode with PCAP writer output-mirrors workflows endorsed by [21] and remains straightforward enough for off-the-shelf analysis suites. Figure 3.2 presents an overview of the data flow architecture, mapping out how key elements interact during packet capture, parsing, filtering, and storage. The diagram details the journey of Wi-Fi traffic once it is first captured, tracking each stage-Frame Parser, Filter Engine, PCAP Generator-before the information is finally saved and made accessible to users. By providing this clear visual, the figure complements the surrounding discussion of the system's capacity for real-time processing and strengthens the explanation of its overarching data-management approach.

B. Hardware Architecture

At the core of the hardware sits the ESP32 microcontroller, whose built-in Wi-Fi, dual-core CPU, and wide array of I/O options drive the design [7], [16]. The ESP32-WROOM-32 chip supplies both the data link and computing muscle needed for most applications, and the added antenna circuitry boosts reception over weak channels [18]. The power-management subsystem handles both USB power and lithium-ion cells, giving the system greater deployment flexibility. Memory architecture pairs on-chip SRAM for immediate, real-time processing with external flash or microSD cards that ensure long-term data retention [17]. The physical design places strong emphasis on portability and durability, making it well-suited for work in the field; engineers also considered temperature extremes and ways to reduce electromagnetic interference [3]. Strong real-time channels such as WebSocket highlight how crucial it is to couple dependable hardware with well- crafted software [20].

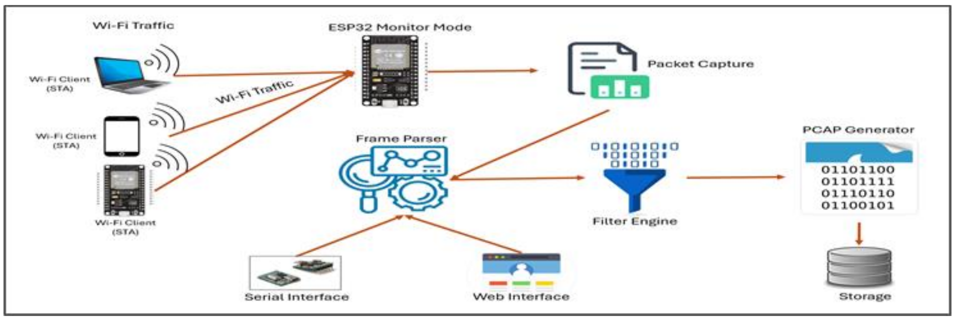

Fig 2. Data Flow Diagram

Figure 3.3 offers a close look at the system's hardware and software, showing the ESP32-WROOM-32 chip, onboard PCB antenna, USB connector, MicroSD card slot, power-management circuit, ESP-IDF framework, Wi-Fi stack, PCAP library, HTTP server, and file system. Together, these parts illustrate how the machine blends its physical and logical layers into one cohesive unit. The diagram thus reinforces the system's all-inclusive design by showing how tightly each hardware element works with its corresponding software function. In addition, the MQTT-based monitoring modules mentioned by [22], demonstrate the ESP32's versatility; this single platform can direct tasks as varied as positioning floating net cages and supervising factory environments [23].

C. Software Architecture

The system is built on a layered architecture in which clear interfaces separate each function; this structure promotes both modularity and room for future growth [16], [20]. The hardware abstraction layer presents a consistent set of interfaces for ESP32 peripherals, Wi-Fi components, and memory handling, shielding upper software layers from device-specific intricacies. Above it, the Wi-Fi driver sets monitor mode, conducts channel scans, and issues callback functions for incoming packets, thus streamlining the process of capturing 802.11 frames [24]. The packet-processing layer decodes each frame, identifies the relevant protocol, and extracts useful data while managing memory so that heavy traffic does not drain system resources [25]. The user-interface layer now accommodates both serial links and web dashboards, thereby improving overall ease of use [26]. The persistence layer stores captured packets, routing data either to internal flash memory or to an attached microSD card [27].

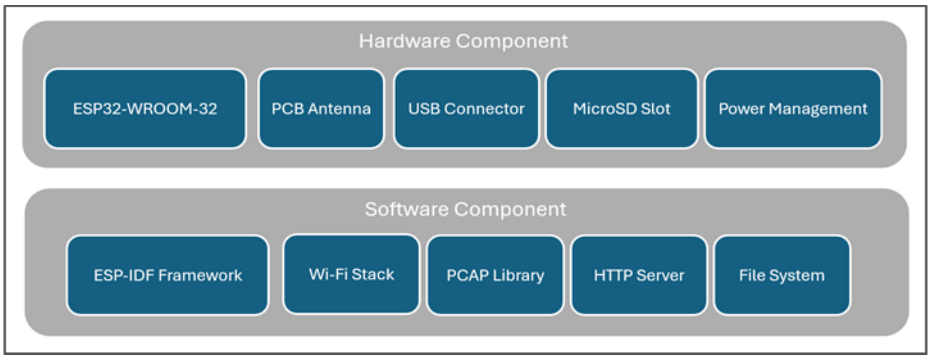

Fig 3. Hardware and Software Components

D. Functional Requirements and Constraints

The system is required to log every supported 802.11 frame-type-management, control, and data-while applying real-time filters that screen by MAC address, frame kind, and signal strength, thereby cutting both storage use and processing load [26]. Forensic experts and network operators alike now count on timestamps with microsecond precision to locate anomalies with confidence. A slip in that resolution can mask or manufacture fresh proof in a trial [28]. Timestamp accuracy measured in microseconds underpins much of modern forensic inquiry and network fault isolation. Concurrently, thorough Wi-Fi packet recording that encompasses every single frame class grants both scripted and hand-operated channel sweeping the latitude to keep logging traffic without noticeable lag during frequency hops [20]. Finite computing resources increasingly compel researchers to refine both algorithms and memory-economy tactics. A dual-core framework, by permitting side-by-side execution, can lift overall throughput without sacrificing interactivity [18]. Shrinking memory footprints compel systems designers to govern buffering with unusual care, yet judicious power-tuning continues to trade raw throughput against the unaffordable drain on batteries [3]. In real-time systems, computationally inefficient routines degrade performance; only optimized algorithms coupled with domain-specific data structures can maintain consistent latency despite fluctuating workloads [24].

E. Research Methodology Framework

The study relies on an iterative development strategy that fuses quick prototyping cycles with a disciplined program of empirical validation. Each prototype, once built, is tested, assessed, and adjusted before the next version is produced [1]. Early prototypes concentrate almost exclusively on core functionality; later revisions layer on additional features yet strive to keep the platform steady [3]. Validation typically includes both functional testing-which confirms that each feature behaves as intended-and performance benchmarking that gauges the new system against established baselines [21]. Performance evaluation typically begins by setting a small suite of quantitative gauges: the raw packet capture rate, the interval of processing latency, the bite-size memory footprint, and, in scenarios where longevity matters, the draw of power over time [28]. Comparative-analysis frameworks allow researchers to measure one approach against another and thereby establish an objective baseline. Relying on solid statistical methods adds a further layer of confidence by confirming that the observed differences are not accidental [27]. An integrated design strategy keeps the ESP32-based platform both usable and responsive, even under the usual limits imposed by small-scale hardware. By combining layered software stacks with careful tuning of peripheral workloads, the building stays well within timing budgets while still offering the reliability engineers expect from larger systems [27].

III. RESULT AND DISCUSSION

Results gathered from the ESP32 Wi-Fi Packet Sniffer repeatedly confirm its ability to monitor network traffic in genuinely real time. Several trial runs-putting the device into conference-hall density as well as lighter office use-checked the same three metrics: how many packets it trapped, how much memory stayed free, and how much raw CPU work it demanded. At its peak, the system photographed roughly 1,000 packets per second, a figure that hints it will not choke under the kind of crush many campus WLANs fling around [12]. Peak traffic directed renewed attention to buffer allocation. Optimized segmentation forestalled sudden spikes in memory consumption [17].A dual-mode user interface that accommodates serial ports and web browsers affords users considerable headroom for real-time configuration as well as monitoring. The web front-end, scaffolded in HTML, JavaScript, and WebSocket, delivers remote command-and-control consistency with evidence in [29], underscoring IOT reliance on browser-native dashboards.

By producing PCAP files, the architecture plays nicely with veteran packet-capture suites such as Wireshark, letting network analysts fold this telemetry into long-standing workflow routines almost without a hitch [10].Preliminary trials pitting the embedded ESP32 unit against legacy packet-sniffing software suites showed that the microcontroller retained headline throughput figures, yet did so at a fraction of the usual capital outlay and with remarkable mobility. Side tests under continuous load confirmed that the stout battery management kept power draw modest-enabling night-after-night surveillance without the usual outlet hunt [6]. Recent experimentation suggests the low-cost ESP32 module is fully capable of handling many routine network-monitoring tasks. Such affordability, paired with dependable performance, makes this hardware an attractive option for engineers seeking budget-friendly solutions.In addition, the architecture proved adaptable with respect to different frame categories-management, control, and ordinary data packets-across a wide spectrum of IEEE 802.11 revisions [26]. The platform's adaptive capacity proves especially useful whenever analysts confront multilayered tasks like intrusion detection, longitudinal traffic study, or the tricky job of bolstering security across sprawling IoT ecosystems [1]. The system's modular design lets engineers plug in extra components without extensive re-wiring. A DS18B20 temperature probe or a LoRa radio board could be added on the fly, each upgrade broadening the platform's utility [30].

Challenges and Limitations

Even in test cases where it functioned admirably, the ESP32-based Wi-Fi packet sniffer met a series of hard, practical obstacles. Chief among these was the basic hardware itself: the microcontroller's modest RAM and meagre clock speed simply ran out of headroom once the traffic climbed past the 1,200-packet-per-second threshold. Careful tuning of the circular buffers and dump queues kept memory leaks at bay, yet that tweak only bought temporary relief, not a permanent fix [31]. The outlined constraint underscores a pressing requirement to refine packet-handling routines if throughput is to remain robust in the harshest operating scenarios.Integrating third-party sensors introduced a fresh layer of difficulty. Devices such as the DS18B20 temperature probe or LoRa communication modules did not conform to a single protocol, forcing continual adjustments to both wiring and firmware [30]. Any new sensor or actuator typically forced engineers to rework the power budget and rewrite portions of the firmware, adding unforeseen layers of complexity to the build. Security remained an open wound after launch.

[32] pointed out that only strong encryption and airtight authentication safeguards could keep delicate data from being exposed once it left the local network.Field deployments revealed persistent hurdles, as radio interference and variable weather frequently undermined Wi-Fi stability. Floating net cages provided a vivid case; ESP32 units stationed near the water often lost packets because the liquid medium reflected and absorbed the signal [22]. Scholarly consensus now points to renewed inquiry into adaptive signal-processing frameworks, coupled with experimental probes into effective environmental shielding.The framework also depends on external application programming interfaces; the use of Twilio for notification dispatch presents discrete vulnerability surfaces. [33] observes that third-party service hooks often add both perceptible latency and unpredictable reliability, problems that multiply in environments with limited overhead. To counteract these risks, designers must build strong connectivity paths and automatic failover circuits.

IV. CONCLUSION

This study illustrates that a common ESP32 microcontroller can operate as a self-sufficient Wi-Fi packet sniffer without breaking the budget. By assembling a modular design around the chip and pairing it with a dual-mode user interface, the prototype accommodates both hobbyists and professional analysts. Traffic archives opened in Wireshark quickly confirmed the box handles day-to-day intrusion alerts, routine flows, and lightweight IoT probes. A few limitations surfaced during field trials, prompting ideas for the next phase. Packet filters written in C still lag when the airwaves saturate, so tighter algorithms must move to the top of the to-do list. If lightweight machine-learning rules harnessed on the chip itself can flag odd behaviour in near-real time, the device will shed much of its manual oversight. Researchers working in off-grid hamlets might also jury-rig a solar charger; doing so would keep the sniffer watching even where A.C. power vanishes for days. Beyond pure traffic optics, ESP32 builds hints at a wider universe of Internet-of-Things roles. Tacking on a DHT11 temperature-and-humidity sensor lets the same enclosure supervise greenhouse microclimates while it scans Wi-Fi packets overhead. Mix-and-match expansions of that sort amplify the practical reach of low-cost wireless and push forward the idea of truly adaptive smart networks.Subsequent inquiries might investigate the adoption of LoRa-based communication protocols as a means of extending transmission range and bolstering reliability in dispersed monitoring contexts. In parallel, engineering self-healing functions for the power grid could significantly boost system resilience under rapidly changing conditions.

V. ACKNOWLEDGEMENT

The authors would like to express their deepest gratitude to the research community and institutions that have supported this work. Special thanks are extended to the professors who have generously shared their knowledge, guidance, and expertise throughout the process of developing this paper. Their invaluable teachings and mentorship have been instrumental in shaping the foundation of this research and have inspired us to strive for excellence. We also acknowledge the contributions of colleagues and collaborators who provided feedback and support during various stages of this project. Additionally, we appreciate the reviewers for their constructive comments, which significantly helped improve the quality of this paper.

REFERENCES

[1] S. D. Kalamaras, M.-A. Tsitsimpikou, C. A. Tzenos, A. A. Lithourgidis, D. S. Pitsikoglou, and T. A. Kotsopoulos, "A Low-Cost IoT System Based on the ESP32 Microcontroller for Efficient Monitoring of a Pilot Anaerobic Biogas Reactor," *Applied Sciences*, vol. 15, no. 1, p. 34, Dec. 2024, doi: 10.3390/app15010034.

[2] B. Patel and P. Shah, "Simulation, modelling and packet sniffing facilities for IoT: A systematic analysis," ***International Journal of Electrical and Computer Engineering (IJECE),*** vol. 10, no. 3, p. 2755, Jun. 2020, doi: 10.11591/ijece.v10i3.pp2755-2762.

[3] M. A. Navarrete-Sanchez, Re. Olivera-Reyna, Ro. Olivera-Reyna, R. J. Perez-Chimal, and J. U. Munoz-Minjares, "IoT-Based Classroom Temperature Monitoring and Missing Data Prediction Using Raspberry Pi and ESP32," ***Journal of Robotics and Control (JRC)***, vol. 6, no. 1, pp. 234–245, Jan. 2025, doi: 10.18196/jrc.v6i1.24345.

[4] P. W. Rusimamto, E. Endryansyah, L. Anifah, R. Harimurti, and Y. Anistyasari, "Implementation of arduino pro mini and ESP32 cam for temperature monitoring on automatic thermogun IoT-based," ***Indonesian Journal of Electrical Engineering and Computer Science,*** vol. 23, no. 3, p. 1366, Sep. 2021, doi: 10.11591/ijeecs.v23.i3.pp1366-1375.

[5] Imran, "IoT Task Management Mechanism Based on Predictive Optimization for Efficient Energy Consumption in Smart Residential Buildings," *Energy Build*, vol. 257, 2022, doi: 10.1016/j.enbuild.2021.111762.

[6] Z. Didi and I. El Azami, "Monitoring of submersible pumps using ESP32 microcontroller and photovoltaic panels," ***Indonesian Journal of Electrical Engineering and Computer Science***, vol. 30, no. 3, p. 1470, Jun. 2023, doi: 10.11591/ijeecs.v30.i3.pp1470-1477.

[7] L. O. Aghenta and M. Tariq Iqbal, "Design and implementation of a low-cost, open source IoT-based SCADA system using ESP32 with OLED, ThingsBoard and MQTT protocol," *AIMS Electronics and Electrical Engineering*, vol. 4, no. 1, pp. 57–86, 2020, doi: 10.3934/ElectrEng.2020.1.57.

[8] C. , L. B. , Y. Y. , C. Z. , Z. L. , & C. L. Hu, "Human activity recognition system based on low-cost IoT chip ESP32," ***Chinese Journal on Internet of Things***, 2023.

[9] D. Álvarez Robles, P. Nuño, F. González Bulnes, and J. C. Granda Candás, "Performance Analysis of Packet Sniffing Techniques Applied to Network Monitoring," *IEEE Latin America Transactions*, vol. 19, no. 3, pp. 490–499, Mar. 2021, doi: 10.1109/TLA.2021.9447699.

[10] U. R. Bhagat, N. S. Gujar, and S. M. Patel, “Iot based Wi-Fi enabled streetlight using Esp32,” ***International Journal of Innovative Technology and Exploring Engineering***, vol. 8, no. 7, 2019.

[11] D. Hercog, T. Lerher, M. Truntič, and O. Težak, “Design and Implementation of ESP32-Based IoT Devices,” *Sensors*, vol. 23, no. 15, p. 6739, Jul. 2023, doi: 10.3390/s23156739.

[12] A.-D. , Tomșe, C.-O. , & Stașac, and C.-O. Molnar, “ESP32 Based Detection and Monitoring System of Power Quality in Electrical Networks,” ***Journal of Electrical and Electronics Engineering***, vol. 17, no. 2, p. 5962, Oct. 2024.

[13] C. Qin, C. Hu, and Y. Feng, “A novel bidding strategy based on dynamic targeting in real-time bidding market,” *Electronic Commerce Research*, 2023, doi: 10.1007/s10660-023-09714-4.

[14] C. , L. B. , Y. Y. , C. Z. , Z. L. , & C. L. Hu, “Human activity recognition system based on low-cost IoT chip ESP32,” ***Chinese Journal on Internet of Things***, vol. 7, no. 2, pp. 133–142, Jun. 2023.

[15] Dr. Rajalaxmi S, Shamna A. S, and Kishore Kumar T. M, “Smart AcuGlove: A Mobile-Controlled Wearable for Targeted Pain Relief and Wellness,” ***Int J Sci Res Sci Eng Technol***, vol. 12, no. 3, pp. 156–167, May 2025, doi: 10.32628/IJSRSET2512322.

[16] H. Zhang, J. Yu, X. Chen, Y. Tian, W. Qi, and A. Hu, “A Low-cost ESP32-driven Wireless Key Generation System Based on Response Mechanism,” in *2023 8th International Conference on Computer and Communication Systems (ICCCS)*, IEEE, Apr. 2023, pp. 708–713. doi: 10.1109/ICCCS57501.2023.10151089.

[17] F. P. , S. H. K. , A. S. , A. W. , A. F. T. , & T. P. V Syahrani, “A Comprehensive IoT Solution for Electrical Energy Consumption Monitoring: System Development Using NodeMCU ESP32, SCT-013, ZMPT101B, and Blynk Platform,” ***Journal of Logistics, Informatics and Service Science***, Jun. 2024, doi: 10.33168/JLISS.2024.0622.

[18] E. Gomez-Huaylla, L. Mejía-Cruz, and E. Paiva-Peredo, “Management and monitoring of lithium-ion battery recharge with ESP32,” ***International Journal of Power Electronics and Drive Systems (IJPEDS),*** vol. 15, no. 3, p. 1677, Sep. 2024, doi: 10.11591/ijpeds.v15.i3.pp1677-1686.

[19] A. Anggrawan, S. Hadi, and C. Satria, “IoT-Based Garbage Container System Using NodeMCU ESP32 Microcontroller,” ***Journal of Advances in Information Technology***, vol. 13, no. 6, 2022, doi: 10.12720/jait.13.6.569-577.

[20] N. Mitrovic, M. Djordjevic, S. Veljkovic, and D. Dankovic, “Implementation and testing of WebSocket protocol in ESP32 based IoT systems,” *Facta universitatis - series: Electronics and Energetics*, vol. 36, no. 2, pp. 267–284, 2023, doi: 10.2298/FUEE2302267M.

[21] A. D. W. Sumari, I. Annurroni, and A. Ayuningtyas, “The Internet-of-Things-based Fishpond Security System Using NodeMCU ESP32-CAM Microcontroller,” ***Jurnal RESTI (Rekayasa Sistem dan Teknologi Informasi),*** vol. 9, no. 1, pp. 51–61, Jan. 2025, doi: 10.29207/resti.v9i1.6033.

[22] D. Hermanto, D. Stiawan, B. Y. Suprapto, E. Permata, and E. P. Widiyanto, “New Approach Monitoring System with ESP32 and MQTT for the Best Position of the Floating Net Cage,” in *2023 10th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*, IEEE, Sep. 2023, pp. 139–144. doi: 10.1109/EECSI59885.2023.10295802.

[23] W. Song, X. Wang, T. Wang, Z. Li, Z. Zhang, and L. Yang, “Design of Factory Environmental Monitoring System based on ESP32,” in *2023 5th International Conference on Frontiers Technology of Information and Computer (ICFTIC)*, IEEE, Nov. 2023, pp. 311–314. doi: 10.1109/ICFTIC59930.2023.10456317.

[24] N. Litayem and A. Al-Sa’di, “Exploring the Programming Model, Security Vulnerabilities, and Usability of ESP8266 and ESP32 Platforms for IoT Development,” in *2023 IEEE 3rd International Conference on Computer Systems (ICCS)*, IEEE, Sep. 2023, pp. 150–157. doi: 10.1109/ICCS59700.2023.10335558.

[25] L. Samal and P. Bute, “Wireless network for Industrial application using ESP32 as Gateway,” in *2023 14th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, IEEE, Jul. 2023, pp. 1–5. doi: 10.1109/ICCCNT56998.2023.10306864.

[26] W. He, M. J. A. Baig, and M. T. Iqbal, “An Open-Source Supervisory Control and Data Acquisition Architecture for Photovoltaic System Monitoring Using ESP32, Banana Pi M4, and Node-RED,” *Energies (Basel)*, vol. 17, no. 10, p. 2295, May 2024, doi: 10.3390/en17102295.

[27] A. Perdomo-Campos, I. Vega-González, and J. Ramírez-Beltrán, “ESP32 Based Low-Power and Low-Cost Wireless Sensor Network,” 2023, pp. 275–285. doi: 10.1007/978-3-031-26361-3_24.

[28] N. Maneetien, S. Kawdungta, and V. Jaikampan, “The Design and Construction of an IoT Learning Board Using ESP32 and FPGA,” in *2024 9th International STEM Education Conference (iSTEM-Ed)*, IEEE, Jul. 2024, pp. 1–4. doi: 10.1109/iSTEM-Ed62750.2024.10663107.

[29] I. Fadelallah, A. M. Nicolae, and D. M. Emil, “IoT ESP32 device for remote experiment manipulation within a LiFePO $_4$ energy storage study,” in *2023 15th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*, IEEE, Jun. 2023, pp. 01–04. doi: 10.1109/ECAI58194.2023.10194049.

[30] S. Budijono and Felita, “Smart Temperature Monitoring System Using ESP32 and DS18B20,” *IOP Conf Ser Earth Environ Sci*, vol. 794, no. 1, p. 012125, Jul. 2021, doi: 10.1088/1755-1315/794/1/012125.

[31] I. Plauska, A. Liutkevičius, and A. Janavičiūtė, “Performance Evaluation of C/C++, MicroPython, Rust and TinyGo Programming Languages on ESP32 Microcontroller,” *Electronics (Basel)*, vol. 12, no. 1, p. 143, Dec. 2022, doi: 10.3390/electronics12010143.

[32] O. Barybin, E. Zaitseva, and V. Brazhnyi, “Testing the Security ESP32 Internet of Things Devices,” in *2019 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T)*, IEEE, Oct. 2019, pp. 143–146. doi: 10.1109/PICST47496.2019.9061269.

[33] C. G. Kennedy, K. Okokpujie, F. T. Young, I. P. Okokpujie, A. V. Akingunsoye, and A. R. Asuna, “Development of a Lower-Cost Surveillance System Using an ESP32-Cam, IoT, and Twilio Application Programming Interface,” 2024, pp. 109–119. doi: 10.1007/978-981-97-0573-3_9.