International Journal of Reconfigurable and Embedded Systems (IJRES)
Vol. 15, No. 3, November 2026, pp. 837∼847
ISSN: 2089-4864, DOI: 10.11591/ijres.v15.i3.pp837-847

Throughput bottleneck shift in an embedded Wi-Fi gateway

Jianan Zhao, Yuanbin Mu
Department of Applied Electronic Technology, School of Internet of Things Technology, Hangzhou Polytechnic, Hangzhou, China

Article Info

Article history:

Received Apr 22, 2026
Revised Aug 24, 2026
Accepted Sep 11, 2026

Keywords:

ESP32-S3
Internet of things
Packet loss
Software access point
Throughput ceiling
Transmission control protocol gateway
Two-factor analysis of variance

ABSTRACT

An ESP32 software access point (SoftAP) often serves as the sole transmission control protocol (TCP) gateway in field deployments without routers; however, prior benchmarks varied only one factor at a time. We varied two factors that differ in practice: TCP connection count (3, 5, and 10) and per-packet payload size (10, 64, and 256 bytes), yielding 9 configurations. Each was tested at increasing send frequencies, with each frequency repeated 3 times, for a total of 246 60-second runs. Two-way analysis of variance (ANOVA) on the 30 Hz data confirms that both factors and their interaction are significant (all $p < 0.001$). With 10 connections and 10-byte payloads, the gateway sustained 3,237 packets per second (pps) with zero loss; switching to 256-byte payloads dropped the safe rate below 946 pps and pushed central processing unit (CPU) utilization past 100%. In our experiments, system constraints shift from per-frame media access control (MAC) overhead to channel-bandwidth saturation as payload size increases. We present a stability-region chart that helps designers understand the limits of system stability, optimize system performance, and choose the appropriate connection count and message size.

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

Corresponding Author:

Yuanbin Mu
Department of Applied Electronic Technology, School of Internet of Things Technology
Hangzhou, China
Email: muyuanbin@qq.com

1. INTRODUCTION

ESP32 serves as a data gateway in regions without wireless network coverage (e.g., agricultural areas or disaster sites) [1], [2], while also performing local data aggregation and backhaul services [3]–[5]. ESP32-enabled gateways are widely deployed in low-cost private deployments [6]. Recent studies in this journal demonstrated ESP32-based internet of things (IoT) systems for flood warning [7] and smart parking [8]. The sensor data they process are time-sensitive and unrecoverable [9], [10]. Unfortunately, prior benchmarks have varied only one factor at a time, leaving the combined effect unknown. As a result, the zero-loss throughput ceiling becomes the primary concern for system designs [11]. This ceiling is a fine-grained stability limit that conventional iperf tests cannot precisely resolve because of their best-effort traffic generation.

The ESP32 family has become very popular as a general-purpose IoT prototyping platform. Prior work characterized it from multiple perspectives: Naik [1] surveyed its messaging protocols and dual-radio wireless fidelity (Wi-Fi)/Bluetooth low energy (BLE) advantages; Maier *et al.* [2] validated its field-readiness; Plauska *et al.* [12] provided compute baselines in C/C++, MicroPython, and Rust, though not for networking workloads; at the system level, ESP32 has been applied in message queuing telemetry transport (MQTT)-based automation [13] and reusable IoT gateways [6], both focusing on application-layer interoperability rather than throughput limits. Recent work explored ESP32 as a smart-home gateway [14] and evaluated its real-time synchronization performance [15], but multi-factor stress testing under concurrent transmission control proto-

col (TCP) loads remains unaddressed. Although the official documentation [3], [16] states that the maximum number of associated stations is 10, this limit provides no indication of the sustainable packet rate without loss. To address this uncertainty, we performed a load-sweep experiment.

Every 802.11 transmission requires an unavoidable overhead before the first bit of payload reaches the air. Bianchi [17] modeled this cost with a Markov-chain model for the distributed coordination function (DCF), characterizing a total delay (DCF inter-frame space, random backoff, and physical layer preamble (PHY preamble)) between 200 and 250 μs per frame, and independent of the payload size. The IEEE 802.11-2020 standard [18] specifies these timing parameters; Tselishchev *et al*. [9] showed that greater contention among stations diminishes aggregate throughput with higher collision probability; however, their simulations assumed idealized media access control (MAC) layer conditions, ignoring the software-stack overhead present in real system on chip (SoC) implementations. Additionally, Montori *et al*. [19] examined the energy efficiency of Wi-Fi for IoT and offer a view of throughput through a MAC-layer constraint. Nevertheless, all these studies treat the radio layer in isolation. For an integrated software access point (SoftAP) gateway, the radio subsystem and the TCP/IP stack share the same dual-core central processing unit (CPU). Consequently, MAC-layer overhead and software processing latencies are tightly coupled: precisely the hardware-software interaction that our experiments were designed to investigate.

The lightweight internet protocol (lwIP) stack, defined by Dunkels [20], remains the most popular TCP/IP stack for resource-constrained embedded systems. Espressif distributes lwIP in the espressif IoT development framework (ESP-IDF) (v2.1.2) [21], which is specifically tuned so that TCP window sizes, maximum segment size (MSS), and memory-pool depth can be set to the required level. Works on the overall IoT-edge-cloud continuum, such as those by Gkonis *et al*. [22], Perera *et al*. [23], point out that TCP congestion at the gateway layer is increasingly becoming a bottleneck in the end-to-end data pipeline. Although this is widely acknowledged, there are few studies of the ESP32-S3's limits in the literature, particularly when it runs SoftAP while simultaneously handling several high-throughput TCP sessions. Our study fills this gap by designing a stress-test procedure to measure the performance ceiling under multi-factor conditions. In future high-density deployments, we can offload TCP/IP processing to a field-programmable gate array (FPGA) fabric or a dedicated very-large-scale integration (VLSI) network processor. This approach significantly reduces software bottlenecks. Identifying bottleneck positions is essential for most hardware acceleration schemes.

Existing ESP32 Wi-Fi benchmarks mainly focus on single-stream TCP/user datagram protocol (UDP) throughput [1], [12]. These benchmarks measure the raw pipe capacity of a gateway, but do not capture its response to synchronized short-packet bursts from multiple sources. Two factors that vary simultaneously in real deployments are the number of concurrent TCP connections and the payload size. How each factor affects the performance bottleneck of a gateway is not well understood. Increasing the connection count puts more pressure at the system level (ACKs, socket-buffer allocation, and lwIP context switching); increasing the message length changes the ratio of fixed 802.11 MAC-layer overhead (distributed inter-frame space (DIFS), backoff, and preamble) to actual data payloads [20]–[24]. We cannot predict from theoretical analysis alone how the performance degradation caused by more connections is amplified or dampened by longer messages; empirical observation is needed to find the stable operating boundaries.

In this paper, we evaluate the ESP32 in SoftAP mode through a controlled two-factor experiment. Our contributions are: i) a test matrix covering 246 trials, crossing connection count (3, 5, and 10) with payload size (10 B, 64 B, and 256 B), repeated three times. These levels represent minimal sensor readings, moderate multi-sensor telemetry, and large diagnostic packets, respectively; the connection counts cover low- to high-density deployments, with 10 being the SoftAP firmware limit; ii) identification of a bottleneck shift: for small payloads, throughput is limited by MAC-layer overhead; for large payloads, by channel-bandwidth saturation. The interaction is confirmed statistically significant ($p < 0.001$) by two-way analysis of variance (ANOVA)—optimizing only one factor is ineffective once the payload size shifts the bottleneck to a different subsystem; and iii) a stability-region chart mapping the safe operating envelope for designers, which also informs the configuration of reconfigurable hardware gateways or FPGA accelerators in high-density IoT.

2. METHOD

2.1. Hardware and software platform

For our experiments, we used an ESP32-S3-N16R8 integrated on a DevKitC-1 development board [16]. To emulate a resource-constrained deployment, we disabled the 8 MB pseudo-static random-access mem-

ory (PSRAM), leaving about 281 KB of usable heap at boot. The software stack ran the Arduino framework on ESP-IDF v4.4.7, with the stock lwIP v2.1.2 stack and default TCP parameters. The Wi-Fi driver was pinned to Core 0 and the application loop to Core 1. Otherwise, Wi-Fi interrupts could steal CPU cycles away from the packet-parsing logic, and we would not be able to tell which core runs out of capacity first. The full configuration is given in Table 1.

Table 1. Full configuration of the experimental platform

Parameter	Value
SoC	ESP32-S3 (Xtensa LX7 dual-core)
CPU frequency	240 MHz
Flash/PSRAM	16 MB/8 MB (PSRAM not enabled)
SRAM (heap)	359 KB total and ∼281 KB free at boot
Wi-Fi	802.11 b/g/n, 2.4 GHz, and PCB antenna
SoftAP config	Channel 6, WPA2-PSK, and max 10 STAs
Firmware	Arduino framework and ESP-IDF v4.4.7
lwIP	v2.1.2, default TCP_WND/MSS, and Nagle on
Task allocation	Wi-Fi on Core 0 and app loop on Core 1

For traffic generation, a laptop-based Python program served as the client, connecting to the ESP32 SoftAP at 1.5 m. The test environment was a single-person office with no competing access points on Channel 6, so co-channel interference was minimal. To model high-concurrency sensor traffic, each TCP connection was implemented as an independent Python thread, with each thread handling data from 11 logical terminals. The physical topology and the two-factor experimental matrix are shown in Figure 1.

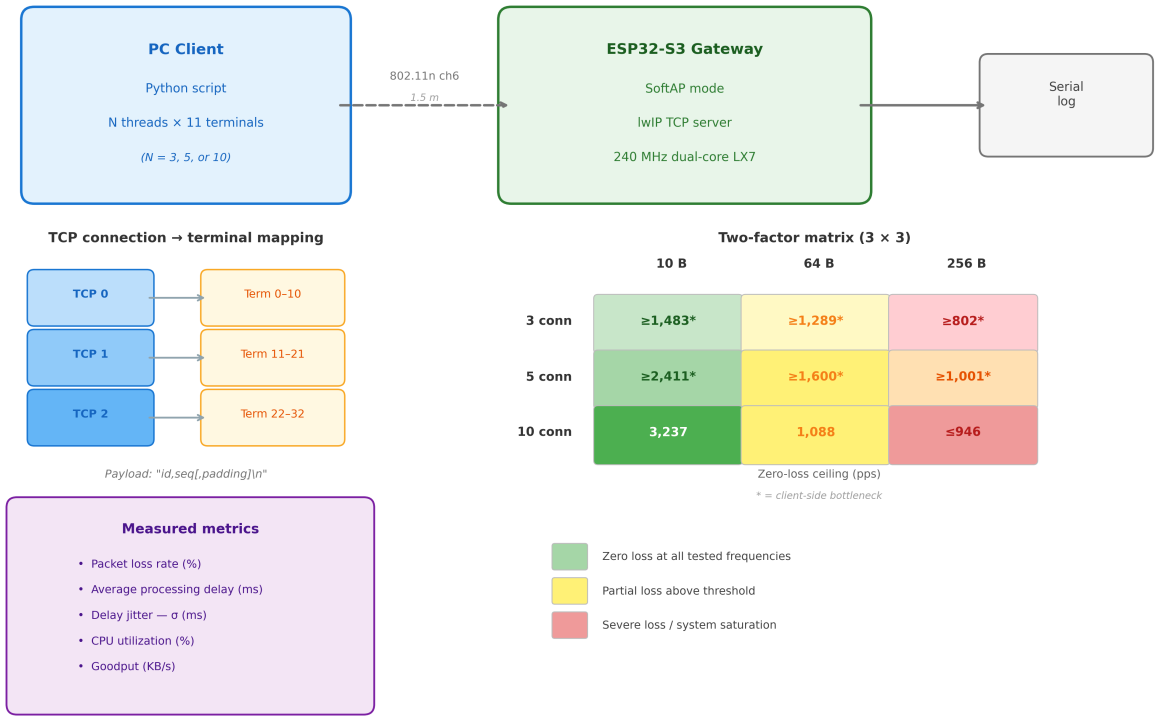


Figure 1. Experimental setup and two-factor test matrix

2.2. Two-factor experimental design

For our evaluation, we used a fully crossed factorial design to test the system for two factors: i) Factor A: concurrent TCP connections: three levels (3, 5, and 10); since each connection multiplexes data for

11 logical terminals, the total count of terminals increased to 33, 55, and 110, respectively and ii) Factor B: payload size: three levels (10, 64, and 256 B). The 10-byte payload packet used for baseline testing consists of a terminal identifier (ID) and an American Standard Code for Information Interchange (ASCII) sequence number. We pad the packet to 64 bytes and 256 bytes using padding characters. In this paper, to ensure consistency in application-layer overhead, the gateway firmware code parses only the terminal ID and sequence number, discarding all padding bytes.

By crossing the two factors, we obtained nine distinct test configurations. For each configuration, we designed eight transmission frequencies ranging from 10 to 50 Hz. Every trial ran for 60 seconds and was repeated three times, yielding a total of 246 valid runs. For the 3- and 5-connection configurations, additional higher frequencies were applied with the intent of probing the Python client's saturation behavior, and thereby localizing the global interpreter lock (GIL) bottlenecks.

2.3. Measurement metrics

We evaluated each experimental run using five metrics: i) zero-loss throughput ceiling (pps): the highest packet rate at which every packet arrived intact; ii) loss rate (%): ratio of lost to total received packets, computed by scanning for gaps in per-terminal sequence numbers; iii) average processing delay (ms): the mean time interval from the start of readBytesUntil() to the completion of sscanf(); iv) delay jitter (ms): computed from the cumulative sum of squared delays, which removes the need for a per-packet buffer; and v) CPU utilization (%): the proportion of each 60-second window spent in the busy loop rather than in the free real-time operating system (FreeRTOS) idle task. Two derived quantities were also computed: goodput and per-frame payload fraction. The former is defined as $G = (\bar{R} \times L)/1024$ (kB/s), where $\bar{R}$ is the mean received packet rate (pps) and L is the payload size (bytes). The latter quantifies the proportion of each frame occupied by useful data, given by $\varphi = L/(L + H)$, where $H = 72$ bytes account for the TCP (20 B), IP (20 B), and 802.11 MAC/logical link control (LLC, 32 B) headers [18], [24], excluding the frame check sequence (FCS, 4 B) and optional TCP timestamp fields.

2.4. Statistical method

Each of the three repetitions per condition delivered one set of measures. We report the means and show error bars ± 1 standard deviation in all figures. To test if connection count and payload size affect the throughput independently or interactively, we ran a two-way fixed-effects ANOVA on the pps values recorded at 30 Hz, the highest frequency at which all nine settings had valid (i.e., not overflowed) data. The significance level was $\alpha = 0.05$. The ANOVA partitions total variance into components for the main effect of connection count, the main effect of payload size, their interaction, and the residual error. This method enables us to assess the main influence of each factor and to examine whether the effect of one factor depends on the level of the other.

3. RESULTS AND DISCUSSION

3.1. Effect of connection count (payload=10 B)

Figure 2 plots packet loss against send frequency for three connection counts, all with a 10-byte payload. The 3- and 5-connection configurations experienced no packet loss, with aggregated throughput peaking at approximately 1,480 and 2,410 pps, respectively. Importantly, these throughput plateaus were caused by the Python client's GIL limit rather than gateway CPU exhaustion; only at 10 connections did the gateway CPU become the bottleneck. Even at a per-terminal frequency of 30 Hz (3,237 pps average), there was zero loss. When the frequency was increased to 32 Hz, there was an immediate jump in loss, growing to above 20%.

For small payloads, the zero-loss ceiling is governed by total packet rate rather than connection count. With 3 or 5 connections, the gateway still had headroom while the Python client saturated due to the GIL, so those measured values are lower bounds on hardware capacity. A GIL-free client (e.g., C-based or multiple machines) is required to obtain the true ceiling.

3.2. Effect of payload size (connections=10)

Figure 3 also uses a connection count of 10 and varies the payload size. The difference with subsection 3.1 is striking. At 10 bytes, the gateway maintains 3,237 pps with zero loss (30 Hz). At 64 bytes, zero-loss holds only at 10 Hz (1,088 pps); by 20 Hz, packets are already lost. At 256 bytes, loss occurred at all tested

frequencies; average pps drops to 200–600, and CPU utilization exceeded 100%. Multiplying the payload sizes by about 25 (from 10 to 256 bytes) lowered the zero-loss ceiling by a factor of three or more.

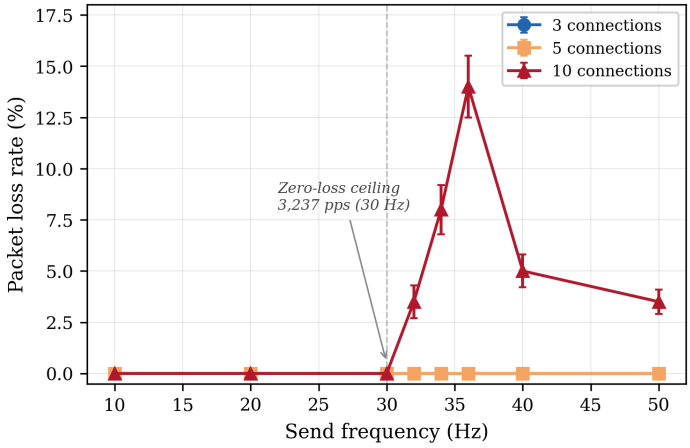


Figure 2. Effect of connection count on packet loss (payload=10 B)

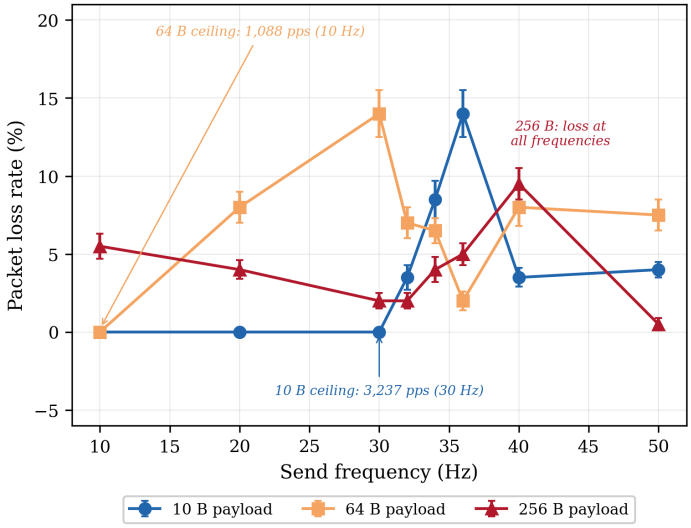


Figure 3. Effect of payload size on packet loss (connections=10)

3.3. Interaction effects and the 3×3 matrix

Table 2 lists the zero-loss ceiling for all nine configurations. The two-way ANOVA at 30 Hz (Table 3) shows significant effects for all three terms.

Table 2. Zero-loss throughput ceiling (pps) for all configurations

Connections	10 B	64 B	256 B
3	≥1,483*	≥1,289*	≥802*
5	≥2,411*	≥1,600*	≥1,001*
10	3,237	1,088	≤946

*Client-side bottleneck; ESP32 has remaining capacity.

The interaction F-value ($F(4, 17) = 77.66$, $p < 0.001$) is significant. The partial eta-squared values ($\eta_p^2 > 0.76$ for all sources) confirm that the observed ANOVA effects are substantively meaningful both statistically and practically. Statistically, with a 10-byte payload, increasing the connection count only raised

the total packet rate (pps), indicating that the client was the limiting factor. With a 256-byte payload, the 3- and 5-connection configurations achieved roughly 800–1,000 pps, while the 10-connection throughput dropped sharply. In other words, connection count has a negligible impact on gateway throughput for small packets, but it significantly degrades performance for large packets. From a practical perspective, engineers should consider multiple factors jointly to predict combined performance, because the interaction between factors can significantly alter the bottleneck. This is because the system bottleneck shifts from MAC-layer frame-rate limitations to channel-bandwidth and buffer-pool capacity constraints.

Table 3. Two-way ANOVA results (dependent variable: pps at 30 Hz)

Source	SS	df	MS	F	p	η_p^2
Connections (A)	1,520,746	2	760,373	27.40	<0.001	0.763
Payload (B)	8,166,606	2	4,083,303	147.13	<0.001	0.945
A × B	8,621,237	4	2,155,309	77.66	<0.001	0.948
Error	471,808	17	27,753			
Total	18,780,396	25				

3.4. Payload fraction analysis

Table 4 breaks down the fixed per-frame overhead. Each frame carries a fixed overhead of 72 bytes (TCP 20 B + IP 20 B + MAC 32 B). However, the larger frames now occupy considerably more channel time. This explains why the pps ceiling decreases as frame size grows. It also challenges a common intuition: a high payload fraction alone does not guarantee high throughput on a resource-constrained gateway. For example, a configuration with a 78% payload fraction saturates the channel at 946 pps, whereas one with only 12.2% sustains 3,237 pps. The latter achieves 3.4× the packet processing rate of the former, delivering significantly higher throughput.

Table 4. Per-frame payload and header fractions by payload size

Payload (B)	Frame total (B)	Payload fraction (%)	Header fraction (%)
10	82	12.2	87.8
64	136	47.1	52.9
256	328	78.0	22.0

3.5. Delay and jitter analysis

Figure 4 presents the average processing delay across all experimental configurations. For the 10-byte payload, the delay remained very low under all connection counts and increased only slightly even at the peak frequency, indicating that the system was far from reaching its computational capacity limit. For the 64-byte payload, the delay of the 10-connection group rose sharply to >1.5 ms once the transmission frequency exceeded 20 Hz, far exceeding the delay observed with the 10-byte payload. Under the 256-byte payload condition, the processing delay reached its maximum at the peak frequency. These observations indicate that the internal buffer pools were exhausted, and system-level network bottlenecks began to emerge.

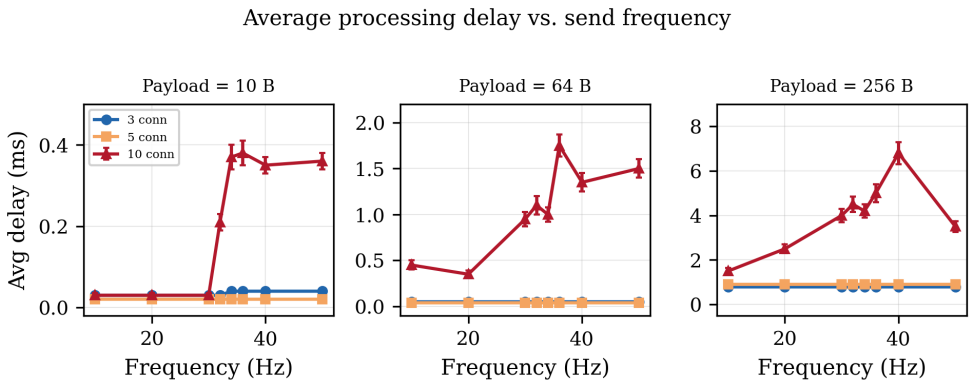


Figure 4. Average processing delay vs. send frequency

3.6. Central processing unit utilization

Figure 5 shows the CPU utilization at a fixed 30 Hz for all nine configurations of the experiments. For the 3-connection case, utilization ranged from 11% to 43%, depending on the payload. Conversely, the 10-connection/256-byte case had a CPU utilization exceeding 100%. This indicates that our application-layer busy-loop counter, which tracks time spent outside the FreeRTOS idle task, fails to capture interrupt-context processing overhead. The cumulative busy time recorded by our counter exceeds the 60-second measurement window, indicating that interrupt-context processing (Wi-Fi ISRs, lwIP timer callbacks) was active during intervals not captured by the application-level timer. While the absolute value above 100% is not physically meaningful, it confirms that no idle cycles remain and the system is fully saturated.

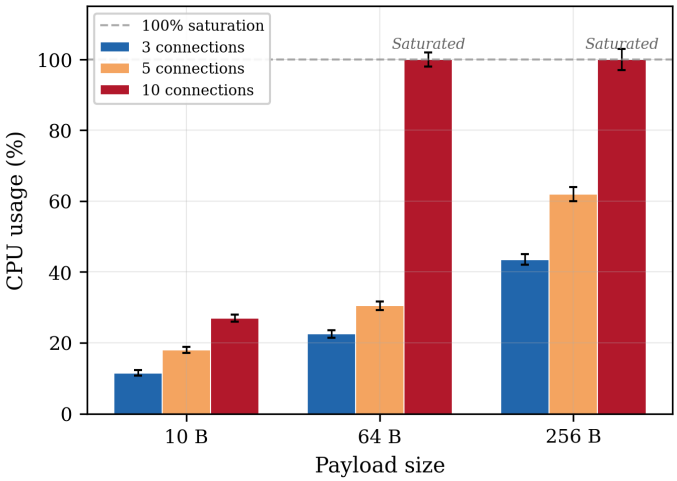


Figure 5. CPU utilization at 30 Hz send frequency

3.7. Bottleneck shift mechanism

Our experiments show two distinct operational regimes, with the transition occurring near a 64-byte payload.

Small-payload regime (10 B). Here, the channel is limited by a fixed per-frame temporal overhead that is independent of the payload. Summing the DIFS (∼50 μs), random backoff (∼67 μs at CW_{min}=15), the PHY preamble (∼40 μs) and the SIFS-plus-ACK exchange (∼50 μs) [17], [18] yields a total non-payload cost of 200–250 μs per transmission. The 10-byte payload takes almost no airtime (∼1 μs). Hence, the frame rate is bound in principle to ∼4,000 frames/s. Our observed ceiling of 3,237 pps agrees with this estimate after accounting for lwIP software delays (interrupt servicing, pbuf allocation, and application-loop jitter). In this regime, neither increasing nor decreasing the TCP connection count changes the bottleneck: the bottleneck is given by the serialized MAC-layer queue.

Large-payload regime (256 B). When the payload is resized to 256 B, the bottleneck shifts from frame rate to channel capacity and memory pressure. In addition to the extra airtime needed for 2,608 bits, each packet needs more space in the lwIP pbuf pool [20]. Specifically, each 256 B pbuf consumes approximately four times the memory of a 10 B pbuf. Given the fixed size of the system buffer pool, larger payloads occupy most available memory, leaving insufficient space for TCP ACK buffers and retransmission queues. Likewise, TCP control-plane complexity (including window management and ACK coalescing) scales with payload size [25]. This 10-connection case is particularly vulnerable here, as 10 streams compete for a finite pbuf pool at the same time that they generate a 10× increase in ACK traffic compared with a single-stream baseline.

Statistical fingerprint. This bottleneck shift, from MAC-cycle limits at small payloads to channel or memory saturation at large payloads, is corroborated by the highly significant ANOVA interaction term ($p < 0.001$), marking the transition from a packet-rate-limited to a throughput-limited system.

3.8. Design guidelines

Our empirical data provide four key design heuristics to maximize ESP32-S3 SoftAP performance [26], [27]: i) for small messages (≤64 B), the throughput ceiling is strictly in terms of the total pps rate rather

than TCP connections. The designer can use up to 10 concurrent sockets without a throughput penalty if the total pps does not approach the saturation point; ii) increasing the payload size from 10 to 256 bytes reduces the zero-loss ceiling by a factor of more than 3. For high-frequency telemetry, sensors need to fragment large payloads into smaller packets or reduce the sampling frequency; iii) operating at 80% of the measured ceiling (∼26 Hz per terminal) is a necessary operating margin against temporary traffic bursts and electromagnetic pollution; iv) processing delay is a more sensitive early-warning indicator than packet loss. Across all setups, the mean latency continues to rise well before the packet drops occur, and a per-packet delay limit of about 0.1 ms should trigger automatic rate limiting. Therefore, a scheme can be designed to be directly integrated into the gateway firmware: once the per-packet latency exceeds a user-configurable threshold, packet reception is throttled.

Figure 6 depicts the safe operating envelope: a multidimensional map of zero-loss operating points across connection counts and payload sizes. The shaded area represents the deterministic stability region, with zero loss observed in all experiments.

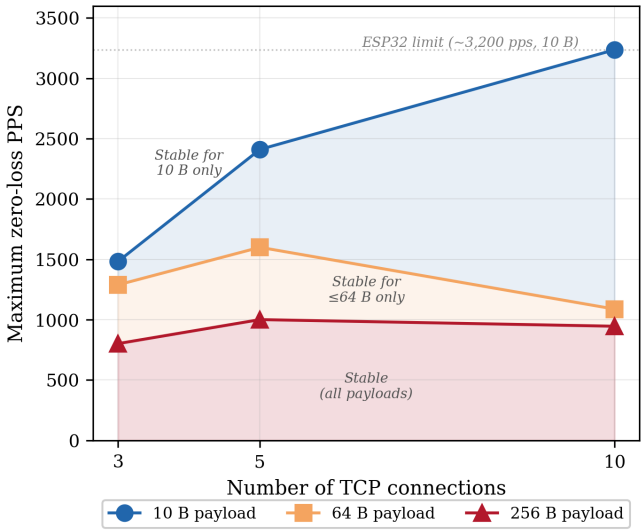


Figure 6. System stability region

3.9. Implications for reconfigurable and embedded systems

The above bottleneck analysis applies not only to the ESP32 platform but also to general embedded gateway systems. When incoming packet rates exceed the software processing capacity of general-purpose SoCs, offloading TCP/IP processing to reconfigurable FPGA hardware becomes an effective optimization solution [28]. This approach has been validated across the IoT domain: FPGA-based accelerators have demonstrated substantial throughput gains in high-density sensor networks [29]. The MAC-layer interrupt overhead observed in small-payload scenarios could potentially be eliminated by deploying hardware-based packet parsing pipelines [26]. This hardware implementation bypasses the entire lwIP software stack and would bypass relevant software constraints. Similarly, a dedicated VLSI network processor with independent packet buffer memory could alleviate the pbuf pool contention caused by large payloads [30]. The stability region chart in Figure 6 provides a quantitative baseline for evaluating hardware acceleration performance. Any FPGA or VLSI acceleration design may improve the zero-loss throughput ceiling beyond the 3,237 pps limit of the pure software gateway implemented in this study. Furthermore, embedding real-time throughput monitoring into IoT gateways supports dynamic system adaptation. The delay-based early-warning indicator proposed in subsection 3.8 could enable reconfigurable systems to adjust hardware resource allocation dynamically according to real-time traffic characteristics. This mechanism could further enhance the operational stability and resource utilization of high-density IoT deployments.

3.10. Limitations

Our findings are subject to some constraints. First, the experiments are performed under a baseline RF situation, a single-occupant indoor environment on a single Wi-Fi channel. Actual functioning in dense

electromagnetic environments would reach a much lower saturation rate owing to the additional contention and interference. Second, we only analyzed TCP. UDP has no ACK generation or flow-control overhead at the transport layer, so its performance envelope is likely to differ. Third, the zero-loss ceilings reported for 3-connection and 5-connection groups at 10 B and 64 B payloads are client-side GIL artifacts; the true ESP32-S3 hardware ceiling for these configurations lies above the measured values.

Consequently, the claim in subsection 3.7 that connection count does not shift the bottleneck in the small-payload regime should be interpreted as: within the tested range where client-side saturation was avoided (10 connections), the MAC-layer frame-rate limit dominates independently of socket count. The small-payload behavior at intermediate connection counts remains to be verified with a non-GIL-constrained client. Fourth, we had disabled PSRAM; increasing external RAM would likely reduce buffer-pool pressure for high-throughput large payloads. Fifth, as this work used the ESP32-S3, our results may not be immediately transposable to other family members (e.g., wireless module (WROOM-32E)) due to variations in radio-frequency front-end and memory architectures [14].

4. CONCLUSION

This study identified a bottleneck-shift phenomenon in an ESP32-S3 SoftAP gateway using a fully crossed two-factor experiment (connection count and payload size, 246 trials in total). Two-way ANOVA confirmed that both factors, and crucially their interaction, are statistically significant ($p < 0.001$), implying that system stability is not additive: a design validated under one payload size cannot be assumed to work under another. Specifically, while small payloads (10–64 B) keep the gateway limited by per-frame MAC-layer overhead, large payloads (256 B) shift the bottleneck to channel bandwidth and lwIP buffer-pool contention, reducing the zero-loss throughput ceiling from thousands of packets per second to below 946 pps. These findings lead to three practical guidelines for embedded gateway design: i) when higher transmission frequencies are required, the packet length should be reduced appropriately; ii) the gateway should be designed with sufficient margin to absorb traffic bursts; and iii) per-packet delay should be considered as an early-warning indicator, and the transmission frequency should be throttled at appropriate times. Beyond the ESP32 platform, the identified bottlenecks (MAC-layer per-frame overhead and pbuf-pool contention) are suitable targets for reconfigurable hardware acceleration. FPGA/VLSI offloading may further raise the zero-loss ceiling, and the reported results can serve as a baseline for evaluating such designs. Future work includes extending the study to UDP, enabling PSRAM, and comparing other ESP32 variants to verify the generality of the observed behavior.

FUNDING INFORMATION

This work was supported in part by the Hangzhou Joint Fund of Zhejiang Provincial Natural Science Foundation of China (No. LHZQN26H180003), the Scientific Research Fund of Zhejiang Provincial Education Department (No. Y202559018), the Zhejiang Province New Talent Program for College Students (No. 2025R463A002, 2025R463A004), and the Key Project of Hangzhou Polytechnic (No. HKYZXZD-2025-3).

AUTHOR CONTRIBUTIONS STATEMENT

This journal uses the Contributor Roles Taxonomy (CRediT) to recognize individual author contributions, reduce authorship disputes, and facilitate collaboration.

Name of Author	C	M	So	Va	Fo	I	R	D	O	E	Vi	Su	P	Fu
Jianan Zhao	✓	✓	✓	✓	✓	✓		✓	✓	✓			✓	
Yuanbin Mu		✓				✓		✓	✓	✓	✓	✓		

C : **C**onceptualization
M : **M**ethodology
So : **So**ftware
Va : **Va**lidation
Fo : **Fo**rmal Analysis
I : **I**nvestigation
R : **R**esources
D : **D**ata Curation
O : Writing - **O**riginal Draft
E : Writing - Review & **E**diting
Vi : **Vi**sualization
Su : **Su**pervision
P : **P**roject Administration
Fu : **Fu**nding Acquisition

CONFLICT OF INTEREST STATEMENT

Authors state no conflict of interest.

DATA AVAILABILITY

The data that support the findings of this study are available from the corresponding author upon reasonable request.

REFERENCES

[1] N. Naik, "Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP," in *2017 IEEE International Systems Engineering Symposium (ISSE)*, Vienna, Austria, 2017, pp. 1–7, doi: 10.1109/SysEng.2017.8088251.

[2] A. Maier, A. Sharp, and Y. Vagapov, "Comparative analysis and practical implementation of the ESP32 microcontroller module for the Internet of Things," in *2017 Internet Technologies and Applications (ITA)*, Wrexham, UK, 2017, pp. 143–148, doi: 10.1109/ITECHA.2017.8101926.

[3] Espressif Systems, "ESP-IDF Programming Guide," 2023. [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/latest/.(Accesed: Dec 1. 2025).

[4] R. Ratasuk, B. Vejlgaard, N. Mangalvedhe, and A. Ghosh, "NB-IoT system for M2M communication," in *2016 IEEE Wireless Communications and Networking Conference*, Doha, Qatar, 2016, pp. 1–5, doi: 10.1109/WCNC.2016.7564708.

[5] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, Oct. 2010, doi: 10.1016/j.comnet.2010.05.010.

[6] D. C. Y. Vargas and C. E. P. Salvador, "Smart IoT Gateway For Heterogeneous Devices Interoperability," in *IEEE Latin America Transactions*, vol. 14, no. 8, pp. 3900-3906, Aug. 2016, doi: 10.1109/TLA.2016.7786378.

[7] A. Hasibuan *et al*., "Design of flood warning prototype using ESP32 module-based ultrasonic sensors," *International Journal of Reconfigurable and Embedded Systems (IJRES)*, vol. 14, no. 1, pp. 126–135, Mar. 2025, doi: 10.11591/ijres.v14.i1.pp126-135.

[8] P. Ranjani and N. Harini, "IoT Based Cloud-Integrated Smart Parking System With Reservation," in *2024 International Conference on System, Computation, Automation and Networking (ICSCAN)*, Puducherry, India, 2024, pp. 935–938, doi: 10.1109/ICSCAN62807.2024.10894637.

[9] Y. Tselishchev, A. Boulis and L. Libman, "Experiences and Lessons from Implementing a Wireless Sensor Network MAC Protocol in the Castalia Simulator," *2010 IEEE Wireless Communication and Networking Conference, Sydney*, NSW, Australia, 2010, pp. 1-6, doi: 10.1109/WCNC.2010.5506096.

[10] C. Bormann, M. Ersue, and A. Keranen, "Terminology for constrained-node networks," *Internet Engineering Task Force (IETF) Request for Comments: 7228*, May 2014, doi: 10.17487/RFC7228.

[11] S. Li, L. D. Xu, and S. Zhao, "5G Internet of Things: A survey," *Journal of Industrial Information Integration*, vol. 10, pp. 1–9, Jun. 2018, doi: 10.1016/j.jii.2018.01.005.

[12] I. Plauska, A. Liutkevičius, and A. Janavičiūtė, "Performance evaluation of C/C++, MicroPython, Rust and TinyGo programming languages on ESP32 microcontroller," *Electronics*, vol. 12, no. 1, p. 143, 2023, doi: 10.3390/electronics12010143.

[13] S. Kumar, N. K. Tomar, L. Gupta, Manav, V. Tripathi, and P. Kalvakota, "Design and Implementation of an IoT-Based Industrial Process Control Simulator using ESP32, MQTT and Android," in *2025 2nd International Conference on Recent Trends in Electrical, Electronics and Computing Technologies (ICRTEECT)*, Warangal, India, 2025, pp. 1–6, doi: 10.1109/ICRTEECT67512.2025.11448784.

[14] F. Serepas, I. Papias, K. Christakis, N. Dimitropoulos, and V. Marinakis, "Lightweight embedded IoT gateway for smart homes based on an ESP32 microcontroller," *Computers*, vol. 14, no. 9, p. 391, 2025, doi: 10.3390/computers14090391.

[15] M. J. Espinosa-Gavira, A. Agüera-Pérez, J. C. Palomares-Salas, J. M. Sierra-Fernandez, P. Remigio-Carmona, and J. J. G. de-la-Rosa, "Characterization and performance evaluation of ESP32 for real-time synchronized sensor networks," *Procedia Computer Science*, vol. 237, pp. 261–268, 2024, doi: 10.1016/j.procs.2024.05.104.

[16] Espressif Systems, "ESP32-S3 Series Datasheet," v1.2, 2023. [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32-s3_datasheet_en.pdf.

[17] G. Bianchi, "Performance analysis of the IEEE 802.11 distributed coordination function," in *IEEE Journal on Selected Areas in Communications*, vol. 18, no. 3, pp. 535-547, Mar. 2000, doi: 10.1109/49.840210.

[18] "IEEE Standard for Information Technology–Telecommunications and Information Exchange between Systems - Local and Metropolitan Area Networks–Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," in *IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016)*, pp.1-4379, Feb. 2021, doi: 10.1109/IEEESTD.2021.9363693.

[19] F. Montori, L. Bedogni and L. Bononi, "A Collaborative Internet of Things Architecture for Smart Cities and Environmental Monitoring," in *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 592-605, Apr. 2018, doi: 10.1109/JIOT.2017.2720855.

[20] A. Dunkels, "Design and implementation of the lwIP TCP/IP stack," *Swedish Institute of Computer Science*, vol. 2, no. 277, 2001.

[21] Espressif Systems, "ESP-IDF lwIP documentation," 2023. [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/latest/esp32s3/api-guides/lwip.html.

[22] P. K. Gkonis, P. T. Trakadas, and D. I. Kaklamani, "A comprehensive study on simulation techniques for 5G networks," *Electronics*, vol. 9, no. 3, p. 468, 2020, doi: 10.3390/electronics9030468.

[23] C. Perera, A. Zaslavsky, P. Christen and D. Georgakopoulos, "Context Aware Computing for The Internet of Things: A Survey," in *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 414-454, 2014, doi: 10.1109/SURV.2013.042313.00197.

[24] P. Gupta and P. R. Kumar, "The capacity of wireless networks," *IEEE Transactions on Information Theory*, vol. 46, no. 2, pp. 388-404, Mar. 2000, doi: 10.1109/18.825799.

[25] S. Ha, I. Rhee, and L. Xu, "CUBIC: A new TCP-friendly high-speed TCP variant," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 5, pp. 64–74, Jul. 2008, doi: 10.1145/1400097.1400105.
[26] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge Computing: Vision and Challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637-646, Oct. 2016, doi: 10.1109/JIOT.2016.2579198.
[27] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2347-2376, 2015, doi: 10.1109/COMST.2015.2444095.
[28] D. Sidler, G. Alonso, M. Blott, K. Karras, K. Vissers, and R. Carley, "Scalable 10Gbps TCP/IP Stack Architecture for Reconfigurable Hardware," in *2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines*, Vancouver, BC, Canada, 2015, pp. 36-43, doi: 10.1109/FCCM.2015.12.
[29] A. Magyari and Y. Chen, "Review of state-of-the-art FPGA applications in IoT networks," *Sensors*, vol. 22, no. 19, p. 7496, 2022, doi: 10.3390/s22197496.
[30] D. Yang, X. Xu, T. Chen, Y. Chen, and J. Zhang, "Low latency TOE with double-queue structure for 10 Gbps Ethernet on FPGA," *Sensors*, vol. 23, no. 10, p. 4690, 2023, doi: 10.3390/s23104690.

BIOGRAPHIES OF AUTHORS



Jianan Zhao received the B.Eng. degree in Electronic Information Engineering from Chongqing University of Posts and Telecommunications, China, in 2018, and the M.Eng. degree in Electronic Information from Zhejiang University of Technology, China, in 2023. His research interests include embedded systems, IoT, and wireless sensor networks. He can be contacted at email: zjn@hzpt.edu.cn.



Yuanbin Mu received the Ph.D. degree in Electronic Science and Technology from Hangzhou Dianzi University, China, in 2025. He is currently an associate professor at Hangzhou Polytechnic, China. His research interests include embedded systems, IoT, wireless sensor networks, flexible, and wearable electronics. He can be contacted at email: muyuanbin@qq.com.