Evaluation of Resource-Aware HTTP/3 Proxies for Smuggling Resilience in IoT Environments

Lorenzo Pisu*, Giovanni Pettorru*†, Leonardo Regano*, Davide Maiorca*, Giorgio Giacinto*‡, Marco Martalò*†
*Department of Electrical and Electronic Engineering, University of Cagliari, Italy
†Consorzio Nazionale Interuniversitario per le Telecomunicazioni (CNIT), Italy
‡Consorzio Interuniversitario Nazionale per l'Informatica (CINI), Italy
E-mail: {name.surname}@unica.it

***Abstract*—The growing integration of IoT devices into Edge and Fog infrastructures, alongside the increasing adoption of low-latency QUIC-based protocols like HTTP/3, has intensified the need for lightweight, resource-efficient security mechanisms to counter emerging threats such as request smuggling. Within this context, proxy-based architectures offer an optimal trade-off to strengthen network security while accommodating the limited computational capacity of IoT devices. In this direction, this paper presents a comprehensive experimental evaluation of the impact of different proxies for HTTP/3 services on resource usage when deployed on platforms such as the Raspberry Pi (RPi), considering diverse traffic patterns, operational conditions, and device configurations. The results highlight that proxies can achieve a promising balance between security and resource overhead, confirming their viability for integration into distributed IoT-based Edge and Fog networks.**

***Index Terms*—Internet of Things (IoT), Edge Computing, HTTP/3, Request Smuggling, Resource Consumption, Security**

I. Introduction

The Internet of Things (IoT)'s rapid growth and integration into Edge and Fog computing paradigms have highlighted the demand for lightweight, distributed platforms capable of delivering efficient and secure data processing [1]. In modern IoT environments, such as Smart Cities, eHealth, precision agriculture, and Industrial IoT, the deployment of multiple nodes in close proximity to data sources is essential to minimize latency, improve responsiveness, and reduce reliance on centralized infrastructures [2].

Single-board computers represent the most commonly adopted solution for this task, and Raspberry Pi (RPi) has emerged as a preferred option due to its well-balanced combination of computational power, energy efficiency, and cost-effectiveness [3]. As these devices are increasingly entrusted with more demanding tasks, they face growing challenges related to computational efficiency and exposure to evolving security threats. Among these, the widespread adoption of modern, low-latency communication protocols (particularly QUIC-based solutions such as HTTP/3), further complicates the scenario by introducing new vulnerabilities, such as protocol-level attacks like request smuggling [4]. In this context, a thorough evaluation of IoT-compliant platforms under realistic operating conditions becomes essential, with a specific focus on the effectiveness and overhead of lightweight security mechanisms [5]–[7].

From a security perspective, Edge and Fog computing networks present distinct challenges due to their distributed architecture and frequent deployment in untrusted or physically exposed environments. These characteristics significantly expand the attack surface and heighten the need for lightweight yet effective mechanisms to enforce access control, ensure data integrity, and secure communications [8], [9]. In this context, proxy-based architectures are frequently adopted to centralize and offload security-related functions, such as Transport Layer Security (TLS) termination, request filtering, and traffic routing, thereby enhancing scalability and mitigating common security threats [10].

While proxies can enhance protection and isolate backend services, their integration into low-cost Edge platforms requires careful consideration of both security and resource usage. Although many studies in the literature emphasize security aspects, few address the impact on *resource usage* (and, consequently, energy consumption), which is a critical concern in IoT-compliant environments. In this context, the authors of [11] propose a latency-aware, energy-efficient data management layer that reduces communication overhead and energy inefficiencies in an Industrial IoT use case by caching data in proxy nodes. Similarly, the work in [12] presents a forward CoAP Proxy server for embedded systems, aimed at improving performance by reducing response times and packet exchanges across multiple hops, with a focus on energy and resource consumption inferred from these performance improvements rather than directly evaluated. Similarly, the evaluation in [13] also focuses on proxy performance improvements, with aspects related to energy efficiency and resource consumption inferred from performance results rather than explicitly measured.

Although existing works acknowledge the importance of integrating security with minimal resource and energy consumption, few directly evaluate these aspects through experiments in real-world settings, especially for devices where such factors are critical, like IoT systems. These considerations motivate the contributions of this work, which are summarized as follows.

- Experimental analysis of proxy-induced resource usage under normal (regular) traffic and HTTP/3 request smuggling, highlighting the potential of proxies to balance

security and efficiency in resource-constrained environments.
- Evaluation of IoT-compliant platforms for secure proxy deployment in Edge/Fog environments, optimizing cost, security, and performance.
- Insights into the selection of proxies, hardware platforms, and Operating System (OS) for real-world IoT applications, grounded in experimental results.

The remainder of the paper is organized as follows. Section II provides an overview of the background relevant to this work. Section III describes the experimental setup, while Section IV presents and discusses the numerical results. Finally, Section V outlines the concluding remarks.

II. Background

A. The Role of QUIC in IoT Communications

The IoT is rapidly transforming how devices communicate and operate within modern networks. As these systems evolve, ensuring low-latency and secure communication becomes critical, especially in scenarios such as smart homes, e-health, environmental monitoring, and industrial automation, where delayed or compromised data can lead to reduced service quality, security breaches, or even safety risks. In this context, transport protocols must strike a balance between performance and security [14].

One of the most prominent protocols for enabling secure and low-latency Internet communications is QUIC, which is the foundation of HTTP/3 and is quickly becoming a key component of modern web transport. QUIC combines the lightweight efficiency of the User Datagram Protocol (UDP) with the robust security guarantees of Transport Layer Security (TLS), offering encrypted and authenticated communication by default [15]. Different characteristics make QUIC, and therefore HTTP/3, a highly suitable choice for scenarios where both speed and security are essential. Notable features include multiplexed streams to avoid head-of-line (HOL) blocking, 0-RTT connection resumption to reduce handshake delays, and built-in encryption that ensures end-to-end data protection with minimal overhead.

In this direction, recent research has increasingly explored the adoption of QUIC within IoT environments, recognizing its potential to meet the stringent requirements of secure and low-latency communication. In [16], [17], the authors compare MQTT over QUIC with traditional TCP/TLS, showing promising results under varying network conditions. Experimental validation of MQTT over QUIC is provided even in [18], where delay and energy consumption are evaluated for IoT applications. Other studies explore QUIC with different protocols, such as the comparison of a QUIC-based HTTP stack with CoAP and MQTT in [19], which shows competitive performance in lossy environments. Similarly, [20] highlights the benefits of integrating CoAP and QUIC. More recently, [21] introduces WebSocket over QUIC, leveraging QUIC's performance and security advantages for efficient IoT gateway communication.

Although QUIC brings significant improvements in both performance and security compared to traditional protocols, it is not immune to vulnerabilities. Despite being relatively young and designed with security in mind from the outset, several studies have already highlighted potential attack vectors that can compromise its intended guarantees, one of which is described in the next section, as it represents the focus of the work.

B. HTTP/3 Request Smuggling

Request smuggling is a vulnerability that poses a serious threat to web applications, allowing malicious actors to exploit inconsistencies in how HTTP requests are parsed by proxies and frameworks. This enables attackers to bypass security controls and carry out malicious activities such as data exfiltration and access control violations.

Request smuggling can arise from non-compliance with the HTTP/3 standard (RFC 9114), which defines how intermediaries should process incoming malicious requests to prevent security issues. From an attacker's perspective, request smuggling involves injecting malicious data into the structure of an HTTP request to trigger a parsing discrepancy between the proxy and the backend server.

Whenever a malicious request of this type is parsed by a proxy, the proxy should reject it and terminate the connection. However, as demonstrated by previous research [4], proxies often diverge from the RFC standard and react differently depending on the specific characteristics of the malicious request. From a device resource consumption perspective, this results in varying impacts on hardware performance depending on how the proxy handles the malformed request.

C. HTTP/3 to HTTP/1.1 Conversion in Proxies

Since HTTP/3 is becoming ever more widespread, but HTTP/1.1 is still widely used in production, proxies have started to support the conversion of HTTP/3 requests to HTTP/1.1. This study focuses on four popular and open source proxies, namely Nginx, Traefik, Caddy, and HaProxy. Table I reports the version, popularity, and security characteristics of the proxies.

Proxy	Version	Popularity		RFC Adherence
		★	⑂	
Caddy	2.9.1	63.5k	4.3k	Strong
Traefik	3.3.5	54.2k	5.3k	Low
Nginx	1.27.4	26.6k	7.2k	Medium
Haproxy	2.7	5.5k	835	Low

★=GitHub stars, ⑂=GitHub forks

TABLE I: Version, popularity, and degree of RFC adherence of each selected proxy. Strong (>70% RFC-compliant rejections), medium (30–70%), and low (<30%).

RFC 9114 Adherence RFC 9114 defines the standard for HTTP/3 implementations and provides explicit recommendations for handling malformed or malicious requests. According to the specification, such requests should be rejected, and the corresponding connection should be closed.

However, the degree of adherence to the RFC varies among different proxy implementations. Proxies that demonstrate strong adherence typically reject malformed requests and terminate the associated connection immediately. In contrast, proxies with medium adherence generally reject malformed requests but may, in certain cases, perform sanitization or allow the connection to timeout. Low adherence is characterized by frequent misvalidation of malformed requests, coupled with inconsistent sanitization or reliance on connection timeouts rather than active rejection.

Below is a detailed overview of the observed behaviors for each proxy, based on the results reported in a prior study [4]:

- **Caddy**. Caddy, which is one of the most popular proxies based on GitHub stars, exhibits strong adherence to RFC 9114. Only two cases were identified where requests containing invalid characters (tabs and whitespaces) were not properly validated. Additionally, three cases resulted in connection timeouts, while 32 malformed requests were sanitized and forwarded instead of being rejected.
- **Traefik**. Traefik demonstrates low adherence to the RFC. Similar to Caddy, it fails to validate requests containing tabs and whitespaces. However, it is more prone to leaving connections idle, with 195 requests resulting in timeouts and only two cases being sanitized.
- **Nginx**. Nginx also falls into the medium adherence category. Unlike Traefik, Nginx frequently applies sanitization to malformed requests (131 cases were observed) and does not leave connections hanging.
- **HAProxy**. HAProxy (version 2.7) shows the lowest adherence to RFC 9114, partly due to a known vulnerability (CVE) associated with HTTP request smuggling. Specifically, five cases were observed where requests containing malicious characters were not properly validated. Furthermore, HAProxy left the connection hanging in 195 cases and sanitized only three malformed requests.

III. Experimental Setup

A. Hardware

To experimentally evaluate the performance of HTTP/3 proxies, two versions of IoT-compliant hardware platforms are used. Specifically, we employed a RPi 4 Model B with a Broadcom BCM2711 quad-core processor and 8 GB of RAM, and a RPi 5 with a Broadcom BCM2712 quad-core processor and 8 GB of RAM. For each device, two OSs are tested to ensure a comprehensive evaluation, resulting in a total of four configurations. In particular, RPi OS (Debian GNU/Linux 12, bookworm) and Ubuntu Desktop 24.04.2 LTS are used, both optimized for ARM-based hardware and suitable for IoT experimentation. The client component consists of lightweight scripts that generate requests to the proxy and are executed on a workstation.[1] For networking, a DWR-932 4G LTE Mobile Wi-Fi Hotspot is used to establish a local private wireless network, isolated from the rest of Internet, that enables communication between the client and the proxy.

[1]Since the evaluation focuses on the behaviour and resource usage of the proxy running on the RPi devices, the client hardware specifications are omitted.

Device	Technical Specifications
RPi 4 Model B	Broadcom BCM2711 quad-core, 8 GB RAM
RPi 5	Broadcom BCM2712 quad-core, 8 GB RAM
DWR-932 4G LTE Mobile Hotspot	802.11 a/b/g/n, 2.4 GHz 150 Mbps down / 50 Mbps up

OS	Release Date
RPi OS (Debian 12, bookworm)	June 10, 2023
Ubuntu Desktop 24.04.2 LTS	February 20, 2025

TABLE II: Devices used in the experiments, with technical specifications and adopted OSs.

This setup provides a realistic and reproducible testbed for evaluating the feasibility of deploying QUIC proxies designed to be secure and resilient to attacks, even when running on resource-constrained IoT platforms. Table II summarizes the devices used in the experiments along with their technical specifications and the corresponding OSs.

B. Software

On the client side, two separate custom scripts are used to simulate real-world traffic patterns: one generates regular HTTP/3 requests over a defined time interval, while the other carries out HTTP/3 smuggling attacks. Both the regular and smuggling scripts issued a total of 900 requests, with a fixed time interval of 0.55 seconds between each, resulting in a total runtime of 8 minutes and 15 seconds.

The regular traffic consisted of an even split: 450 `GET` requests and 450 `POST` requests, with each `POST` request carrying a simple 4-byte payload containing the string `test`. This setup was designed to simulate a realistic traffic scenario in which both request types are commonly observed. In contrast, the malicious script was crafted to perform request smuggling attack attempts. It sent requests containing forbidden bytes in both header names and values, as well as requests with forbidden or conflicting headers, such as the presence of a `transfer-encoding` header or a `content-length` field that did not match the actual body size. Additionally, the script generated requests missing mandatory pseudo-headers (e.g., `:method`, `:scheme`, `:path`, and `:authority`). The primary goal of this malicious script was to assess how the validation mechanisms of different proxies handle invalid requests and how these behaviors affect resource consumption.

On the proxy side, each RPi runs a single proxy instance encapsulated within a Docker container. This approach ensures environmental consistency and facilitates reproducibility in different test scenarios. Moreover, each proxy is paired with a simple Python backend implemented using `Flask`, representing a realistic scenario in which the proxy is responsible for routing the incoming traffic to the appropriate endpoints and balancing the load of the web application. The proxy and backend are deployed in separate Docker containers, with the proxy configured to translate incoming HTTP/3 requests into HTTP/1.1 requests for the backend. As a result, only requests that successfully pass through the proxy reach the

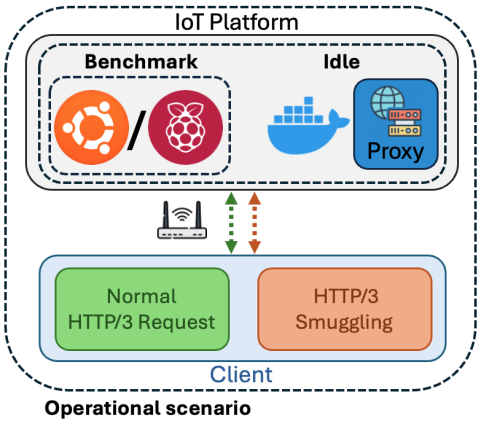

Fig. 1: Overview of the experimental scenarios with different proxy activity states.

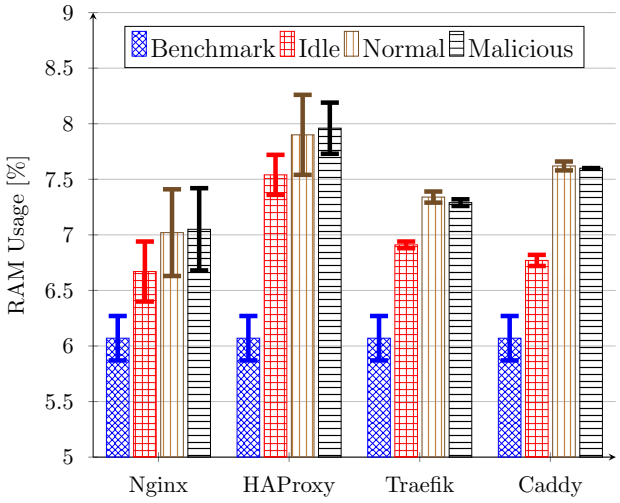

Fig. 2: RAM % usage under different conditions and proxies.

backend and trigger a corresponding response. Conversely, if the proxy filters out a request, the backend remains inactive.

Alongside the proxy, a dedicated monitoring script continuously collects system performance metrics to evaluate the overhead introduced by the proxy under various operational conditions. To this end, we employed Psutil [22], a cross-platform Python library that offers fine-grained access to system-level statistics. Specifically, we monitored CPU temperature and utilization, memory usage, and the total number of active processes.

C. Scenario Configuration

To evaluate the resource consumption associated with deploying a proxy on IoT-compliant devices, a set of controlled test scenarios has been considered. These are intended to isolate and quantify the system overhead introduced by the proxy under various conditions, ranging from idle operation to active handling of regular and malicious traffic. A visual summary of all these scenarios is provided in Fig. 1.

The *benchmark scenario* serves as a reference point. In this configuration, the device intended to host the proxy is connected to the private network and left in its baseline operational state, running only essential system processes. A remote SSH session, initiated from a separate supervisory device, is established and consistently maintained across all scenarios to control and monitor the device under test. This initial setup enables us to determine the platform's idle resource consumption and assess the incremental impact of the additional workloads introduced in the other scenarios.

In *idle scenario*, the proxy software is deployed and is actively running on the device, but does not receive any client requests. This setup enables us to isolate the overhead due solely to the proxy's background operations, such as listening for incoming connections and maintaining the internal state, without the influence of external traffic.

The *operational scenario* is divided into two distinct phases to emulate realistic proxy usage. In the normal phase, a single client interacts with the proxy by issuing only legitimate HTTP/3 requests over a fixed time interval, simulating standard user traffic. In the malicious phase, the client switches to sending malformed HTTP/3 requests crafted to exploit smuggling vulnerabilities, as outlined in Section III-B. This separation allows for a more granular evaluation of the proxy's behaviour under varying traffic conditions. By comparing these phases with the idle scenario, we assess the runtime impact of handling both regular and malicious traffic on resource-constrained platforms.

IV. Experimental Results

A. Proxy Performance Metrics

To establish the security baseline of each proxy, the same tests provided by [4] have been performed, confirming the same results in terms of timeouts, rejections, sanitizations, and acceptances of the malicious requests. Building on this, the trade-off between security and performance is assessed by evaluating the computational overhead of proxy-based mechanisms on resource-constrained IoT platforms, starting the analysis with a representative configuration: the RPi 4 running RPi OS. This setup serves as a reference to ascertain the behaviour of all proxies across the three operational scenarios. It was chosen for its ability to encapsulate typical behaviours observed in all test cases, a generalization supported by the aggregated summary in Table III.

As shown in Fig. 2, all proxies exhibit similar RAM usage across idle, normal, and malicious scenarios. The average RAM consumption for each proxy remains stable, typically between 7% and 8%, regardless of the operational state. The key differences arise in the standard deviation: Nginx and HAProxy show more variability in RAM usage, while Traefik and Caddy maintain a more consistent and stable profile.

A different picture arises when analyzing CPU usage, as illustrated in Fig. 3. While the general trends across proxies and scenarios are still aligned in terms of both average CPU usage and variability, the normal scenario leads to higher CPU consumption compared to both idle and malicious cases.

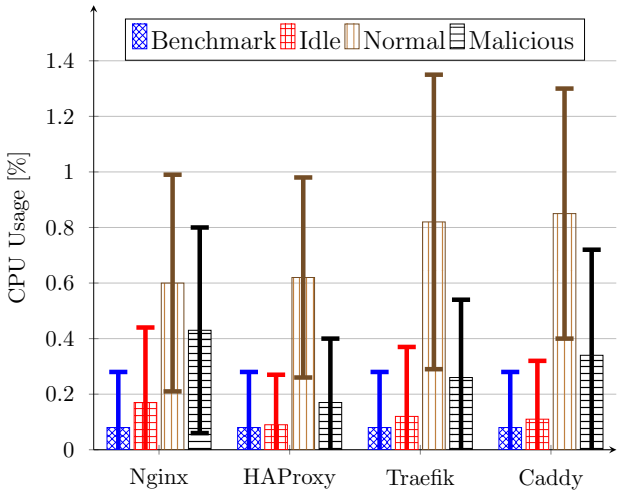

Fig. 3: CPU % usage under different conditions and proxies.

This behaviour is observed in most cases, except for a few anomalies primarily seen in RPi5 and Raspbian, where occasional peaks lead to higher average values. These peaks, though limited in magnitude and not indicative of any issues with the IoT platform, are accompanied by a scenario in which the malicious traffic leads to worse performance than the normal one.

In general, the higher CPU usage in the normal scenario can be attributed to the correct functioning of the proxy [6]. When the proxy successfully mitigates malicious traffic, it prevents communication from reaching the backend, saving CPU resources. This highlights both the security benefits of using secure proxies and their positive impact on resource utilization, optimizing performance in real-world traffic scenarios. However, if the malicious scenario leads to worse performance than the normal one, it indicates the proxy is not functioning correctly and is failing to handle malicious traffic properly.

B. Impact of Hardware and OS

In the following phase, the tests shift the focus from proxy performance to assessing the impact of the underlying hardware platforms and OSs. Fig. 4 illustrates how different hardware platforms and OSs affect resource usage, using HAProxy as the reference proxy. Although the general trends in resource usage are similar across both RPi platforms, the initial resource usage (*benchmark*) levels differ, with Ubuntu showing higher starting values compared to RPi OS. Despite this, the impact of proxy usage on resource consumption remains minimal in both configurations. These findings underscore the importance of selecting an appropriate OS, while also highlighting the efficiency of both IoT platforms in handling proxy-related tasks.

Table III provides a comprehensive summary of the variations in resource usage, expressed as Δ [%] from the benchmark values, across different proxies and operating conditions within the experimental setup. The results highlight that,

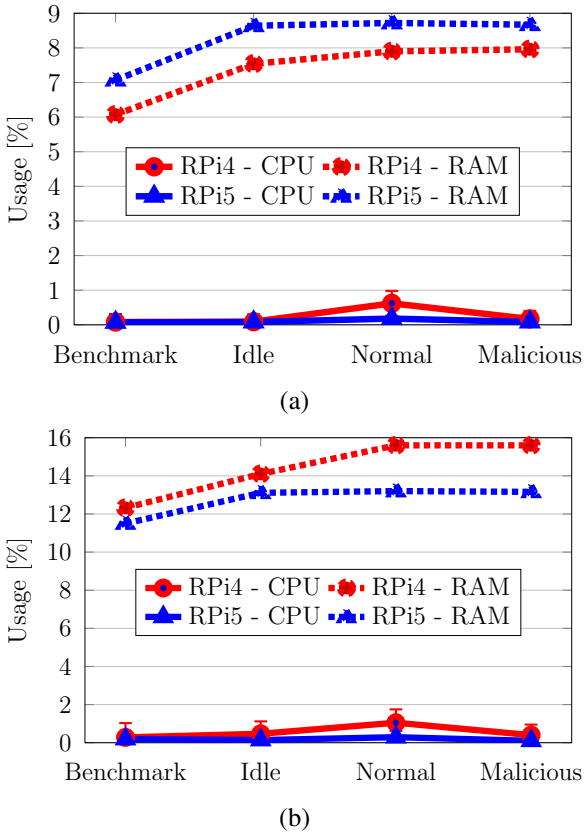

Fig. 4: Impact of hardware platform and OS on percentage resource usage: (a) RPi OS and (b) Ubuntu.

regardless of the hardware platform, OS, or proxy employed, deviations in CPU and RAM consumption remain minimal. As an example, the maximum increase in RAM usage is observed on the RPi 4 running Ubuntu with the Traefik proxy under normal conditions, reaching 4.28%. On the other end, the smallest delta of just 0.6%, occurs on the RPi 4 with RPi OS using Nginx in idle mode. CPU usage shows even more contained variations, remaining below 1% for the vast majority of tests and conditions, with a peak of 2.23% on the RPi 4 with Ubuntu and Nginx under normal conditions.

These results suggest that, in the considered scenarios, the impact of the hardware platform and OS on proxy-related resource consumption is negligible. In general, the minimal deviations observed between benchmark and operational states, whether under normal or malicious traffic, further support the suitability of IoT-compliant deployments across varying conditions. Note that additional metrics, such as CPU temperature, frequency, active processes, and cache usage, were also monitored during testing. However, they are omitted here due to space constraints and their limited relevance, as no significant variations were observed.

V. Conclusions

The increasing demand for efficient and secure data processing in IoT environments, particularly within Edge and Fog computing paradigms, has introduced new vulnerabilities,

			Nginx			HAProxy			Traefik			Caddy		
			Idle	*Normal*	*Malicious*	*Idle*	*Normal*	*Malicious*	*Idle*	*Normal*	*Malicious*	*Idle*	*Normal*	*Malicious*
RPi 4	RPi OS	RAM [%]	0.6	0.95	0.98	1.47	1.83	1.89	0.84	1.27	1.22	0.7	1.55	1.53
		CPU [%]	0.09	0.52	0.35	0.01	0.54	0.09	0.04	0.74	0.18	0.03	0.77	0.26
	Ubuntu	RAM [%]	0.88	4.2	3.93	1.78	3.28	3.28	1.08	4.28	4.26	0.88	2.15	2.18
		CPU [%]	0.15	2.23	1.38	0.19	0.77	0.12	0.20	1.27	0.68	0.22	1.05	0.44
RPi 5	RPi OS	RAM [%]	0.86	0.68	0.73	1.56	1.65	1.60	1.05	1.03	1.03	0.80	0.91	0.91
		CPU [%]	0.11	0.08	0.05	0.00	0.11	0.01	0.02	0.11	0.26	0.53	0.17	0.56
	Ubuntu	RAM [%]	0.73	0.71	0.71	1.61	1.70	1.66	0.93	1.06	0.90	0.64	0.71	0.70
		CPU [%]	0.06	0.00	0.00	-0.05	0.10	-0.09	0.03	0.07	-0.08	0.02	0.24	-0.01

TABLE III: Resource usage Δ [%] from the benchmark for each device, OS, and proxy.

such as request smuggling in Web-based services, further complicating the security landscape. Building on this, this work provides a comprehensive analysis of how proxies running on IoT-compliant devices impact resource usage, exploring their dual role in enhancing both security and efficiency under varying traffic conditions, including malicious HTTP/3 traffic.

The experimental results offer actionable insights for selecting the most suitable proxies, OSs, and platforms in real-world IoT deployments. Furthermore, the study demonstrates the feasibility of employing low-cost, IoT-compliant devices in Edge and Fog layers, supported by an in-depth evaluation of resource allocation across different operational scenarios. These findings provide a solid foundation for the efficient and secure integration of proxy-based architectures in future distributed IoT systems.

In future work, the analysis will be extended by including additional metrics, such as network-related parameters, and by evaluating system scalability to better understand the impact of multiple devices connecting to the same proxy.

ACKNOWLEDGMENT

This work was supported by M.D. 351 and SERICS project (PE00000014), National Recovery and Resilience (NRRP), funded by the European Union, NextGenerationEU. It was also supported by the POSIDONIA project under the third open call of the NGI Sargasso project (GA n. 101092887).

REFERENCES

[1] Q. Xu and J. Zhang, "pifogbed: A fog computing testbed based on raspberry pi," in *IEEE IPCCC*, 2019, pp. 1–8.

[2] P. G. V. Naranjo, Z. Pooranian, M. Shojafar, M. Conti, and R. Buyya, "Focan: A fog-supported smart city network architecture for management of applications in the internet of everything environments," 2017. [Online]. Available: https://arxiv.org/abs/1710.01801

[3] R. Mahmud and A. N. Toosi, "Con-pi: A distributed container-based edge and fog computing framework," *IEEE Internet of Things Journal*, vol. 9, no. 6, p. 4125–4138, March 2022. [Online]. Available: http://dx.doi.org/10.1109/JIOT.2021.3103053

[4] L. Pisu, F. Loi, D. Maiorca, and G. Giacinto, "Http/3 will not save you from request smuggling: A methodology to detect http/3 header (mis) validations," in *IEEE NCA*. IEEE, 2024, pp. 97–104.

[5] A. Alwarafy, K. A. Al-Thelaya, M. Abdallah, J. Schneider, and M. Hamdi, "A survey on security and privacy issues in edge computing-assisted internet of things," 2020. [Online]. Available: https://arxiv.org/abs/2008.03252

[6] M. S. Aslanpour, S. S. Gill, and A. N. Toosi, "Performance evaluation metrics for cloud, fog and edge computing: A review, taxonomy, benchmarks and standards for future research," *Internet of Things*, vol. 12, p. 100273, 2020.

[7] W. Trappe, R. Howard, and R. S. Moore, "Low-energy security: Limits and opportunities in the internet of things," *IEEE Security & Privacy*, vol. 13, no. 1, pp. 14–21, 2015.

[8] A. M. Alwakeel, "An overview of fog computing and edge computing security and privacy issues," *Sensors*, vol. 21, no. 24, 2021.

[9] A. A. Patwary, A. Fu, R. K. Naha, S. K. Battula, S. K. Garg, M. A. K. Patwary, and E. Aghasian, "Authentication, access control, privacy, threats and trust management towards securing fog computing environments: A review," *CoRR*, vol. abs/2003.00395, 2020.

[10] O. A. Khashan, "Hybrid lightweight proxy re-encryption scheme for secure fog-to-things environment," *IEEE Access*, vol. 8, pp. 66 878–66 887, 2020.

[11] A. Ghaderi and Z. Movahedi, "Joint latency and energy-aware data management layer for industrial iot," in *IEEE ICWR*, 2022, pp. 70–75.

[12] I. Amezcua Valdovinos, P. E. F. Millán, J. A. Guerrero-Ibáñez, and R. E. C. Valdez, "Design, implementation, and evaluation of an embedded coap proxy server for 6lowpan," *IEEE Access*, vol. 12, pp. 15 594–15 608, 2024.

[13] Q.-M. Nguyen, L.-A. Phan, and T. Kim, "Load-balancing of kubernetes-based edge computing infrastructure using resource adaptive proxy," *Sensors*, vol. 22, no. 8, 2022. [Online]. Available: https://www.mdpi.com/1424-8220/22/8/2869

[14] M. Martalò, G. Pettorru, and L. Atzori, "A cross-layer survey on secure and low-latency communications in next-generation iot," *IEEE TNSM*, vol. 21, no. 4, pp. 4669–4685, 2024.

[15] T. Shreedhar, R. Panda, S. Podanev, and V. Bajpai, "Evaluating QUIC performance over Web, Cloud storage, and video workloads," *IEEE TNSM*, vol. 19, no. 2, pp. 1366–1381, June 2022, DOI: 10.1109/TNSM.2021.3134562.

[16] P. Kumar and B. Dezfouli, "Implementation and analysis of QUIC for MQTT," *Elsevier Computer Networks*, vol. 150, no. 26, pp. 28–45, February 2019, DOI: 10.1016/j.comnet.2018.12.012.

[17] F. Fernandez, M. Zverev, P. Garrido, J. R. Juarez, J. Bilbao, and R. Aguero, "And QUIC meets IoT: performance assessment of MQTT over QUIC," in *IEEE WiMob*, virtual conference, 2020, pp. 1–6, DOI: 10.1109/WiMob50308.2020.9253384.

[18] S. Jeddou, F. Fernández, L. Diez, A. Baina, N. Abdallah, and R. Agüero, "Delay and energy consumption of MQTT over QUIC: An empirical characterization using commercial-off-the-shelf devices," *MDPI Sensors*, vol. 22, no. 10, 2022, DOI: 10.3390/s22103694.

[19] E. Liri, P. K. Singh, A. B. Rabiah, K. Kar, K. Makhijani, and K. Ramakrishnan, "Robustness of IoT application protocols to network impairments," in *IEEE LANMAN*, Washington, DC, USA, June 2018, pp. 97–103, DOI: 10.1109/LANMAN.2018.8475048.

[20] R. Herrero, "Analysis of QUIC transported CoAP," *SN Computer Science*, vol. 2, no. 2, pp. 1–11, April 2021, DOI:10.1007/s42979-021-00468-0.

[21] G. Pettorru and M. Martalò, "QUIC and WebSocket for secure and low-latency IoT communications: an experimental analysis," in *IEEE ICC*, Rome, Italy, May 2023.

[22] G. Rodola. psutil. [Online]. Available: https://pypi.org/project/psutil/