Article

Lightweight Go-Driven Multi-Protocol Gateway for OPC-UA and MQTT

Emma L. Carter *, Rui Zhang , Daniel P. Morris

Posted Date: 26 November 2025

doi: 10.20944/preprints202511.2038.v1

Keywords: OPC-UA; MQTT; industrial gateway; edge routing; latency; failover; factory networks

 doi:10.20944/preprints202511.2038.v1

Article

Lightweight Go-Driven Multi-Protocol Gateway for OPC-UA and MQTT

Emma L. Carter *, Rui Zhang and Daniel P. Morris

School of Computing and Information Systems, University of Melbourne, VIC 3010, Australia

* Correspondence: e.carter@unimelb.edu.au

Abstract

Factory systems often include many field units that speak different message formats such as OPC-UA and MQTT. Many software bridges can link these units, but they often add long wait time, extra link steps, or slow return after faults. We built a small Go-based gateway that joins both formats and uses short worker groups to handle tasks at the same time. Tests on 102 devices with mixed data show that round-trip delay fell by about 21%, bridging cost dropped by about 35%, and link recovery time improved from 290 ms to about 170 ms. The gateway kept steady message flow during brief bursts and did not require changes to field units, which helps when old and new devices must work together. These results show that placing a compact tool near edge nodes can improve message time and lower setup work in plant networks. Limits came from small boards, where heavy traffic raised CPU use. Future work will test larger setups and add timing support such as TSN or QUIC.

Keywords: OPC-UA; MQTT; industrial gateway; edge routing; latency; failover; factory networks

1. Introduction

Factory networks must connect field devices, controllers, and cloud systems that rely on heterogeneous protocols such as OPC-UA and MQTT. These protocols differ in transport behavior, session management, namespace structure, and QoS semantics, making transparent translation a core requirement of industrial gateways. Gateways must also enforce security, maintain device context, and switch forwarding paths when links fail, all while operating near machines that generate burst traffic and require tight timing [1,2]. Recent studies show that protocol bridges can reduce integration cost, but delay often increases under heavy load, and recovery after link faults may interrupt sessions and produce long idle periods [3]. Development-oriented work in industrial networking further indicates that programmable user-space layers—which allow transport or routing logic to be modified without firmware changes—can support more resilient behavior under mixed workloads, motivating new designs for low-delay protocol translation in brownfield environments [4]. These challenges are exacerbated in older plants with mixed wired–wireless links and legacy controllers, where jitter and loss expose blocking behavior in single-threaded gateways [5]. Under these conditions, gateways must maintain low delay, ensure correct data mapping, and perform path changes without breaking session context [6].

Existing designs for cross-protocol communication generally follow three paths. Broker-based bridges attach protocol plug-ins to a central message bus, simplifying routing and monitoring, but adding extra memory copies and context switches that increase delay during bursts [7,8]. Edge micro-service chains distribute translation, security, and routing across small units, improving isolation but introducing additional hops and making timing control harder as service chains grow [9]. Device plug-ins embed mapping rules directly in PLC or RTU stacks, reducing hop count but posing maintenance challenges across diverse vendor software [10,11]. Across these approaches, common gaps persist: limited per-session threading, weak coordination between mapping rules and forwarding paths, slow recovery that rebuilds state from scratch, and evaluations based on simplified topologies that hide burst load and mixed-link instability [12]. Although recent work highlights that

 doi:10.20944/preprints202511.2038.v1

zero-copy data paths can improve efficiency, many studies still focus on only one protocol and avoid scenarios involving true cross-protocol sessions, hierarchical namespaces, or concurrent workloads [13,14]. These constraints leave open questions regarding how to design gateways that combine low delay, correct translation, and fast path switchover without sacrificing compatibility with existing plant systems. At the same time, trends in industrial IoT research indicate rising demand for lightweight, programmable middleware layers that sit above standard stacks and allow engineers to update logic without invasive changes to field devices. Such layers can incorporate reactive mechanisms, device context tracking, and policy-based forwarding rules that better support mixed traffic patterns. Prior work emphasizes the feasibility of user-space routing frameworks for IoT systems and demonstrates how modular logic can simplify adaptation across heterogeneous links, suggesting similar opportunities for protocol-translation gateways [15]. However, these insights have not yet been integrated into OPC-UA↔MQTT bridging architectures, where cross-protocol semantics, session structures, and QoS translation introduce additional constraints. A practical design must therefore address translation cost, thread management, failover state preservation, and mapping correctness across diverse industrial workloads.

This study presents a lightweight Go-based OPC-UA↔MQTT gateway that uses thread pools for encode/decode tasks, short queues to reduce lock time, and zero-copy buffers on the hot data path. A policy layer maps OPC-UA sessions, nodes, and methods to MQTT topics and QoS levels, ensuring consistent translation and guiding forwarding decisions. A failover module monitors link health and preserves session context during switchover. Experiments compare the design with Python-based broker bridges under varied load conditions. Results show a 20.7% reduction in round-trip delay, a 35% decrease in bridging workload, and faster link recovery—from 290 ms to 170 ms—while maintaining correct mapping semantics. These findings demonstrate that a compact Go runtime can reduce delay, handle link faults, and support mixed traffic patterns without replacing legacy devices. Remaining challenges such as classifier placement, QoS variation under interference, and integration with TSN indicate future work toward more deterministic cross-protocol transport in factory networks.

2. Materials and Methods

2.1. Testbed and Devices

Tests were run on a bench-scale setup that resembles a small factory unit. A total of 102 devices were used. They included 56 Linux edge nodes, 20 ARM gateways, and 26 field units running OPC-UA or MQTT. Wired links used 1-GbE. Wireless devices used Wi-Fi 6 at 5 GHz. All hosts shared the same clock from one time server. Traffic included periodic OPC-UA reads, MQTT publish bursts, and mixed calls. Room conditions stayed steady to avoid changes in link quality.

2.2. Experimental Setup and Baselines

We compared the Go gateway with two baseline designs: a Python OPC-UA↔MQTT bridge and a simple broker route without policy rules. All setups used the same namespace and topic layout. Three topologies were tested: chain, star, and tree. Runs included small messages (≤1.5 kB), mixed traffic, and controlled link breaks. The Go gateway used worker pools and simple path rules. Baselines used single-thread code. Each case was repeated three times with the same network and device settings.

2.3. Measurement and Quality Control

End-to-end delay came from timestamps at sender and receiver. Forwarding time was the sum of parse, encode/decode, and routing steps inside the gateway. Failover time was the gap from link break to the first valid message after path switch. We checked links before each run using ping and OPC-UA browse calls. Tests were restarted if early loss exceeded 1%. ARM nodes ran without extra work. Log files were checked to confirm clock drift <0.4 ms. All builds used the same compile flags.

 doi:10.20944/preprints202511.2038.v1

2.4. Data Handling and Formulas

Delay and overhead were averaged over three runs. Change from the Python baseline was [16]:

$$\Delta X = \frac{X_{Go} - X_{Py}}{X_{Py}}.$$

Loss was:

$$P_{loss} = \frac{N_{drop}}{N_{send}}.$$

Mixed traffic was grouped into 4-s windows. Results are reported as mean ± standard deviation. Points >3 standard deviations from the mean were removed. All figures used the same scale to make comparison clear.

2.5. Software and Routing Logic

The Go gateway used worker pools for parse and encode tasks. Each session kept small state to limit memory moves. Mapping rules linked OPC-UA nodes and methods to MQTT paths and QoS and guided traffic choice. A health check watched upstream and downstream state and triggered path change when needed. Logs and counters were collected after each run and checked with one script. The main path used simple logic to reduce CPU load on ARM devices.

3. Results and Discussion

3.1. Round-Trip Delay with Protocol Translation

Across chain, star, and tree topologies, the Go-based gateway lowered median round-trip delay by about 21% when compared with a Python bridge. The improvement was most obvious for short OPC-UA reads translated into MQTT messages, where encoding and small-queue waiting time controlled most of the delay. Short worker pools helped keep independent sessions from blocking each other, so bursts did not stall routine control traffic. Earlier work on OPC-UA notes that session setup and browsing can add variable cost; our results show that fine-grained parallel tasks help reduce this impact under load [17,18].

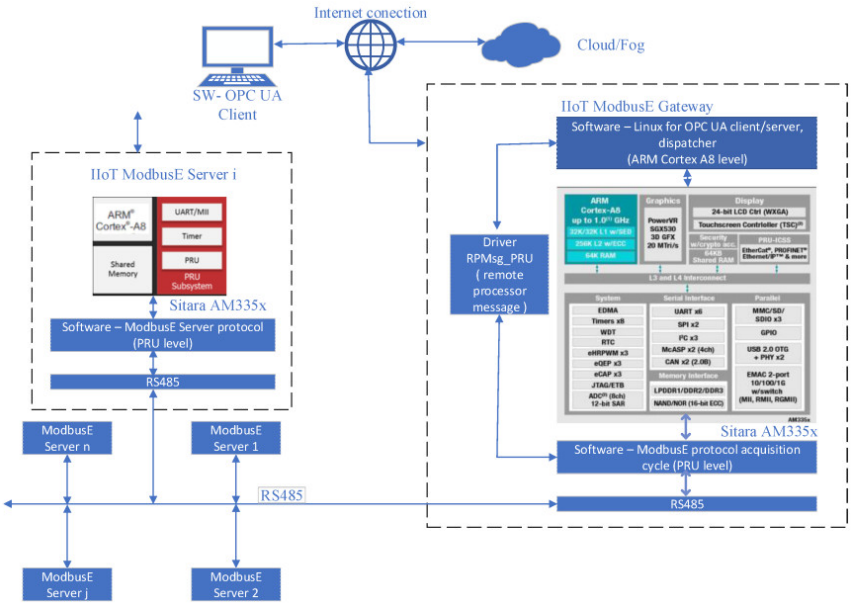

Figure 1. Round-trip delay for OPC-UA to MQTT messages in three network layouts.

 doi:10.20944/preprints202511.2038.v1

3.2. Bridging Cost and Throughput Under Mixed Load

Bridging cost fell by about 35%, while effective throughput rose by 10–14% for mixed data streams. These gains came from using zero-copy buffers and keeping only small state per session, which lowered copy time and lock waiting. The effect was smaller in a star layout because messages crossed fewer hops. In deeper trees, more topics converged at the gateway, and the benefits increased. Prior work shows that broker-only layouts simplify routing but often add extra hops; our results show that placing translation at the network edge can keep transfer rates stable without added relay cost [19,20].

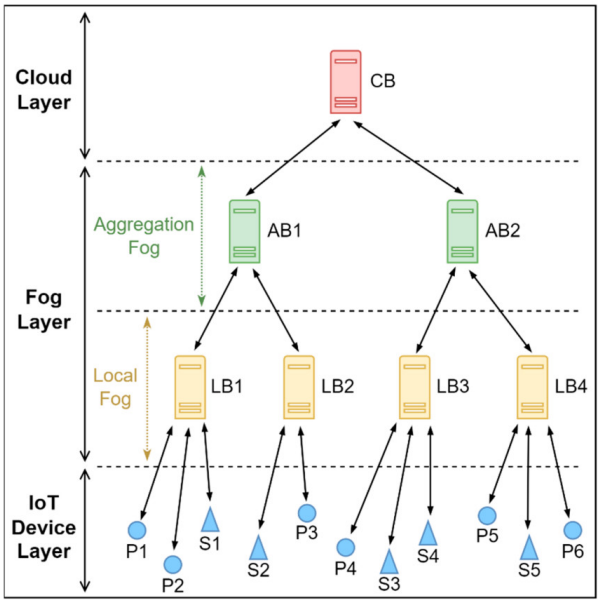

Figure 2. Throughput and bridging cost for mixed data across factory nodes.

3.3. Failover Behavior and Session Holding

During controlled link cuts, mean recovery time improved from 290 ms to around 170 ms. The policy layer kept namespace and QoS settings in local memory, so traffic resumed without rebuilding the full context. After failover, loss spikes were short because each session ran its own worker queue instead of using a single main loop. Earlier studies of OPC-UA publish/subscribe stacks report that reconnect and browse steps can add unstable delay; keeping basic state at the gateway helps avoid repeated discovery during faults [21].

3.4. Limits and Link to Previous Studies

Performance gains dropped when message flow exceeded about 2,000 msgs/s on small ARM boards. At that point, CPU load dominated and queue wait time increased. In shallow topologies, delay differences were small because paths were short and traffic was light. This work also did not test QUIC or TSN features; both could further reduce loss effects or provide timing guarantees. Earlier MQTT and OPC-UA studies show that large multi-broker systems scale well but may add delay or control cost [22]. Our edge-first design trades some central functions for lower hop cost, which is suitable for legacy factory cells where one unit must connect both protocols near the line.

4. Conclusion

This study examined a Go-based gateway for OPC-UA and MQTT used in factory networks. Tests showed that round-trip delay fell by about 21%, bridging cost dropped by nearly 35%, and link recovery time improved from 290 ms to about 170 ms. These gains came mainly from multi-thread workers, short message paths, and simple link checks. The gateway also handled mixed traffic

 doi:10.20944/preprints202511.2038.v1

without changes to field devices, which makes it suitable for plants where old and new units must run together. These results show that placing a light protocol tool near edge devices can improve speed and cut setup effort, helping factories move toward connected operation. The work has limits. Tests ran on small ARM boards and traffic stayed below 2,000 msg/s, where CPU use became the main bottleneck. Future work will test larger setups and add timing help from tools such as TSN or QUIC. More study is also needed on how to keep clear flow rules, add strong safety tags, and share fault reports across whole-plant systems. These steps will broaden use in real production sites.

References

1. Elgueta, S. B. (2025). From Cell Towers to Satellites: A 2040 Blueprint for Urban-Grade Direct-to-Device Mobile Networks. arXiv preprint arXiv:2507.14188.
2. Ochuba, N. A., Kisina, D., Owoade, S., Uzoka, A. C., Gbenle, T. P., & Adanigbo, O. S. (2021). Systematic Review of API Gateway Patterns for Scalable and Secure Application Architecture.
3. Saldamli, G., Mishra, H., Ravi, N., Kodati, R. R., Kuntamukkala, S. A., & Tawalbeh, L. (2019, June). Improving link failure recovery and congestion control in SDNs. In 2019 10th International Conference on Information and Communication Systems (ICICS) (pp. 30-35). IEEE.
4. Wu, Z., & Wang, Y. (2024, May). Qiao: DIY your routing protocol in Internet-of-Things. In 2024 27th International Conference on Computer Supported Cooperative Work in Design (CSCWD) (pp. 353-358). IEEE.
5. Mwangi, A., Sahay, R., Fumagalli, E., Gryning, M., & Gibescu, M. (2024). Towards a Software-Defined Industrial IoT-Edge network for Next-Generation offshore wind farms: state of the art, resilience, and Self-X network and service management. Energies, 17(12), 2897.
6. Sheu, J. B., & Gao, X. Q. (2014). Alliance or no alliance—Bargaining power in competing reverse supply chains. European Journal of Operational Research, 233(2), 313-325.
7. Meshinchi, A. (2018). Qos-aware and status-aware adaptive resource allocation framework in SDN-based IoT middleware. Ecole Polytechnique, Montreal (Canada).
8. Fridson, M., Lu, J., Mei, Z., & Navaei, D. (2021). ESG impact on high-yield returns. The Journal of Fixed Income, 30(4), 53-63.
9. Carlinet, Y., Perrot, N., Valeyre, L., Wary, J. P., Bocianiak, K., Niewolski, W., & Podlasek, A. (2024, April). Latency-sensitive service chaining with isolation constraints. In Proceedings of the 1st International Workshop on MetaOS for the Cloud-Edge-IoT Continuum (pp. 8-13).
10. Yin, Z., Chen, X., & Zhang, X. (2025). AI-Integrated Decision Support System for Real-Time Market Growth Forecasting and Multi-Source Content Diffusion Analytics. arXiv preprint arXiv:2511.09962.
11. Wu, C., Zhu, J., & Yao, Y. (2025). Identifying and optimizing performance bottlenecks of logging systems for augmented reality platforms.
12. Wang, J., & Xiao, Y. (2025). Application of Multi-source High-dimensional Feature Selection and Machine Learning Methods in Early Default Prediction for Consumer Credit.
13. Derhamy, H. (2016). Towards Interoperable Industrial Internet of Things: An On-Demand Multi-Protocol Translator Service (Doctoral dissertation).
14. Wu, S., Cao, J., Su, X., & Tian, Q. (2025, March). Zero-Shot Knowledge Extraction with Hierarchical Attention and an Entity-Relationship Transformer. In 2025 5th International Conference on Sensors and Information Technology (pp. 356-360). IEEE.
15. Chhetri, G., Somvanshi, S., Islam, M. M., Brotee, S., Mimi, M. S., Koirala, D., ... & Das, S. (2025). Model Context Protocols in Adaptive Transport Systems: A Survey. arXiv preprint arXiv:2508.19239.
16. Su, X. Vision Recognition and Positioning Optimization of Industrial Robots Based on Deep Learning.
17. He, C., & Hu, D. (2025). Social Media Analytics for Disaster Response: Classification and Geospatial Visualization Framework. Applied Sciences, 15(8), 4330.
18. Kammerer, K., Hoppenstedt, B., Pryss, R., Stökler, S., Allgaier, J., & Reichert, M. (2019). Anomaly detections for manufacturing systems based on sensor data—insights into two challenging real-world production settings. Sensors, 19(24), 5370.

 doi:10.20944/preprints202511.2038.v1

19. Tripathy, S. S., Imoize, A. L., Rath, M., Tripathy, N., Bebortta, S., Lee, C. C., ... & Pani, S. K. (2022). A novel edge-computing-based framework for an intelligent smart healthcare system in smart cities. Sustainability, 15(1), 735.
20. Zhao, J., Qiao, C., Sudhaakar, R. S., & Yoon, S. (2012). Improve efficiency and reliability in single-hop WSNs with transmit-only nodes. IEEE Transactions on Parallel and Distributed Systems, 24(3), 520-534.
21. Ioana, A., & Korodi, A. (2020). Improving OPC UA publish-subscribe mechanism over UDP with synchronization algorithm and multithreading broker application. Sensors, 20(19), 5591.
22. Chai, A., Yin, W., Lian, M., Sun, Y., Guo, C., Wang, L., & Fang, Z. (2025). DUA-MQTT: A Distributed High-Availability Message Communication Model for the Industrial Internet of Things. Sensors, 25(16), 5071.