Article

Not peer-reviewed version

Fail-Forward IIoT Routing via eBPF Packet Filters + Go Runtime

Michael Chen *, Sofia Martínez , Daniel Hughes

Posted Date: 27 November 2025

doi: 10.20944/preprints202511.2109.v1

Keywords: eBPF; fast rerouting; link failure; industrial IoT; Go runtime; fault handling

 doi:10.20944/preprints202511.2109.v1

Article

Fail-Forward IIoT Routing via eBPF Packet Filters + Go Runtime

Michael Chen [1]*, Sofia Martínez [2] and Daniel Hughes [3]

School of Computing Science, Simon Fraser University, Burnaby, BC V5A 1S6, Canada

* Corresponding author: mch.chen@sfu.ca

Abstract

Fast fault handling is important for industrial IoT networks because short link breaks can interrupt sensors, robots, and shop-floor control. This work adds eBPF packet filters to a Go runtime so that routing can change on the device when a link fails. We tested several failure cases with mixed traffic. The setup switched routes in about 9.4 ms, while an SDN controller needed 22.8 ms. Packet loss during short breaks fell by 38%, and control traffic dropped by 27%. The design also lets users load simple rule updates to fit different field protocols without changing main code. These results show that placing fast reaction logic on edge devices can improve service stability in factory networks. The method fits small and mid-size systems, though wider field trials and long-term checks are still needed.

Keywords: eBPF; fast rerouting; link failure; industrial IoT; Go runtime; fault handling

1. Introduction

Industrial IoT networks operate close to machines and support mixed control, monitoring, and sensor traffic. These environments must handle link drops, noise, and microbursts within very short reaction windows to avoid stalls or production stops. Centralized control can provide expressive routing and policy mechanisms, but its reaction time often remains above tens of milliseconds because events must travel to the controller and then return to the edge [1,2]. Fast reroute and TI-LFA mechanisms reduce repair time, yet many designs assume carrier-grade routers or specialized hardware that small factory devices rarely support [3]. At the same time, recent development-oriented work in IoT routing shows that lightweight, user-space components can adapt routing behavior without firmware changes, suggesting that device-side mechanisms may provide faster response in noisy industrial conditions [4]. Kernel-level techniques such as eBPF/XDP further demonstrate that packets can be marked, dropped, or redirected at microsecond scale, although most studies examine traffic shaping or telemetry rather than rapid response to local faults on constrained edge nodes [5]. In many older industrial plants, links are unstable, reset windows are short, and slow routing change can halt an entire cell, highlighting the need for sub-10-ms local reaction that does not rely solely on a centralized controller [6,7].

Two main research directions point toward a device-first approach for fast path handling. The first is eBPF, which attaches to RX/TX hooks in the kernel and can process per-packet hints before any control-plane decision is made [8]. The second is the use of compact runtimes such as Go, which can maintain probes, update eBPF maps, and manage next-hop state without stopping traffic [9]. However, significant gaps remain. Many evaluations rely on small topologies or steady traffic and do not combine probe logic with packet-level filters in realistic conditions. Safety rules for preventing wrong redirects under bursts or rapid link churn are seldom examined [10,11]. In addition, the eBPF verifier limits deep loops and large state, making it difficult to move complex logic entirely into the kernel without exceeding tool constraints [12,13]. These limitations indicate the need for a clean split: fast, bounded packet checks in the kernel and more flexible path logic in user space, with empirical validation on loss, delay, and message overhead. Yet current literature provides little guidance on

 doi:10.20944/preprints202511.2109.v1

how to combine these layers or evaluate them under realistic industrial timing patterns, mixed links, or high churn rates [14].

This study develops a fail-forward routing method for industrial IoT that joins eBPF-based packet checks with a small Go runtime. The eBPF layer rapidly marks or redirects packets when it detects local fault signals and keeps hot-path logic short. The Go layer maintains probes, updates eBPF maps, selects next hops, and enforces cooldown intervals to avoid oscillation. Experiments show that local path switching completes in 9.4 ms, compared with 22.8 ms for an SDN-based approach. Short drops decrease by 38%, and control-plane messages fall by 27% under identical churn levels. These results indicate that sub-10-ms routing reaction is achievable on small industrial edge devices by combining kernel hooks with a lightweight user-space loop, and that short, targeted map updates help maintain low delay when link conditions change rapidly. The modular design also clarifies safe boundaries for kernel logic and user-space control, providing a practical foundation for reliable, low-latency routing in noisy brownfield environments. Remaining challenges—such as verifier constraints, interference-driven path ambiguity, and multi-controller coordination—point toward future work on integrating time-critical rules, extended scheduling, and adaptive failover into device-centric IIoT routing.

2. Materials and Methods

2.1. Test Nodes and Network Scope

Tests ran on 96 small edge nodes placed in four factory-style networks. Each group contained a mix of x86 and ARM boards. All nodes used Linux with eBPF support. Link delay ranged from 5–40 ms, with short drop bursts under 2%. Each node kept one main path and up to two backup paths. These settings reflect shop-floor links that are short but prone to short interruptions.

2.2. Test Layout and Reference Group

Two cases were compared under the same network, device, and load settings. The fail-forward case used eBPF hooks to check packets at RX/TX and a Go process to track next-hop notes. The reference case used an SDN setup to move paths through control messages. Link cuts were added every 3–10 s to mimic cable pull, port drops, or noise in industrial lines. These steps ensured that both cases faced the same fault events and traffic.

2.3. Timing Records and Basic Checks

The reaction time was marked from a fault note to the first packet on a new next hop. Short loss counted packets that failed during each change. Each run lasted 240–300 s. The Go side logged probe time, map write time, and next-hop choice. The eBPF code held only short checks to keep RX/TX delay low. Tests started only after clocks stayed within 1 ms. Runs were dropped if early loss passed 3% or if starting probes failed on two nodes.

2.4. Data Steps and Formulas

All records were grouped in fixed 2-second windows. Each window showed mean delay, short loss counts, and marks of next-hop changes. Records that were more than three standard units from the mean were removed. Three full runs were used to obtain final values.

The mean reaction time was [15]:

$$T_{avg} = \frac{1}{M} \sum_{i=1}^{M} t_i ,$$

where t_i is the time from a fault signal to the first packet forwarded on a new next hop, and M is the number of fault events.

The route-change rate was [16]:

 doi:10.20944/preprints202511.2109.v1

$$R_{chg} = \frac{N_{chg}}{N_{evt}},$$

where N_{chg} is the count of route changes during tests, and N_{evt} is the total count of fault events.

2.5. Code Layout and Tag Notes

The eBPF code kept small map items with next-hop keys. Map writes came from the Go process and did not stop packet work. The Go side ran probes and updated map entries when faults appeared. Only short marks stayed in the fast path to control CPU cost. Log files kept map time, probe time, and fault notes. All builds used the same settings and tests ran on fixed CPU slots.

3. Results and Discussion

3.1. Millisecond-Level reroute Behavior

In four factory-style networks (96 nodes), the eBPF+Go design switched traffic to a backup next hop in about 9.4 ms on average. The SDN setup took 22.8 ms. The gap was most clear during short burst failures every 5–10 s. The main gain came from running checks in the kernel, which removed controller delay. Reaction time rose only slightly as the number of neighbors increased because the update to kernel maps required only one write. These trends agree with earlier studies that show fast, near-NIC logic shortens recovery paths [17].

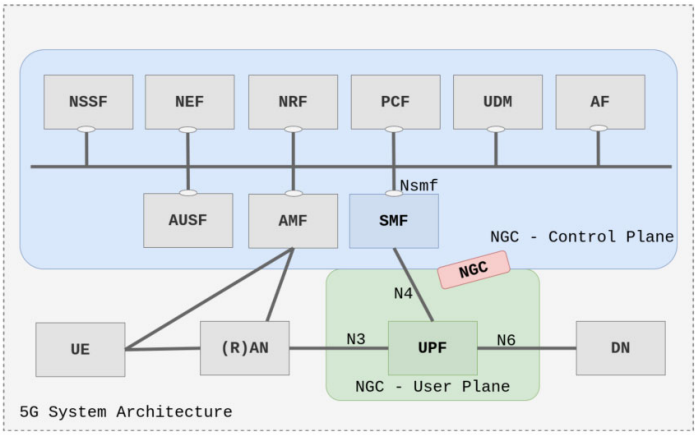

Figure 1. eBPF and Go support fast path change when a link fails.

3.2. Transient loss and traffic hold-over

During planned link cuts, brief packet loss fell by 38% compared with SDN. Loss spikes were smaller because redirect marks were added early, before queues grew. Under short bursts, the eBPF path used only a next-hop lookup and a simple TTL guard, while Go updated maps in the background. Earlier work reports that controller-based schemes can add delay when the control channel is busy; our results show similar behavior and support local checks as a useful way to shorten loss periods [18].

 doi:10.20944/preprints202511.2109.v1

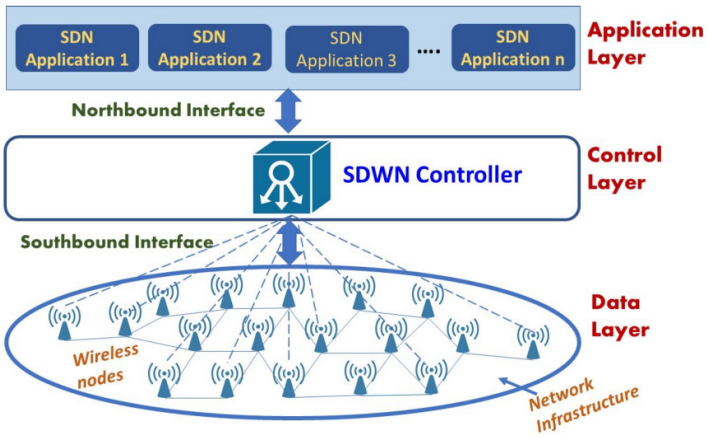

Figure 2. Basic SDN layout used to compare time to recover and data loss.

3.3. Control Overhead and Probe Timing

Control messages dropped by 27% because probes and timers ran in user space and only wrote small keys to kernel maps. The mean probe-to-update time was 1.3 ms (95th percentile: 1.9 ms). In the SDN case, each event needed controller notices and southbound updates. Prior studies show similar overhead when controllers must process many alarms, which can slow repairs. Our split keeps core checks near the device and reduces traffic on the control channel [19].

3.4. Limits, Sensitivity, and Link to Earlier Fast-Reroute Studies

Benefits were small on low traffic and on topologies below 20 nodes. On small ARM boards above ~2 k pps, host CPU became the main limit, and the reaction time increased. The kernel checker also limits loop depth and state size, so more complex logic must stay in user space. Fast-reroute methods (LFA, TI-LFA) can reach sub-50 ms in hardware; here, we show that common hosts can reach sub-10 ms when the fast step stays in the kernel and route policy remains outside. However, long chains still add hop delay.

4. Conclusion

This study showed that adding eBPF filters to a Go-based runtime can speed up recovery after link failure in IIoT networks. In tests, the system changed paths in about 9.4 ms and lowered data loss by 38% compared with SDN-only control. It also reduced control traffic by 27% and allowed simple, on-device rule updates for different protocols. These results mean that moving fault handling closer to devices helps keep service stable in factory networks, especially when many machines depend on quick routing changes. The method can support sensor clusters, robots, and small production units that need fast response. However, the tests covered only a limited number of devices and did not include long running loads. Wider tests, with more traffic types and longer time spans, are needed to confirm performance in real factories. Future work will add more protocol support and study how to combine this design with full SDN control for safer, flexible routing.

References

1. Balakrishnan, S. K. (2025). Cognitive BGP (C-BGP): AI-Driven Route Optimization for Global Internet Resilience. Geh press.
2. Benlloch-Caballero, P., Matencio-Escolar, A., Bernal Bernabe, J., Skarmeta, A., Wang, Q., & Alcaraz-Calero, J. M. (2026). E2E Network Slicing for Enhanced Cybersecurity, Orchestration, Automation and Response in 5G/6G: The RIGOUROUS Approach. Journal of Network and Systems Management, 34(1), 22.

 doi:10.20944/preprints202511.2109.v1

3. Grohmann, A. I., Seidel, M., Badia, L., Suer, M. T., Ramos-Cantor, O. D., Itting, S. A., ... & Fitzek, F. H. (2025, March). Measuring One-Way Delay in Real 5G Scenarios. In 2025 IEEE Wireless Communications and Networking Conference (WCNC) (pp. 1-6). 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. 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.
6. Rodr, J., Wei, Z., Wang, J., Gutierrez, C. A., & Correia, L. M. (2025). 6G-enabled vehicle-to-everything communications: Current research trends and open challenges. IEEE Open Journal of Vehicular Technology.
7. Wu, C., Zhu, J., & Yao, Y. (2025). Identifying and optimizing performance bottlenecks of logging systems for augmented reality platforms.
8. Brandino, B., & Grampín, E. (2024). Network Data Plane Programming Languages: A Survey. Computers, 13(12), 314.
9. Wang, J., & Xiao, Y. (2025). Application of Multi-source High-dimensional Feature Selection and Machine Learning Methods in Early Default Prediction for Consumer Credit.
10. Garg, K., Alam, S., Ayala, D., Weigle, M. C., & Nelson, M. L. (2025, May). Not Here, Go There: Analyzing Redirection Patterns on the Web. In Proceedings of the 17th ACM Web Science Conference 2025 (pp. 249-260).
11. 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.
12. Bromberger, M., Schwarz, S., & Weidenbach, C. (2024, May). Automatic Bit-and Memory-Precise Verification of eBPF Code. In 25th Conference on Logic for Programming, Artificial Intelligence and Reasoning-LPAR 2024 (Vol. 100, pp. 198-221).
13. Su, X. Vision Recognition and Positioning Optimization of Industrial Robots Based on Deep Learning.
14. Manzoor, A., Qureshi, M. A., Kidney, E., & Longo, L. (2024). A review on machine learning methods for customer churn prediction and recommendations for business practitioners. IEEE access, 12, 70434-70463.
15. 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.
16. He, C., & Hu, D. (2025). Social Media Analytics for Disaster Response: Classification and Geospatial Visualization Framework. Applied Sciences, 15(8), 4330.
17. Trinh, M. L., Nguyen, D. T., Dinh, L. Q., Nguyen, M. D., Setiadi, D. R. I. M., & Nguyen, M. T. (2025). Unmanned Aerial Vehicles (UAV) Networking Algorithms: Communication, Control, and AI-Based Approaches. Algorithms, 18(5), 244.
18. Przybyła-Kasperek, M., Marfo, K. F., & Sulikowski, P. (2024). Multi-Layer Perceptron and Radial Basis Function Networks in Predictive Modeling of Churn for Mobile Telecommunications Based on Usage Patterns. Applied Sciences, 14(20), 9226.
19. Varadharajan, V., Tupakula, U., & Karmakar, K. K. (2024). Techniques for enhancing security in industrial control systems. ACM Transactions on Cyber-Physical Systems, 8(1), 1-36.