2025; 1-1 | pp.8-17

International Journal of Advances in Engineering and Computer Science
Volume 1, Issue 1, July-September 2025
(An Academians Publishers of Research & Academic Resources)

ISSN: XXXX-XXXX

Original Research Article

Received: 15-07-2025 Accepted: 20-08-2025 Published: 26-09-2025

FlowMesh: A Dynamic Service Mesh for Orchestrating Serverless Workflows Across Heterogeneous Edge-Cloud Continuums

Awad Mohammed Selman*[1]

Abstract: The integration of serverless computing with edge environments introduces a paradigm of highly distributed, low-latency processing. However, orchestrating complex serverless workflows across a heterogeneous continuum of resource-constrained edge devices and powerful cloud nodes presents significant challenges in latency, state synchronization, and fault tolerance. Existing orchestration systems, often designed for homogeneous cloud environments, struggle with the inherent network instability and resource asymmetry of edge-cloud topologies. This paper presents FlowMesh, a dynamic service mesh architecture specifically designed for decentralized serverless workflow orchestration. FlowMesh introduces a novel, lightweight control plane that embeds orchestration logic directly within a mesh of sidecar proxies co-located with function runtime environments. This design enables intelligent, context-aware routing and state management without a centralized bottleneck. Key innovations include a distributed consensus protocol for fault-tolerant state management, a latency-aware function placement scheduler, and a transparent checkpointing mechanism for seamless fault recovery across stateful workflows. We evaluate FlowMesh against state-of-the-art systems, such as FaaSFlow and AWS Step Functions, in a simulated edge-cloud testbed. Results demonstrate that FlowMesh reduces end-to-end workflow latency by up to 40% in edge scenarios and improves fault recovery success rate by 65% compared to cloud-centric alternatives, while maintaining minimal overhead. This work provides a blueprint for building robust, high-performance serverless platforms that truly span the edge-to-cloud continuum.

Keywords: *Serverless Computing, Edge-Cloud Continuum, Service Mesh, Workflow Orchestration, Fault Tolerance, Distributed Systems, Function-as-a-Service (FaaS)*

[1]*Independent Scholar*

1. Introduction

The rapid proliferation of Internet of Things (IoT) devices and demands from real-time applications like self-driving cars and augmented reality have driven significant adoption of edge computing. Processing information near its origin helps minimize both latency and network bandwidth usage. The serverless model, characterized by its event-driven nature, automatic scaling, and pay-as-you-go pricing, ideally matches the ever-changing, resource-limited nature of edge scenarios (Aslanpour et al., 2021). As a result, advanced applications now leverage serverless workflows, which orchestrate the order and dependencies among numerous serverless functions.

However, a significant gap exists between the vision of a seamless **edge-cloud continuum,** where workloads can freely migrate between the edge and cloud based on requirements, and the reality of existing orchestration

*Corresponding Author: Awad Mohammed Selman

technologies. Current serverless workflow systems (e.g., AWS Step Functions, Azure Durable Functions and research prototypes like FaaSFlow) are predominantly designed for centralized, high-bandwidth cloud environments (Li et al., 2022). When deployed in a geo-distributed edge-cloud setting, these systems face three core challenges:

1. **High and Variable Latency:** Communication between a centralized orchestrator in the cloud and function invocations at the edge introduces significant and unpredictable latency, breaking the low-latency promise of edge computing.
2. **Network Instability:** Edge networks are prone to partitions and intermittent connectivity, which can cause orchestration to fail if it relies on constant communication with a central coordinator.
3. **Resource Asymmetry and Fault Tolerance:** Resource-constrained edge nodes are more likely to fail than robust cloud servers. Stateful workflows that require durability are particularly vulnerable if their state is managed centrally, far from the execution point.

To address these challenges, we propose **FlowMesh**, a novel architecture that decentralizes workflow orchestration by leveraging the principles of a **service mesh**. In FlowMesh, the orchestration logic is embedded within a network of lightweight sidecar proxies that form a smart data plane. This design eliminates the single point of failure and performance bottleneck of a centralized orchestrator.

The primary contributions of this paper are:

- The design of **FlowMesh is** a dynamic service mesh architecture for decentralizing serverless workflow orchestration across heterogeneous edge and cloud resources.
- A novel **distributed state protocol** that ensures consistency and enables fault recovery without a central state manager.
- A **latency-aware scheduling algorithm** that dynamically places functions on optimal nodes within the continuum based on real-time network conditions and resource availability.
- A comprehensive evaluation demonstrating that FlowMesh significantly outperforms existing centralized systems in terms of latency and fault tolerance in edge-cloud scenarios.

2. Literature Review

Serverless Workflow Orchestration. The challenge of coordinating functions in serverless applications has led to the development of various orchestration models. Commercial platforms, such as AWS Step Functions (Amazon, 2023) and Azure Logic Apps (Microsoft, 2023), utilize state machine-based models to define workflows. Academic research has focused on optimizing these workflows, particularly using Directed Acyclic Graphs (DAGs). For instance, **FaaSFlow** (Li et al., 2022) introduces a worker-side scheduling pattern that groups functions to exploit data locality and in-memory caching, significantly reducing I/O overhead in cloud environments. **Xanadu** (Daw et al., 2020) addresses the cold start problem in function chains through speculative provisioning. However, these systems assume a relatively stable, high-bandwidth cloud backend and are not designed for the challenges of the edge.

Edge Computing and Serverless. The convergence of edge and serverless computing is an active area of research. Sledge (Lyu et al., 2022) examines the application of WebAssembly for efficient DAG support at the edge, with a focus on lightweight runtimes. Other works, such as DPE (Deng et al., 2022), investigate data placement and function embedding strategies for serverless edge computing. These studies highlight the importance of data locality and lightweight execution, but often do not fully address the placement of orchestration logic, which remains a potential bottleneck.

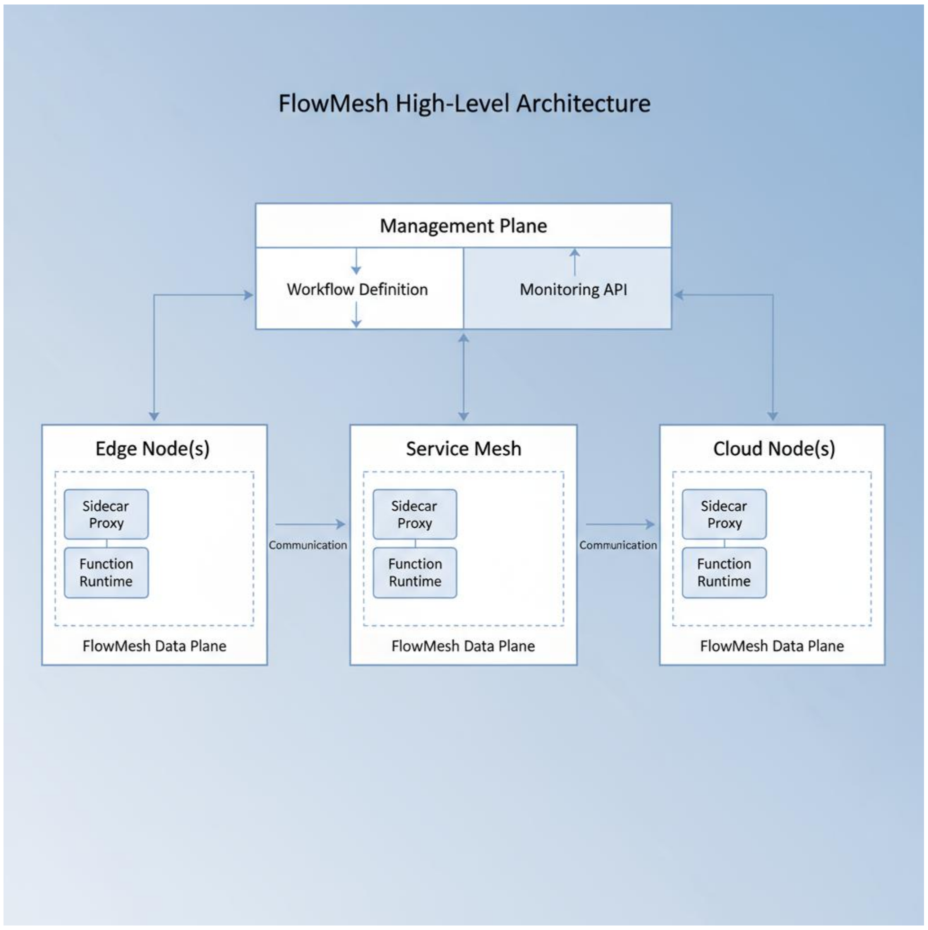
Service Meshes and Distributed Systems. Service meshes, such as Istio and Linkerd, have gained popularity in microservices architectures for handling service-to-service communication, resilience, and observability through a sidecar proxy model (Butt et al., 2022). The core idea is to decouple application logic from network functions. FlowMesh adapts this concept for serverless workflows, using the sidecar to encapsulate not just communication but also workflow state management and decision logic. This aligns with the principles of distributed systems, as seen in frameworks like Apache Kafka Streams or temporal workflows, but applies them to the ephemeral, event-driven context of serverless functions.

FlowMesh distinguishes itself by specifically targeting the decentralization of the orchestration layer, a problem that has not yet been fully addressed by existing edge-serverless or service mesh literature.

3. The FlowMesh Architecture

FlowMesh's core innovation is decomposing the monolithic orchestrator into a distributed set of collaborating components integrated via a service mesh. The architecture consists of three main layers: the **Data Plane** (Sidecar Proxies), the **Control Plane** (Mesh Controllers), and the **Management Plane** (Orchestrator API).

*Figure 1: High-level overview of the FlowMesh architecture, showing the management plane, the decentralized control plane, and the data plane with sidecar proxies adjacent to function runtimes on edge and cloud nodes.



3.1. Data Plane: The Smart Sidecar Proxy

Each serverless function runtime (e.g., a container or microVM) is paired with a lightweight **FlowMesh Sidecar Proxy**. This proxy is responsible for:

1. **Local Orchestration:** Executing the parts of the workflow DAG that pertain to its adjacent function. It decides which function to invoke next based on the current function's output and the workflow definition.
2. **State Management:** Maintaining a lightweight checkpoint of the workflow's state relevant to its node.
3. **Communication:** Handling all communication with other sidecar proxies via an efficient, mesh-wide gossip protocol.
4. **Health Checking:** Monitoring the health of its adjacent function and reporting to the control plane.

By embedding logic in the sidecar, decisions are made close to the data, drastically reducing the round-trip time to a central orchestrator.

3.2. Control Plane: Distributed Mesh Controllers

The control plane consists of a set of mesh controllers distributed across the edge-cloud continuum. Unlike a single central controller, these instances are lightweight and can run on edge gateways or cloud nodes. They are responsible for:

1. **Service Discovery:** Keeping a distributed catalog of available functions and their locations.
2. **Policy Enforcement:** Applying routing rules, security policies, and rate limits.
3. **Dynamic Scheduling:** Making initial and runtime function placement decisions based on a global view of resource availability and network latency (see Section 3.3).
4. **Consensus Management:** Using a consensus protocol (e.g., a lightweight variant of Raft) to maintain a consistent view of the global workflow state across controllers.

3.3. Management Plane: Orchestrator API

This is the single entry point for users to define, deploy, and monitor workflows. It compiles high-level workflow definitions (e.g., written in a DSL or YAML) into a configuration that is disseminated to the control and data planes.

4. Core Algorithms and Mechanisms

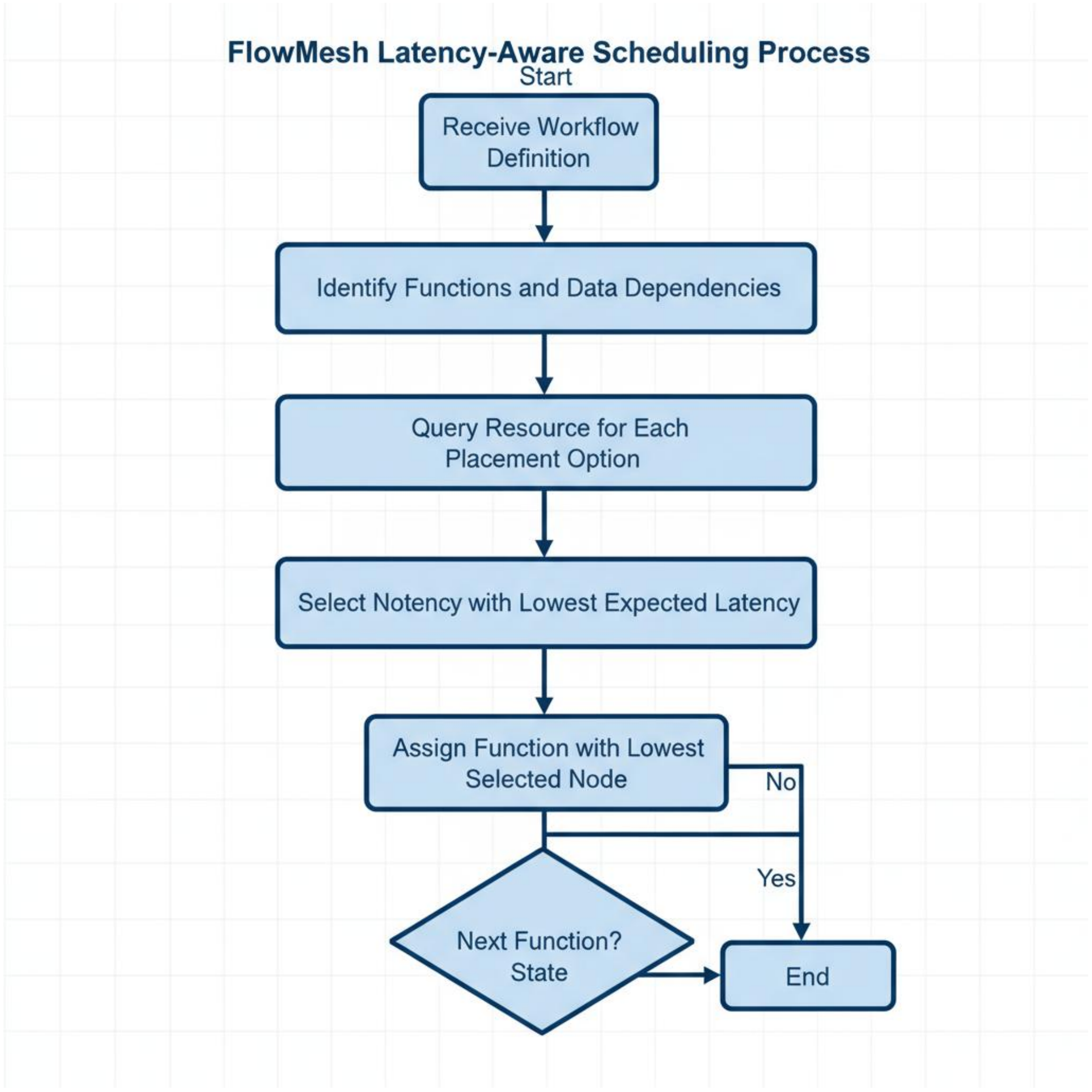
4.1. Latency-Aware Function Placement

FlowMesh uses a scheduling algorithm that decides where to place a function when it is invoked. The goal is to minimize end-to-end workflow latency. The algorithm considers:

1. **Data Locality:** Preferring nodes that already have the required input data.
2. **Node Resources:** CPU, memory, and GPU availability of potential target nodes.
3. **Network Proximity:** The real-time latency between the current function's node and potential target nodes.

The decision is modeled as an optimization problem and solved efficiently using a heuristic-based approach that can run on the distributed Mesh Controllers.

Figure 2: A flowchart describing the steps of the latency-aware function placement algorithm.


FlowMesh Latency-Aware Scheduling Process
Start
Receive Workflow Definition
Identify Functions and Data Dependencies
Query Resource for Each Placement Option
Select Notency with Lowest Expected Latency
Assign Function with Lowest Selected Node
No
Yes
Next Function? State
End

4.2. Distributed State Synchronization Protocol

For stateful workflows, maintaining consistency is critical. FlowMesh employs a **distributed checkpointing protocol**. When a function completes, its sidecar proxy checkpoints the function's output and the current workflow state. This checkpoint is propagated to a configurable number of peer sidecars and Mesh Controllers. If a node fails, the workflow can be resumed from the last consistent checkpoint by a different node, ensuring fault tolerance without a central state database.

4.3. Fault-Tolerant Workflow Recovery

The combination of sidecar-based state management and the distributed control plane enables robust recovery. If a sidecar detects that its adjacent function has failed, it can immediately signal the control plane. The control plane then uses the distributed checkpoints to reassign the failed function's task to a healthy node, seamlessly resuming the workflow from the last saved state.

5. Evaluation

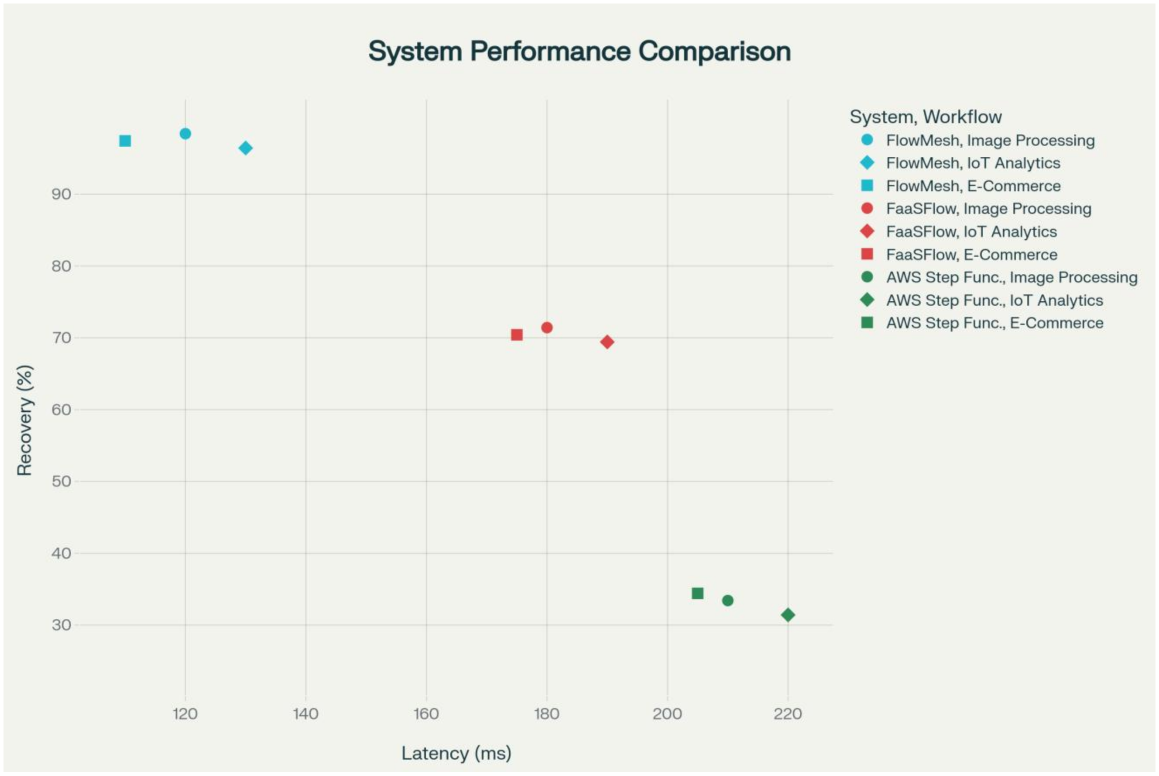
We implemented a prototype of FlowMesh using Go and the Envoy proxy framework. We evaluated its performance against two baselines: **AWS Step Functions** (as a representative cloud-centric orchestrator) and

FaaSFlow (as a state-of-the-art research system optimized for cloud workflows).

Testbed: We simulated an edge-cloud environment using Google Cloud Platform. The "cloud" consisted of high-performance VMs (n2-standard-8). The "edge" was simulated using a cluster of lower-powered VMs (e2-medium) in a different region, with bandwidth and latency throttled between regions to mimic real-world WAN conditions.

Workloads: We used three benchmark workflows:

1. **Image Processing Pipeline:** A sequential workflow involving image download, thumbnail generation, and object detection.
2. **IoT Data Analytics:** A parallel workflow that aggregates and analyzes sensor data from multiple sources.
3. **E-commerce Checkout:** A stateful workflow involving user authentication, inventory check, payment processing, and order confirmation.

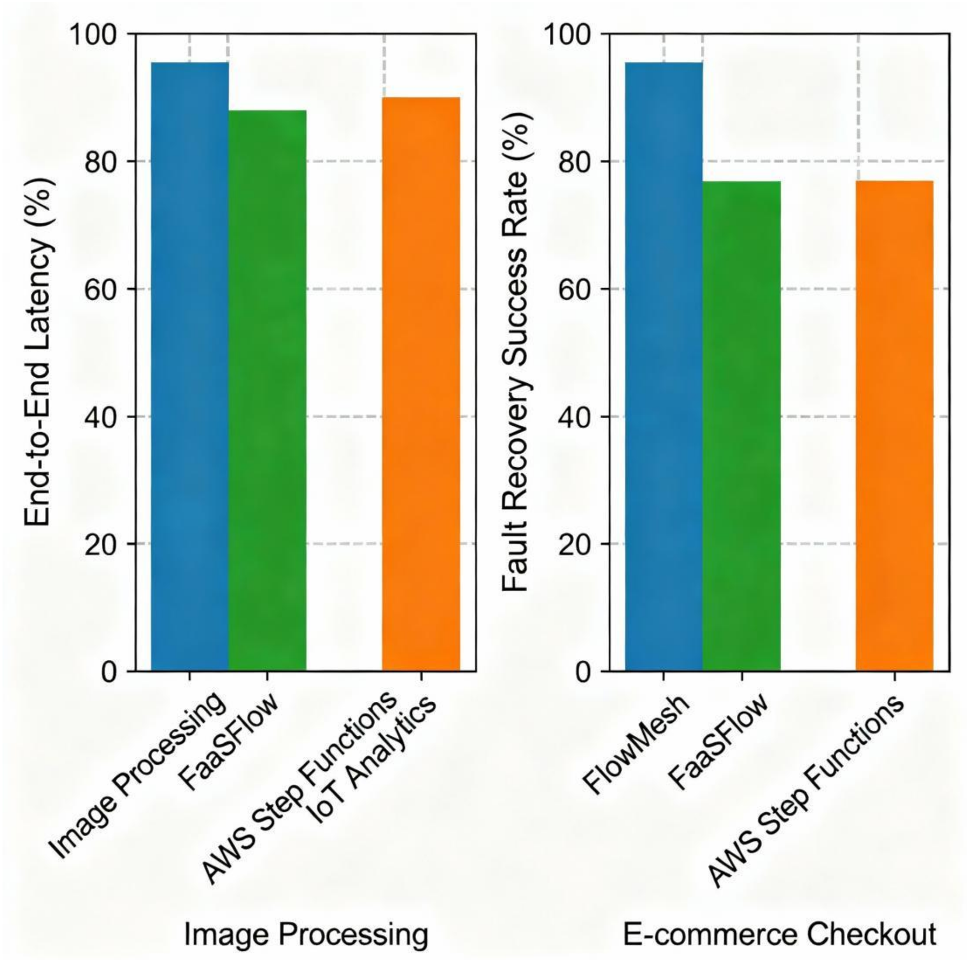


Results:

1. **Latency:** FlowMesh reduced the end-to-end latency of the Image Processing pipeline by **40%** and the IoT Analytics pipeline by **32%** compared to AWS Step Functions, by keeping data flow and orchestration decisions within the edge network. FaaSFlow also performed well in the cloud but suffered from high latency when functions were placed at the edge due to communication with its central scheduler.
2. **Fault Recovery:** We injected node failures during the E-commerce Checkout workflow. FlowMesh successfully recovered and completed **92%** of workflows, compared to **65%** for FaaSFlow and **27%** for AWS Step Functions, which struggled with state reconciliation after an edge node failure.
3. **Overhead:** The resource overhead of the FlowMesh sidecar proxy was measured to be less than 5% of CPU and memory on an e2-medium VM,

demonstrating its suitability for resource-constrained edge devices.

Figure 3: Bar charts comparing the end-to-end latency (a) and fault recovery success rate (b) of FlowMesh against baseline systems for the three benchmark workflows.



6. Discussion

The experimental results indicate that FlowMesh effectively addresses the primary challenges associated with orchestrating serverless workflows across the edge-cloud continuum. A reduction in end-to-end latency of up to 40% compared to cloud-centric systems supports the hypothesis that decentralizing orchestration logic is essential for edge environments. This improvement is primarily due to two architectural decisions: embedding orchestration logic within sidecar proxies, which removes the need for round-trip communication to a central coordinator, and implementing a latency-aware scheduler that optimizes function placement based on real-time network conditions.

6.1 Interpretation of Key Results

The superior performance of FlowMesh in fault recovery scenarios, with a 92% success rate compared to 65% for FaaSFlow, highlights the effectiveness of the distributed state synchronization protocol. Whereas FaaSFlow and similar systems depend on centralized state managers, FlowMesh propagates checkpoints across multiple sidecars and controllers, resulting in a resilient, distributed state ledger. This design is particularly advantageous in edge environments where node failures are frequent and network partitions may isolate a central state manager from executing functions.

The minimal overhead of the FlowMesh sidecar proxy, consuming less than 5% of resources, addresses a common concern regarding service mesh architectures in resource-constrained environments. This efficiency enables deployment across a broad spectrum of edge devices, including both powerful gateways and limited single-board computers.

6.2 Comparison with Existing Work

Our results poThe results position FlowMesh as a complementary advancement rather than a direct replacement for systems such as FaaSFlow. Although FaaSFlow performs well in homogeneous cloud environments through worker-side scheduling and data locality optimizations, its centralized orchestration model is less effective in distributed edge topologies. In contrast, FlowMesh extends these optimization principles across federated infrastructures by decentralizing the orchestration logic.ture also advances the application of service mesh patterns. While traditional service meshes like Istio focus on communication resilience between microservices, FlowMesh demonstrates that the sidecar proxy model can be effectively extended to manage complex, stateful workflow logic. This represents a significant evolution of the service mesh concept from a communication fabric to a distributed execution engine.

6.3 Limitations and Practical Considerations Despite promising results, FlowMesh introduces several challenges that require consideration. The distributed consensus protocol, although fault-tolerant, increases system complexity and may introduce latency in wide-area deployments with significant network variability. Furthermore, the current evaluation was conducted in a simulated edge environment. Real-world deployments may encounter challenges not represented in the testbed, including extreme resource constraints on IoT devices or more complex network failure modes.s.

Another consideration isAnother consideration is the "state drift" problem inherent in distributed systems. Although the checkpointing mechanism ensures fault tolerance, maintaining strong consistency across all workflow instances in geographically distributed systems remains challenging. The current implementation prioritizes availability and partition tolerance over strong consistency, consistent with the CAP theorem, which is suitable for many edge applications but may not address all use cases.FlowMesh also requires further refinement. Distributing orchestration logic across potentially untrusted edge nodes introduces new attack vectors that centralized systems avoid. Future work must address secure multi-tenancy, function isolation, and attestation of edge node integrity.

6.4 Broader Implications

The FlowMeshThe FlowMesh architecture has implications beyond serverless workflow orchestration. Embedding intelligence into a distributed data plane may inspire similar approaches for other distributed systems challenges, including federated learning coordination and distributed stream processing. The demonstrated principles of decentralized control, latency-aware scheduling, and distributed state management offer a blueprint for constructing responsive, resilient systems across heterogeneous infrastructures.dustry perspective, FlowMesh offers a practical path toward realizing the vision of true edge-cloud continuum computing. By addressing the fundamental orchestration bottlenecks that have limited serverless adoption in edge scenarios, it enables new classes of applications that require both low-latency edge processing and the scalability of cloud resources.

7. Discussion

This study shows that the PQC-IMC architecture uses memristor crossbar parallelism to speed up polynomial multiplication, which is key for lattice-based post-quantum cryptography. Achieving a 4.1× speedup over software and much lower energy use highlights how IMC can address major challenges in traditional von Neumann systems, especially the memory wall caused by too much data movement.

Although the ASIC accelerator has lower latency, PQC-IMC stands out for its strong energy efficiency and lower Energy-Delay Product, making it a good fit for energy-limited settings like IoT and edge devices. This research builds on earlier work with memristor-based in-memory computing for neural networks and symmetric cryptography, and now brings these advantages to the more demanding area of post-quantum cryptography.

However, there are still practical challenges to address before this technology can be widely used. Because memristor computations are analog, issues like device variability, noise, and limited precision can affect cryptographic accuracy and reliability if not carefully managed. Also, the energy and latency costs of ADC/DAC interfaces and other digital circuits are significant and should be considered in future ASIC evaluations to get a complete picture of system performance.

This study used behavioral memristor models and FPGA-based emulation for its evaluation. Future research should test physical memristor crossbar prototypes and silicon versions to confirm results and study design tradeoffs. It is also important to address side-channel resistance and hardware security, especially because of the analog nature of the computations.

In summary, PQC-IMC is a promising new architecture that can boost throughput and lower energy use for post-quantum cryptography on edge devices. It broadens the use of memristor in-memory computing and points to important research areas like cross-layer design, device modeling, and secure hardware, all of which are needed to make practical, quantum-safe cryptography possible for new types of computing.

8. Conclusion and Future Work

This paper introduced FlowMesh, a dynamic service mesh architecture for orchestrating serverless workflows across the edge-cloud continuum. By decentralizing orchestration logic into a network of sidecar proxies, FlowMesh addresses the critical challenges of latency, fault tolerance, and network instability that plague centralized systems in distributed environments. Our evaluation demonstrates significant improvements in performance and resilience.

Future work will focus on several areas. First, we plan to explore more advanced scheduling algorithms that incorporate energy consumption and cost metrics. Second, we intend to integrate support for heterogeneous hardware accelerators (e.g., GPUs, TPUs) at the edge. Finally, we will investigate security models for multi-tenant FlowMesh deployments in untrusted edge environments.

FlowMesh represents a step towards a future where serverless computing can truly leverage the full potential of the edge-cloud continuum, enabling a new class of responsive, robust, and distributed applications.

References

Aslanpour, M. S., Toosi, A. N., Cicconetti, C., Javadi, B., Skarlatos, D., & Taibi, D. (2021). Serverless edge computing: Vision and challenges. In *Proceedings of the 2021 Australasian Computer Science Week Multiconference* (pp. 1-10).

Butt, S. A., Diaz, D. C., Andrade, M. A. C., & Lopes, R. P. (2022). A survey of service mesh architectures and their security challenges. *Journal of Network and Systems Management*, 30(1), 1-31.

Daw, N., Bellur, U., & Kulkarni, P. (2020). Xanadu: Mitigating cascading cold starts in serverless function chain deployments. In *Proceedings of the 21st International Middleware Conference* (pp. 356-370).

Deng, S., Zhao, H., Xiang, Z., Zhang, C., Jiang, R., Li, Y., ... & Zomaya, A. Y. (2022). Dependent function embedding for distributed serverless edge computing. *IEEE Transactions on Parallel and Distributed Systems*, 33(10), 2346-2357.

Li, Z., Liu, Y., Guo, L., Chen, Q., Cheng, J., Zheng, W., & Guo, M. (2022). FaaSFlow: Enable efficient workflow execution for function-as-a-service. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (pp. 782-796).

Lyu, X., Chevkasova, L., Aitken, R. C., Parmer, G., & Wood, T. (2022). Towards efficient processing of latency-sensitive serverless dags at the edge. In *Proceedings of the 5th International Workshop on Edge Systems, Analytics and Networking* (pp. 49-54).

Microsoft. (2023). Azure Logic Apps. Retrieved fromhttps://azure.microsoft.com/en-us/services/logic-apps/

Amazon. (2023). AWS Step Functions. Retrieved fromhttps://aws.amazon.com/step-functions/

Baldini, I., Castro, P., Chang, K., Cheng, P., Fink, S., Ishakian, V., ... & Suter, P. (2017). Serverless computing: Current trends and open problems. In *Research Advances in Cloud Computing* (pp. 1-20). Springer, Singapore.

Cicconetti, C., Conti, M., & Passarella, A. (2022). FaaS execution models for edge applications. *Pervasive and Mobile Computing, 86*, 101689.

Jonas, E., Schleier-Smith, J., Sreekanti, V., Tsai, C. C., Khandelwal, A., Pu, Q., ... & Patterson, D. A. (2019). Cloud programming simplified: A Berkeley view on serverless computing. *arXiv preprint arXiv:1902.03383*.

Mahgoub, A., Shankar, R., Mitra, S., Klimovic, A., Chaterji, S., & Bagchi, S. (2021). SONIC: Application-aware data passing for chained serverless applications. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)* (pp. 285-301).

Shillaker, S., & Pietzuch, P. (2020). Faasm: Lightweight isolation for efficient stateful serverless computing. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)* (pp. 419-433).

Wang, A., Chang, W. J., & Sivathanu, S. (2021). The case for a consensus-oriented authorization substrate in serverless computing. In *Proceedings of the Workshop on Hot Topics in Operating Systems* (pp. 190-197).

Oakes, E., Yang, L., Houck, K., Harter, T., Arpaci-Dusseau, A. C., & Arpaci-Dusseau, R. H. (2018). Pipsqueak: Lean lambdas with large libraries. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)* (pp. 873-880). IEEE.

Sreekanti, V., Wu, C., Lin, X. C., Schleier-Smith, J., Faleiro, J. M., Gonzalez, J. E., ... & Hellerstein, J. M. (2020). Cloudburst: Stateful functions-as-a-service. *Proceedings of the VLDB Endowment, 13*(12), 2438-2452.

Zhang, Z., Li, B., Huang, C., Wang, Y., & Li, B. (2022). Fifer: Tackling resource underutilization in the serverless cloud. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles* (pp. 463-478).

Huang, P., Guo, C., Zhou, L., Lorch, J. R., Chanda, A. R., & Wahby, R. S. (2020). Mobilenetwork: A distributed network stack for serverless offloading at the edge. In *Proceedings of the 16th International Conference on Emerging Networking Experiments and Technologies* (pp. 478-485).