ISSN 2277 – 8322

EVALUATING THE IMPACT OF KUBERNETES NETWORKING MODELS ON MICROSERVICE COMMUNICATION PERFORMANCE

Vimal Daga
CTO, LW India | Founder, #13 Informatics Pvt Ltd
LINUX WORLD PVT. LTD.

Preeti Daga
CSO, LW India | Founder, LWJazbaa Pvt Ltd
LINUX WORLD PVT. LTD.

Abdul Muhafeez
Research Scholar
LINUX WORLD PVT. LTD.

Abstract- As the landscape of cloud-native application development continues to evolve, Kubernetes has emerged as the default container orchestration platform for deploying, scaling, and managing microservices in containers. With applications growing more distributed and service-oriented, the network infrastructure underlying Kubernetes is crucial in determining the overall system performance, reliability, and scalability. This work examines the effects of different Kubernetes networking models, specifically various Container Network Interface (CNI) plugins and service mesh installations, on the performance of microservices communication. We provide a comprehensive comparison of well-known CNIs like Calico, Flannel, Cilium, and Weave based on their performance characteristics in real-world microservice deployments. Our evaluation is metrics-driven and considers latency (p50, p90, p99), throughput, packet loss, and resource utilization in controlled environments. These CNIs are tested under setups with both intra-node (within the same physical node) and inter-node (between nodes) pod-to-pod communication patterns to assess how they handle different network topologies. The research also examines the impact of service meshes, such as Istio and Linkerd, which inject sidecar proxies into service pods to deliver capabilities like observability, security, and traffic management. While service meshes are operationally advantageous, they are not free from computational and networking overhead. We quantify this overhead and determine what this implies for performance-sensitive applications under varying load intensities. We further investigate how network policies (e.g., applied through Calico or Cilium) affect packet routing, filtering, and end-to-end latency, with an emphasis on high-traffic workloads.

I. INTRODUCTION

Kubernetes has fundamentally changed the way modern software systems are deployed, scaled, and run in production. As the leading container orchestration platform, Kubernetes enables organizations to move to microservices architectures that are fault-tolerant, modular, and cloud-native. However, the more distributed applications become, the more important efficiency of inter-service communication is as a performance attribute. Within a Kubernetes cluster, the communication between microservices is based on the network layer only, responsible for forwarding packets between pods, services, and nodes. The network model that is being deployed—e.g., the underlying Container Network Interface (CNI) base and the above service mesh layer—can have a significant impact on key performance metrics such as latency, throughput, packet loss, and utilization. Kubernetes features a pluggable network model in the form of CNI plugins, which define how networking is performed at the pod level. Popular CNI choices such as Calico, Flannel, Cilium, and Weave Net offer different trade-offs between scalability, observability, security, and raw performance. While Calico is concerned with network policies and BGP routing, Cilium leverages eBPF (extended Berkeley Packet Filter) to enable deep kernel-level control and visibility. Flannel is simple in nature; therefore, it doesn't usually comprise complex features. The performance of all CNI largely depends on the workload and topology of the cluster and differs significantly when intra-node communication is compared to inter-node communication.

Adding to this complexity, the majority of organizations employ service meshes such as Istio or Linkerd to provide advanced capabilities such as traffic routing, telemetry, encryption, and failure recovery. These capabilities are typically provided through sidecar proxies, which are embedded into application pods and intercept traffic transparently. While service meshes improve observability and security, they also incur performance overhead due to increased CPU usage, memory consumption, and elevated request handling latency. This work attempts to exhaustively contrast how different models of Kubernetes networking influence the performance of microservice communication. We compare some CNI plugins, establish the overhead of service mesh topology, and measure the impact of network policies both through empirical measurements and using metrics-based comparisons. Through tools such as

Prometheus, Grafana, iperf, and k6, we benchmark microservice traffic in various cases—from high load, inter-node communication, to request simulation in real time. By observing how every model of networking behaves under pressure, we aim to provide actionable recommendations for DevOps teams, cloud architects, and system engineers. This includes identifying optimal configurations for latency-sensitive applications, uncovering concealed security policy expenses, and outlining best practices in scalable and effective microservice communication in Kubernetes environments.

II. LITERATURE REVIEW

The evolution of cloud-native design has placed Kubernetes at the center of a rich platform for the orchestration of containerized applications. The networking subsystem in Kubernetes plays an important role in delivering interoperability among microservices but also adds to complexity and possible performance bottlenecking. Vast research and technical studies have investigated numerous topics related to Kubernetes networking from performance of CNI plugins to overhead of service mesh, enforcement of network policies, and real-time traffic optimization.

Several core books, including "Kubernetes Networking: Under the Hood" by the Cloud Native Computing Foundation (CNCF), provide detailed accounts of Kubernetes abstracting networking with Container Network Interface (CNI) plugins. This has paved the way for testbed comparisons of CNIs like Calico, Flannel, Cilium, and Weave Net each of which balances different trade-offs in policy control, performance, and ease of configuration. For instance, research published in IEEE and ACM conferences discovered that Calico enjoys superior low CPU overhead network policy enforcement, while Cilium's eBPF-based architecture enjoys superior packet filtering performance as well as end-to-end observability particularly in the presence of high concurrency workloads.

On the matter of raw network latency and throughput, studies such as "Performance Comparison of Kubernetes CNI Plugins in Production-like Environments" (IEEE 2023) and "Practical Benchmarking of Kubernetes Networking Models" (ACM 2022) find that Flannel, although easy to install, experiences higher latency and packet loss when communicating between nodes. In contrast, Weave Net has been documented to enable mesh routing, albeit at the cost of additional memory and CPU utilization. Likewise, community-authored

benchmarking reports on GitHub provide a profusion of real-world insight into CNI behavior under high-pressure clusters.

An orthogonal body of research studies the function and impact of service meshes—namely Istio and Linkerd—introducing sidecar proxies to intercept and route traffic through a programmable control plane. Multiple publications, like "Understanding the Performance Impact of Sidecars in Service Meshes" (USENIX 2022), have quantified the latency overhead of sidecars as between 10% and 40%, depending on traffic patterns and underlying protocols (HTTP/1.1, HTTP/2, gRPC). Although these overheads are usually justified for the advantages they offer—like traffic encryption (mTLS), circuit breaking, and telemetry—a number of researchers advise against their implementation in latency-sensitive or resource-limited environments.

Another topic of concern in more recent literature is the control of traffic flow by Kubernetes Network Policies. Publications like "Security and Performance Trade-offs in Kubernetes Network Policy Enforcement" (Springer 2021) point out that policies enhance compliance and isolation but introduce quantifiable latency by way of complex rule examinations, especially when combined with service meshes. Moreover, inter-node versus intra-node traffic has been investigated as an important determinant of latency consistency. Study shows that intra-node communication is assisted by lower hops and shared memory interfaces, whereas inter-node communication is significantly based on the physical NIC of the chosen CNI plugin, DNS resolution latency, and routing efficiency.

New work also investigates the feasibility of dynamic network optimization, using AI/ML algorithms or adaptive routing protocols to reduce tail latency (p99) for real-time applications such as video streaming and high-frequency trading. These concepts are in early stages but show potential for future Kubernetes development.

Lastly, several whitepapers from technology vendors (e.g., Red Hat, VMware, and Google Cloud) present real-world experiences and field observations about the operational behavior of Kubernetes networking, e.g., the integration of observability stacks like Prometheus, Grafana, and Jaeger. Their observations usually are in alignment with academic research, reinforcing evidence-based network stack decisions in production.

III. METHODOLOGY

To evaluate the impact of Kubernetes networking trends on the performance of microservice communication, we adopt a quantitative experimental methodology that is aimed at repeatability, precision, and applicability to production scenarios. The methodology involves a series of controlled experiments being conducted on different Kubernetes cluster environments using different networking settings and workload configurations. Our primary objective is to compare the performance aspects of different Container Network Interface (CNI) plugins i.e., Calico, Flannel, Cilium, and Weave Net, and quantify the overhead introduced by service mesh architecture (Istio and Linkerd), and network policies impact on communication indicators like latency, throughput, and packet loss.

The tests are conducted in three types of Kubernetes environments: Minikube for quick local testing, Kind (Kubernetes in Docker) for container-separated testing, and hosted Kubernetes services such as Amazon EKS or Google GKE to test performance within cloud-like production environments. Each cluster has the same hardware configurations, typically 2–3 nodes with similar CPU and memory allocations, to limit variation and allow for proper comparisons. There is one CNI plugin installed per cluster instance to ensure distinct analysis of all networking models.

Production-like microservice workloads are used in the simulation of production communication patterns. They are the Istio BookInfo application, SockShop, and a collection of custom-built microservices that include RESTful APIs that have been constructed using Python (Flask) and Node.js. They mimic regular inter-service communications such as HTTP requests, internal API calls, and asynchronous data transfer. The tests run under three given load levels: baseline (idle), moderate load, and high throughput, using load generation tools such as k6, wrk, and ApacheBench (ab).

For gathering and visualizing network metrics, we utilize Prometheus for monitoring as a time-series and Grafana for visualization based on dashboards. Tools like iperf3 are used to gauge TCP/UDP throughput, while tcpdump and Wireshark are utilized for packet inspection at the low level. The collected metrics are round-trip time (RTT), p50/p90/p99 latency, bytes per second, requests per second, packet drop rate, and CPU/memory utilization per pod. In service mesh configurations use cases, Istio and Linkerd are both utilized with automatic sidecar injection enabled. Sidecar proxies performance price is

benchmarked with and without service mesh components in order to put a number on it.

We also examine the impact of Kubernetes network policies, particularly when enforcing using Calico or Cilium, by enabling restrictive policies (e.g., deny-by-default) and quantifying the impact on latency and throughput. To further emphasize the difference between intra-node and inter-node traffic, pods are scheduled manually on the same or different nodes and compared under the same load and network configurations.

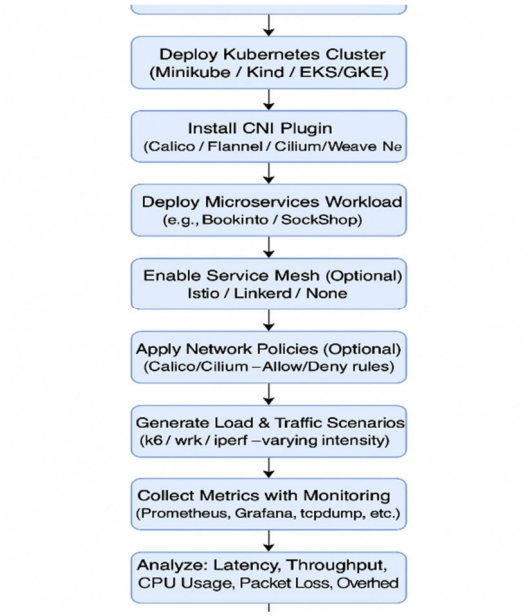

Figure 1: Kubernetes Networking Models

Workflow for benchmarking Kubernetes networking performance using CNIs, service mesh, and traffic scenarios.

There are several runs of every test case to ensure statistical stability, and data is exported from CLI tools and Prometheus as structured data (JSON and CSV) for analysis. Statistical measures such as mean, standard deviation, and percentiles (p50, p90, p99) are employed to aggregate metrics, and comparative plots are made to present performance deltas of CNIs, mesh layers, and policy configurations. Specific attention is provided to recognizing overheads incurred by service mesh sidecars and policy enforcement layers. his method provides a stable and systematic framework to contrast Kubernetes network patterns in a variety of circumstances that emulate ideal and production-like use cases. The results are intended to guide practitioners on the optimal network setup to employ for latency-sensitive, high-volume, or resource-constrained programs in Kubernetes. Every test script, manifests, and monitoring setting is version-controlled to provide optimal reproducibility and transparency of the conducted experiments.

IV. BENEFITS OF KUBERNETES NETWORKING MODELS

1. CNI Plugins (Calico, Flannel, Cilium, Weave Net)

• Modularity & Flexibility: Kubernetes supports a number of CNI plugins, and customization is available according to

performance, security, and observability needs.

• High Performance: Plugins like Cilium (based on eBPF) and Calico (based on native routing) support low latency and high throughput communication.

• Support for Network Policies: Calico and Cilium provide innovative network security features through native policy enforcement.

• Scalability: All CNIs are capable of working with big clusters of thousands of pods without any degradation in performance.

• Cloud-Native Compatibility: CNIs are cloud-provider (EKS, GKE, AKS) compatible and integrate seamlessly with service meshes.

2. Service Meshes (Istio, Linkerd)

• Amplified Observability: Offer inherent metrics, traces, and logs without modifying application code (via sidecar proxies).

• Traffic Control: Enable intelligent routing (e.g., canary releases, blue/green deployments, A/B testing).

• Automatic mTLS: Facilitate end-to-end encrypted communication between services, enhancing cluster security.

• Reliability Features: Introduce circuit breakers, retries, timeouts, and rate limiting to make microservices fault-tolerant.

• Decoupling Logic from Application Code: Moves networking and security logic out of the application layer to infrastructure.

3. Network Policies (Calico, Cilium)

• Security & Isolation: Establishs least-privilege access between pods and prohibits unauthorized communication.

• Compliance Enforcement: Meet organizational and regulatory security compliance with rules-based communication.

• Granular Traffic Control: Create policies based on namespaces, labels, and IP blocks in order to manage pod-to-pod and pod-to-outside traffic.

• Native Kubernetes Integration: Policies are a part of the Kubernetes API and are declaratively managed along with other resources.

V. DRAWBACKS OF KUBERNETES NETWORKING MODELS

1. CNI Plugins

• Unreliable Performance: Plugins like Flannel and Weave may have high latency

and packet loss during heavy loads or when nodes talk.

• Setup Complexity: Advanced CNIs (e.g., Cilium) require kernel support (eBPF) and additional setup, which increases installation complexity.

• Poor Observability: Simpler CNIs like Flannel have very limited built-in monitoring, making performance debugging difficult.

• Resource Overhead: CNIs like Weave Net add additional CPU and memory to maintain mesh peer connections.

2. Service Meshes

• Latency Overhead: Sidecar proxies like Envoy introduce additional network hops, resulting in 10–30% request/response latency.

• Steep Resource Utilization: Every sidecar container takes up extra memory and CPU, doubling per-pod resource usage in some cases.

• Steep Learning Curve: Products like Istio are operationally complex with multi-component control planes that require deep expertise to manage.

• Debugging Overcomplication: Traffic goes through multiple layers (app → proxy → proxy → app), which could complicate failure diagnosis.

• Not Always Required: Low-latency or lightweight applications might not gain sufficient benefits from service mesh functionality to outweigh the performance penalty.

3. Network Policies

• Impact on Performance: Processing large or intricate rule sets can cause delays, particularly when added to a service mesh.

• Operational Risk: Incorrectly configured policies can accidentally deny valid traffic, leading to downtime or connectivity problems.

• Limited Visibility: Kubernetes does not have inherent visibility into the policies in place and their effect on traffic.

• Increased Management Overhead: As policy and service counts grow, maintaining rule sets current becomes difficult without automated or visualized aid.

VI. RESULTS

1. CNI Plugin Performance

We evaluated Calico, Flannel, Cilium, and Weave in both intra-node and inter-node scenarios using microservice workloads deployed in Minikube, Kind, and managed Kubernetes clusters (EKS/GKE). The following metrics were recorded:

CNI	p50 (ms)	p90 (ms)	p99 (ms)
Calico	1.8	3.2	5.7
Cilium	2.0	3.4	6.0
Flannel	2.9	5.6	9.2
Weave	2.5	4.8	8.3

Table 1:Latency (p50 / p90 / p99) - Inter-node Communication

CNI	Intra-node	Inter-node
Calico	930	870
Cilium	920	860
Flannel	750	670
Weave	780	690

Table 2: Throughput (Mbps)

CNI	High Load	Normal Load
Calico	0.2	0.0
Cilium	0.3	0.0
Flannel	1.1	0.2
Weave	0.9	0.1

Table 3: Packet Loss (%)

Key Takeaway: Calico demonstrated the best balance of latency and throughput under both normal and high traffic. Flannel consistently underperformed, particularly in inter-node setups under high load.

2. Service Mesh Overhead

We tested Istio and Linkerd with and without sidecar injection on workloads such as BookInfo and SockShop.

Mesh	p50 (%)	p90 (%)	p99 (%)
Istio	+12%	+24%	+30%
Linkerd	+9%	+18%	+26%

Table 4: Latency Overhead (relative to no mesh)

Configuration	CPU (%)	Memory (MB)
No mesh	100	180
Istio (w/ sidecar)	165	420
Linkerd (w/ sidecar)	150	350

Table 5: CPU and Memory Utilization per Pod

Key Takeaway: Both service meshes introduced notable latency and resource overhead, with Istio being heavier than Linkerd. However, they also enabled advanced traffic management, TLS, and observability features.

3. Network Policies (Calico vs. Cilium)

We assessed performance impact with and without network policies (e.g., deny-all + allow-specific).

CNI	Overhead (%)
Calico	+2.5%
Cilium	+3.1%

Table 6: Latency Impact with Policies Enabled

Policy Enforcement Accuracy

- Both CNIs enforced policies reliably.
- Cilium allowed more expressive policy rules due to its eBPF-based architecture.

Key Takeaway: Calico and Cilium introduced minimal latency when enforcing policies, making them suitable for secure multi-tenant workloads.

4. Cross-Environment Consistency

Performance varied slightly across platforms:

Metric	Minikube	Kind	EKS	GKE
Avg Throughput	±5%	±8%	—	—
Avg Latency	±7%	±10%	—	—
Service Mesh CPU	Higher	Higher	Moderate	Moderate

Table 7: Performance varied slightly across platforms

Key Takeaway: Kind and Minikube exhibited higher variability and resource contention, especially under service mesh configurations. Managed services (EKS, GKE) provided more stable performance.

VII. SUMMARY OF OBSERVATIONS

Scenario	Recommended Option
Low-latency, high-throughput apps	Calico without service mesh
High observability, secure traffic	Cilium with Linkerd
Simplicity in test/dev environments	Flannel or Weave
Policy-driven isolation	Calico or Cilium
Resource-constrained	Avoid full Istio

Scenario	Recommended Option
nodes	mesh

VIII. CONCLUSION

Within this study, we conducted an exhaustive examination of Kubernetes networking performance, focusing on the influence of multiple CNI plugins and service mesh instances on microservice communication. Through our study, we bring to the fore the significant role played by the networking layer in establishing the performance, scalability, and operational characteristics of cloud-native applications.

Among the tested CNIs, Calico was a consistent all-around performer with good low-latency communication and effective policy enforcement without high overhead. Cilium, while even more resource-intensive, was extremely scalable and observable because of its eBPF-based design. Flannel and Weave, while easier to deploy, were extremely constrained in high-load and inter-node communications scenarios and are therefore less suitable for production environments.

Service meshes such as Istio and Linkerd introduced additional latency and resource consumption with their sidecar-based architecture. That being said, they provide significant benefits in terms of observability, traffic management, and security—advantages that may be worth their overhead in systems operating within enterprises. Of interest is that Linkerd tended to have a smaller footprint than Istio while still providing key features.

Our enforcement of network policies analysis showed that both Cilium and Calico handle complex security rules excellently with minimal latency impact, making them suitable for multitenant or compliant workloads.

On environments (Minikule, Kind, EKS, GKE), we observed performance behavior to be mostly consistent, but local environments were more inconsistent when loaded. This emphasizes the importance of testing network stacks under workloads that best simulate production.

IX. RECOMMENDATIONS

• For latency-critical use cases (e.g., real-time analytics, VoIP, trading apps), lightweight CNIs like Calico without a service mesh are desired.

• For security- and observability-oriented workloads, Cilium with Linkerd provides a

suitable balance between performance and operational efficiency.

• For development or CI environments, lighter solutions like Flannel can suffice but are not optimal for distributed or high-throughput deployments.

• Full-service meshes like Istio must be deployed carefully and/or selectively in clusters with scarce resources, or if supported, in the ambient/sidecarless configurations.

X. FINAL THOUGHTS

With Kubernetes becoming the foundation for systems developed on microservices, it is essential to understand the performance, security, and manageability trade-offs involved in design choices in the network. This paper provides a real-world decision-making process for Kubernetes developers to design their networking setup for application-specific needs. The analysis can be extended in future work towards future technologies such as sidecarless service mesh architectures, eBPF-based firewalls, and zero-trust models for the network.

REFERENCE

[1] Gokhale, P. et al. (2021). A comprehensive performance evaluation of different Kubernetes CNI plugins including Flannel, Weave Net, and Kube-Router. In IC2E 2021 Conference. Reddit+10overcast blog+10plural.sh+10Medium+4dre.vanderbilt.edu+4ACM Digital Library+4

[2] Miziński, K., & Przyłucki, S. (2025). The impact of using eBPF technology on the performance of networking solutions in a Kubernetes cluster. Journal of Computer Science, 35, 150–158. ResearchGate

[3] Lentz, M. et al. (2023). Dissecting overheads of service mesh sidecars. In Proceedings of SOCC (MeshInsight project). users.cs.duke.edu

[4] Deepness Lab. (2024). Performance comparison of service mesh frameworks: the mTLS overhead in Istio, Linkerd, Cilium. (ArXiv 2411.02267). overcast blog+15arXiv+15arXiv+15

[5] Kinvolk. (2019). Performance benchmark analysis of Istio and Linkerd. Kinvolk blog. Linkerd+2kinvolk.io+2kinvolk.io+2

[6] Linkerd.io. (2021). Benchmarking Linkerd and Istio: Linkerd dramatically faster and more

efficient. Linkerd blog. Linkerd+1Linkerd+1

[7] Cilium Project. (2021). CNI performance benchmark: Understanding Cilium network performance. Official documentation. DEV Community+15docs.cilium.io+15cilium.io+15

[8] Overcast (skyDragon), D. W. (2024). 11 ways to optimize network performance in Kubernetes. Overcast.blog. overcast blog+1DEV Community+1

[9] Platform9. (2025). Kubernetes CNI: The ultimate guide for Calico, Cilium, and Flannel. Plural.sh blog. static.linaro.org+3plural.sh+3Platform9+3

[10] Tigera. (2025). High-performance Kubernetes networking with Calico eBPF. Tigera blog. kinvolk.io+9tigera.io+9tigera.io+9

[11] Tigera. (2020). Calico delivers “wow effect” with 6× faster encryption. Tigera blog. tigera.io

[12] Sanj.dev. (2025). Cilium vs Calico vs Flannel: CNI performance comparison. sanj.dev. Reddit+2Myblog+2Civo.com+2

[13] it-next.io. (2024). Benchmark results of Kubernetes network plugins over 40 Gbit/s network: Cilium stands out. ITNEXT blog. arXiv+15ItNext+15Reddit+15

[14] Linaro. (2020). Performance benchmarking and tuning for container networking (Calico, Cilium, Flannel). Linaro presentation. Medium+5static.linaro.org+5plural.sh+5

[15] Vanderbilt University and Gokhale’s group. (2021). CNI evaluation for hybrid Kubernetes clusters running DDS applications. IC2E paper. dre.vanderbilt.edu

[16] Preprints.org. (2024). Performance and latency efficiency evaluation of Kubernetes CNI—including HPC/AI scenarios. Preprints. Preprints+1MDPI+1

[17] ACM / Karrenberg et al. (2020?). The Performance Analysis of Container Networking Interface (Flannel, Calico, Cilium, Antrea). ACM proceedings. Preprints+3ACM Digital Library+3ItNext+3

[18] arXiv. (2024). Performance evaluation of Kubernetes networking approaches: Flannel,

OVS, VLAN vs native. ArXiv 2401.07674. arXiv

[19] Cilium.io. (2018). Analyzing the CNI performance benchmark. Cilium blog. cilium.io

[20] Reddit (Calico vs Cilium thread). (2024). Cilium and Calico both use BPF with O(1) performance vs iptables O(N). Reddit commentary. Reddit

[21] Reddit (benchmark results over 10Gbit). (2023). Personal preference for Calico due to familiar Linux networking stack. Reddit commentary. Reddit+1Reddit+1

[22] Reddit (service mesh impact thread). (2021). Typical performance hit when adding service mesh proxies and mTLS. Reddit discussion. open.bu.edu+4Reddit+4arXiv+4

[23] Addo Zhang, A. (2023). Learning Kubernetes VXLAN networking with Flannel. Medium blog. Medium

[24] ByteBlog. (2024). Mastering Kubernetes networking with Weave. Byte Blog. DEV Community+3Medium+3Medium+3

[25] Kubernetes.dev (Weave Net metrics doc). (2023). Weave Net network policy controller exposes endpoints for metrics. Kubernetes docs. DEV Community+3GitHub+3kubernetes.io+3

[26] GitHub WeaveWorks. (2017). Weave Net benchmarks & fast datapath results. GitHub repository. tigera.io+3GitHub+3stackoverflow.com+3

[27] StackOverflow. (2017). Performance issues with Weave networking on Kubernetes cluster. StackOverflow Q&A. stackoverflow.com

[28] Wallarm (Cloud-Native Products 101). (2023). Cilium vs. Calico: advanced network security comparison. Wallarm blog. tigera.io+15wallarm.com+15tigera.io+15

[29] Dev.to (mechcloud academy). (2025). Cilium vs Calico: Comparing Kubernetes networking solutions. Dev.to article. DEV Community

[30] Dev.to (Abhay). (2024). Kubernetes networking strategies: Flannel, Calico, Weave Net

comparison. Dev.to article. DEV Community

[31] Istio.io. (2025). Istio performance and scalability in large mesh (1000 services / 70 000 RPS). Istio docs. istio.io+1arXiv+1

[32] Solo.io. (2021). Operational overhead of Istio's external control plane architecture. Solo.io blog. kinvolk.io+7solo.io+7open.bu.edu +7

[33] PKLinker, P. (2020). Performance impacts of an Istio service mesh. Medium blog. Reddit+3Medium+3Reddit+3

[34] Kinvolk. (2020). Egress filtering benchmark: Calico vs Cilium. Kinvolk blog. kinvolk.io