The Service Mesh

Solving Microservice Chaos

(And When You Actually Need One)

Mofesola Babalola
Staff Reliability Engineer

Hannah Olukoye
Engineering Manager

THE LINUX FOUNDATION
OPEN SOURCE SUMMIT
Embedded Linux Conference
NORTH AMERICA

Agenda

What we'll cover today

01 The Microservice Tax

02 What Is a Service Mesh?

03 Pillar 1 — Reliability

04 Pillar 2 — Observability

05 Pillar 3 — Security

06 Real-World Benchmarks

The Microservice Tax

The hidden complexity nobody warned you about

The Promise vs. The Reality

Why microservices create a network engineering problem

✓ The Promise

- → Deploy services independently
- → Scale per service
- → Pick any language/framework
- → Small, focused teams
- → Faster release cycles

✗ The Reality

- ✗ Hand-coding retry logic everywhere
- ✗ Inconsistent timeout configs
- ✗ No uniform metrics or tracing
- ✗ mTLS across 50 polyglot services?!
- ✗ Became part-time network engineers

"We're all part-time network engineers now."

The Three Microservice Taxes

Boilerplate that's holding every team back

⚙️ Reliability Tax

Retry logic, timeouts, circuit breakers — coded by hand, in every service, in every language. Inconsistently.

📊 Observability Tax

50 services, 50 different logging formats. No unified tracing. The 'golden signals' aren't golden — they barely exist.

🔒 Security Tax

Enforcing mTLS across polyglot services requires per-language TLS libs, certs, rotation logic. One misconfig = plaintext traffic.

What Is a Service Mesh?

Introducing Istio

What Is a Service Mesh?

An infrastructure layer — not application code

Data Plane Proxy

Sidecar proxies are deployed with each instance of a service that needs to communicate with other services.

Control Plane

Istio — distributes config, certs, and policy to every proxy in real time.

Sidecar vs. Ambient Mode

Two paths to the mesh

Sidecar Mode (Traditional)

- ✗ Envoy proxy injected into every pod
- ✗ ~0.20 vCPU + 60 MB per pod
- ✗ 1,000 pods = 200 vCPUs idle
- ✗ Security patches force full rollout
- ✓ Rich L7 observability out-of-the-box
- ✓ mTLS keys inside the pod (RCE risk)

Ambient Mode (Istio 2026)

- ✓ ztunnel: shared node-level agent (L4)
- ✓ Waypoint proxy: opt-in per service (L7)
- ✓ ~0.06 vCPU + 12 MB per node
- ✓ Patch mesh without touching apps
- ✓ mTLS keys isolated at node boundary
- → L7 tracing requires Waypoint deployment

Pillar 1
Reliability

Automatic retries, timeouts & circuit breakers — for free

Reliability: No More Hand-Coded Resilience

The mesh handles this — in every language, consistently

Automatic Retries

BEFORE (your code)

```
if err != nil { // retry? how many times?
// exponential backoff?
// don't retry POST? }
```

→

AFTER (mesh YAML)

```
VirtualService:
  retries:
    attempts: 3
    retryOn: 5xx,reset
```

Timeouts

BEFORE (your code)

```
httpClient.Timeout = ?
// Set per-client, inconsistently
// across 50 services
```

→

AFTER (mesh YAML)

```
VirtualService:
  timeout: 5s
// Enforced mesh-wide
```

Circuit Breaker

BEFORE (your code)

```
// Implement Hystrix/Resilience4j
// per service, per language
// Different behavior everywhere
```

→

AFTER (mesh YAML)

```
DestinationRule:
  outlierDetection:
    consecutiveErrors: 5
    interval: 30s
```

Pillar 2
Observability

Uniform golden signals for every service — zero code changes

Observability: The Golden Signals, Automated

📈 Traffic

Requests/sec per service, per route, per version. Automatic. No instrumentation needed.

2,000 RPS
benchmark baseline

🚨 Errors

5xx rates, connection resets, circuit-open events — all surfaced in Prometheus/Grafana.

p99.9 tail latency
25ms → 10ms

⏱️ Latency

p50 / p99 / p99.9 histograms per service. No need to add tracing libraries.

15× more stable
with ambient mode

Distributed Tracing: The Important Caveat

Ambient mode has an L7 observability blind spot — here's the fix

⚠️ The Gap (L4-only mode)

- ✗ ztunnel sees TCP — not HTTP headers
- ✗ No trace spans in L4-only mode
- ✗ Service graphs appear 'broken'
- ✗ Traditional sidecar: full L7 by default (MITM)

✓ The Fix: Waypoint Proxy (opt-in L7)

Pay only where you need it.

Pillar 3
Security

Automatic mTLS and fine-grained authorization policies

Security: mTLS Everywhere, No App Code Required

From per-language TLS libraries to a mesh-enforced security baseline

Automatic mTLS

✓ All service-to-service traffic encrypted by default

✓ Certificates auto-rotated — no manual management

✓ Ambient: keys at node boundary (not inside pod)

✓ Sidecar compromise ≠ identity theft

AuthorizationPolicy

✓ Declare WHO can call WHAT — in YAML

✓ Works across Go, Java, Python, Node.js equally

✓ Namespace, service-account, or request-level rules

✓ Deny-by-default enforcement in seconds

Zero-Downtime Patches

✓ Sidecar model: patch Envoy = rolling restart of EVERY pod

✓ Ambient model: patch ztunnel/Waypoint — app stays up

✓ Decoupled lifecycle = faster CVE remediation

✓ No more coordinating 100% pod rollouts with dev teams

Real-World Benchmarks

Data from a simulation

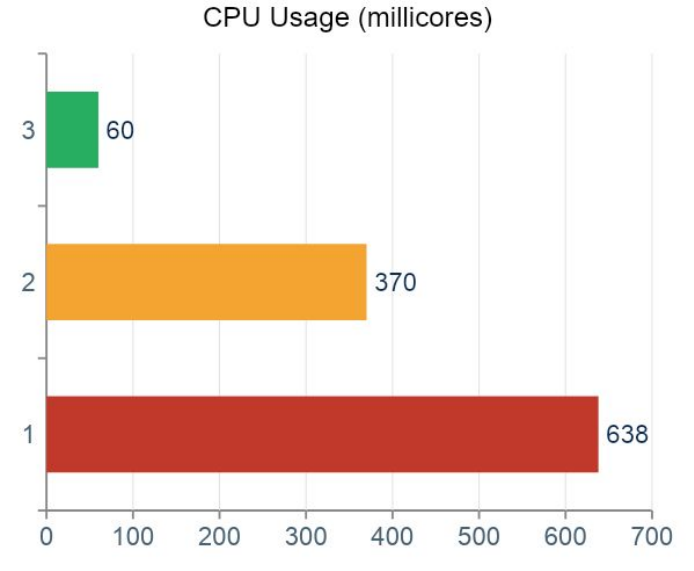
Resource Consumption: The 76% Memory Win

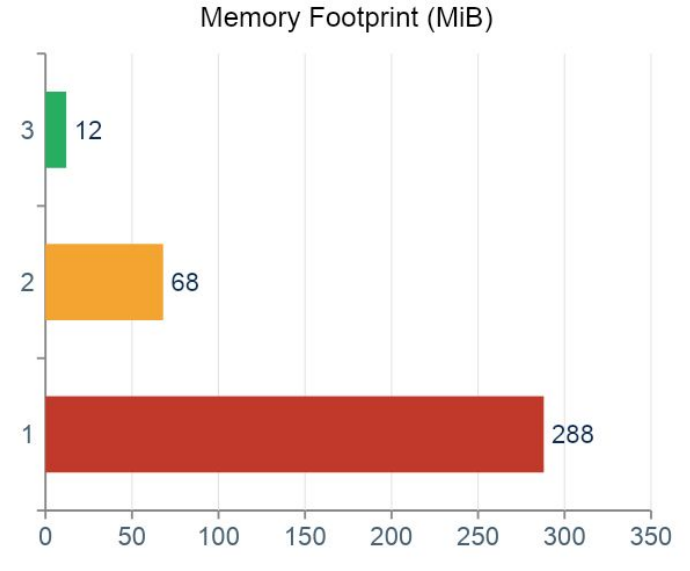
At 2,000 RPS — EKS m6i.xlarge, 1,000 pod cluster

Ambient L4 (ztunnel)

Ambient L7 (Waypoint)

Traditional Sidecar (Envoy)

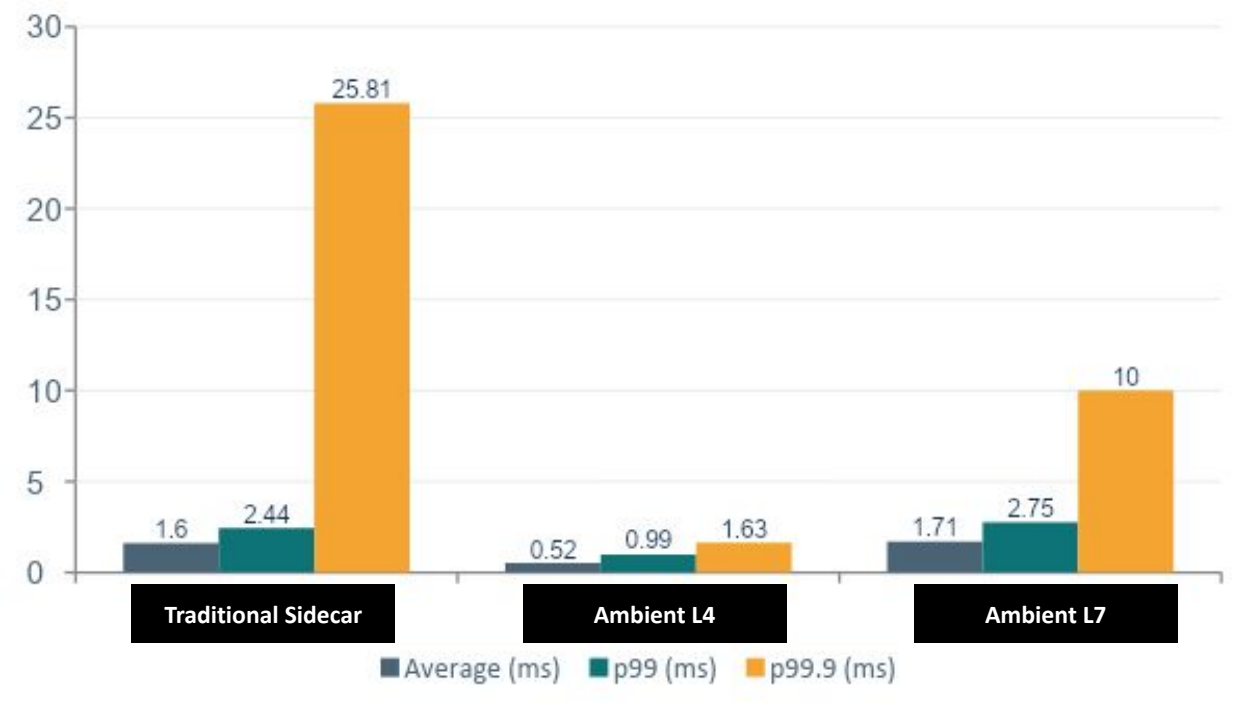




Ambient L4 (ztunnel) uses 91% less CPU and 96% less memory than a traditional sidecar.

Latency: The Real Story

Ambient beats sidecar even with the extra Waypoint hop



15×

more stable tail latency with Ambient L4

60%

reduction in tail latency spikes (L7)

91%

less CPU vs traditional sidecar

When Should You Deploy a Service Mesh?

An honest decision guide — not every team needs one day one

Question	If YES	If NO
Do you have > 5 services talking to each other?	✓ Mesh pays off immediately	✗ Probably overkill right now
Are you hand-coding retries/timeouts/circuit breakers in each service?	✓ Stop. Use a mesh.	✗ Still worth the observability gains
Do you need mTLS enforcement across polyglot services?	✓ Mesh is the cleanest path	✗ App-level TLS may suffice
Are you running on Kubernetes?	✓ Istio/Linkerd are production-ready	✗ Consul or Envoy directly
Is your team < 10 engineers?	✓ Start with Linkerd (simpler ops)	✗ Istio ambient scales better

Thank You

- https://mofesola.me/securing-the-path-from-edge-to-mesh-with-istio-fa7f4b2b258e
- https://www.linkedin.com/learning/kubernetes-service-mesh-with-istio-25332503/making-microservices-simple-with-istio

Mofesola Babalola
https://www.linkedin.com/in/mofesola/

Hannah Olukoye
https://www.linkedin.com/in/hannaholukoye/