CONF42 CLOUD 2026

Beyond Dashboards

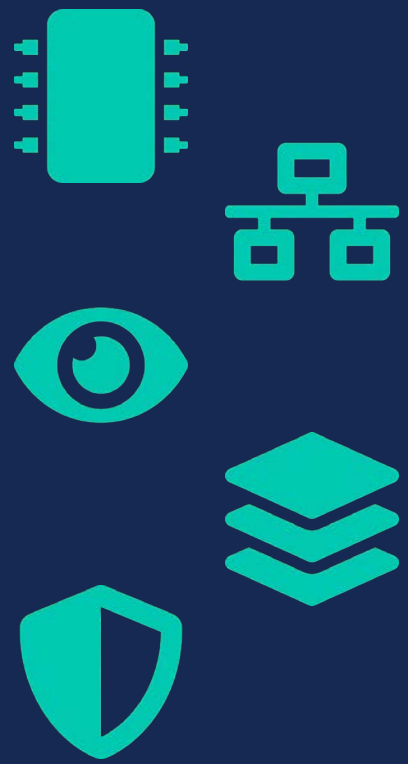
Deep-Dive Network Observability with eBPF

Udit Misra

Senior Software Engineer · Platform Engineering · Salesforce

Previously: Microsoft | Speaker: PlatformCon 2025

Open Source Contributor: Microsoft Retina



What We'll Cover Today

A no-fluff blueprint you can apply tomorrow

01 The Problem

Why dashboards show green while users scream

02 Monitoring vs Observability

The paradigm shift that changes everything

03 The Kubernetes Black Box

iptables limits & what we're flying blind on

04 eBPF for Visibility

Leveraging kernel hooks for network insight

05 How the Tools Work

Retina & Cilium/Hubble — architecture deep dives

06 Results & Comparison

Real cluster output, screenshots & feature matrix

The Problem: Everything Looks Green

...but something is clearly wrong

A Real-World Incident

🟢 CPU: Normal
🟢 Memory: Normal
🟢 Error Rate: 0%

❌ Users: Can't checkout
❌ Support tickets: spiking
❌ Revenue: dropping

Root cause: A NetworkPolicy dropped inter-service traffic silently. No alert fired. No log emitted.

The Visibility Gap

Traditional monitoring tells you WHEN a system breaks. It rarely tells you WHY—especially at the network layer.

Silent drops

NetworkPolicy blocks traffic with zero log output

DNS timeouts

Race conditions during pod startup, invisible to dashboards

Wrong backends

Misconfigured selectors route to wrong pods silently

Monitoring vs. Observability

The paradigm shift that unlocks real debugging power

MONITORING

Watching metrics you already decided matter

- Pre-defined thresholds & alerts
- Known failure modes only
- Aggregate counters (RPS, error %)
- Tells you something is wrong
- Works great for stable systems

→

OBSERVABILITY

Ability to ask any question — even ones you didn't know to ask

- ✓ Flow-level records (not just counters)
- ✓ Which pod made the request?
- ✓ What HTTP path caused the drop?
- ✓ Why was the packet dropped?
- ✓ DNS queries per namespace

"Observability is the ability to understand the internal state of a system from its external outputs — even for failures you never anticipated."

The Kubernetes Network Black Box

What traditional iptables-based approaches can't show you

iptables Limitations

Sequential rule scanning

O(n) lookup: latency grows with every NetworkPolicy rule added

Layer 3/4 only

IP addresses & ports only — no HTTP method, no DNS name, no gRPC path

No flow identity

Packets have IPs, not pod names. You need external tooling to correlate

Silent drops

When a rule drops a packet, there is no log, no trace, no breadcrumb

Static rule table

Rules compiled at policy-create time, not evaluated dynamically

Flying Blind On...

Why traffic between two pods stopped

Which DNS queries are timing out and why

Whether your NetworkPolicy is doing what you think

The real service dependency graph at runtime

Packet drop reasons at the kernel level

Budigiri et al. (IEEE 2021): latency degrades measurably as NetworkPolicy rule count scales

Leveraging eBPF for Network Visibility

From kernel hooks to actionable network intelligence — what you actually gain

What eBPF Unlocks

Per-Flow Records

- Every TCP connection logged individually
- Source pod, dest pod, port, verdict
- Not sampled — every single packet

DNS Visibility

- Which pods are querying which names
- DNS latency and failure reasons
- Query patterns per namespace

Drop Reason Telemetry

- Exact kernel reason a packet was dropped
- NetworkPolicy rule ID that caused it
- No more silent failures

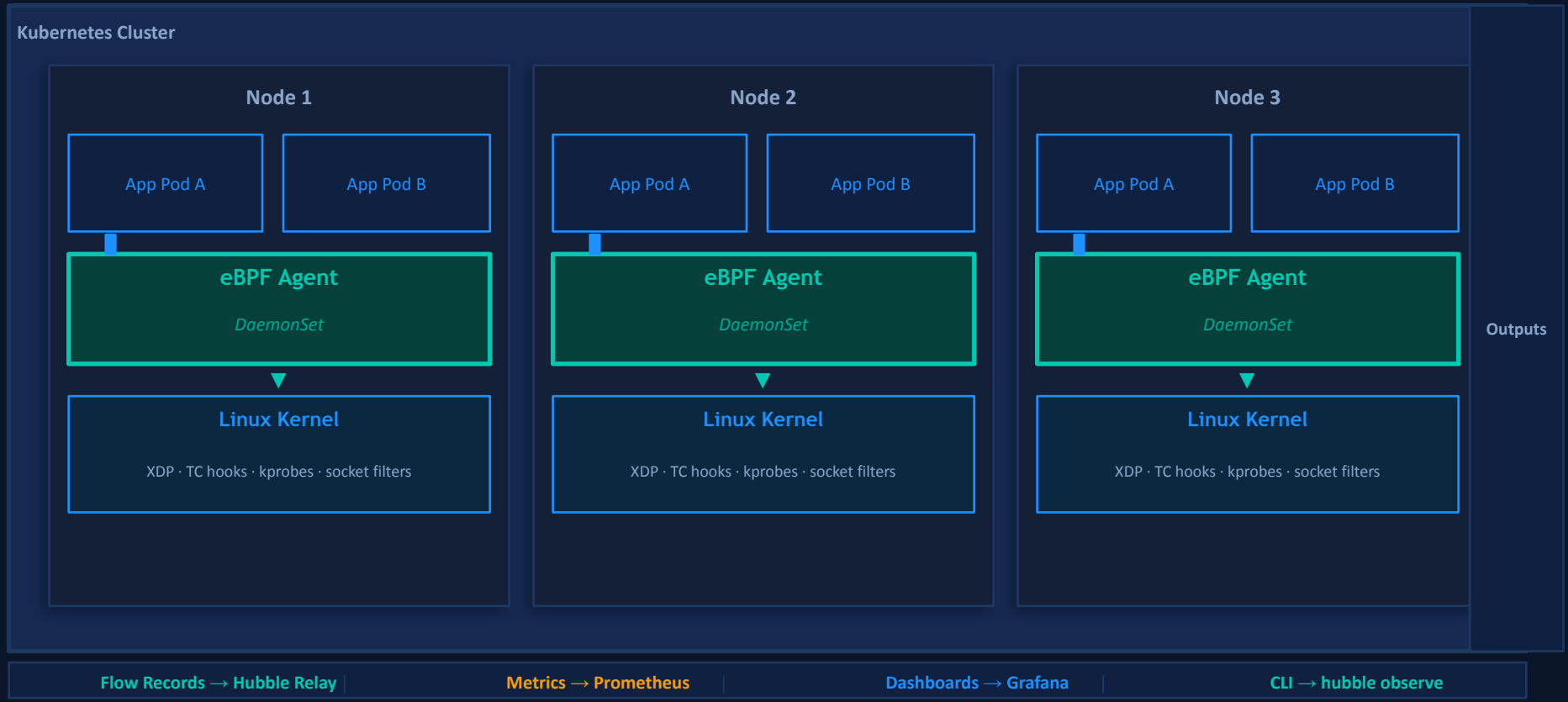
Questions You Can Now Answer

- Which pod initiated this connection?
- Why did this packet get dropped?
- Which services does pod X depend on?
- Is DNS the reason this call is slow?
- Is my NetworkPolicy actually working?
- What HTTP path is generating 5xx errors?
- Which pods talk to the internet?

All of this—without a sidecar proxy, without modifying application code, and without restarting pods.

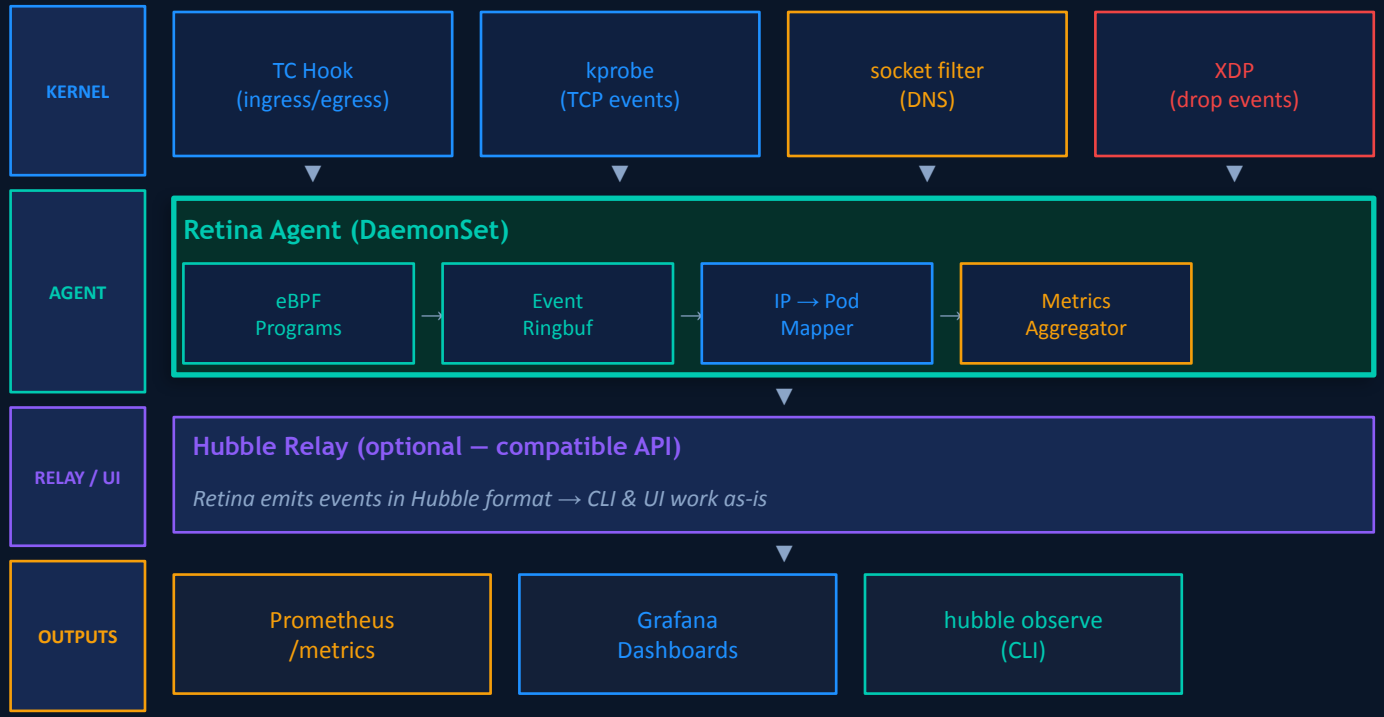
eBPF Agent as DaemonSet — Cluster Architecture

One agent per node, kernel-level hooks, zero application changes



How Retina Works — Architecture Deep-Dive

Microsoft Retina · Open-sourced 2024 · Observability-only, CNI-agnostic



Key Design Traits

CNI-agnostic
Works alongside any CNI

Observability-only
No policy enforcement

Host-network view
Sees traffic at NIC level

IP→Pod mapping
User-space correlation

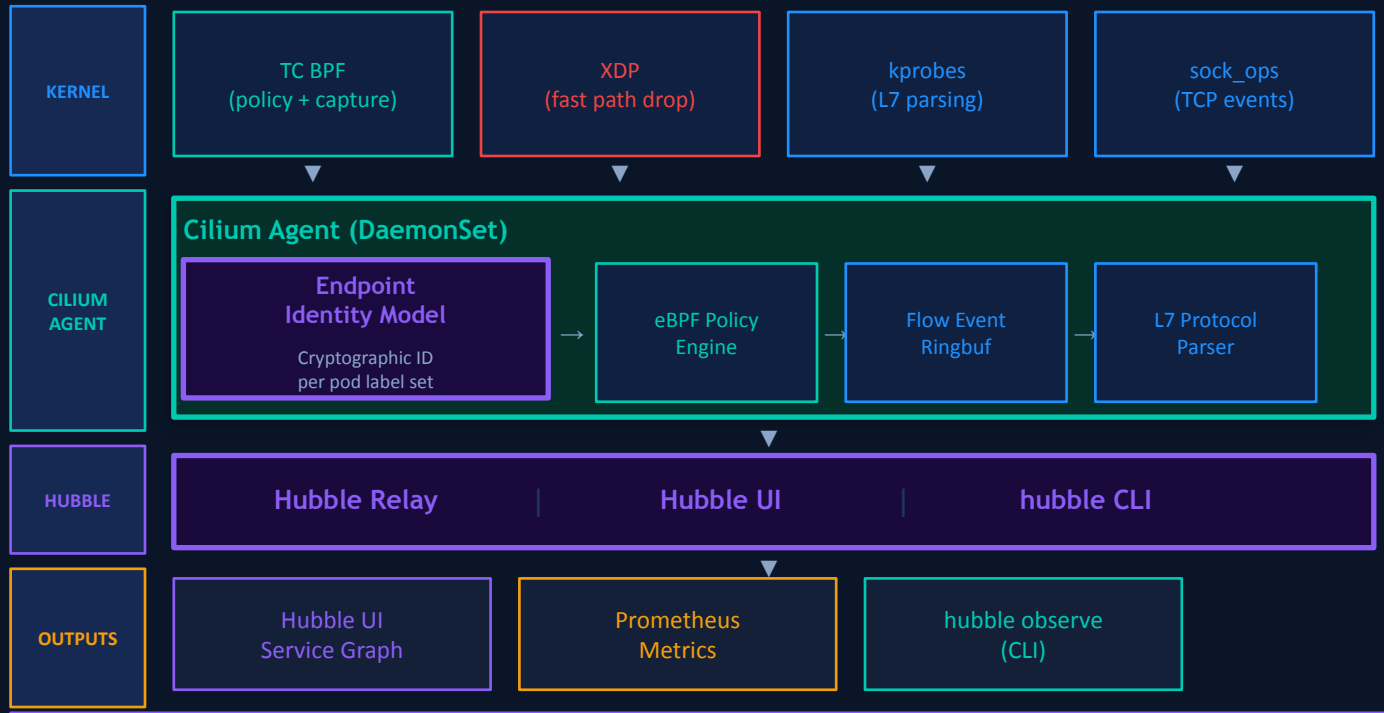
Prometheus-native
No extra scrape config

Hubble-compatible
Reuse familiar CLI/UI

⚠ Identity gap: Retina maps IPs to pods in user-space. Without Cilium's cryptographic endpoint model, destination identity requires explicit pod filter — UI shows raw IPs.

How Cilium + Hubble Works — Architecture Deep-Dive

CNCF Graduated · Chaining mode on EKS · Identity-aware eBPF data plane



Key Design Traits

Cryptographic identity

Every pod has a kernel-level ID

L7 visibility

HTTP method, path, gRPC

Policy enforcement

NetworkPolicy via eBPF (not iptables)

Chaining mode

VPC CNI keeps IPAM; Cilium adds eBPF

Service graph (UI)

Live dependency map

Both-side identity

Src + dst pod resolved automatically

★ Cilium's identity model is the key differentiator: every eBPF program in the kernel knows exactly which pod it is processing — automatically, without user-space correlation.

Architecture Comparison — Side by Side

Same eBPF foundation, fundamentally different design philosophies

Aspect	Retina	Cilium+Hubble
CNI Role	Observability only Plugs into any existing CNI	Full CNI or chaining mode Can replace or augment existing CNI
Identity Model	IP ⟶ Pod mapping User-space correlation	Cryptographic endpoint ID Kernel-level, per-pod label hash
L7 Visibility	Not supported Layer 3/4 only	HTTP, gRPC, DNS Full application-layer parsing
Policy Engine	None — observe only	eBPF-native NetworkPolicy Replaces iptables chains
Hubble Compat.	Relay mode Partial identity in UI	Native Full identity in UI + CLI
Memory / Node	~166 MiB (47% lighter)	~312 MiB Identity subsystem + relay overhead
CPU under load	Nearly flat (~0% increase)	+31% per node at 500 QPS

Results: Resource Overhead

Per-node CPU & memory — idle vs 500 QPS sustained load (AWS EKS benchmark)

47%
Less memory per node (Retina)

31%
CPU spike under load (Cilium)

~0%
CPU delta under load (Retina)

500
QPS sustained (5 min test)

Cilium — kubectl top (19 nodes under 500 QPS)

```
umisra@umisra-ltmf4q2 retina % kubectl top pods -n kube-system | grep cilium
cilium-6hsf7                                50m          335Mi
cilium-6jg9h                                12m          386Mi
cilium-7frh8                                40m          326Mi
cilium-9jk5z                                33m          320Mi
cilium-9lnhw                                52m          316Mi
cilium-bclgv                                47m          331Mi
cilium-fbq9p                                50m          318Mi
cilium-fzsdb                                41m          285Mi
cilium-kgldr                                20m          315Mi
cilium-kgrfn                                37m          324Mi
cilium-klggj                                33m          316Mi
cilium-nq4td                                23m          271Mi
cilium-nscrw                                24m          304Mi
cilium-nxszf                                29m          135Mi
cilium-operator-75fb867f9-5lvhp             9m           131Mi
cilium-operator-75fb867f9-f66p4             0m           32Mi
cilium-p85lc                                43m          285Mi
cilium-svghz                                19m          228Mi
cilium-tnr5g                                26m          304Mi
cilium-vtfrn                                29m          316Mi
cilium-xtnvv                                23m          305Mi
```

Retina — kubectl top (16 nodes under 500 QPS)

```
umisra@umisra-ltmf4q2 retina % kubectl top pods -n kube-system | grep retina
retina-agent-5mr7n                          6m           139Mi
retina-agent-6ksrg                          41m          179Mi
retina-agent-7c4vn                          11m          157Mi
retina-agent-7c64l                          11m          127Mi
retina-agent-8xkbx                          45m          181Mi
retina-agent-c99jm                          22m          158Mi
retina-agent-dk28r                          26m          158Mi
retina-agent-f2fvj                          10m          132Mi
retina-agent-hvntq                          40m          177Mi
retina-agent-j2nt7                          30m          161Mi
retina-agent-l8w7j                          39m          176Mi
retina-agent-ljnd6                          40m          179Mi
retina-agent-mtzsv                          48m          153Mi
retina-agent-qq6km                          44m          177Mi
retina-agent-r4zzx                          10m          125Mi
retina-agent-rdnzf                          23m          154Mi
retina-operator-758cfcbcb6-2b7b6            1m           47Mi
umisra@umisra-ltmf4q2 retina %
```

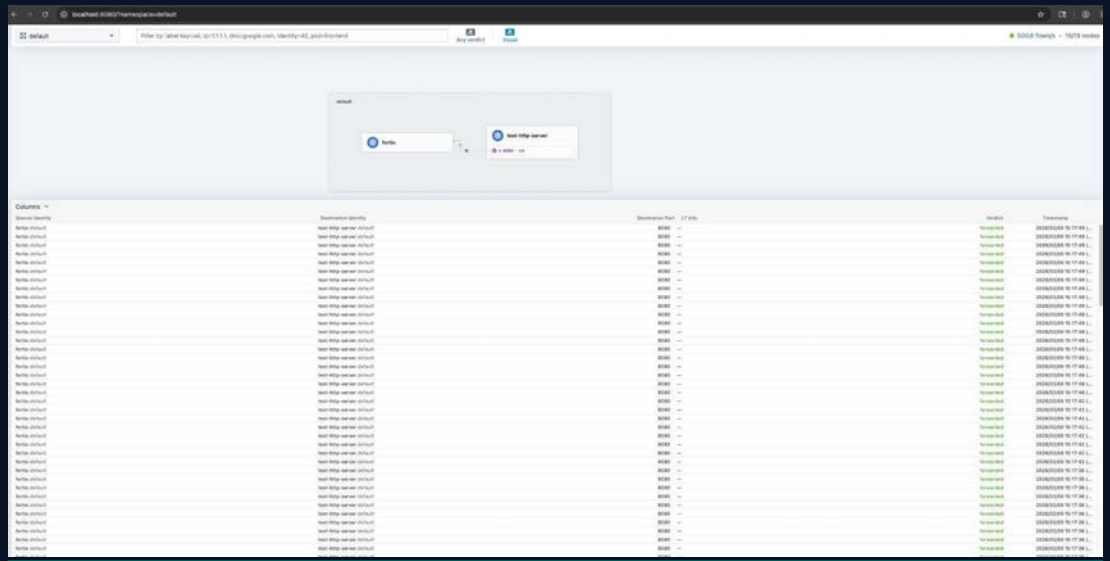
Cilium: 12m–52m CPU/pod · 131Mi–386Mi memory — wide variance driven by identity processing

Retina: 6m–48m CPU/pod · 125Mi–181Mi memory — tight, predictable band · operator adds only 47Mi

Results: What Visibility Actually Looks Like

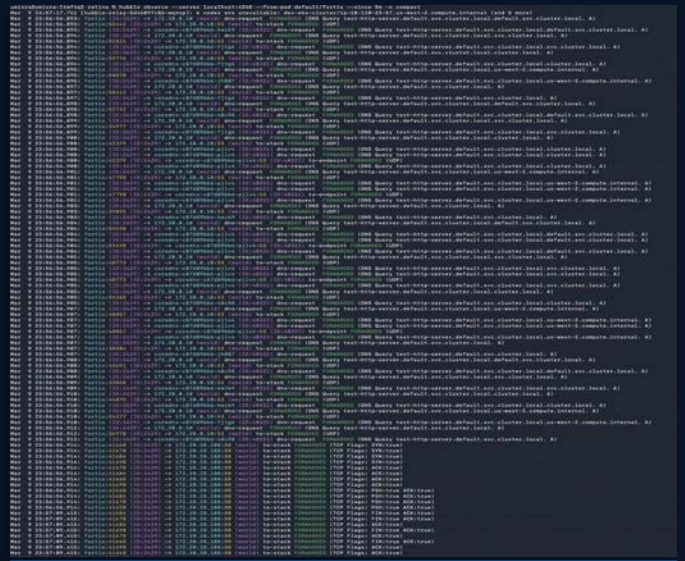
Real Hubble output from the benchmark cluster — Cilium vs Retina

Cilium / Hubble UI — Live Service Dependency Graph



- ✓ Source + dest pod fully resolved
- ✓ Port, protocol, verdict per flow

Retina — hubble observe --from-pod



- ✓ DNS + TCP flows visible
- ⚠ Dest shown as 'world' / raw IP

⚠ Identity gap: Retina observes at the host NIC level — without Cilium's kernel identity model, destination pods show as raw IPs in the UI unless an explicit --from-pod filter is used. Prometheus metrics remain fully accurate.

Results: Flow Visibility Feature Matrix

10 observability capabilities — what each tool gives you during an incident

Observability Feature	Retina	Cilium	Note
Flows visible in Hubble CLI	✓	✓	Table stakes — both ✓
Flows visible in Hubble UI	~	✓	Retina: partial identity
Source pod identity	~filter	✓	Retina: only with --from-pod filter
Destination pod identity	✗	✓	Retina shows 'world' / raw IP
HTTP method & path	✗	✓	Cilium only (L7 parser)
DNS query visibility	✓	✓	Retina via Prometheus
Packet drop reason	✓	✓	Retina via Prometheus
Live service dependency graph	✗	✓	Hubble UI — Cilium only
Prometheus metrics export	✓	✓	Both ✓ (different paths)
L7 gRPC / HTTP metadata	✗	✓	Cilium only

How to Choose: A Decision Framework

Real trade-offs — pick the right tool for where you are

Choose Retina when...

- ✓ You already use Prometheus + Grafana
- ✓ You need low-cost network traceability
- ✓ L7 metadata is not yet a requirement
- ✓ You want drop reasons & DNS metrics fast
- ✓ Cost-per-node is a constraint at scale
- ✓ Starting your observability journey

Choose Cilium + Hubble when...

- ✓ You need accurate pod identity in flows
- ✓ Debugging HTTP/gRPC application failures
- ✓ Building a Zero Trust network model
- ✓ You want the live service dependency graph
- ✓ NetworkPolicy enforcement via eBPF
- ✓ Security auditing inter-service traffic

→ *Not permanent choices — start with Retina for lightweight visibility, migrate to Cilium as requirements mature. Both coexist in a cluster.*

The Blueprint: Your First 3 Steps

Apply this to your cluster — today

01

Deploy Retina as DaemonSet

~5 minutes. Get TCP flows, DNS, drop reasons in Prometheus immediately

```
helm repo add retina https://microsoft.github.io/retina/
helm install retina retina/retina-agent \
  --set operator.enabled=true \
  --set hubble.enabled=true
```

02

Query with Hubble CLI

Instantly see which pods are talking, DNS lookups, and drop reasons

```
# Watch flows from a specific pod
hubble observe --from-pod default/my-service
# Filter dropped traffic only
hubble observe --verdict DROPPED
```

03

Upgrade to Cilium for Identity + L7

When you need HTTP metadata, pod identity, and the service dependency graph

```
helm install cilium cilium/cilium \
  --set cni.chainingMode=aws-cni \
  --set hubble.enabled=true \
  --set hubble.ui.enabled=true
```

Key Takeaways

What to remember from this session

eBPF is the only way to see the full picture

iptables is blind to L7, pod identity, and drop reasons. eBPF hooks into the kernel and changes that.

Retina: 47% lighter, great for Prometheus-native teams

Lightweight, CNI-agnostic, operational in minutes. The smart first step.

Cilium: identity-aware, L7-deep, policy-enforcing

When you need to debug HTTP failures, build Zero Trust, or map real service dependencies.

Network observability is a requirement, not a nice-to-have

As clusters scale, 'ask any question about your network' is the only sustainable debugging strategy.

Thank You!

Questions & Discussion

Udit Misra

Senior Software Engineer · Platform Engineering · Salesforce

Open Source Contributor: Microsoft Retina · Speaker: PlatformCon 2025 · NC State University Affiliate

github.com/microsoft/retina

docs.cilium.io/observability/hubble

uuditmisra@gmail.com
linkedin.com/in/uudit-misra-b3080a49