Challenges and Opportunities in Performance Benchmarking of Service Mesh for the Edge

Mrittika Ganguli (mrittika.ganguli@intel.com)

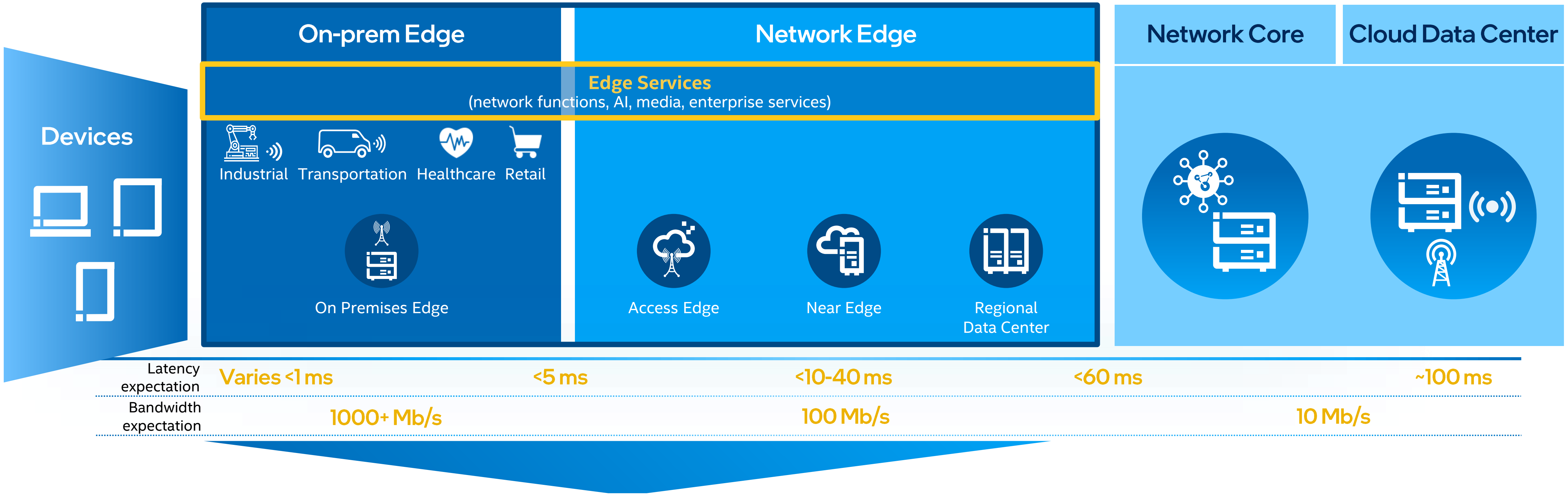
Sunku Ranganath (sunku.ranganath@intel.com)

Subhiksha Ravisundar, Abhirupa Layek, Dakshina Ilangovan, Edwin Verplanke

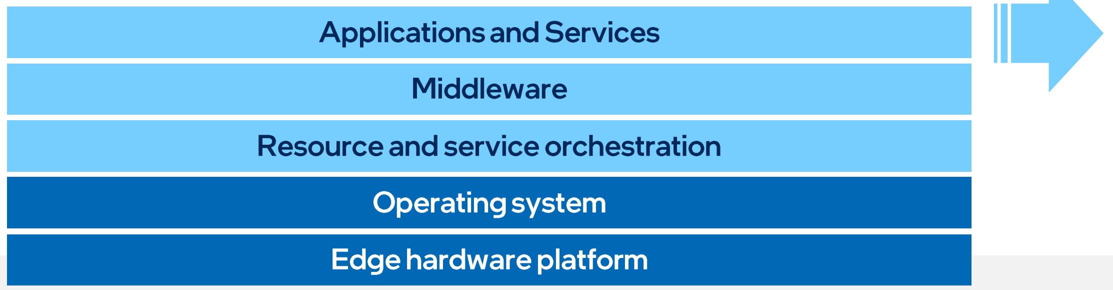
Agenda

- Service mesh vs. Edge deployment requirements
- Challenges at each layer of networking
- Experiment methodology and Results
- Micro-architecture analysis
- Summary and call to action

Edge Native Platforms & Applications



Unified Platforms Across Different Type of Edge Deployments

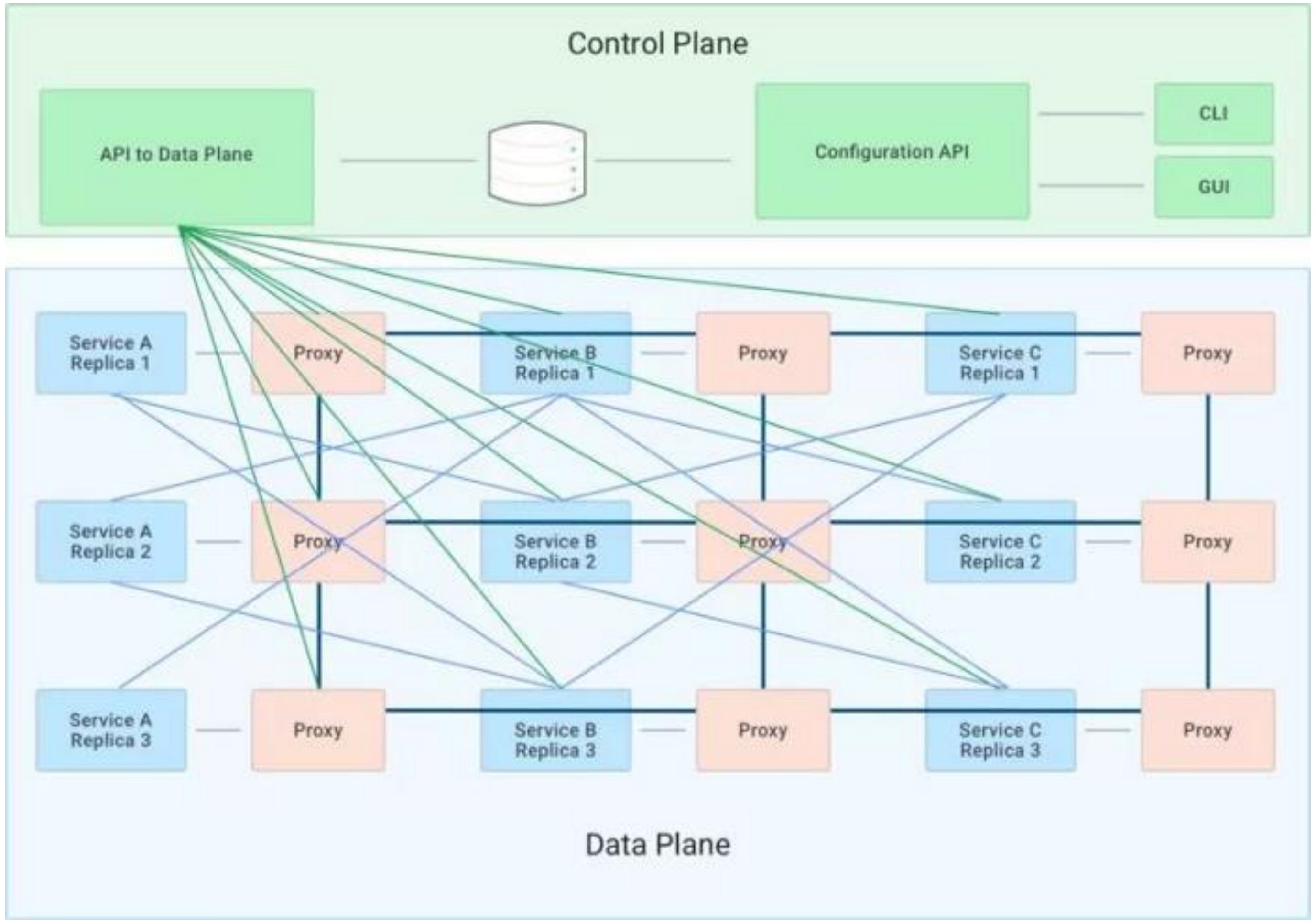


Key challenges to overcome

- Mobility & Federation across MEC domains
- Resource awareness & optimal performance for low latency applications
- Edge Native Application scalability

intel.

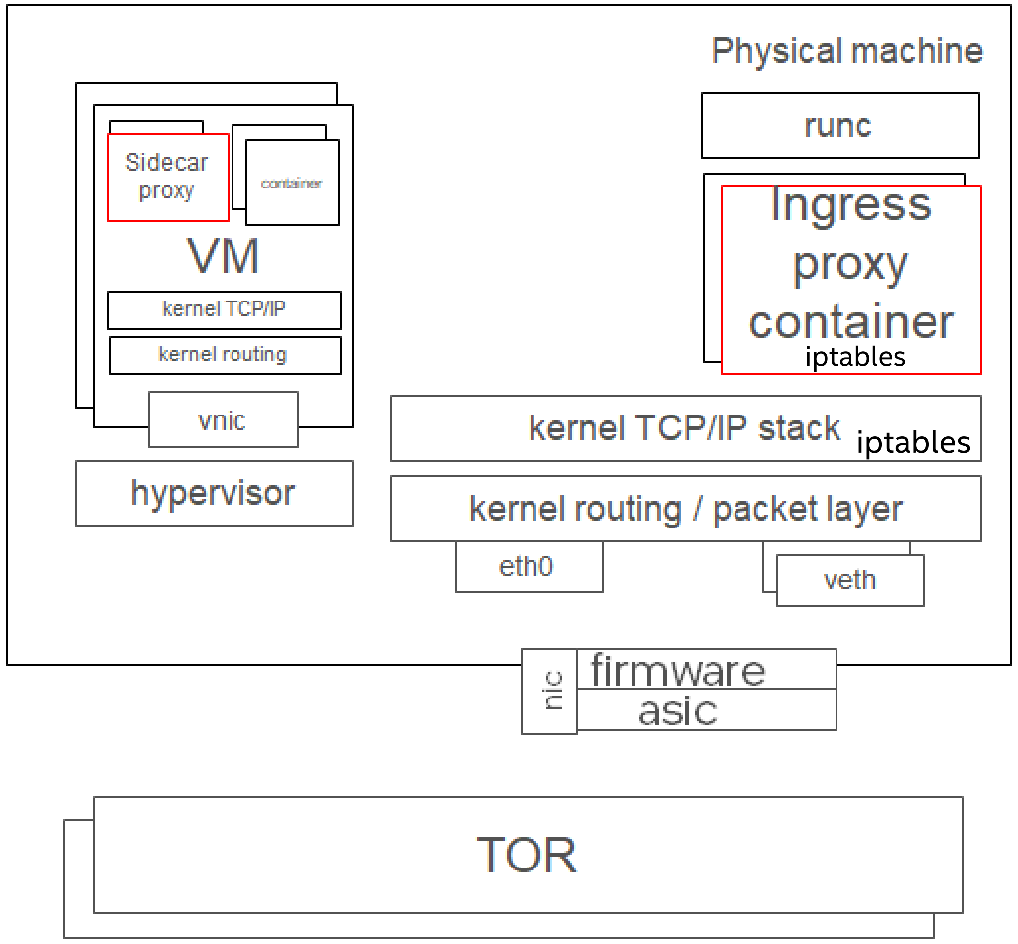
Conceptual Service Mesh



Edge Application Requirements vs. Service Mesh

Attribute	Edge Native Applications	Service Mesh
Awareness & Discovery	Registered & Discoverable, geolocation for enhanced QoS	Side-car proxy to discover set of available services within a cluster and additional meta-data such as QoS or location or traffic requirements, etc., that the application utilize during run-time
Resiliency	Ability to self-heal and be resilient across application restarts or heavy load conditions or unpredictable failure situations	Health checks, traffic reroutes, circuit breaking functionality are all taken care by the mesh control plane
Scalability	Ability to scale as per traffic and load conditions on-demand	Service mesh can threshold the traffic surges and rerouting the traffic to application pods that can handle the increase in requests
Low-latency offloads	Offload the services from end devices, to edge environment to the cloud compute infrastructure on need basis to satisfy low latency requirements	Facilitate interaction between clusters, as well can utilize hardware offload constructs such as offloads to Smart-NIC or utilize acceleration hardware for lower RTT
Security & Privacy	Necessary security network functions are introduced across the Edges to provide secure boundaries across Edge to Cloud communication continuum	Service mesh control plane can help in this domain by offloading communication security aspects such as TLS termination, Ipsec offloads, etc., using side-car proxies

Cloud Native Network Characterization

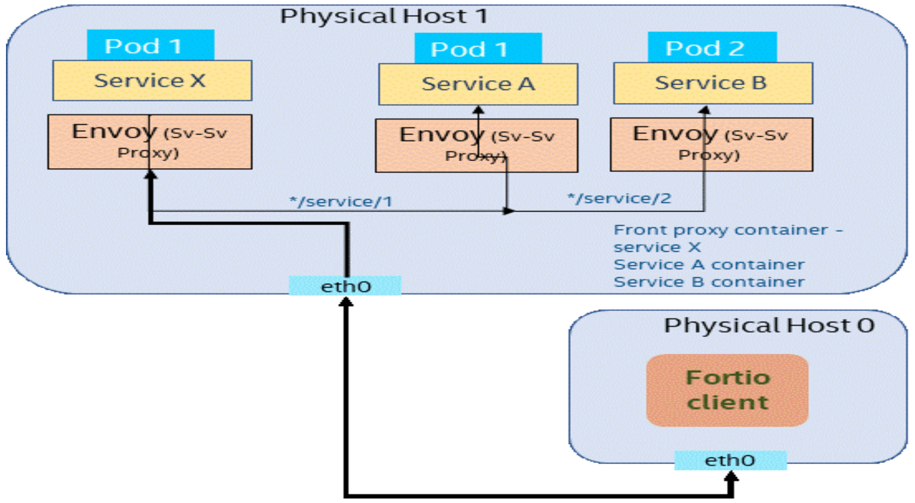


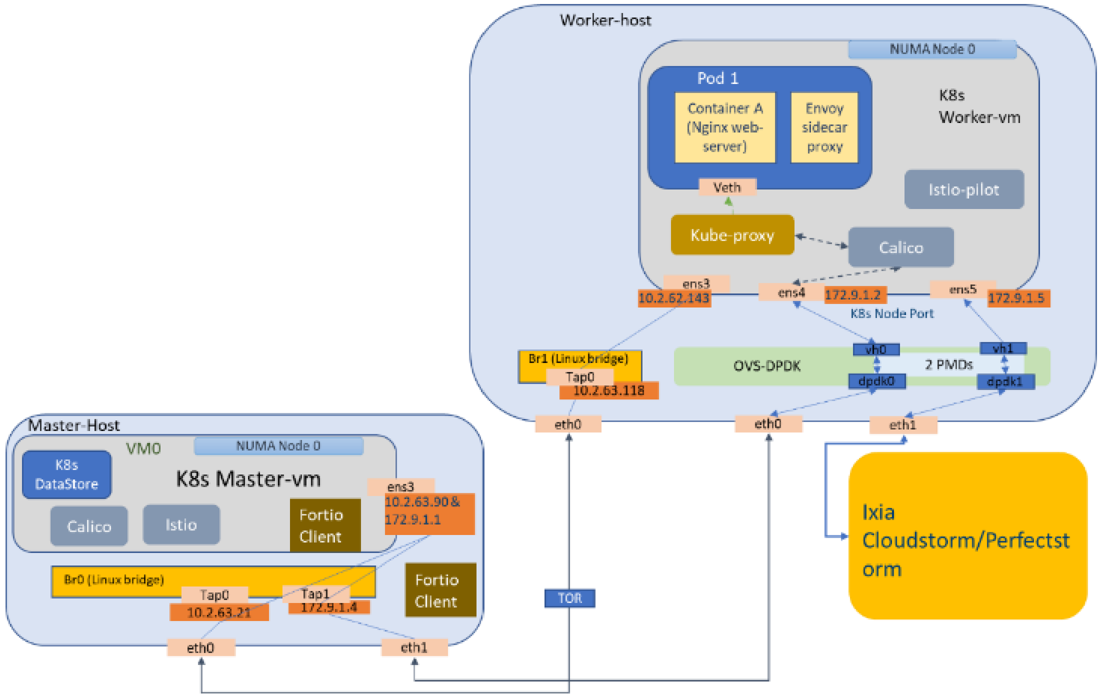
- Microservice architectures are decoupling applications from maintaining & managing infrastructure operations to only perform required business logic
- Kubernetes CNI operates at Layer 3 while Service Mesh takes care of Layer 4 to Layer 7 communication
- Service mesh deployment uses iptables to establish network connections between pods and nodes, managing the networking and port forwarding rules
- Kubernetes pods can scale up to 1000, creating thousands of IP addresses which can be efficiently managed by iptables rules
- Attributes of edge aware applications could be directly attributed to functionality of a typical service mesh

- Each layer adds overhead.
- Tail latencies and microarchitectural analysis will drive optimizations required and offloads that address the performance bottlenecks

intel.

How was benchmarking attempted





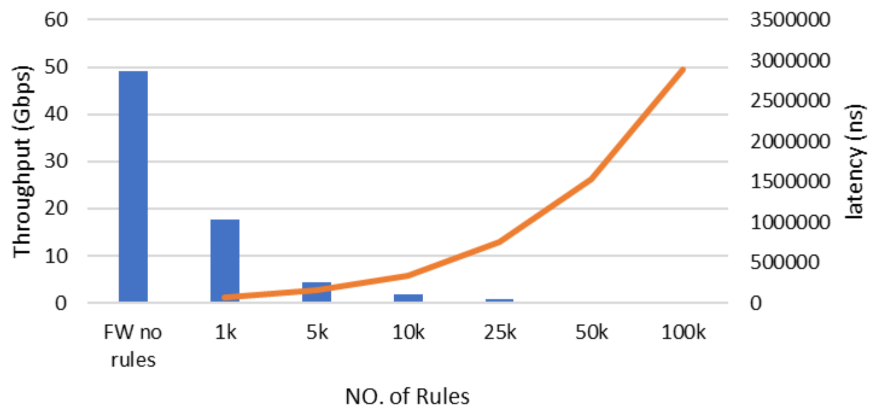
The experiment's goals are to:

1. Measure envoy front proxy performance by increasing the number of queries per second.
2. Increase the number of client connections to obtain maximum QPS resolved successfully
3. 1 & 2 tested and compared on a 48 core Xeon vs a 32 core Xeon with no core pinning.
4. Core scaling experiments done in 20 core Xeon with increasing QPS, connections and clones

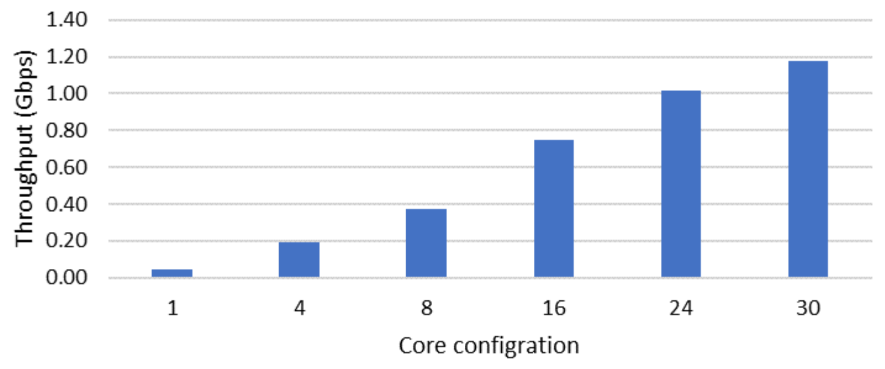
Iptables: Performance overview and bottlenecks

- Performance Overview
  - Throughput drops and latency increases with number of rules. Latency increase with no. of rules
  - Performance scales linearly with cores
  - Additional NAT rules over filter rules has minimal impact on performance.
- Bottlenecks
  - Time to load 100k rules is approx. 2hrs 45min
  - Filter table does a linear search. Position of rule matters
- Workarounds/ Next Steps
  - Ipset, nftables, Ebpf filter
  - Hyperscan implementation
  - Offloading iptables functionality

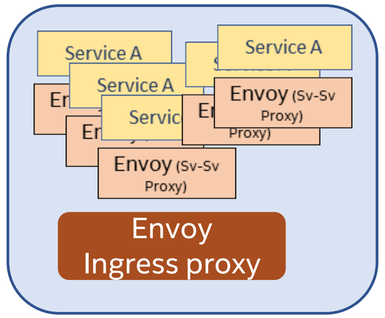
IPtables - Rule scaling Performance
Throughput vs Latency
1024B packets, 1k flows, 16C32T



IPtables - Core scaling performance
25k rules matching at 25kth position
1024B UDP packets, 1024 flows

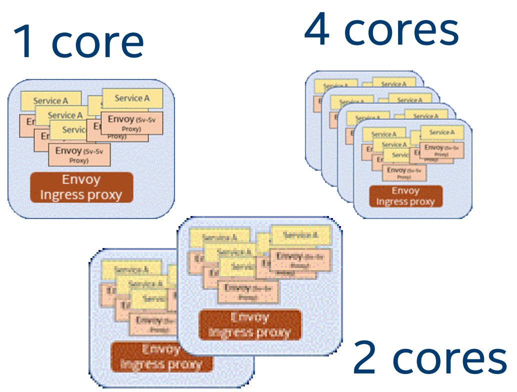


Summary for 100 uSvcs and 64 connections



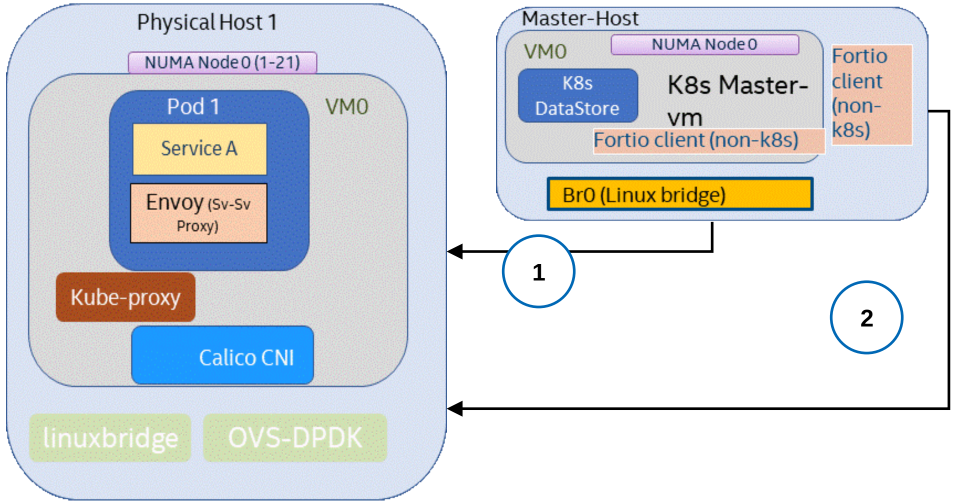
Bare metal

CPU	QPS	Latency (ms)	NBW Mb/s
Xeon1	7.5 k	13.6	250
Xeon 2	12.1k	9.5	370



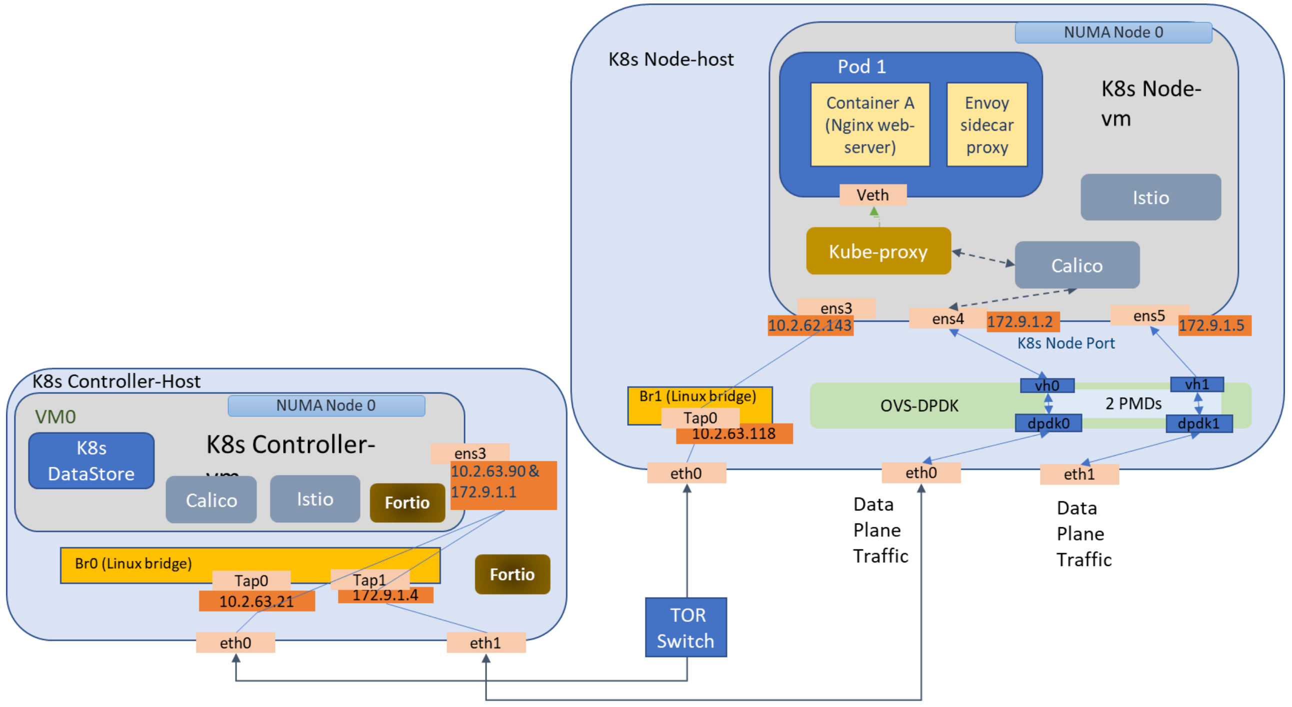
Bare metal pinned cores

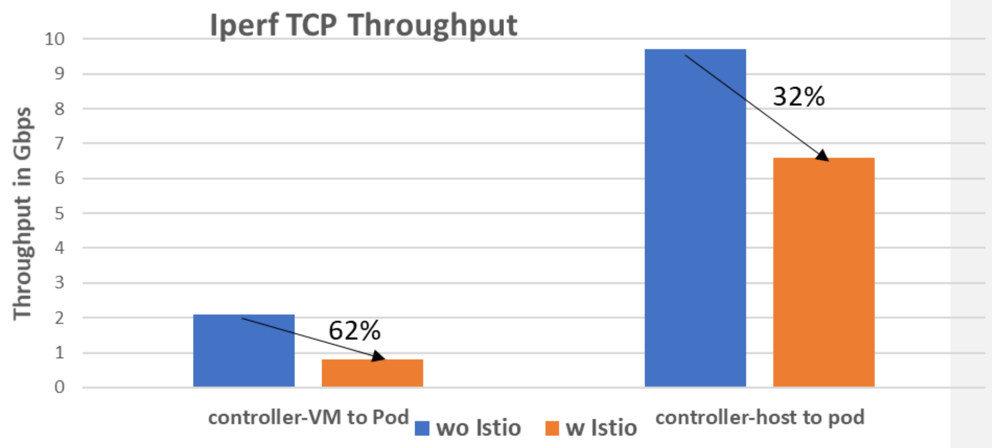
core	QPS	Latency (ms)	NBW
1	200	1300	7 Mbps
2	400	900	18 Mbps
4	1300	1000	54 Mbps

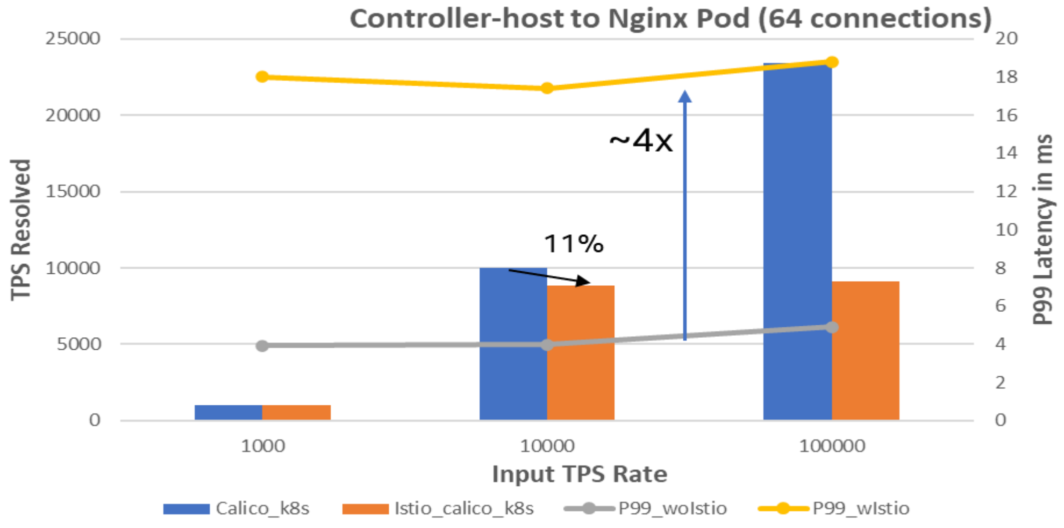


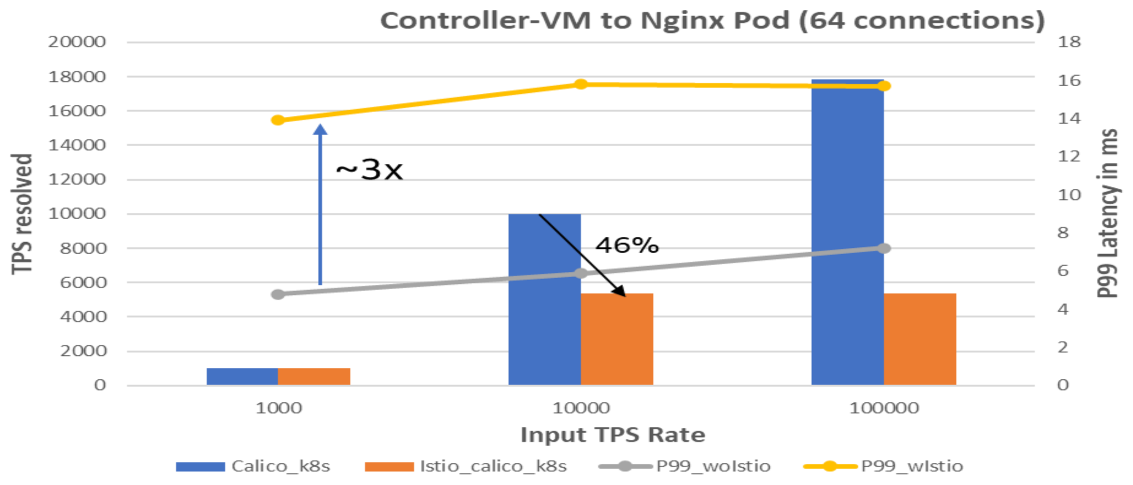
metric	K8s + Calico (no proxy) – 10 cores		K8s + Calico + Istio + Envoy – 10 cores	
	Client in VM	Host client	Client in VM	Host client
QPS	17 k	23k	5k	9k
Latency (P99)	7 ms	4 ms	15 ms	18 ms
NBW (iperf)	2 Gbps	9.6 Gbps	0.8 Gbps	7 Gbps

Performance in Virtualized Environments



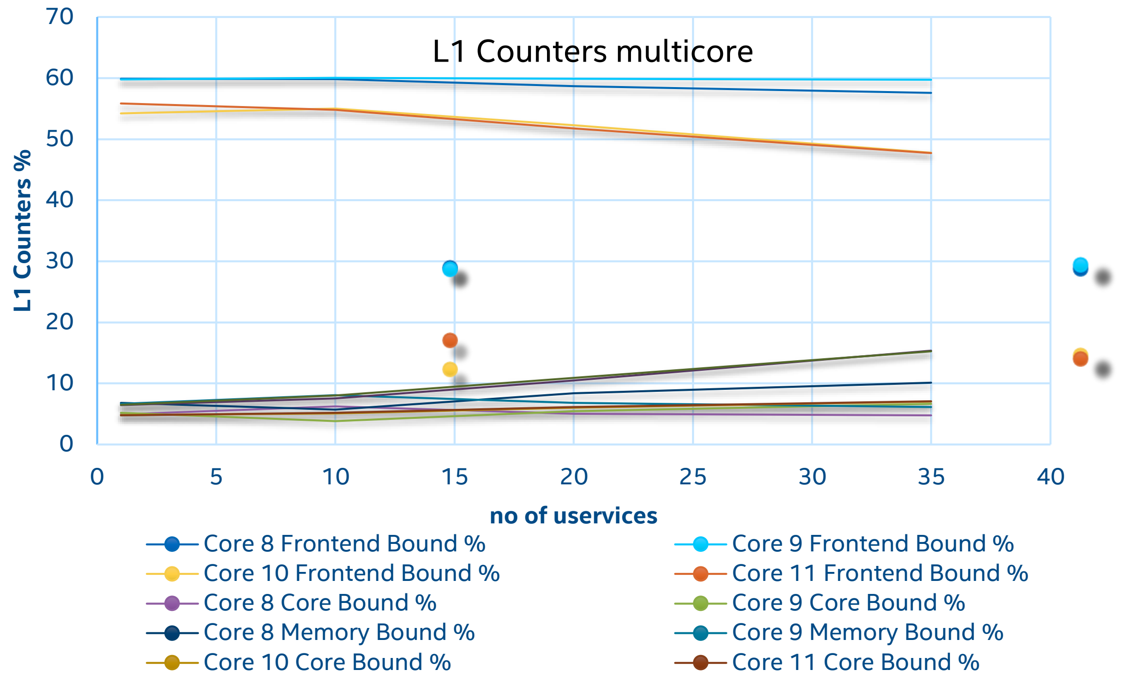


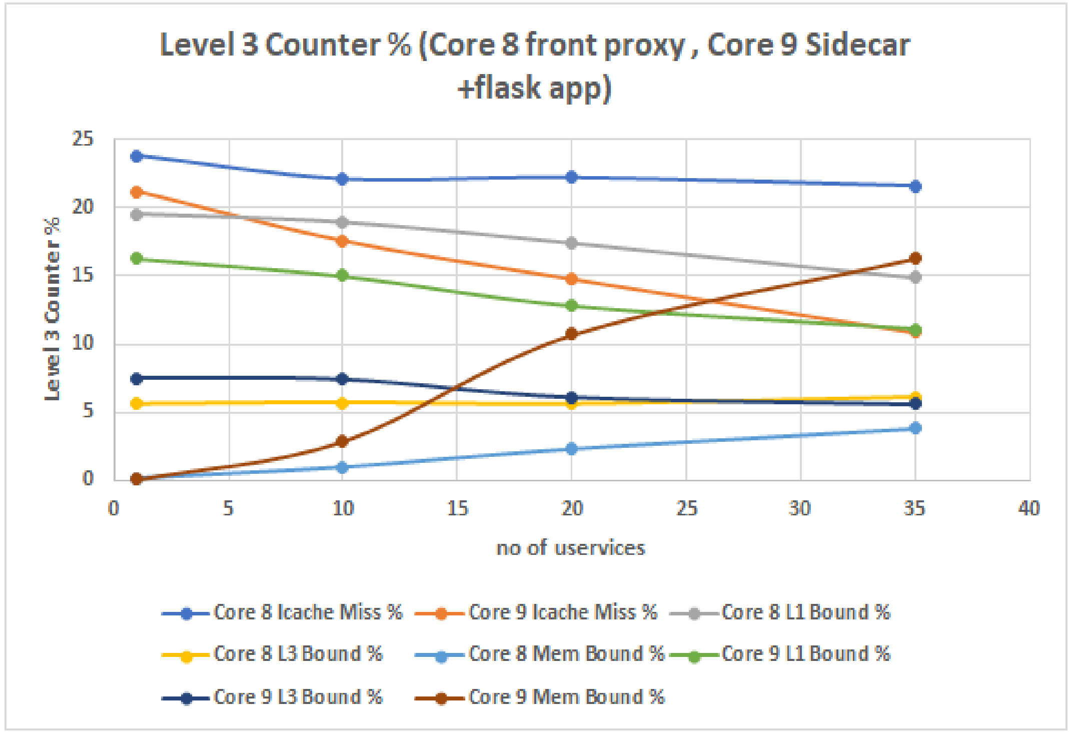




intel.

Micro-Architecture analysis of 40uSvcs multi-core





- Frontend Bound % decreases with increase in number of uservices
- Core Bound and Memory Bound % increases with increase in number of uservices
- L1 and L3 Bound % generally increases for core 8 and decreases for core 9 with number of uservices
- L1 and L3 Bound % decreases for core 10 and 11 with the number of uservices.
- Mem Bound % increases for front proxy but at a larger scale for side car+ flask with number of uservices
- L3 cache misses increase for 0.9-31%

CPU cycle analysis

Layer	Description	%
kernel	Linux forwarding	20%
kernel	Entry Linux switching	20%
Other layers	Other functions	60%

1 core multiple clones front proxy or BE sidecar

Layer	Description	%
kernel	Linux forwarding	20%
kernel	Entry Linux switching	13%
kernel	Libc	16%
Envoy	Envoy static match	22-30%
Envoy	Buffer+water mark	18-30%

1 core sidecar +app 1 clone or multiple clones

Layer	Description	%
kernel	Linux forwarding	20%
kernel	Entry Linux switching	13%
kernel	Libc	16%
Envoy	Envoy-memcpy	20%
Other layers	Other functions	60%

multiples core multiple clones front proxy or BE sidecar

Summary

- Service mesh forms a ***very important software architectural framework*** for Edge computing that can directly correlate with ETSI MEC framework.
- Due to the ***performance impact*** of introducing service mesh and complexity across kernel stack and deployment, the immediate utilization of off the shelf service mesh software components for production usage at the Edge is delayed.
- To deploy microservices with a service mesh it is important to ***identify the right profiling environment*** to estimate what QPS to latency ratio is tolerable for the number of clones deployed.
- To maximize CPU usage number of microservices may need to be increased but to keep a 99-percentile latency in milliseconds, number of concurrent client connections need to be lower.
- EMON and TMAM analysis can help in initial workload characterization but bottlenecks in the CNI layer for bridging and envoy TCP stack traversals through Linux kernel need to be profiled and identified. Calico iptables rule processing analysis revealed a bottleneck in lookup for NAT table traversals.
- Although efforts & proposals are under way in CNCF's Network SIG to standardize some of the traffic generator tools to have consistent performance across test runs, the biggest challenge is to address the need for multiple layers of benchmarks and address the challenges.

Call To Action

- Build **one benchmark** which will
  1. Follow a standard layer 7 benchmarking process with multiple KPIs and configuration modes,
  2. Model different workload patterns
  3. Run on one system and generate primitive based data which can be used to estimate the cluster capacity and performance need
  4. Address different virtualized environment setups with resource (cores, memory and queues) combinations to model different application infrastructure environments