Calico/VPP: All You Can Eat Networking

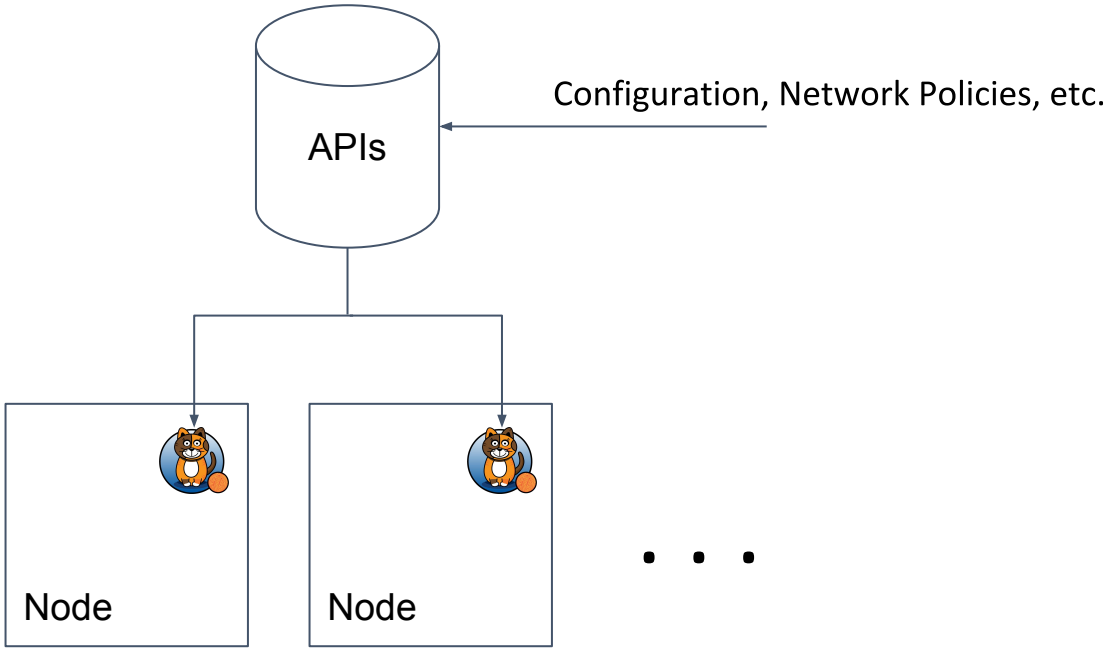
Bringing Kubernetes Goodness to your Hungriest Workloads

Aloÿs Augustin, Casey Davenport

What is Calico?

- Open-source Kubernetes networking and network policy
- Kubernetes pods, nodes, VMs, and legacy workloads
- Rich network policy APIs
- Battle-tested: deployed in production at scale

What is Calico?



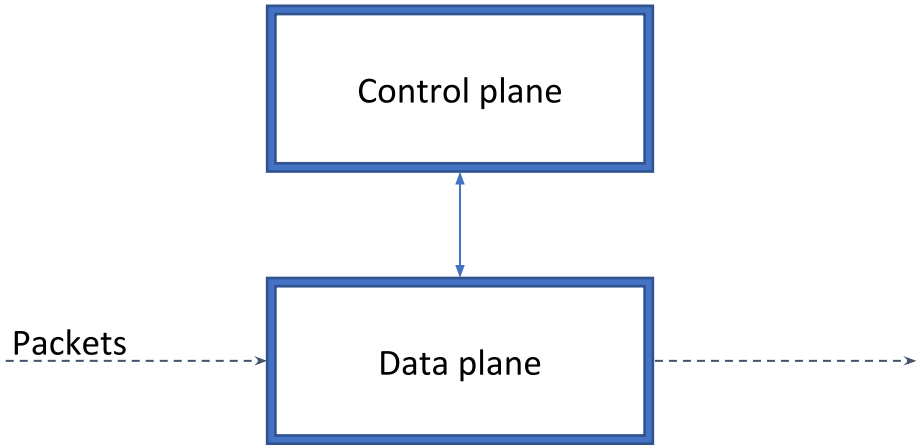
Under the hood

- **CNI plugin / IPAM plugin:**
  - Called by the container runtime on pod ADD / DEL on a per-pod basis
  - Configures pod network namespace with routes, devices, etc.
- **calico/node:**
  - Runs on every node as a DaemonSet
  - Makes routing and policy decisions, make sure they are enforced
  - two main subcomponents: **felix** and **BIRD**

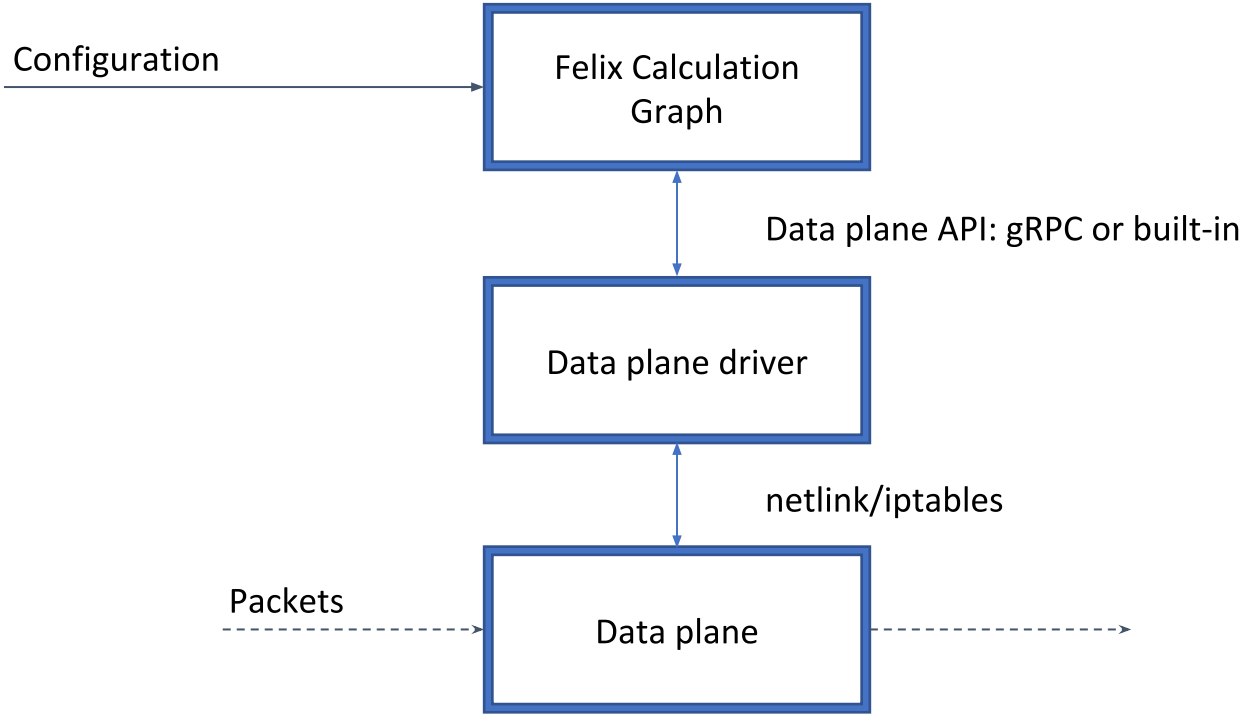
Calico design philosophy

Use the right tool for the job

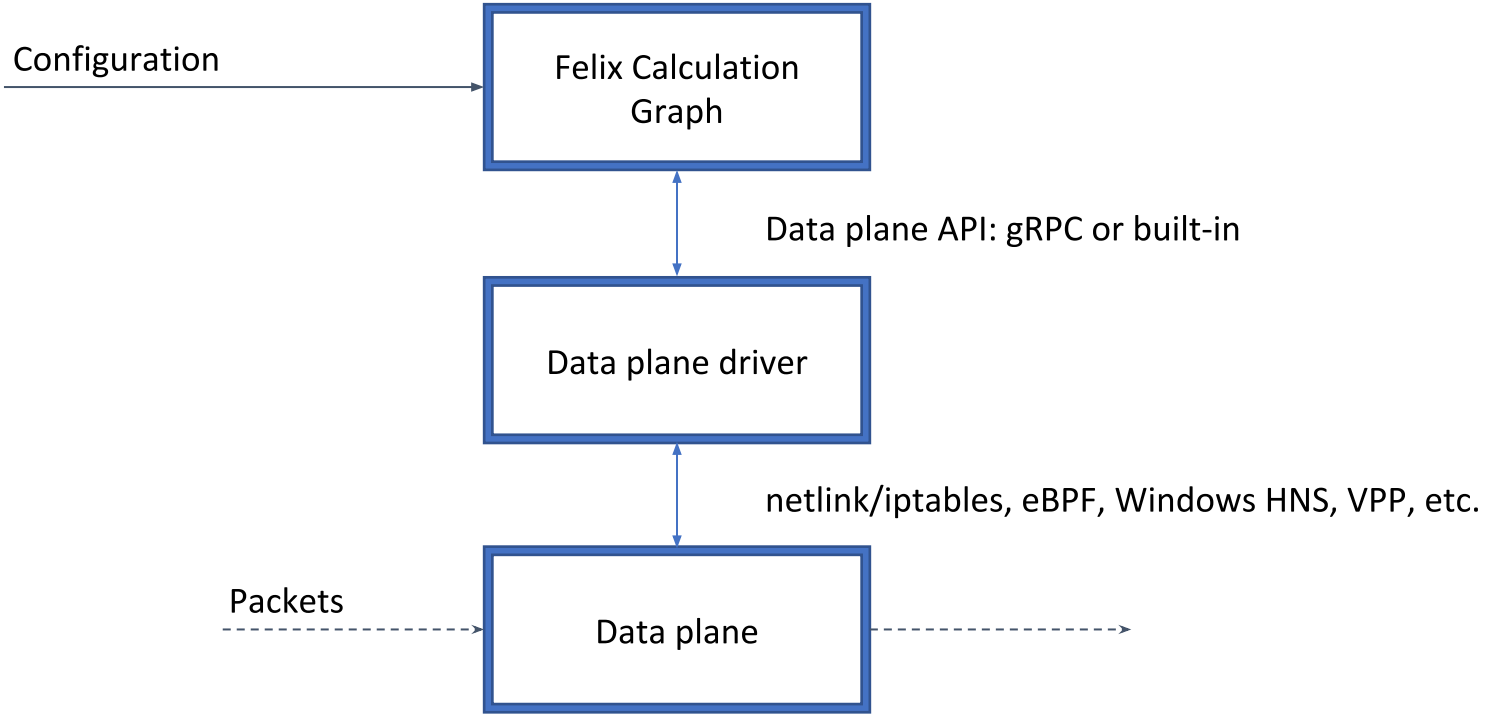
Calico design philosophy



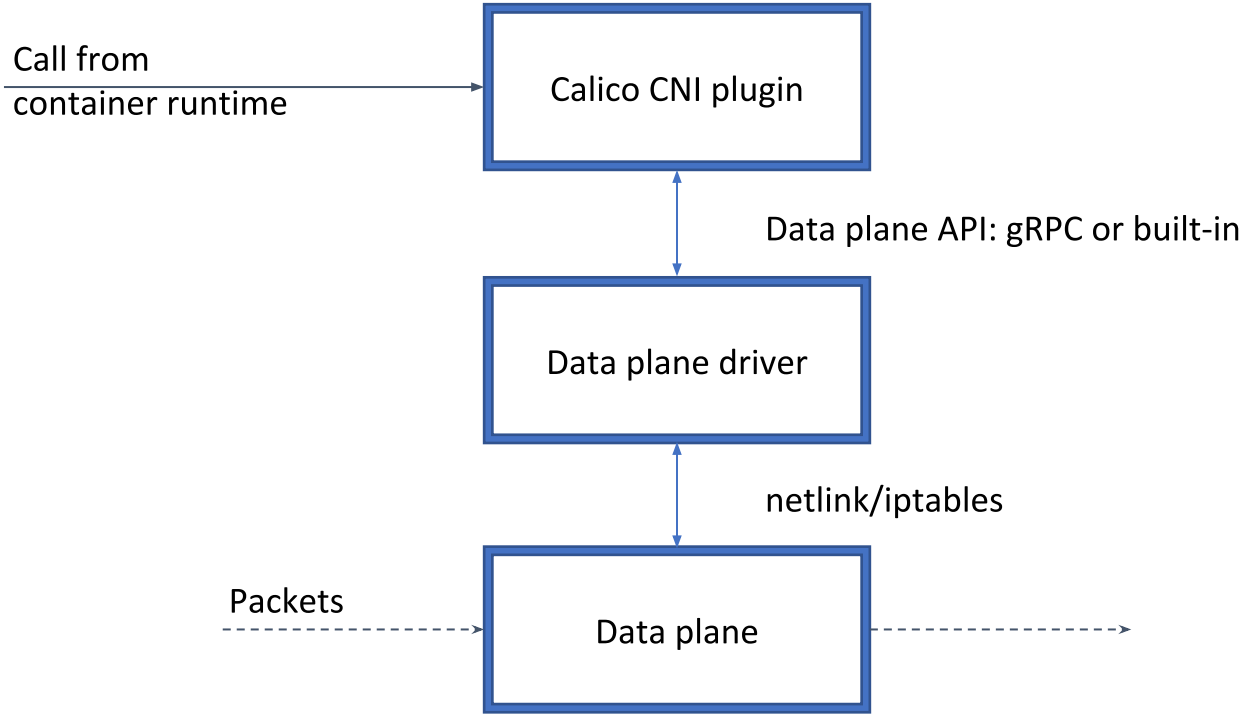
Calico design philosophy



Calico design philosophy



Calico design philosophy



Active community

- 200+ contributors on GitHub
- Regular quarterly releases
- Active slack community of users and developers

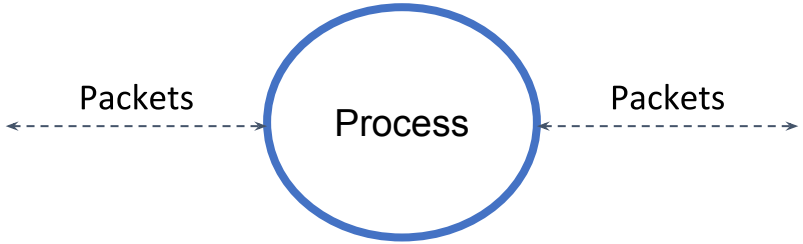
Calico/VPP integration

What is VPP?

- Fast, open-source userspace networking dataplane - https://fd.io/
- Feature-rich L2-3-4 networking: tunneling, NAT, ACL, crypto, TCP, Quic,...
- Easily extensible through plugins
- Supports virtual and physical interfaces
- Fast API: > 200k updates/second
- Highly optimized for performance: vectorization, cache efficiency
- Multi-architecture: x86, ARM

Userspace networking?

- Packet processing done in a regular userspace process
  - Examples: OpenVPN, DPDK based applications, VPP, ...
- Benefits:
  - Performance
  - Development and deployment velocity
  - The network is just another software component in the stack
- Possible thanks to specific interface types (tun/tap) and drivers (uio, vfio)



Riding the kernel modularisation

The linux kernel increasingly offers rich hook points/APIs

- eBPF allows to inject code in the kernel
  - Great for kernel telemetry!
- AF_XDP allows to implement userspace networking functions
- TUN/TAP interfaces with fast virtio backend/multiqueue/GSO/... makes it possible to have efficient userspace <-> Linux communications

It is now possible to leverage high performance userspace networking to accelerate containerized Linux application & microservices

Making VPP container-friendly

CPU

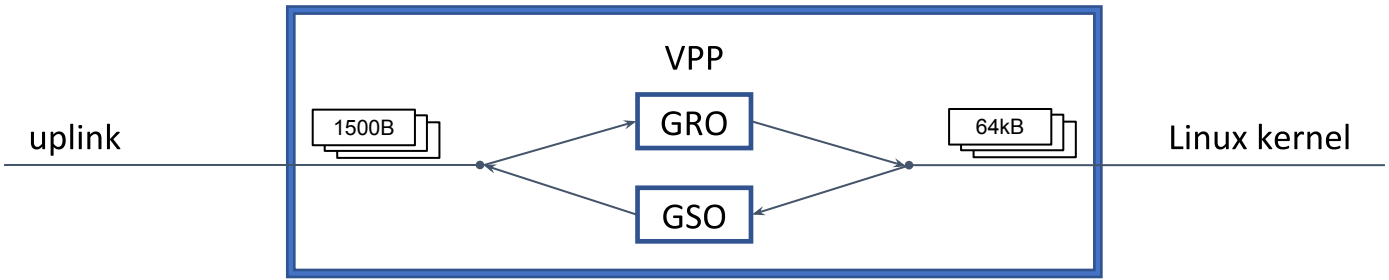
- Poll mode -> Interrupt (or adaptive) mode to reduce CPU consumption
- Scheduler configuration for efficient CPU sharing with pods w/o impact on latency

Memory

- No mandatory reliance on hugepages to ease deployment

Linux integration

- GSO / GRO support to reduce the load on the linux TCP stack (x3-5 speed up)



Optimizing VPP for K8s / Calico

- Custom NAT implementation for K8s
  - Service load balancing (optimized kube-proxy behavior)
  - Source-NAT for outgoing connections

- Custom ACL plugin for Calico policies
  - Stateful ACLs that implement the dataplane API of Felix

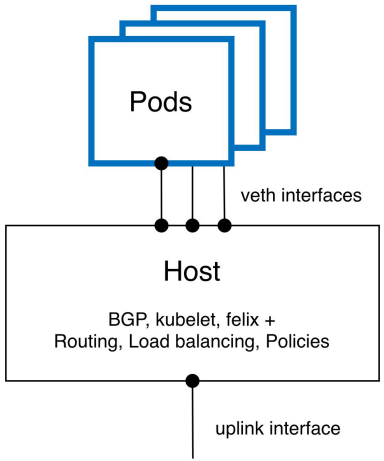
Simplified operations

- VPP is packaged as a regular container
- It can be upgraded/restarted without impacting the pods lifecycle
  - Helpful for seamless upgrades in case of security or bug fixes
- Very limited kernel dependencies
  - Full control over the network stack even in public clouds where the kernel may be maintained by the cloud provider
- Better control over dedicated resources to container networking (memory/CPU/scheduling)

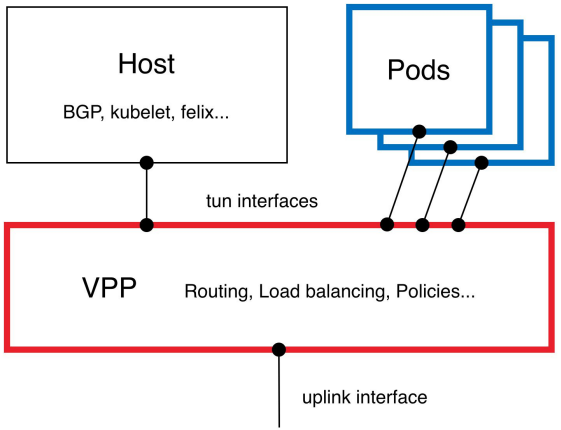
Logical network topology

- VPP inserts itself between the host and the network
- Pure layer 3 network model (no ARP/mac address in the pods)

Regular Calico topology

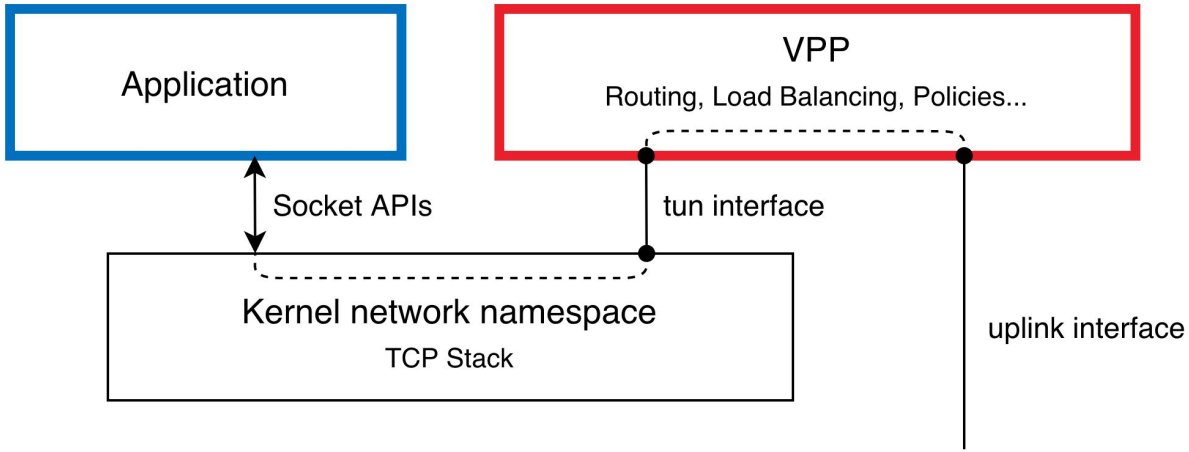


Calico/VPP topology

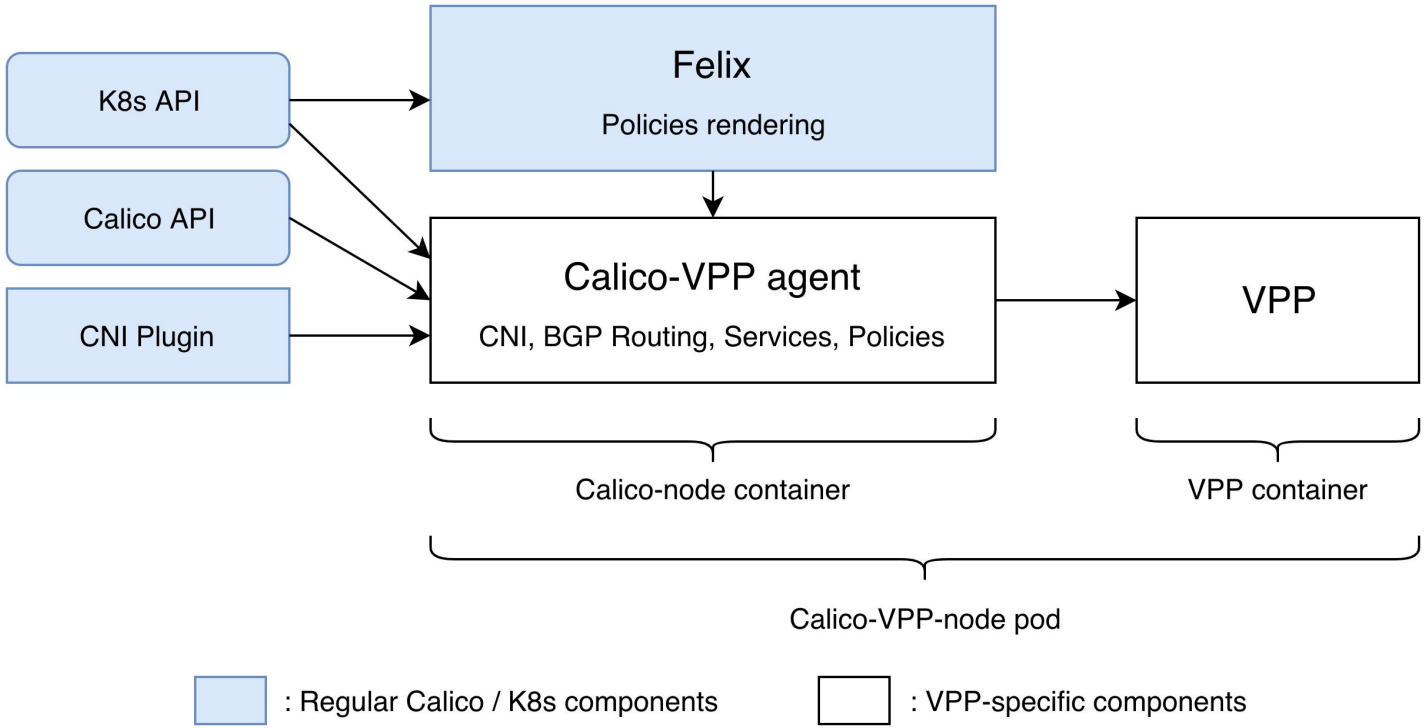


Packet flow

- One tun interface per pod
- No changes required to the applications
- Kernel provides pod isolation / namespacing



Software architecture



Project status

- Open-source on Github
  - https://github.com/projectcalico/vpp-dataplane
- Alpha status
- Calico incubation project
  - Most Calico features are now supported
- Initial performance benchmarks are very promising

Performance benchmarks

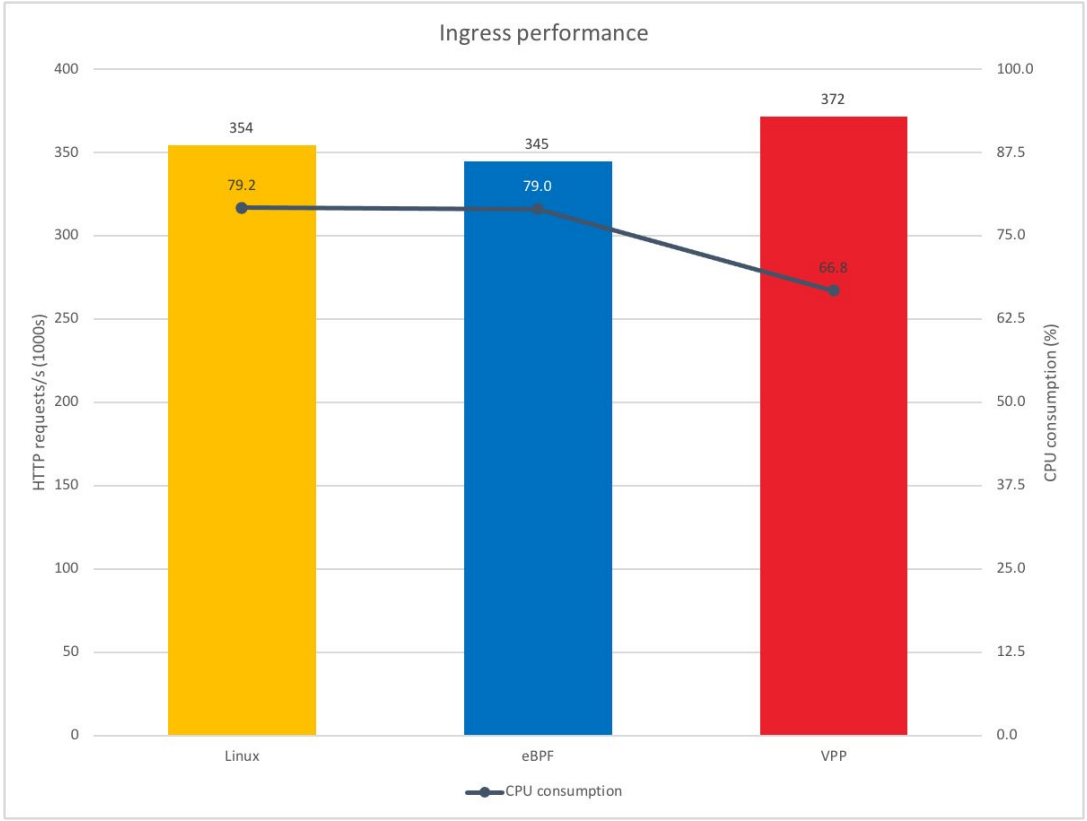
Testbed configuration

- **Hardware**: 2x Cisco C220-M5 UCS with
  - Intel Xeon Gold 6146 CPU (12t, 24c)
  - 192GB 2666MHz DDR4
  - Intel XL710 40G NIC - configured with 1500 bytes MTU
- **Software**
  - Ubuntu 18.04, kernel 5.3.0-51
  - Kubernetes v1.18, Calico v3.16
  - nginx, iperf from Ubuntu packages
  - wrk master from https://github.com/wg/wrk
- **Methodology**
  - Results averaged over 3 runs

HTTP requests/s tests

- Single-node cluster
  - One ClusterIP service pointing to one nginx pod
- wrk client outside the cluster
  - Running 4kB HTTP requests continuously to the service IP
  - 10 threads, 1000 connections in parallel
- Simulates external clients connecting to services
- CPU consumption measured on the server during the tests

HTTP requests/s test results

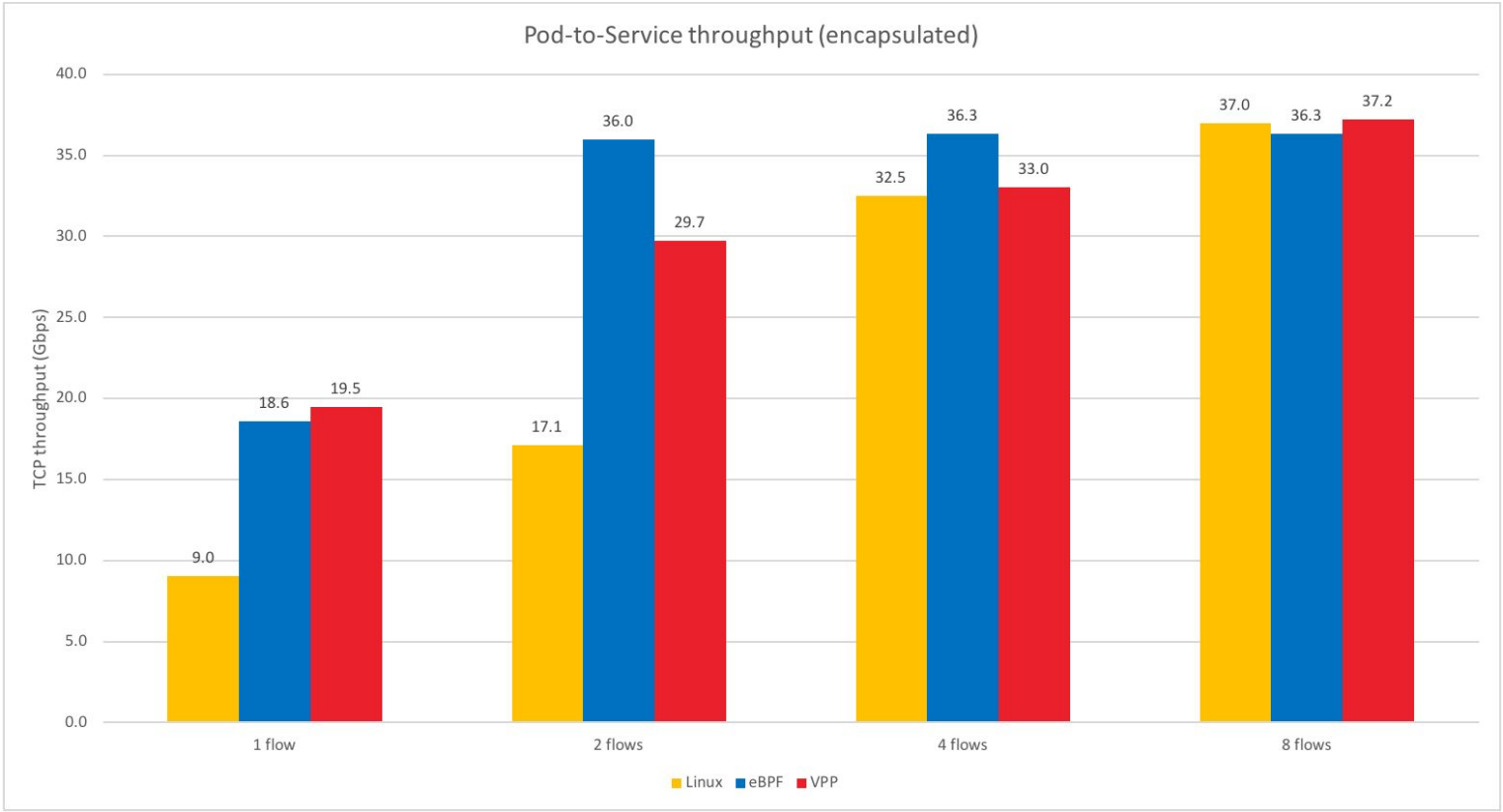


TCP throughput tests

- Two-node cluster
  - IPIP encapsulation for Linux / VPP, VXLAN for eBPF
  - One service pointing to one iperf server pod
  - One iperf client pod
  - Each pod is pinned to one of the nodes

- Test runs with varying number of connections between the client and server

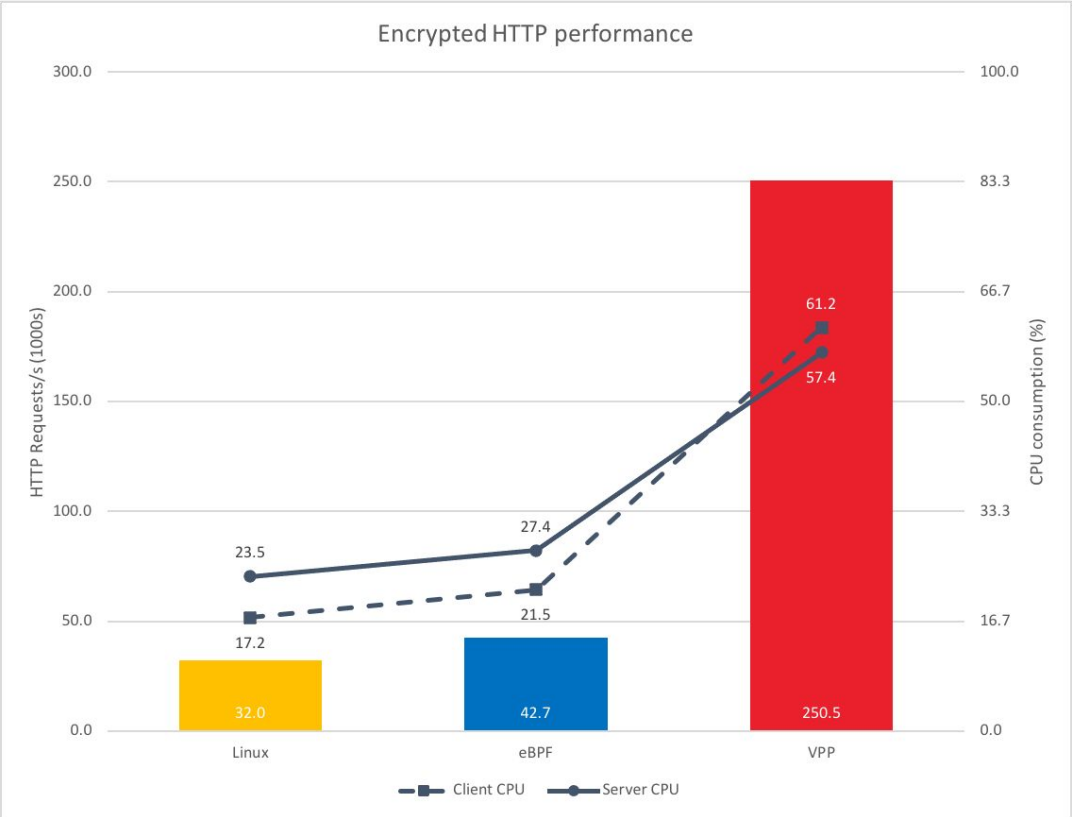
TCP throughput test results



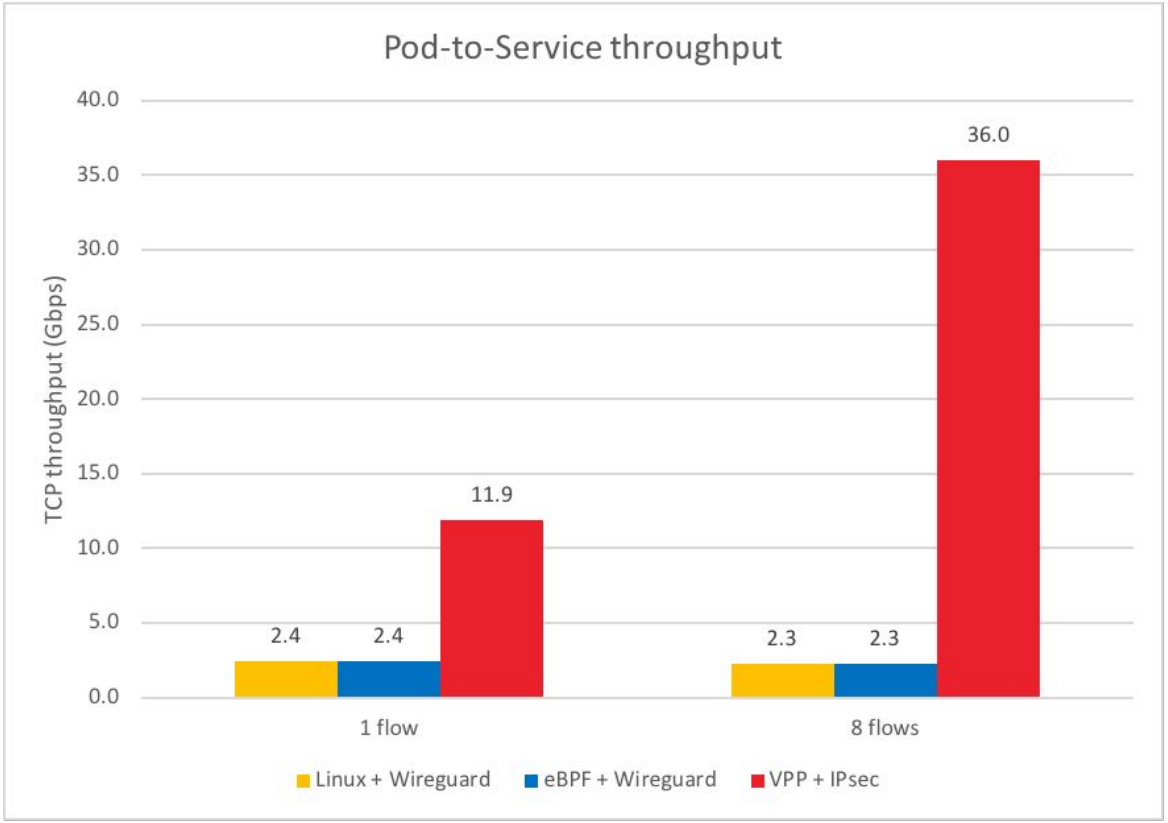
Encryption tests

- Two-node clusters
  - Linux and eBPF dataplanes configured with Wireguard
  - VPP dataplane configured with IPsec
- Same cross-node iperf and wrk tests
  - wrk client now inside the cluster

HTTP Encryption test results



TCP Encryption test results

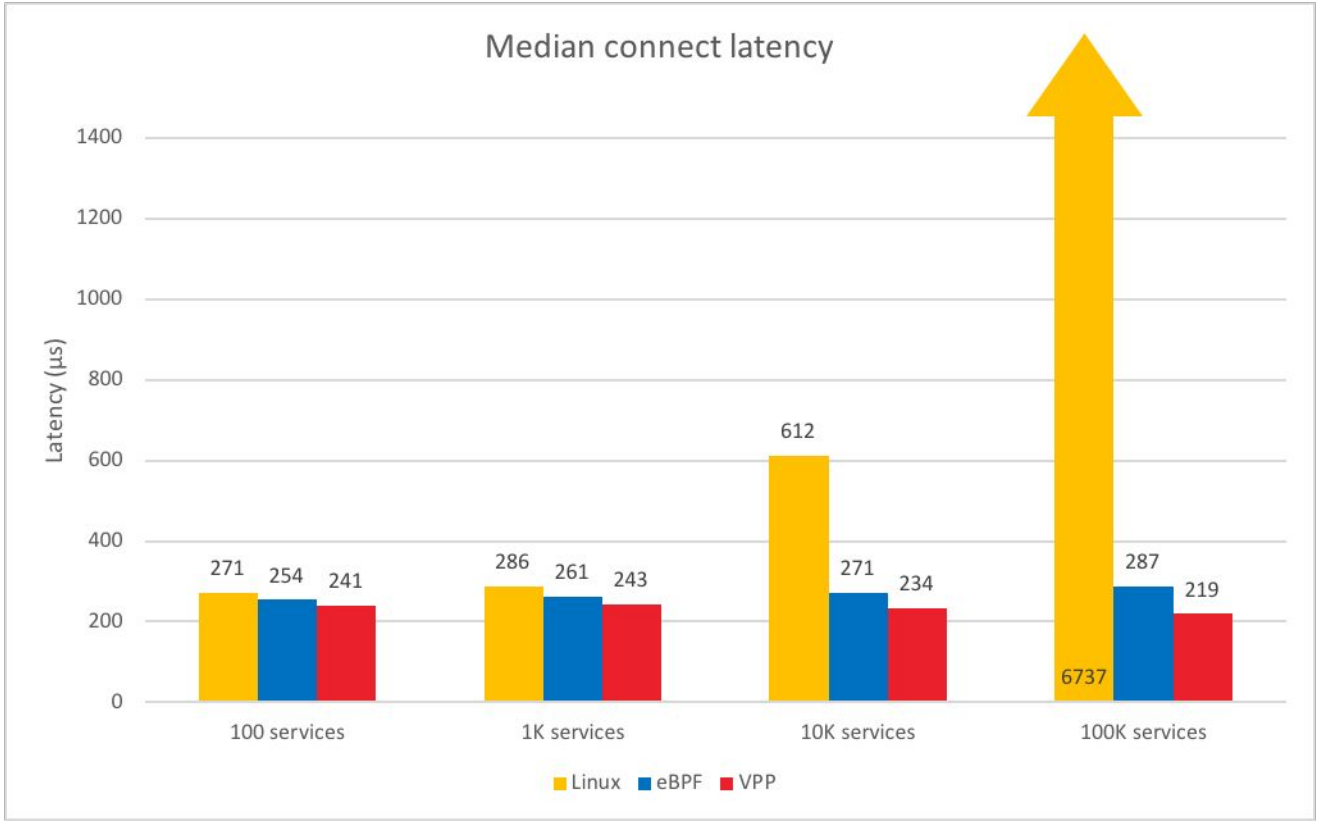


Service scalability tests

- Single-node cluster
  - Large number of services, all pointing to the same nginx pod
  - Standard Linux dataplane uses kube-proxy

- Custom test client, outside of the cluster
  - Sending HTTP requests at constant rate (1k requests/s) for 100s
  - Each request goes to a service chosen at random
  - Connect and request latency measured separately
  - https://github.com/AloysAugustin/go-wrk

Service scalability tests



Next steps: optimizations

- Improved VPP adaptive mode to reduce CPU consumption
- Virtio 1.1 support
- Calico-specific plugins optimization
- 100G IPsec support

- Performance testing with pod churn

Next steps: features

- Wireguard support
- Leverage VPP Telemetry Infrastructure
- Expose additional connectivity options in containers
  - VPP L4 stack
  - memif packet interface
- Envoy TCP/TLS acceleration using VPP L4 stack

- GA status in Calico 😊

Acknowledgements

KubeCon CloudNativeCon North America 2020 Virtual

Many thanks to:

- The FD.io VPP team for their continued support
- The Calico team for their support and feedback

References

- Calico: https://www.projectcalico.org/
- FD.io/VPP: https://fd.io/
  - Continuous performance testing: https://docs.fd.io/csit/master/trending/introduction/dashboard.html
- Calico dataplane driver for VPP:
  - Code: https://github.com/projectcalico/vpp-dataplane
  - Doc: https://github.com/projectcalico/vpp-dataplane/wiki
  - Slack channel: https://calicousers.slack.com/archives/C017220EXU1
- 40Gbps pod-to-pod IPSec for Calico with VPP:
  - https://medium.com/fd-io-vpp/getting-to-40g-encrypted-container-networking-with-calico-vpp-on-commodity-hardware-d7144e52659a


KEEP CLOUD NATIVE
EVERYWHERE
KubeCon
CloudNativeCon
North America 2020
Virtual
HELM
KV