KubeCon | CloudNativeCon

North America 2024

KubeCon CloudNativeCon North America 2024

How the Tables Have Turned

Kubernetes says “Goodbye!” to iptables

Dan Winship, Red Hat

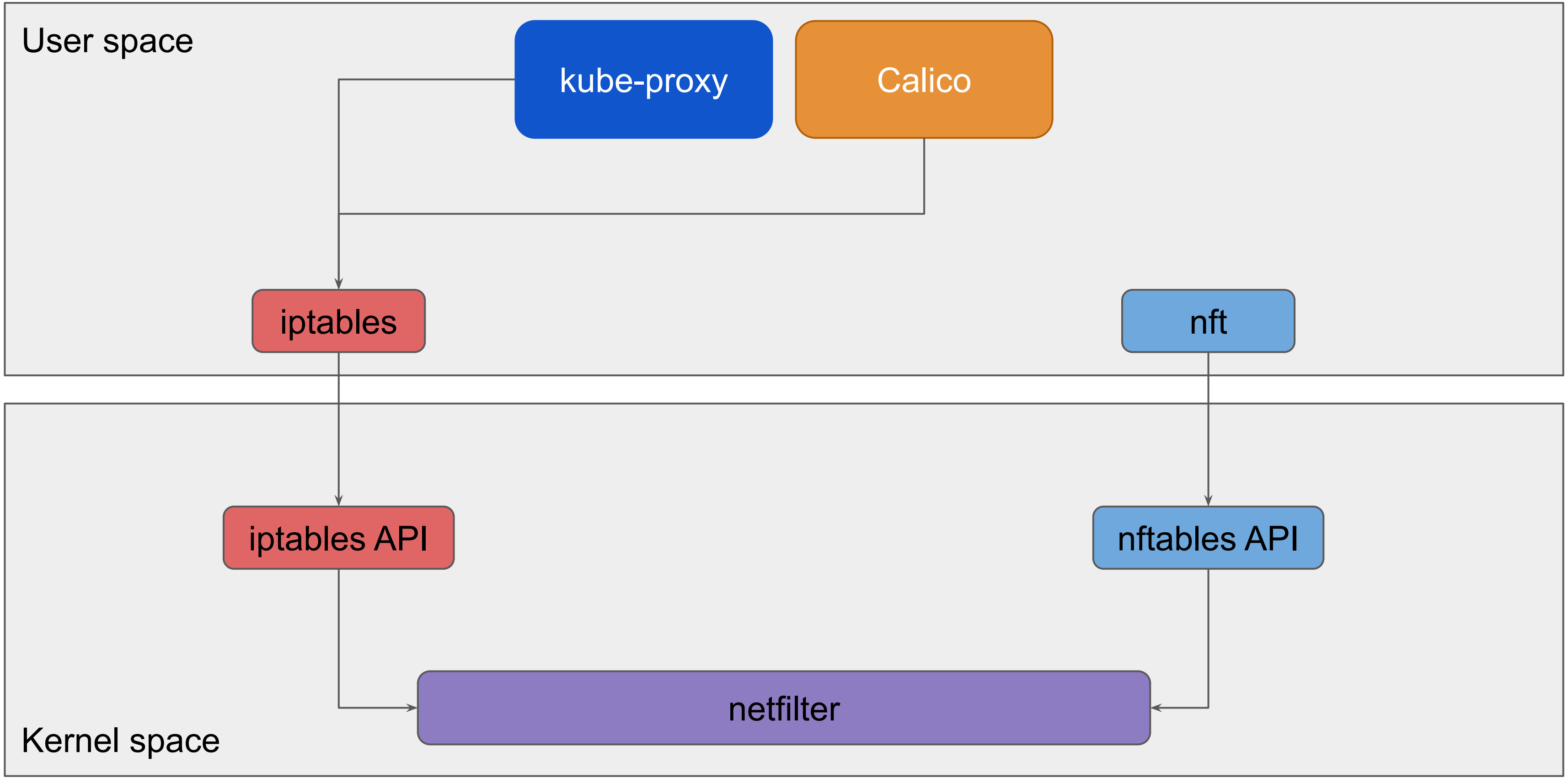
Casey Davenport, Tigera

Overview

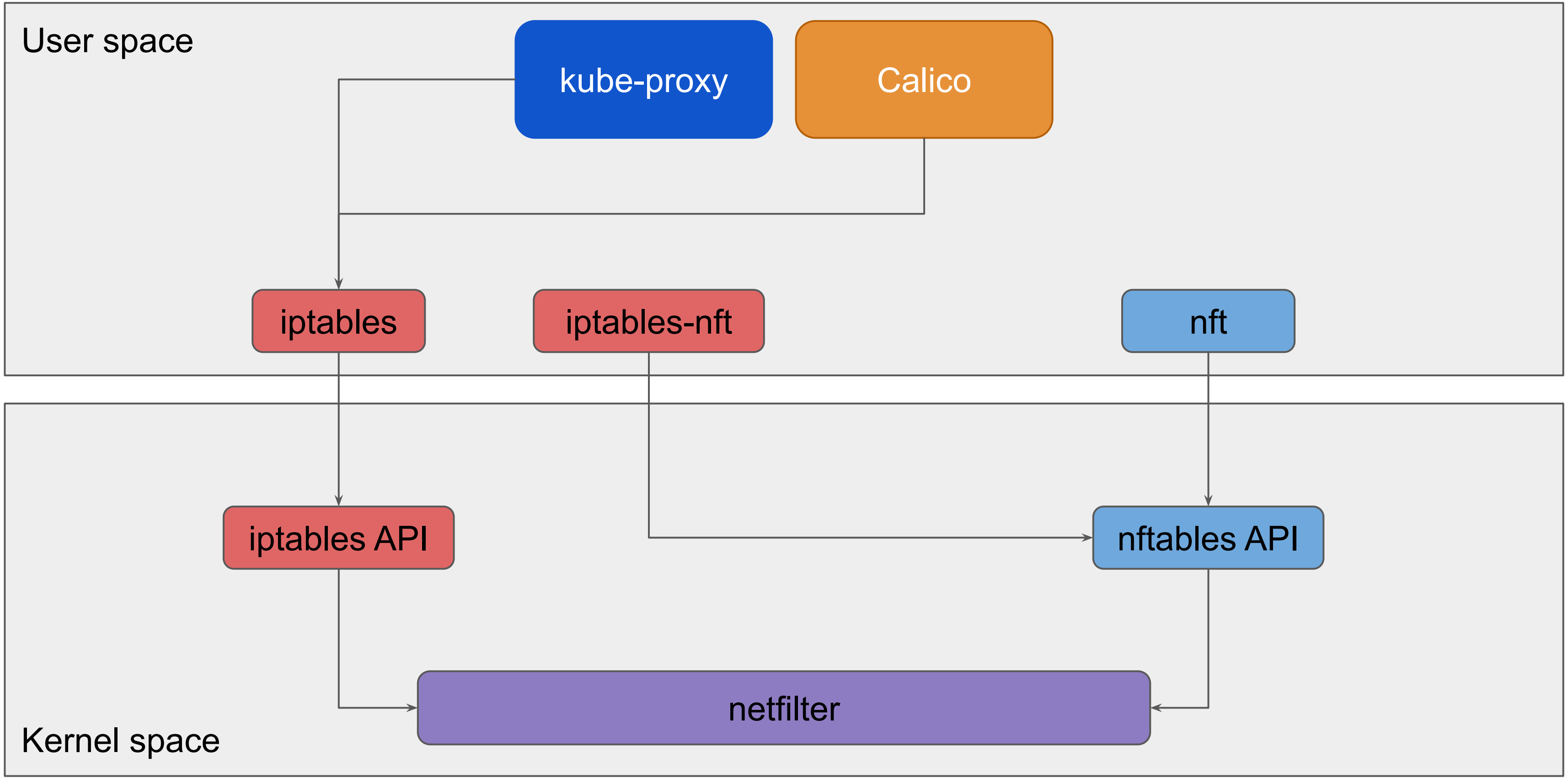
- Overview of iptables + nftables
- Why are we leaving iptables?
- Why did we pick nftables?
- Initial results

iptables? nftables? iptables-nft???

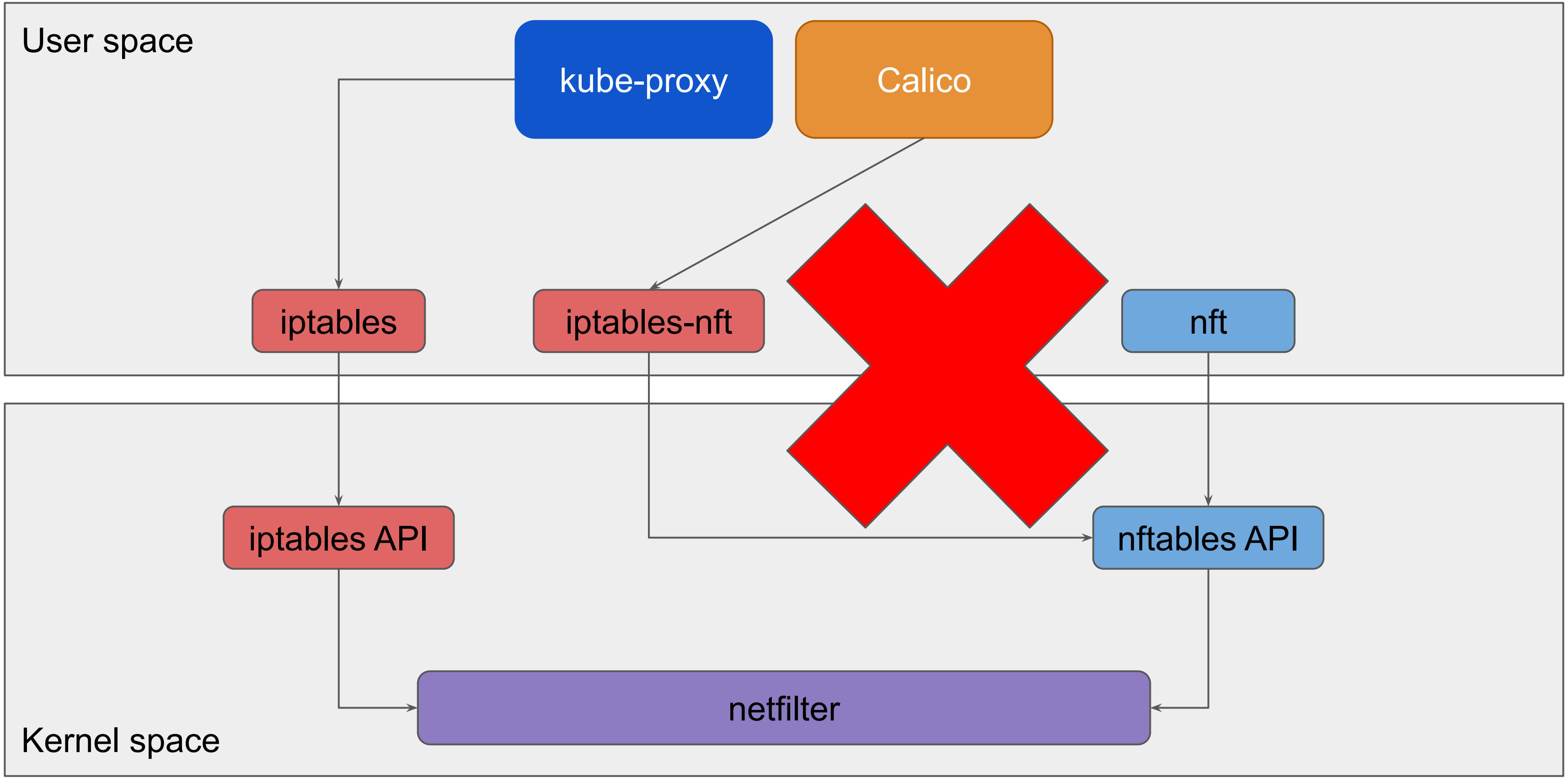
Evolution - The Past (2015-2018)



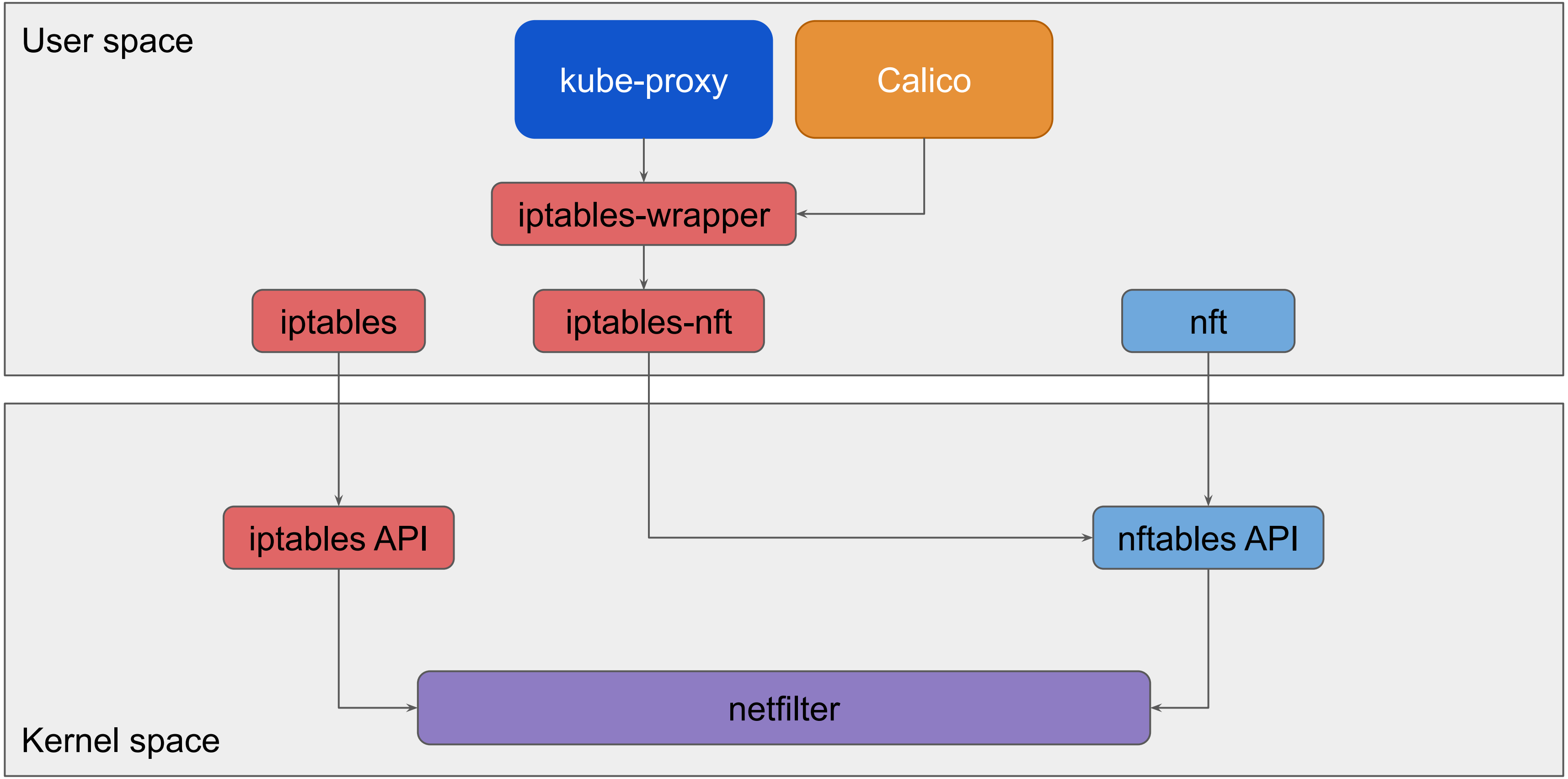
Evolution - iptables-nft transition (2018-2019)



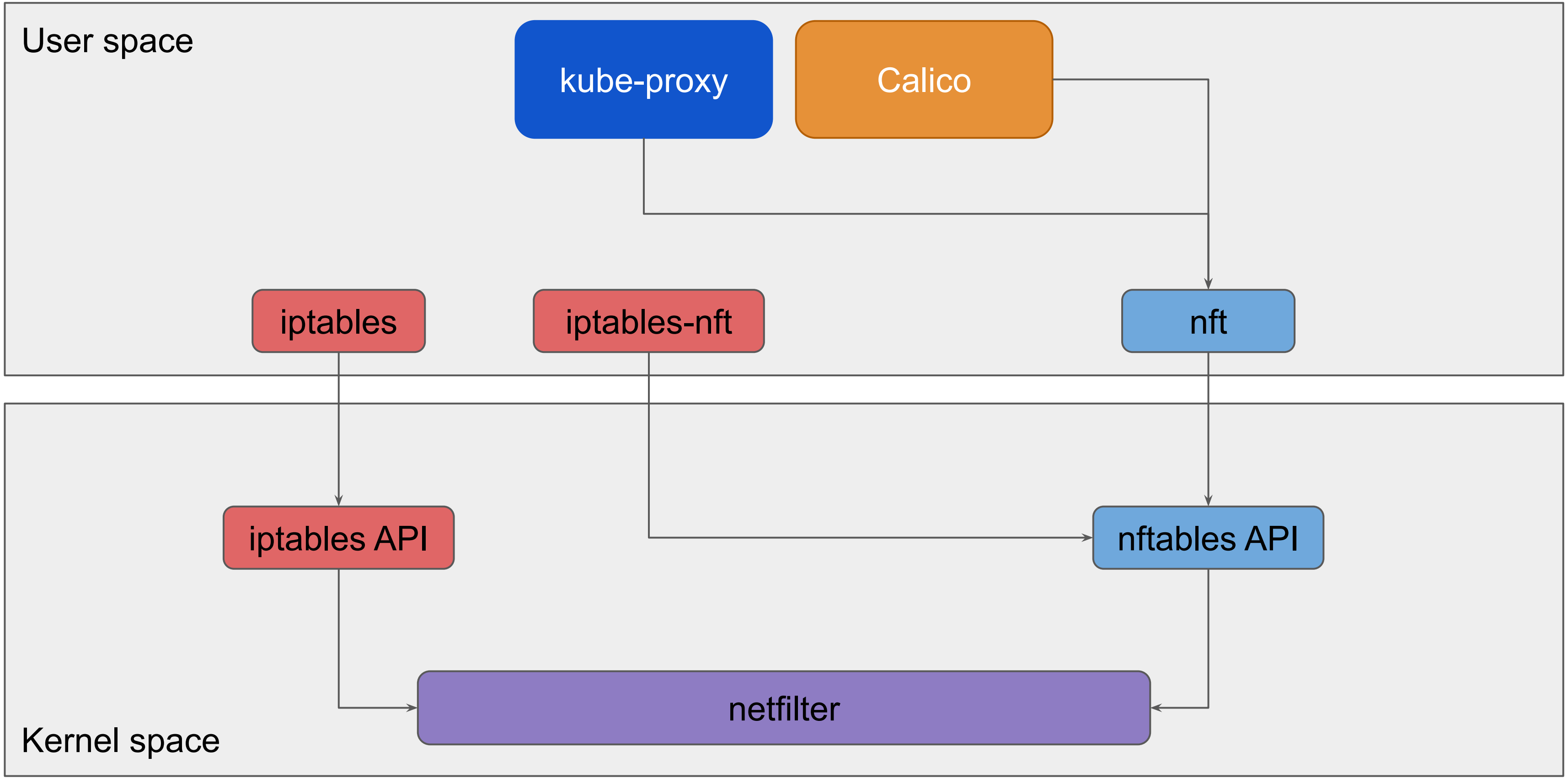
Evolution - iptables-nft transition (2018-2019)



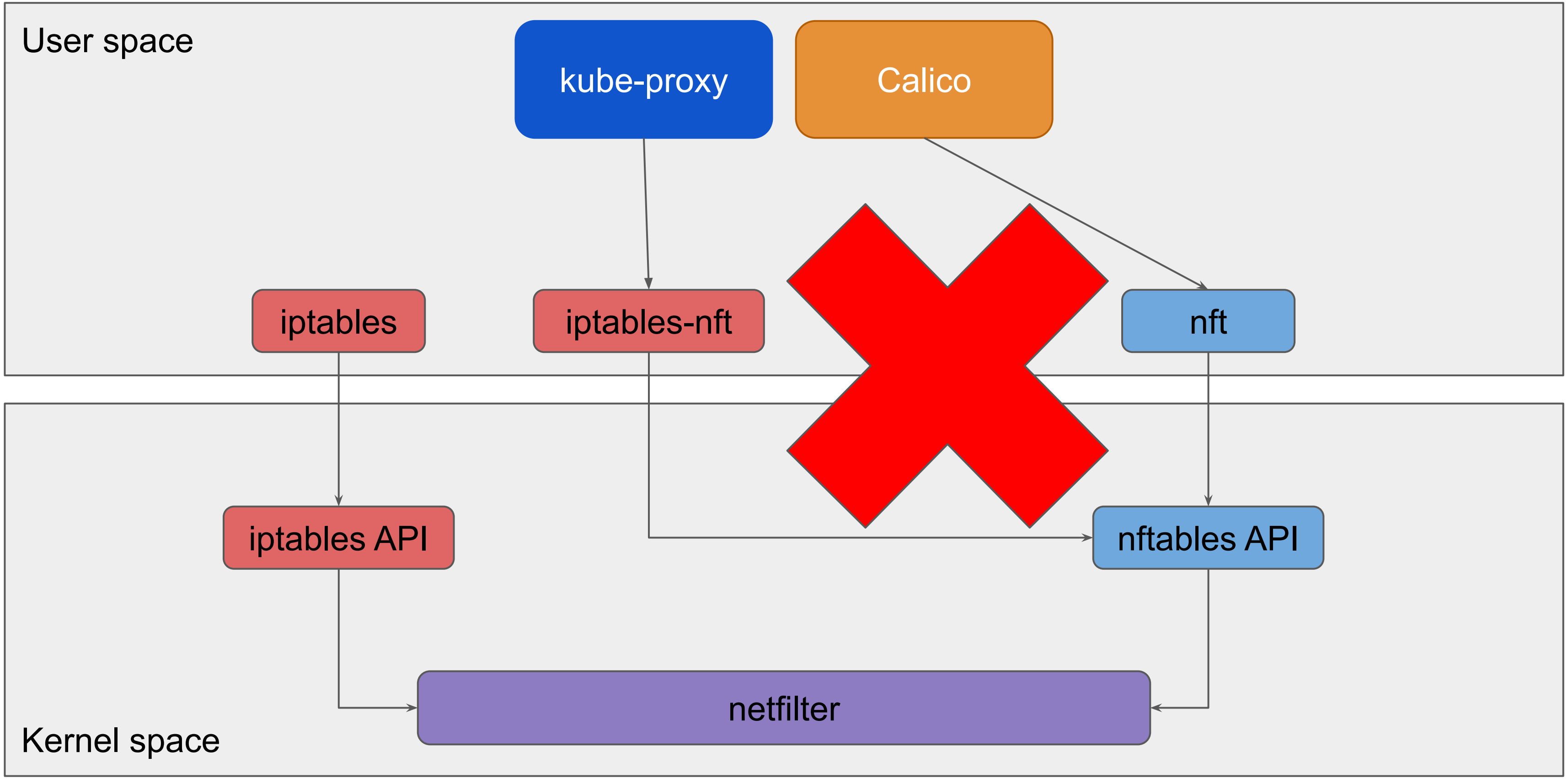
Evolution - The Present (2019-2024)



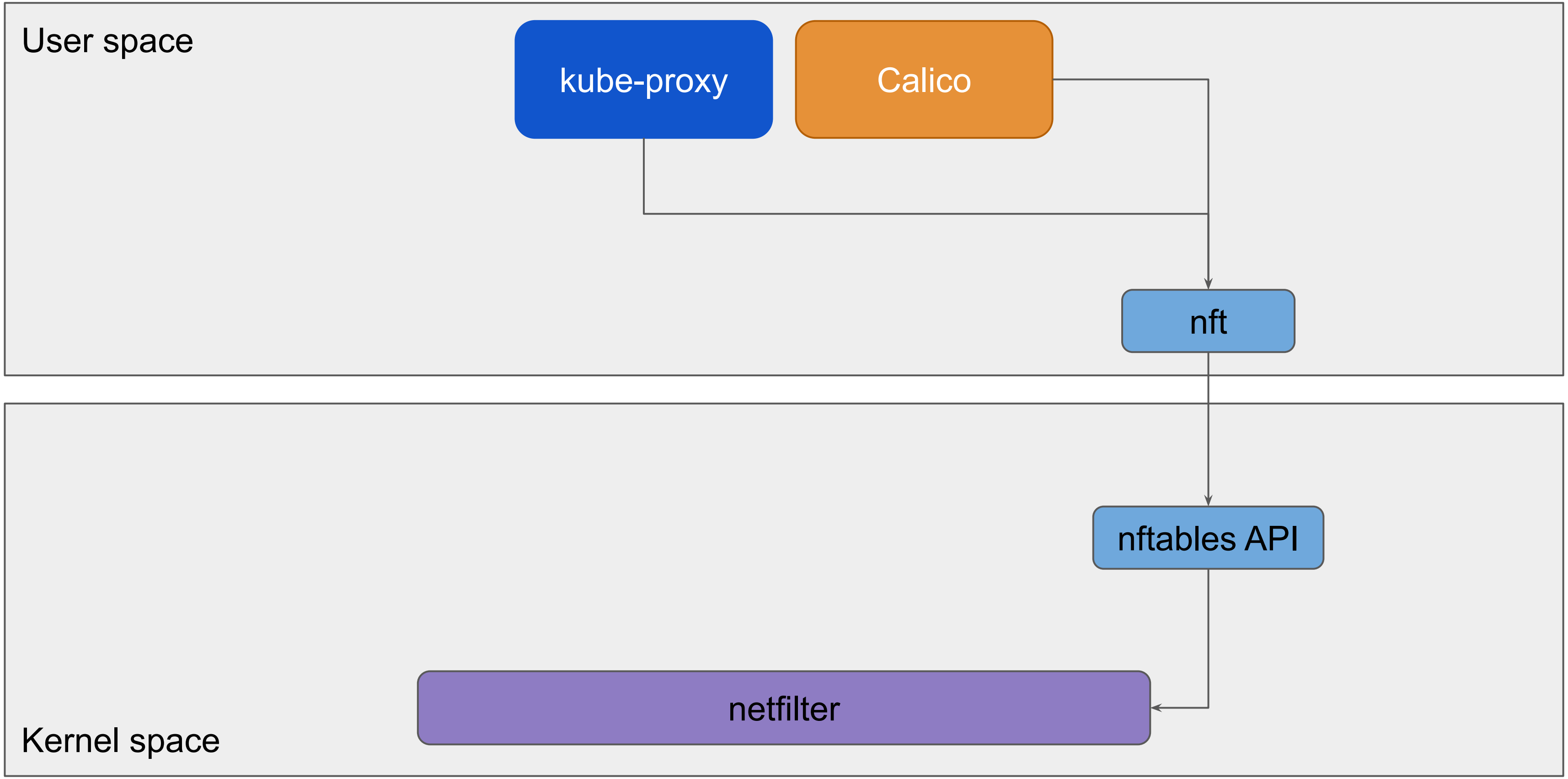
Evolution - The Present (2024)



Evolution - iptables → nftables transition



Evolution - The Future (?)



Why are we ditching iptables?

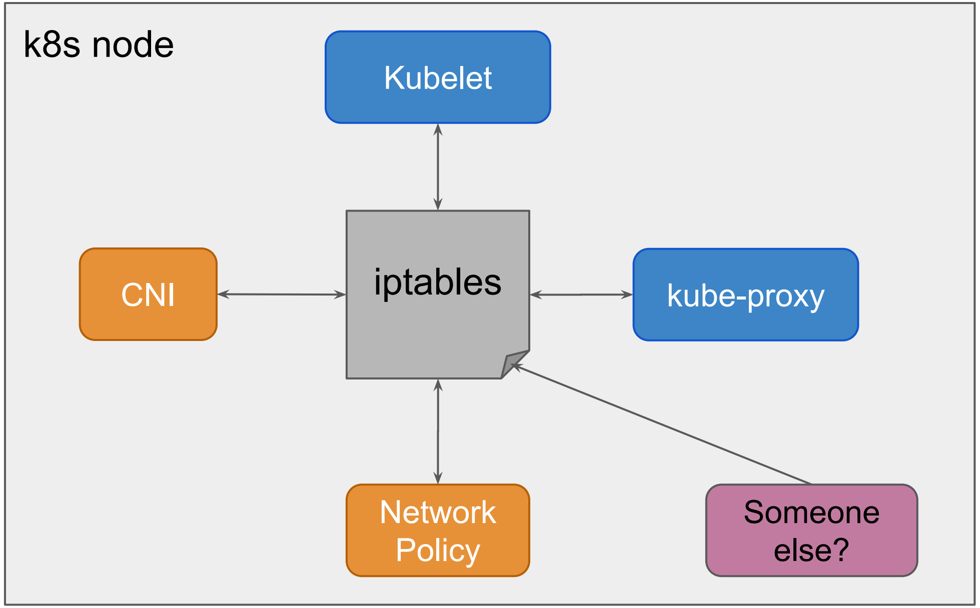
What's wrong with iptables?

- iptables API not designed for Kubernetes scale
- To change a single rule:
    a. Acquire node-wide lock
    b. Download the entire ruleset from kernel
    c. Make your change
    d. Upload the entire ruleset to the kernel
    e. Release the lock
- For large rulesets, this can be **very** slow, and it blocks other iptables users while you're doing it.

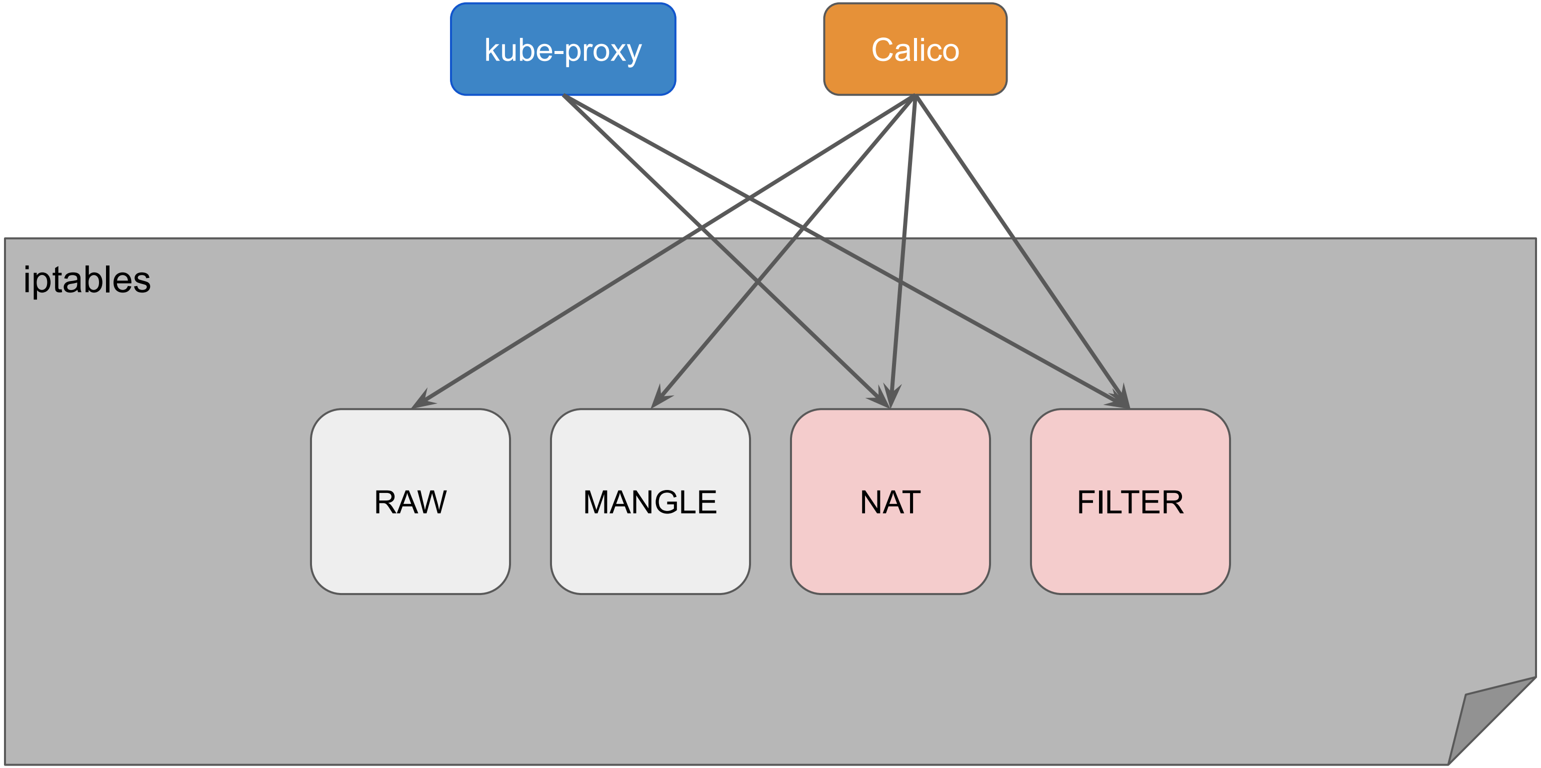
What's wrong with iptables?

iptables API not suited for multiple users

- Agree on tool version
- Everyone shares the same tables and top-level chains
- Fighting over chain ordering
- Fighting for global lock



iptables contention



Sharing iptables

- Pre-defined “hook” chains, with hard-coded priorities
  - e.g., `PREROUTING`, `INPUT`, `FORWARD`, `OUTPUT`, `POSTROUTING`
  - Users of iptables must share these
  - Difficult to manage ordering within these top-level chains
    - e.g., what about custom user-defined rules?

```
-A FORWARD -j cali-FORWARD
-A FORWARD -j KUBE-PROXY-FIREWALL
-A FORWARD -j KUBE-FORWARD
-A FORWARD -j KUBE-SERVICES
-A FORWARD -j KUBE-EXTERNAL-SERVICES
-A FORWARD -m mark --mark 0x10000/0x10000 -j ACCEPT
```

iptables is legacy technology

- Development on iptables has largely stopped
- New distros are moving away from iptables
  - RHEL 9 (deprecated)
  - RHEL 10 (planned removal)
  - Debian 11 (iptables optional)
- New features and fixes instead going into **nftables**

nftables to the rescue!

nftables to the rescue

- The nftables API learned from these mistakes!
  - Allows for operations on individual rules / objects
  - Each component can have a different table
  - Each component can define their own hook chains, with distinct priorities
  - No global lock eliminates “noisy neighbor” issues
- Expanded feature set
  - Verdict maps turn O(n) lookups into O(1)
  - Multiple actions per rule allow for (small) consolidations
  - Flexible map and set constructs

Sharing nftables

- nftables allows for user-defined hooks in each table
  - Can precisely define the priorities of each project’s rules
  - Easy for users to add their own table + hooks
  - Can easily avoid conflicting hook priorities
- Drops take precedence!
  - Can’t bypass network policy drop decisions

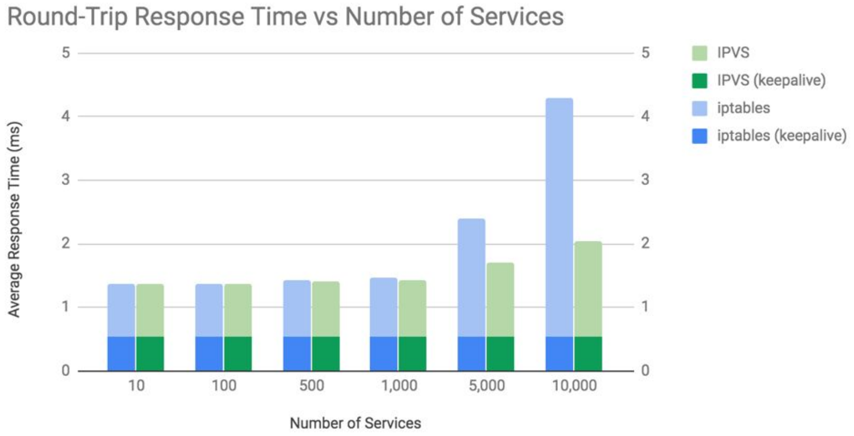
```
table ip kube-proxy {
  chain filter-forward {
    type filter hook forward priority -110; policy accept;
    ct state new jump service-endpoints-check
    ct state new jump cluster-ips-check
  }
}
```

```
table ip calico {
  chain filter-FORWARD {
    type filter hook forward priority 0; policy accept;
    jump filter-cali-FORWARD
    meta mark & 0x00010000 == 0x00010000 accept
  }
}
```

Alternatives?

What about IPVS?

- IPVS is a kernel-based load balancer implementation.
- kube-proxy has had a backend based on IPVS for several years.
- It definitely has better performance than iptables mode in larger clusters:



from https://www.tigera.io/blog/comparing-kube-proxy-modes-iptables-or-ipvs/

What about IPVS?

- But overall IPVS isn’t really the right API for a Kubernetes service proxy or network plugin.
- IPVS *only* does load balancing, but we need more than that.
  - kube-proxy can’t use *just* IPVS because it also needs to drop some packets, masquerade some packets (but not others), etc.
  - Calico can’t use IPVS at all for what it’s doing (NetworkPolicy, etc).
- Kubernetes wants Service proxying to happen in a very specific way.
  - Topology, traffic policy, etc, make the implementation more complicated
- Some of the best features of IPVS can’t be used effectively in a distributed multi-node service proxy anyway.

Why not eBPF?

- eBPF is great when you need to do things the kernel doesn't have native APIs for, but the kernel has perfectly good APIs for routing and rewriting packets.
- eBPF is a software Swiss Army Knife. It's great for doing small things quickly, but it's not designed for massive engineering projects.
- eBPF doesn't help performance as much as you think (and where it does, nftables flowtables can do some of the same things).

Why not eBPF?

- eBPF is, like, a whole *thing*.
- The tooling / libraries are notoriously complicated to work with. The documentation is often lacking.
- Many eBPF features require a very very recent kernel.
- There's no support for Go-to-eBPF translation; code would have to be written in C or Rust.
- … so it would be hard for SIG Network to suddenly shift kube-proxy to eBPF.
    - (But Calico has an eBPF backend.)

Alternatives?

- IPVS would probably be great as the basis for an implementation of Gateway API…
- eBPF is great for probes and observability, and can be used for bigger things if you know what you’re doing.
- OVS is great if you’re building a whole data plane, but hard to use as a single piece of a larger data plane (e.g., kube-proxy).
- nftables is the most general-purpose Linux network stack API, and is probably a good choice for most use cases.

Performance: Fixes

kube-proxy rule optimization

iptables: O(# Services)

```
iptables -A KUBE-SERVICES -d 172.30.12.34 -j KUBE-SVC-ABCD
iptables -A KUBE-SERVICES -d 172.30.34.56 -j KUBE-SVC-EFGH
iptables -A KUBE-SERVICES -d 172.30.56.78 -j KUBE-SVC-IJKL
iptables -A KUBE-SERVICES -d 172.30.78.89 -j KUBE-SVC-MNOP
. . .
```

- Match rule per-Service (O(n))
- NAT chain per endpoint

nftables: O(1)

```
nft add rule ip daddr vmap @services

map services {
    172.30.12.34 : jump KUBE-SVC-ABCD
    172.30.34.56 : jump KUBE-SVC-EFGH
    172.30.56.78 : jump KUBE-SVC-IJKL
    172.30.78.89 : jump KUBE-SVC-MNOP
}
```

- A single rule that does an O(1) map lookup.
- NAT chain per endpoint (for now?)

Calico rule optimization

iptables: O(# local pods)

```
iptables -A cali-fw-dispatch -i caliABC
iptables -A cali-fw-dispatch -i caliBCD
iptables -A cali-fw-dispatch -i caliEFG
iptables -A cali-fw-dispatch -i caliHIJ
. . .
```

- Match rule per local pod
- 1+ chains per-policy applied to local pods

nftables: O(1)

```
nft add rule iifname vmap @cali-fw-dispatch

map cali-fw-dispatch {
    caliABC : jump cali-fw-ABC
    caliBCD : jump cali-fw-BCD
    caliEFG : jump cali-fw-EFG
    caliHIJ : jump cali-fw-HIJ
}
```

- A single match rule
- 1+ chains per-policy applied to local pods

kube-proxy rule updates

iptables

```
iptables -A KUBE-SERVICES -d 172.30.12.34 -j KUBE-SVC-ABCD
iptables -A KUBE-SERVICES -d 172.30.34.56 -j KUBE-SVC-EFGH
iptables -A KUBE-SERVICES -d 172.30.78.89 -j KUBE-SVC-MNOP
. . .
iptables -A KUBE-SERVICES -d 172.30.99.11 -j KUBE-SVC-WXYZ
iptables -X KUBE-SVC-IJKL
```

- Always has to rewrite entire KUBE-SERVICES chain due to how `iptables-restore` works

nftables

```
nft add element ip kube-proxy services \
  { 172.30.99.11 : jump KUBE-SVC-WXYZ }

nft delete element ip kube-proxy services \
  { 172.30.56.78 }
nft delete chain ip kube-proxy KUBE-SVC-IJKL
```

- Can add and remove individual elements and chains.

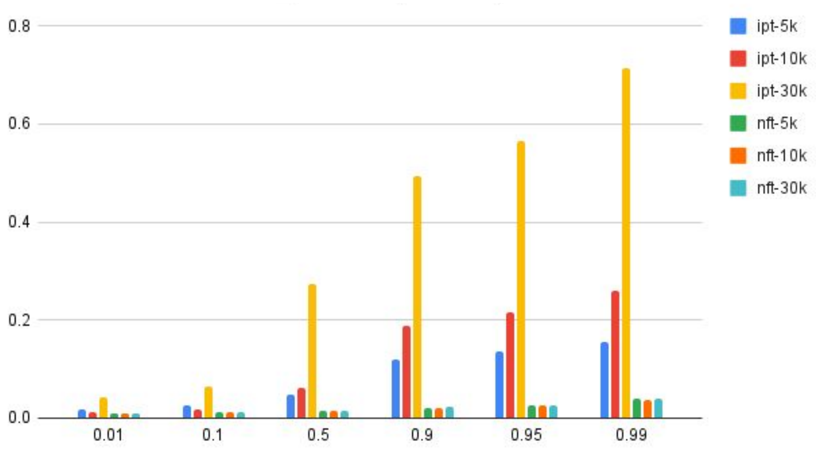
Performance: Results

Time to program / Resource utilization

- Initial tests with kube-proxy initial startup in a large cluster showed it was a few times *slower* than iptables when doing a full sync of a large ruleset.
  - Already improved with very latest `nft` binaries.
  - We may work around this in kube-proxy.
- In the gce-100Nodes performance test, iptables programming latency is around 10s, while in nftables-100Nodes it's about 1.5s… but this is apples-to-oranges…
  - The iptables version uses "`minSyncPeriod: 10s`" because otherwise the iptables updates used up too much CPU. The nftables 100 Node test doesn't need to do that.

Dataplane latency

- p50 latency for nftables is much lower than iptables, especially at larger scale (yellow/cyan = 30,000 Services)
- iptables has *much* larger variance between p50 (center) and p99 (right)



(Graph from [Nadia and Antonio's talk] "From Observability to Performance".)

Dataplane latency

- The iptables latency is mostly because of the number of rules in the `nat` table.
- In the past, large increases to the number of iptables rules in the `filter` table have resulted in total network meltdowns at scale (#56164, #95252), forcing us to revert and redesign the ruleset.
- This blocked us from being able to implement some features (like rejecting connections to unused Service IP ports). In the nftables backend, we can implement features like this without problems.

Dataplane throughput

- Base throughput is similar for iptables and nftables.
- One optimization we are experimenting with in nftables kube-proxy is the use of "flow tables"
  - Once you figure out where a connection is going, you can add it to a flowtable, providing a "fast path" for further packets, bypassing much of the network stack.
  - Packets are diverted at "Ingress" phase - before prerouting hooks
- Can be offloaded to (some) hardware
- Proof-of-concept PRs have been posted
  - kubernetes #128392 for kube-proxy -- shows a ~15% boost
  - projectcalico/calico #9458 for Calico PoC

Performance

- We've had 8 years of optimization for iptables mode, and 8 *months* of optimization for nftables, so this is a work in progress…
- Programming time should generally be better, because of differences in locking, incremental updates, etc.
  - This will let people get rid of "`minSyncPeriod`" in large clusters, so that updates happen sooner.
- Packet latency should be improved by O(1) rather than O(n) rulesets.
- We can also do things with nftables that just wouldn't work with iptables.

KubeCon | CloudNativeCon

North America 2024