KubeCon CloudNativeCon
North America 2024

From Observability to Performance

Nadia Pinaeva, Red Hat
Antonio Ojea, Google

KubeCon CloudNativeCon

North America 2024

Nadia Pinaeva

Senior Software Engineer

Red Hat

Antonio Ojea

Staff Software Engineer

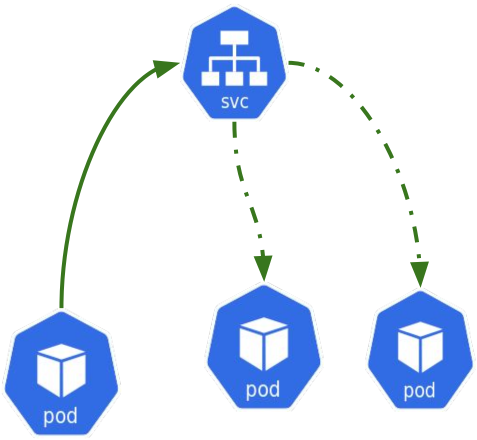
Google

Kubernetes Services

- Network performance is a critical aspect of any system, especially in containerized environments

- Kubernetes Services provide Service Discovery and Load Balancing that allows to decouple the application logic from the underlying infrastructure

- Understanding the performance of Kubernetes Services is crucial for both users and implementers

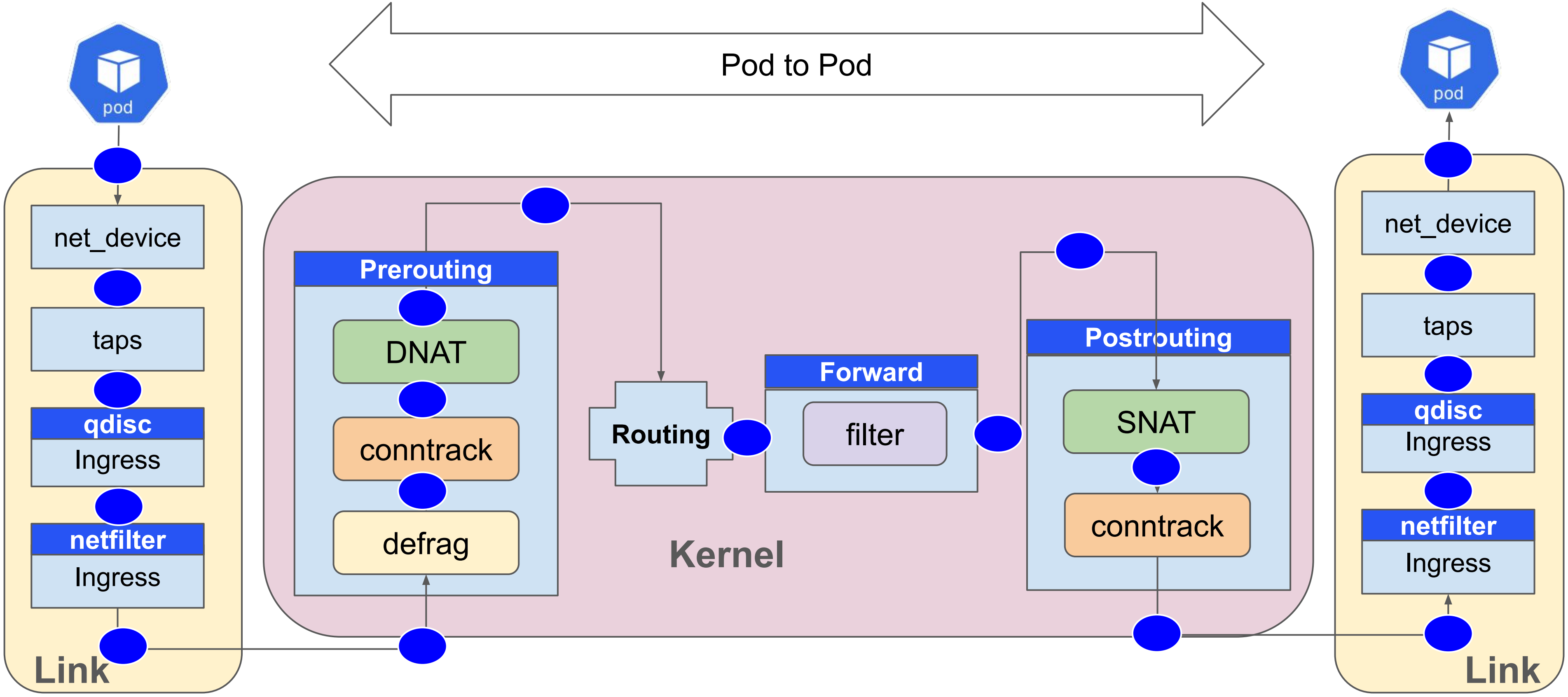



```yaml
apiVersion: v1
kind: Service
metadata:
  name: service
spec:
  clusterIP: 10.96.0.1
  clusterIPs:
  - 10.96.0.1
  internalTrafficPolicy: Cluster
  ipFamilies:
  - IPv4
  ipFamilyPolicy: SingleStack
  ports:
  - name: https
    port: 443
    protocol: TCP
    targetPort: 6443
  type: ClusterIP
```

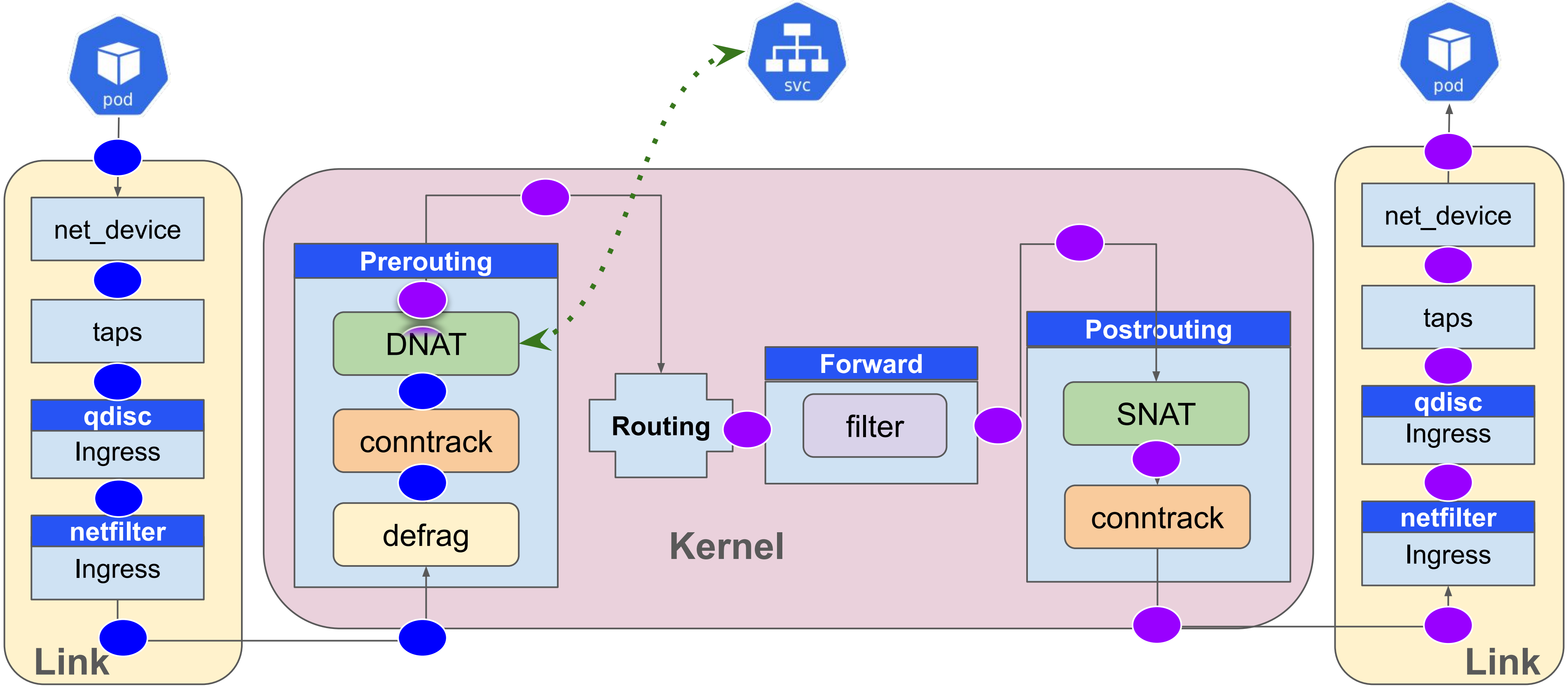
(2) No matter how hard you push and no matter what the priority, you can't increase the speed of light.

RFC 1925: The Twelve Networking Truths

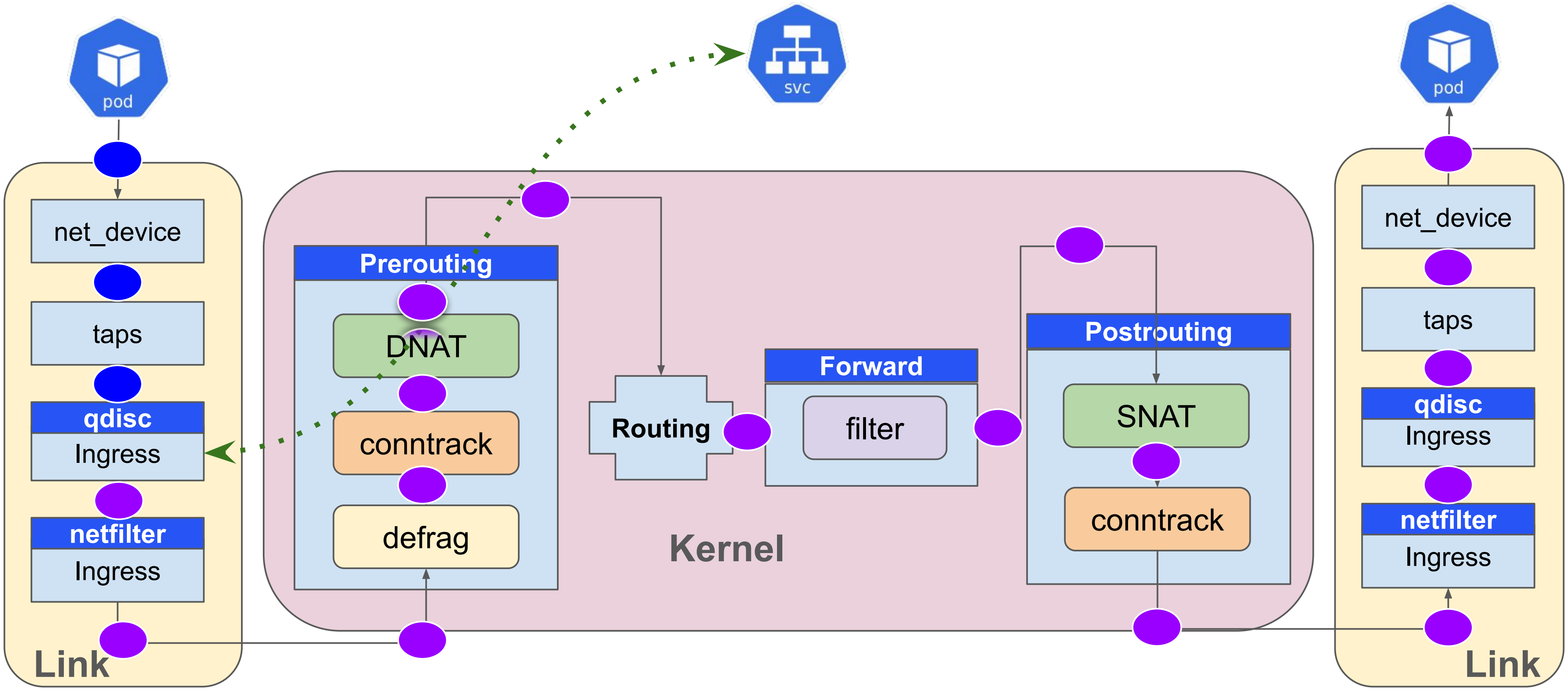
The Complexity of Kubernetes Networking



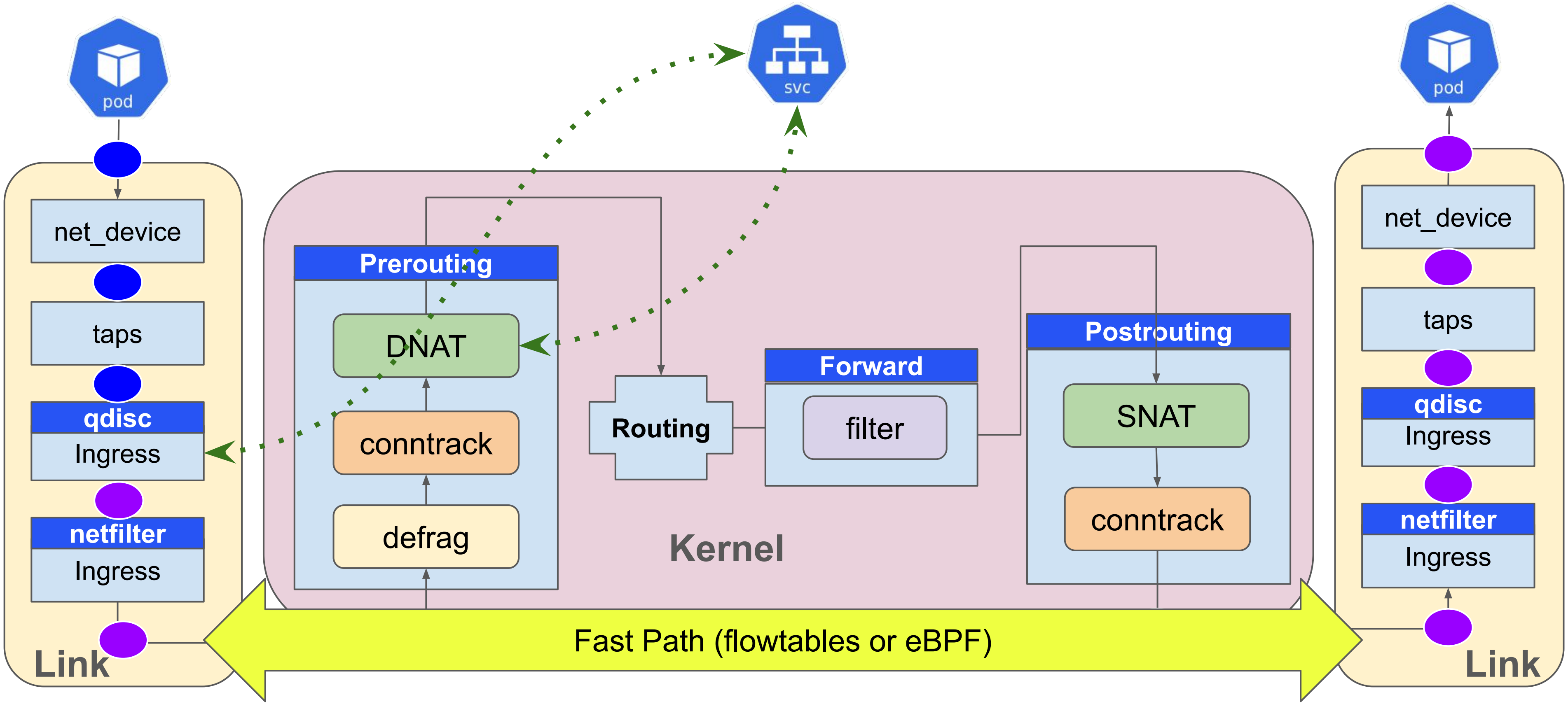
The Complexity of Kubernetes Networking



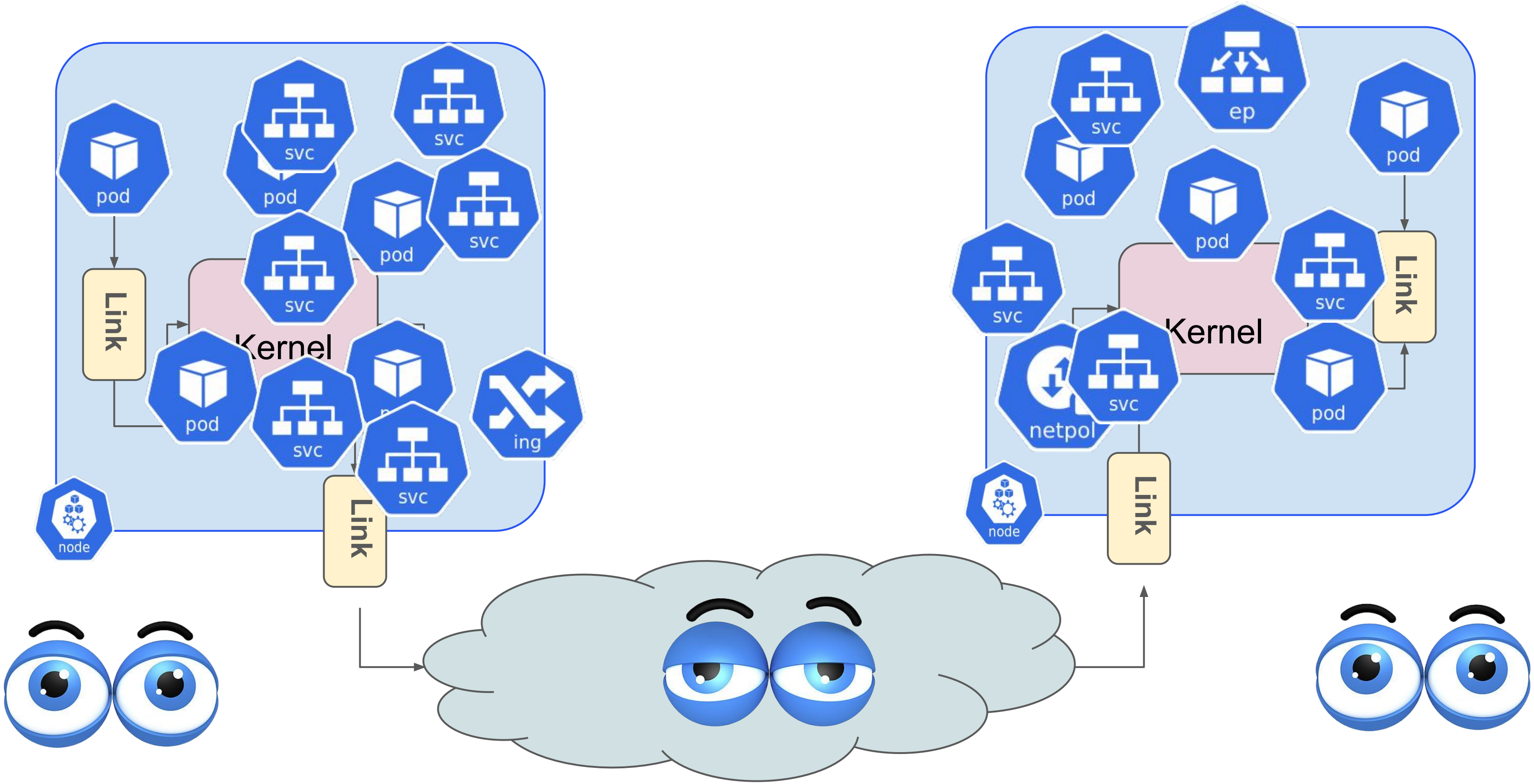
The Complexity of Kubernetes Networking



The Complexity of Kubernetes Networking



Challenges in Measuring Network Performance



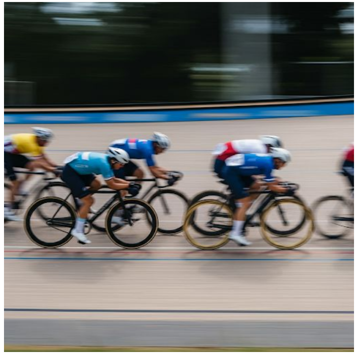
The Power of Service Level Indicators (SLIs)

Programming Latency

time to update the dataplane with the latest configuration

First Packet Latency

time between first packet is sent from the client and first reply packet is received



Connection Total Latency

time between first and last packet of a connection

Throughput

rate of successful data transfer

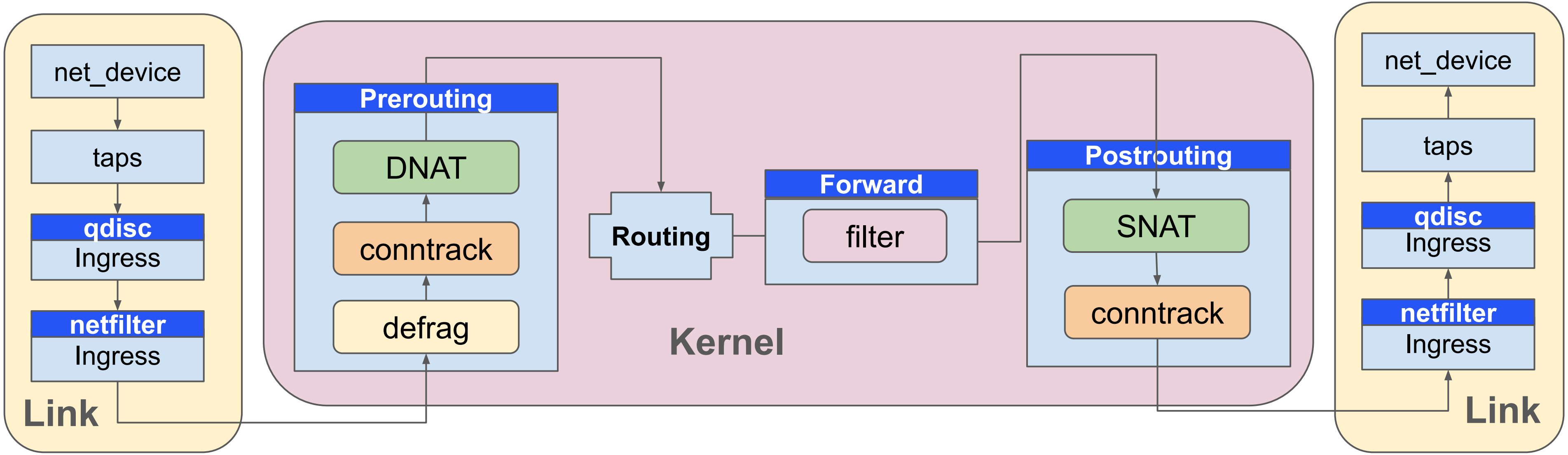
Moving Beyond Synthetic Tests

- Custom benchmarks often use unrealistic workloads and configurations.
- SLIs provide a more accurate picture of performance in real-world production environments.
- Standardized SLIs enable meaningful comparisons across different environments and deployments.

KubeCon | CloudNativeCon
North America 2024

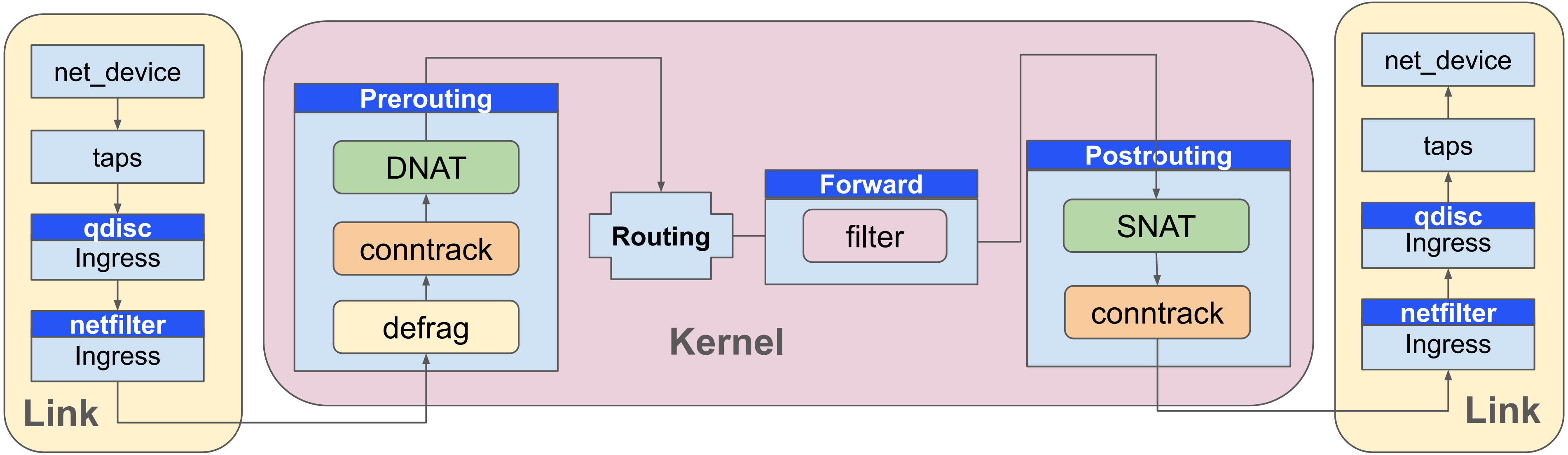
User-Driven Performance Validation

Instrumenting the network: conntrack events



Conntrack generates events for every connection. What can we find there?

Instrumenting the network: conntrack events

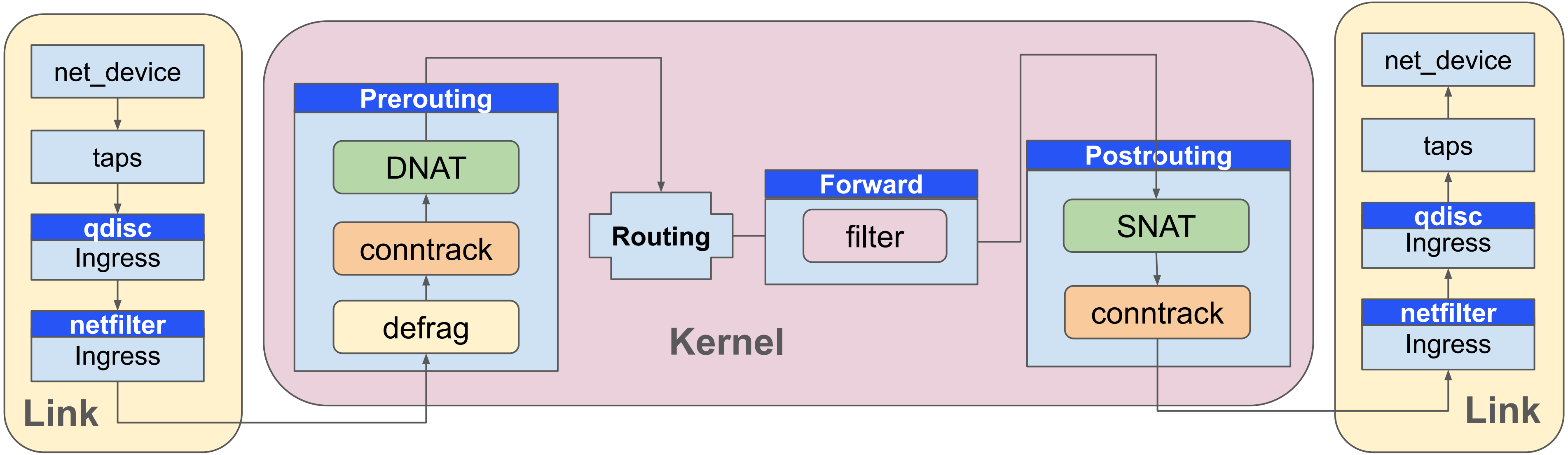


Conntrack generates events for every connection. What can we find there?

- **start** - first packet of a connection is seen
- **seen_reply** - first reply packet of a connection is seen

first packet latency

Instrumenting the network: conntrack events

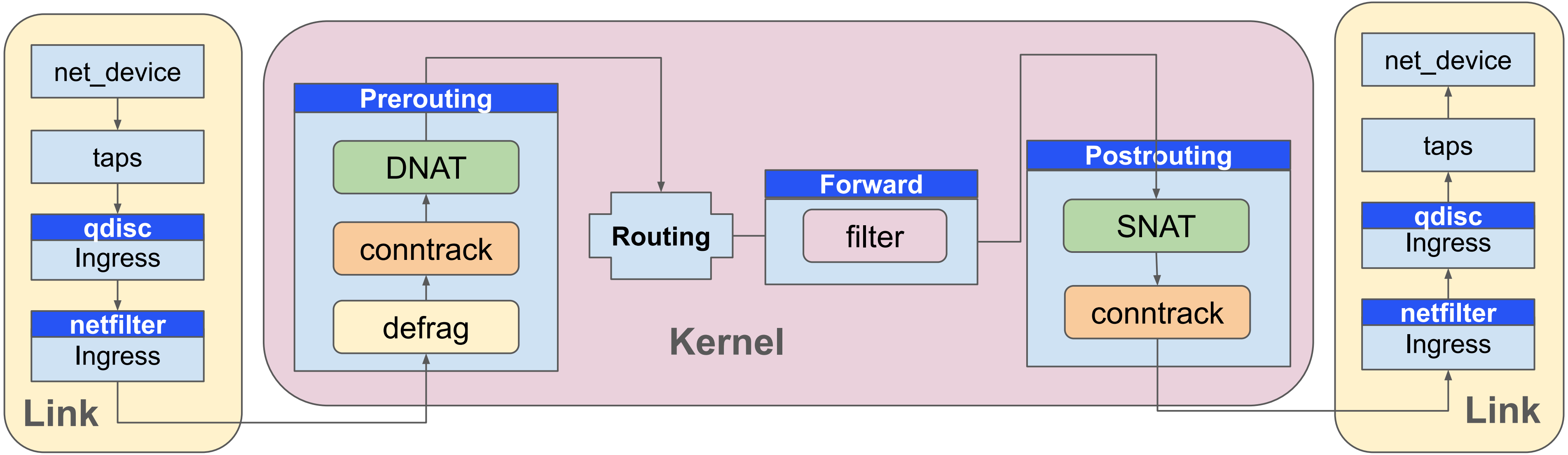


Conntrack generates events for every connection. What can we find there?

- **start** - first packet of a connection is seen
- seen_reply - first reply packet of a connection is seen
- **TCP_FIN** - first TCP connection close packet is seen

total connection latency

Instrumenting the network: conntrack events



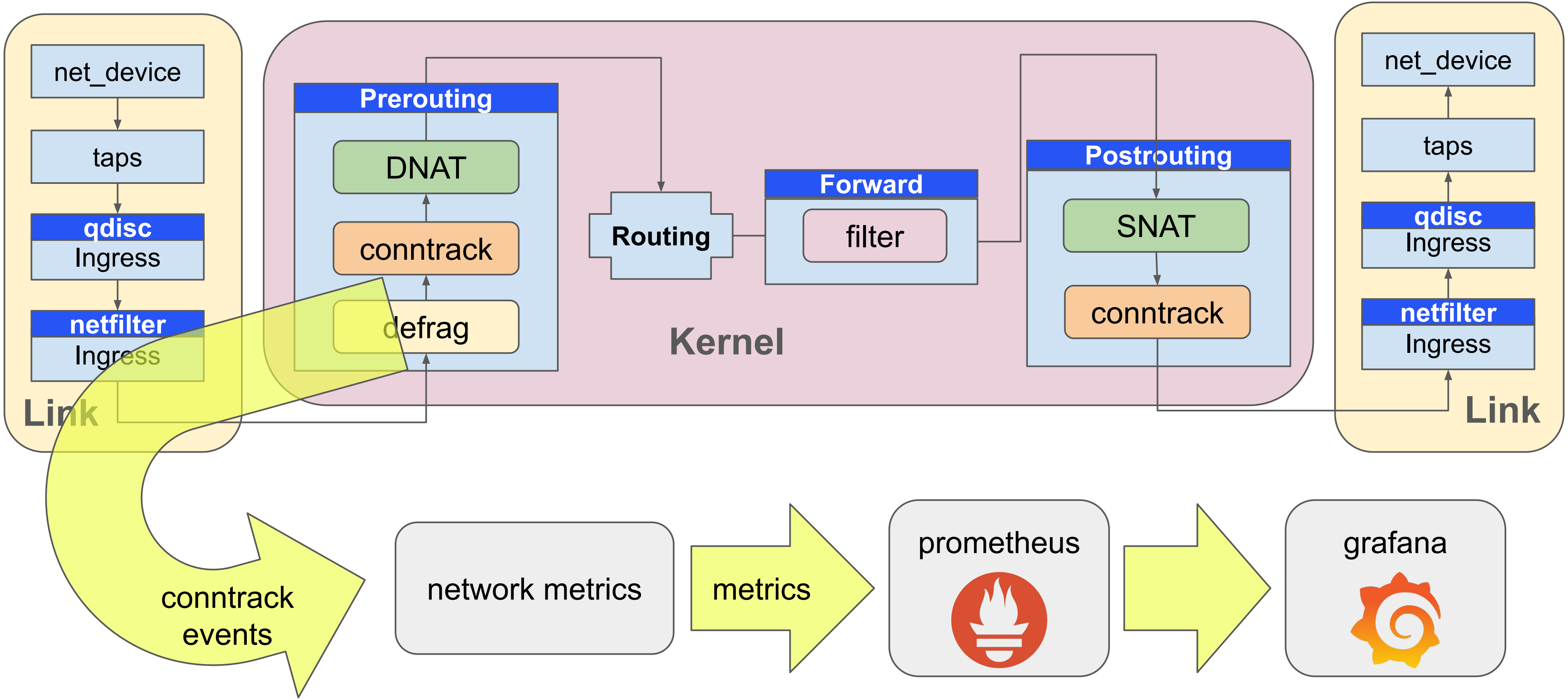
Conntrack generates events for every connection. What can we find there?

- start - first packet of a connection is seen
- seen_reply - first reply packet of a connection is seen
- TCP_FIN - first TCP connection close packet is seen
- **bytes/packets** - amount of data sent/received when connection dies

total connection latency

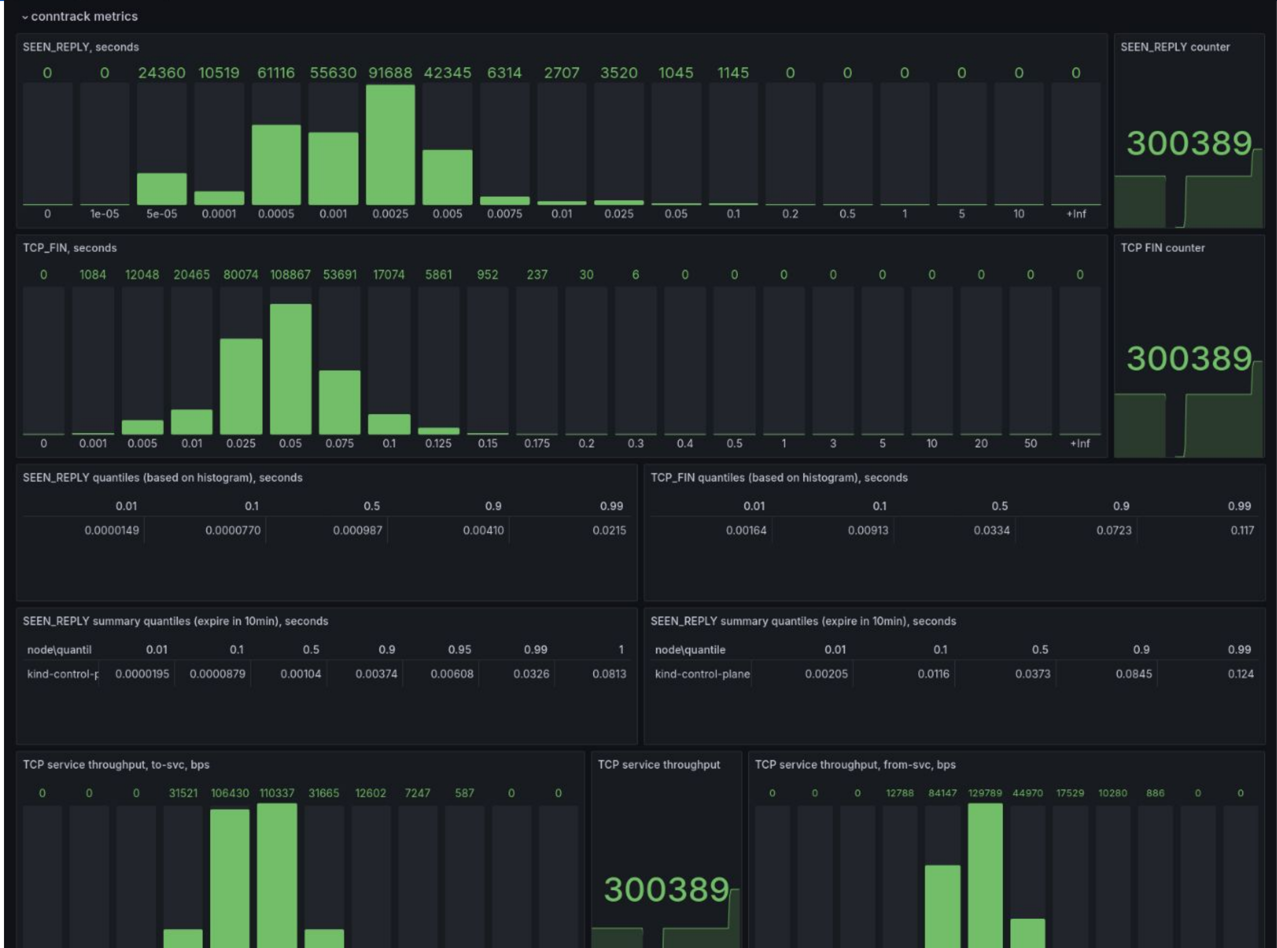
throughput

Instrumenting the network: conntrack events



conntrack metrics

- SEEN_REPLY time, sec
- TCP_FIN time, sec
- TCP throughput, bps



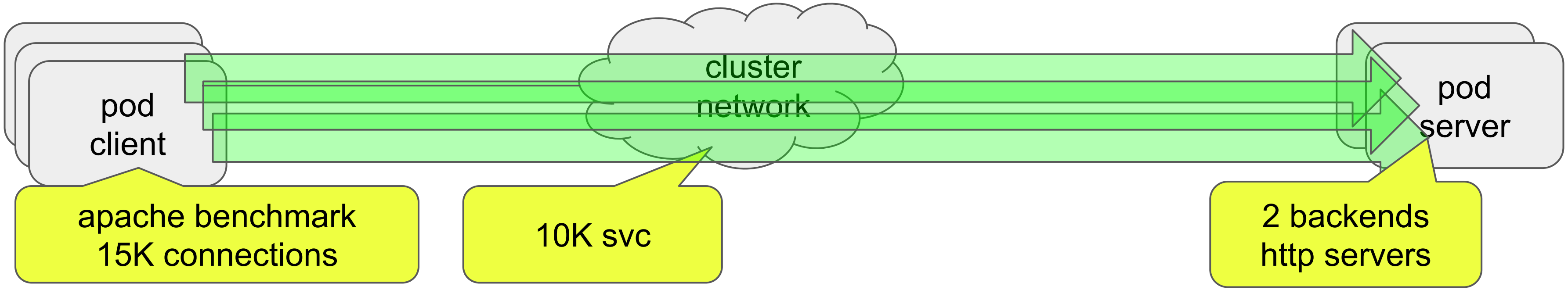
Test setup

- c2-standard-30 VM
- kind 0.24.0
- k8s 1.31.0
  - 1 node

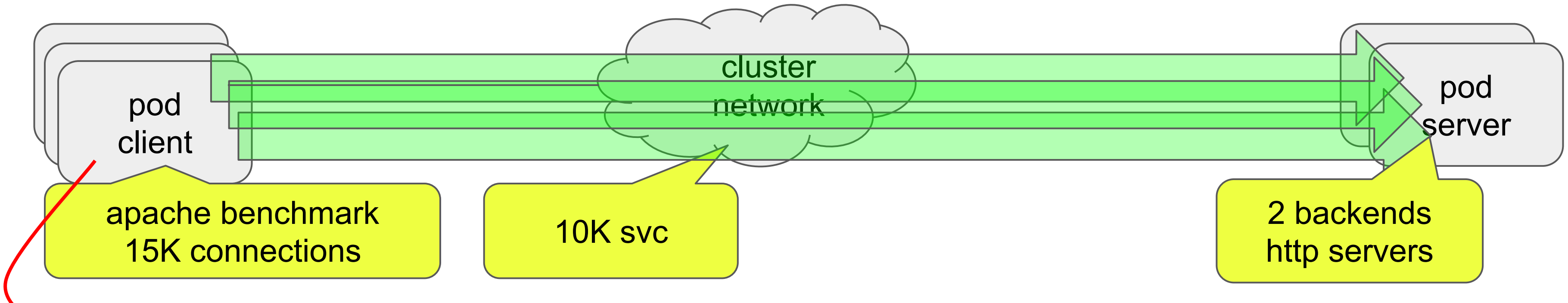
Workload:

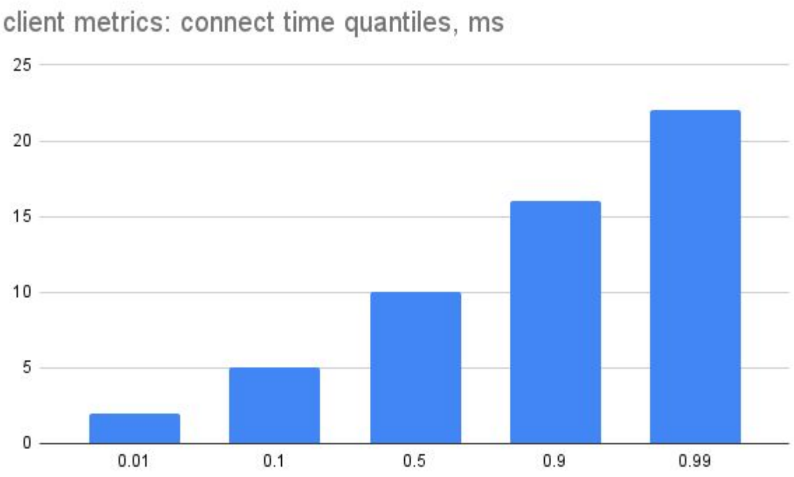
- 2 backends
  - agnhost http server
- 10 clients
  - apache benchmark
  - concurrency = 100
  - connections per client = 15000
  - total connections = 150000

client vs network metrics

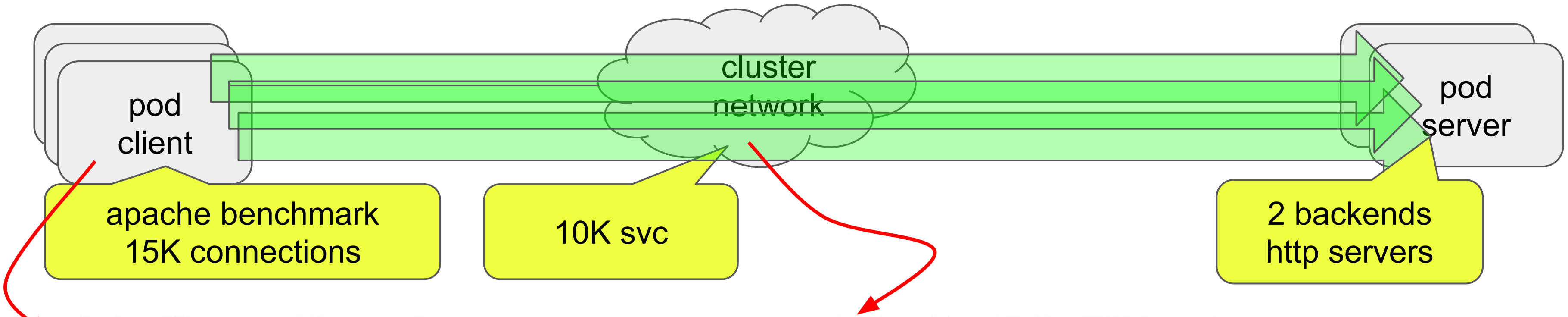


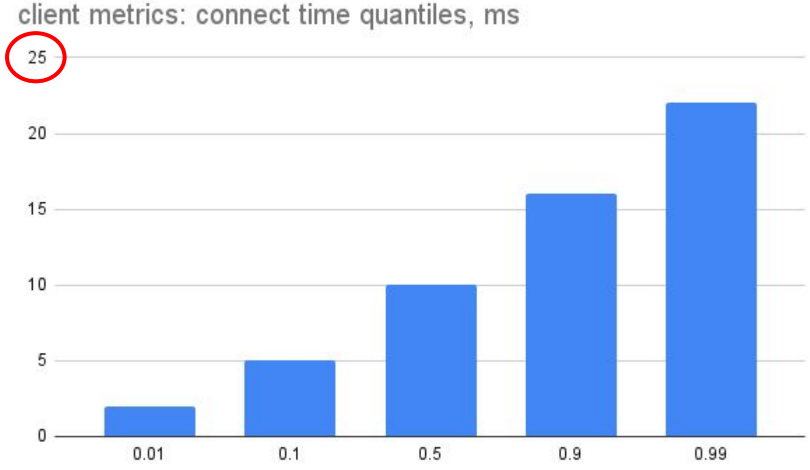
client vs network metrics

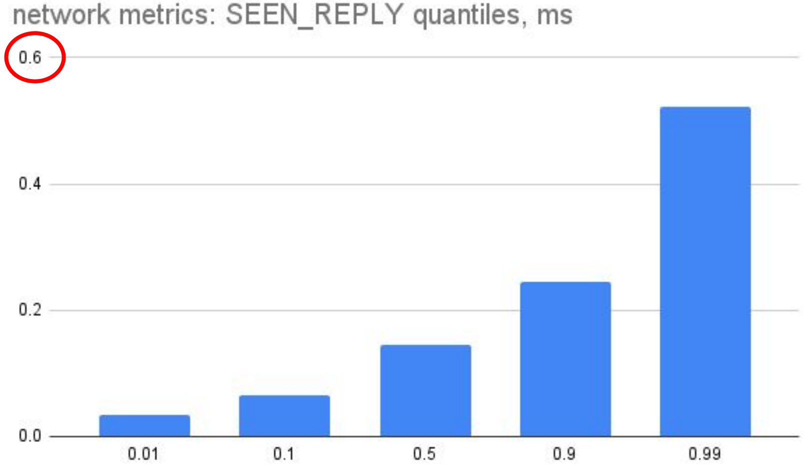




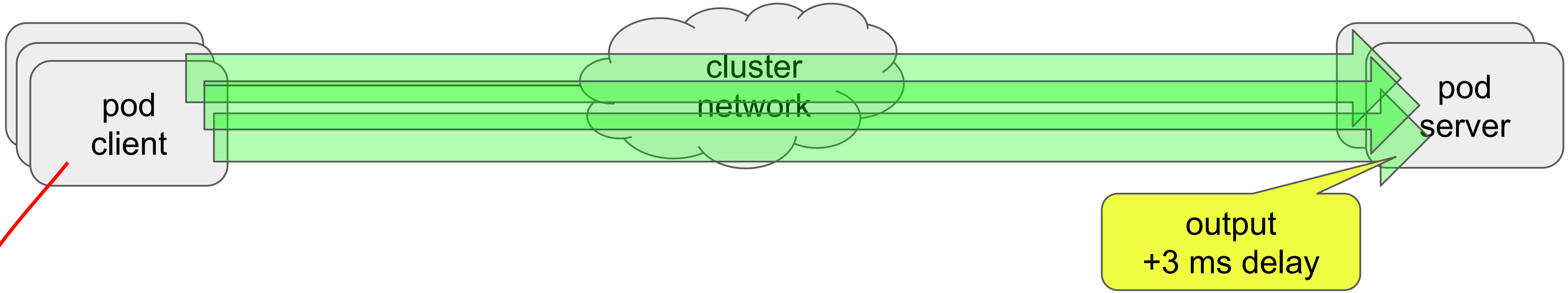
client vs network metrics

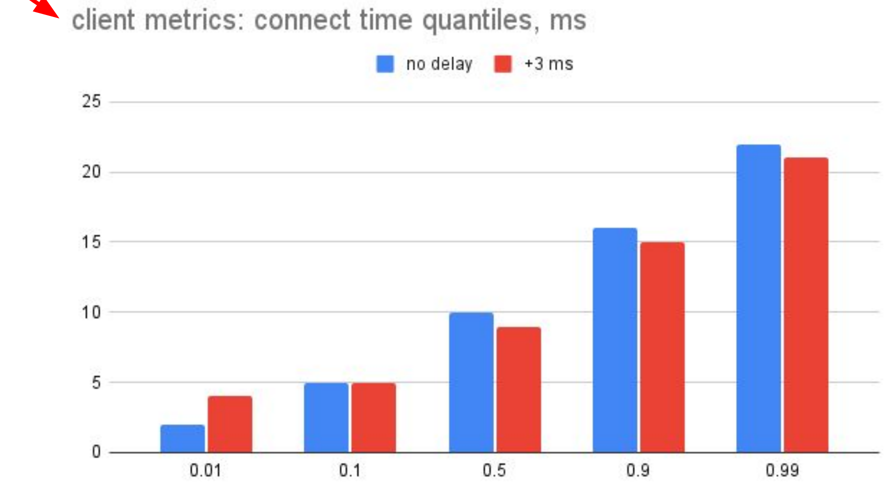




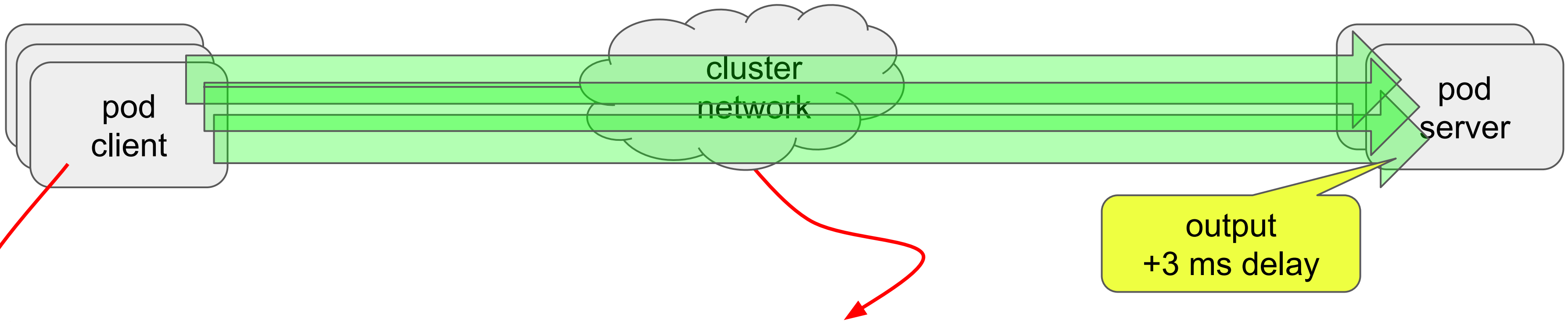


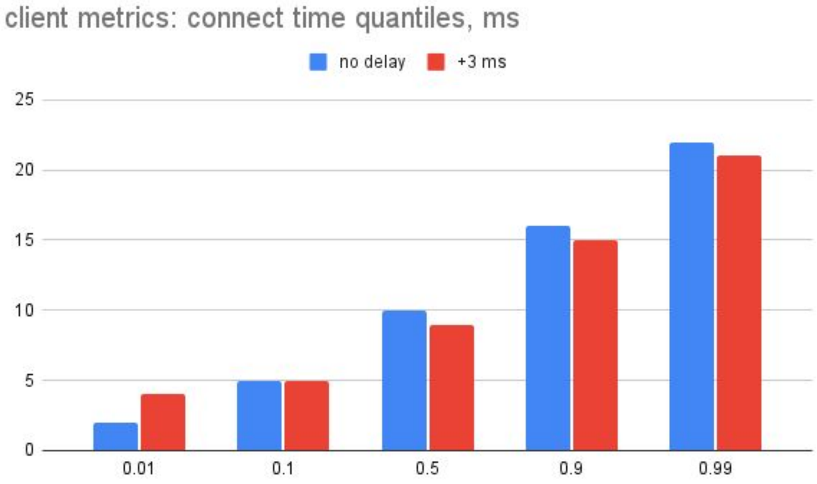
client vs network metrics

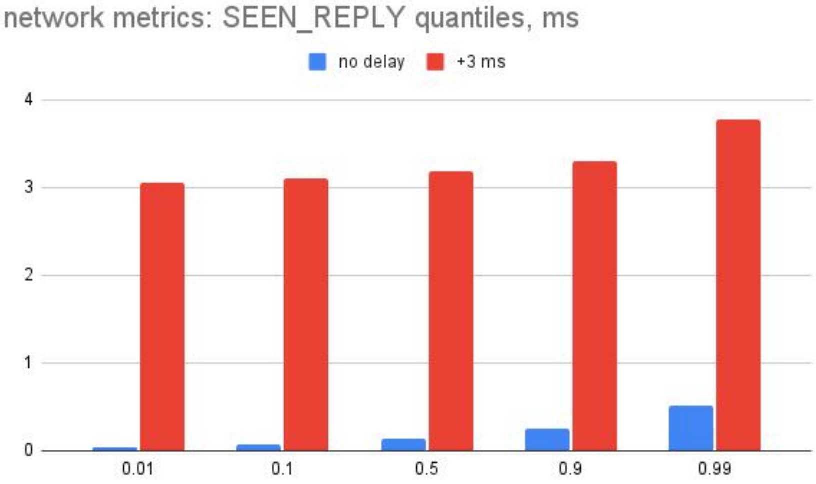




client vs network metrics







Outcomes

- **every benchmarking/measuring tool has its limitations**

KubeCon | CloudNativeCon

North America 2024

Bridging the Gap

From Monitoring Insights to Kube-proxy Enhancements

iptables vs nftables kube-proxy

How the Tables Have Turned: Kubernetes Says Goodbye to Iptables

Casey Davenport, Tigera & Dan Winship, Red Hat

Thursday November 14, 2024 2:30pm - 3:05pm MST

iptables vs nftables kube-proxy

Theory: iptables performance is limited mainly by two reasons:

- **Programming latency** caused by the need to save and restore all the lines to the kernel in each transaction
- **Latency on the first packet** of a connection caused by the linear search rule matching

iptables vs nftables kube-proxy

Theory: iptables performance is limited mainly by two reasons:

- **Programming latency** caused by the need to save and restore all the lines to the kernel in each transaction
- **Latency on the first packet** of a connection caused by the linear search rule matching

Workload to compare nftables vs iptables:

- Create **a ServiceA with 100k endpoints** (no need to create pods), measure the time to program the dataplane

iptables vs nftables kube-proxy

Theory: iptables performance is limited mainly by two reasons:

- **Programming latency** caused by the need to save and restore all the lines to the kernel in each transaction
- **Latency on the first packet** of a connection caused by the linear search rule matching

Workload to compare nftables vs iptables:

- Create **a ServiceA with 100k endpoints** (no need to create pods), measure the time to program the dataplane
- Create a **second ServiceB** with a real http server backend, ensure this service rules are evaluated after the first service

- ServiceA (1.1.1.1)
  - backend1
  - ...
  - backend100K
- ServiceB (1.1.1.2)
  - backend1

iptables vs nftables kube-proxy

Theory: iptables performance is limited mainly by two reasons:

- **Latency on the first packet** of a connection caused by the linear search rule matching
- **Programming latency** caused by the need to save and restore all the lines to the kernel in each transaction

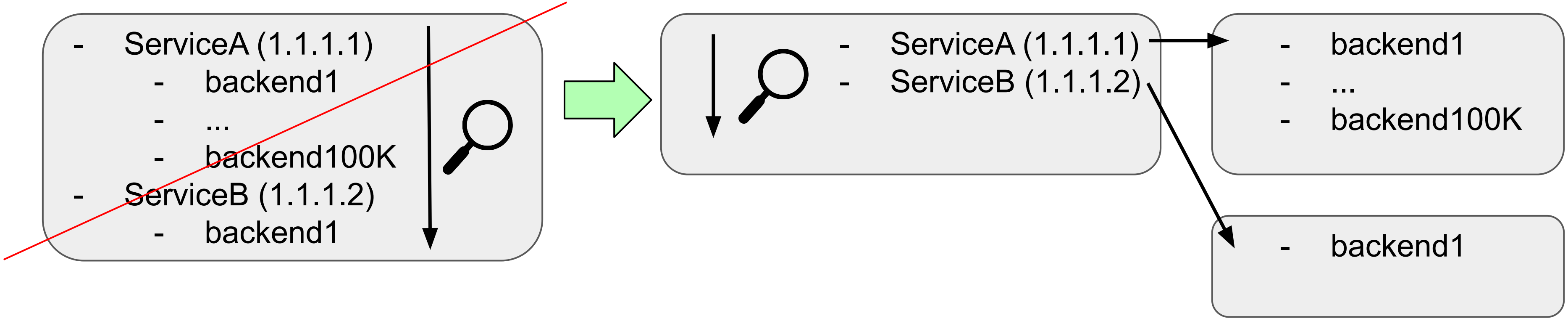
Workload to compare nftables vs iptables:

- Create **a ServiceA with 100k endpoints** (no need to create pods), measure the time to program the dataplane
- Create a **second ServiceB** with a real http server backend, ensure this service rules are evaluated after the first service

Conclusion

- ab with iptables times out
- kube-proxy **nftables** seems to **solve** the iptables scalability and performance problems.

iptables vs nftables kube-proxy



the linear search rule matching

- only grows with the number of services
- number of endpoints only affects a specific service chain

Create **a Service with 100k endpoints**

- doesn’t stress linear search
- only stresses one workload type for control plane

Outcomes

- every benchmarking/measuring tool has its limitations
- **designing test workload is hard. Think about your workloads and stress your cluster with that instead of copy pasting from internet**

iptables vs nftables kube-proxy

ab with iptables times out - WHY?

Let’s check metrics!

iptables vs nftables kube-proxy

ab with iptables times out - WHY?

Let's check metrics

- kubeproxy_sync_proxy_rules_iptables_total
  - Total number of iptables rules owned by kube-proxy

iptables vs nftables kube-proxy

ab with iptables times out - WHY?

Let's check metrics

- kubeproxy_sync_proxy_rules_duration_seconds
    - the latency of one round of kube-proxy syncing proxy rules.

iptables vs nftables kube-proxy

ab with iptables times out - WHY?

Let’s check metrics

- container_cpu_usage_seconds_total{container="kube-proxy"}

iptables vs nftables kube-proxy

Config time for a service with 100K endpoints: nftables **~1m15s** vs iptables **~20m**
What if we run ab again?

iptables vs nftables kube-proxy

Config time for a service with 100K endpoints: nftables **~1m15s** vs iptables **~20m**
What if we run ab again?

```
Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    4   1.4      4       7
Processing:     0    4   1.5      4       8
Waiting:        0    2   1.6      1       7
Total:          5    8   0.4      8      10

Percentage of the requests served within a certain time (ms)
  50%      8
  66%      8
  75%      8
  80%      8
  90%      8
  95%      9
  98%      9
  99%      9
 100%     10 (longest request)
```

same result as with nftables!

Outcomes

- every benchmarking/measuring tool has its limitations
- designing test workload is hard. Think about your workloads and stress your cluster with that instead of copy pasting from internet
- **confirmation bias is real**

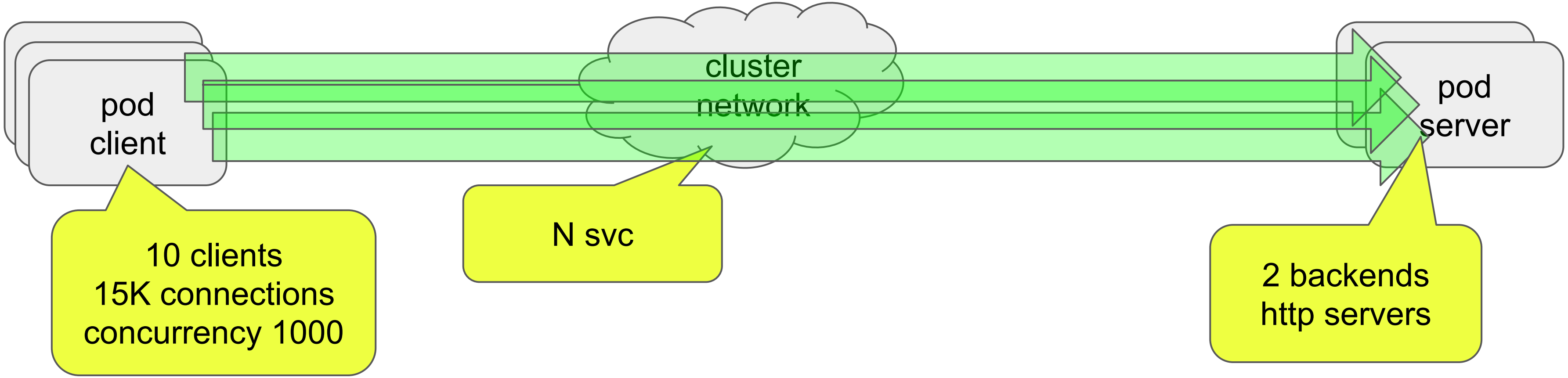
KubeCon | CloudNativeCon

North America 2024

Iterating Towards Improvement

Addressing Kube-proxy Performance Bottlenecks

iptables vs nftables: first packet latency



test variables:

- kube-proxy mode iptables vs nftables
- amount of services: 5K, 10K, 30K

measurements:

- client metrics and network metrics

iptables vs nftables kube-proxy

Now when we are done with large svc test, we want the number of services to be the test parameter.

We want to test kube-proxy performance on different service workloads, at the same time we want to minimize the impact of client/server performance. To achieve that, we will use the exact same amount of service backends and the same configuration for clients.

The main tests parameters are:

- **number of services** (**SERVICES**, using the same amount of backends)
- **number of clients** (**CLIENTS**, when number of services is more than number of clients, each client will ping its own service by picking equally distributed service ips, e.g. for 10 clients and 100 services, client-1 pings service-10, client-2 pings service-20, etc.)
- **number of requests per client** (**CONN_PER_CLIENT**, ab -n value)
- **client concurrency** (**CLIENT_CONCURRENCY**, number of parallel connections per client, ab -c value)

The total number of requests = CONN_PER_CLIENT * CLIENTS

iptables vs nftables: first packet latency

Which kube-proxy mode has better programming latency?

...

iptables vs nftables: first packet latency

Which kube-proxy mode has better programming latency?

...

iptables

Outcomes

- every benchmarking/measuring tool has its limitations
- designing test workload is hard. Think about your workloads and stress your cluster with that instead of copy pasting from internet
- confirmation bias is real
- **do not overgeneralize**

iptables vs nftables: first packet latency

nftables breaks at 30K and can't converge... why?

iptables has a "partialSync" mode, which only applies on a per-service basis.
Let's fix it for nftables: https://github.com/kubernetes/kubernetes/pull/126013

Results:
Creation of 10K services, 2 endpoints each

- before: 25 min
- after: 9 min

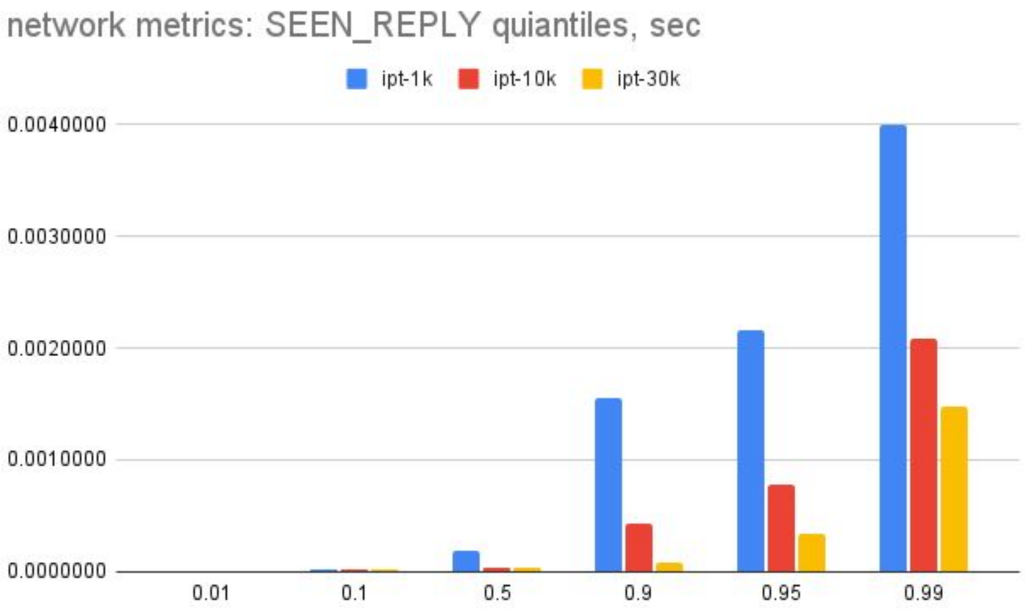
Adding one more service after

- before: 8 min
- after: 141 ms

Outcomes

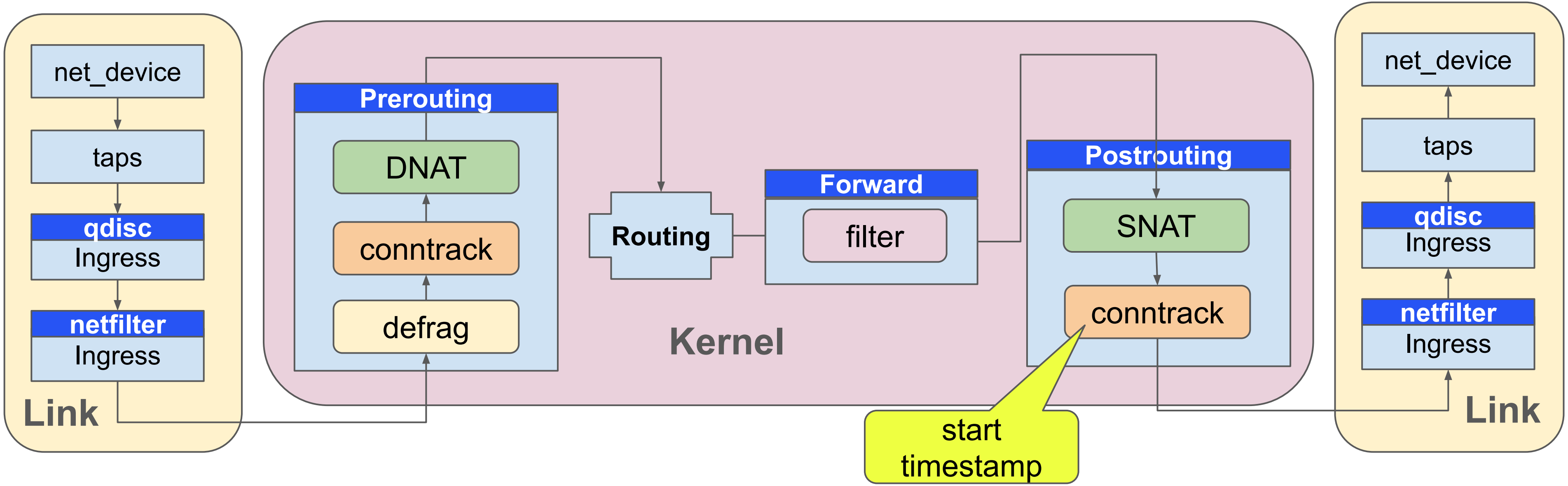
- every benchmarking/measuring tool has its limitations
- designing test workload is hard. Think about your workloads and stress your cluster with that instead of copy pasting from internet
- confirmation bias is real
- do not overgeneralize
- **SLI-driven performance testing makes software better :)**

iptables first packet latency: first try



latency drops when the number of services grows?

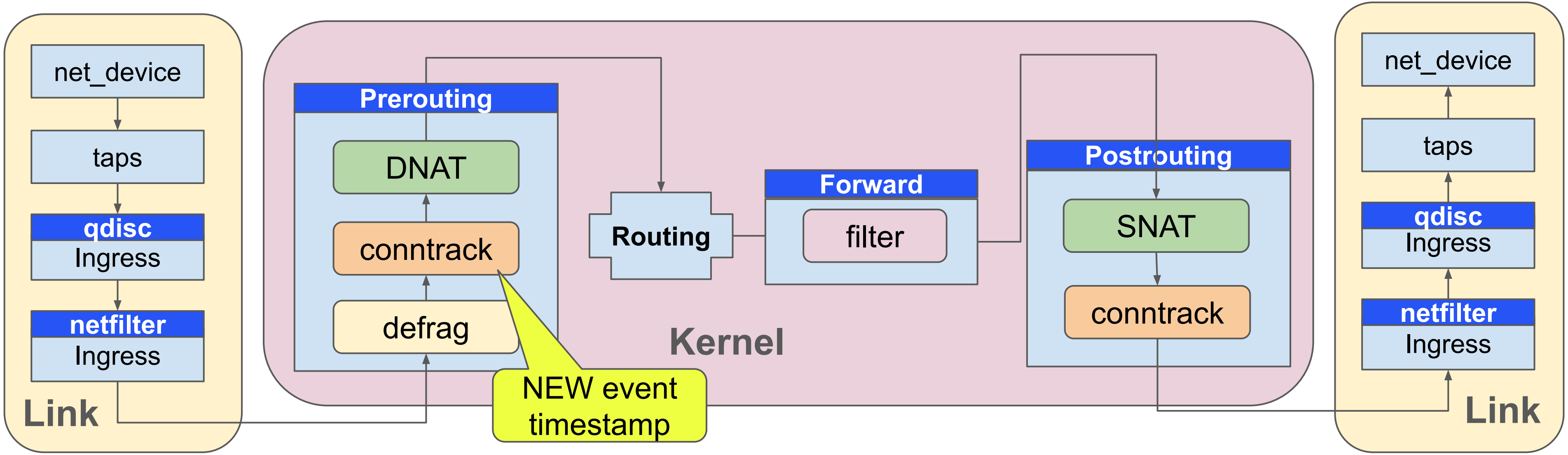
iptables first packet latency: conntrack timestamp



From “netfilter: nf_conntrack_tstamp: add flow-based timestamp extension” patch:

... we use two 64-bits variables to store the **creation timestamp once the conntrack has been confirmed** and the other to store the deletion time.

iptables first packet latency: conntrack timestamp



We can record NEW event timestamp earlier.

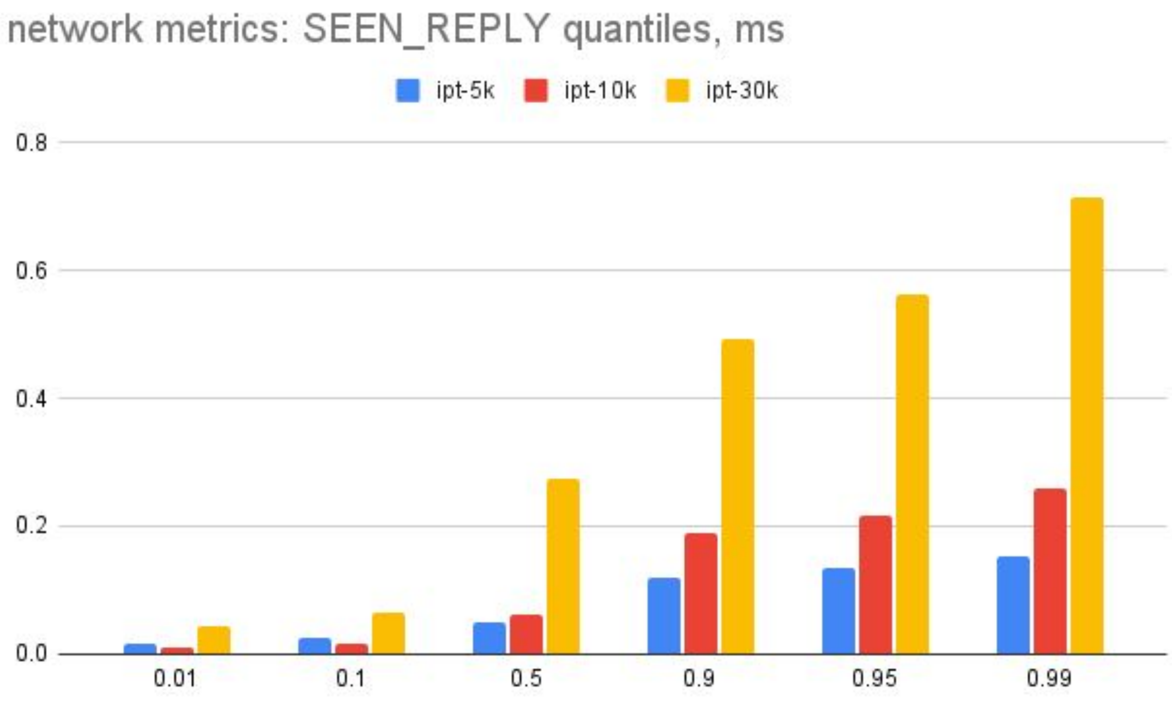
Let's timestamp every event in the kernel!

Thanks to Florian Westphal for this patch ==========>

conntrack: changing timestamping

hook prerouting {
-0000000400 ipv4_conntrack_defrag [nf_defrag_ipv4]
-0000000200 **ipv4_conntrack_in** [nf_conntrack]
-0000000100 nf_nat_ipv4_pre_routing [nf_nat]
}
hook input {
+0000000100 nf_nat_ipv4_local_in [nf_nat]
+2147483647 nf_confirm [nf_conntrack] => **NEW event**
}
hook forward {
0000000000 chain ip filter FORWARD [nf_tables]
}
hook output {
-0000000400 ipv4_conntrack_defrag [nf_defrag_ipv4]
-0000000200 ipv4_conntrack_local [nf_conntrack]
-0000000100 nf_nat_ipv4_local_fn [nf_nat]
0000000000 chain ip filter OUTPUT [nf_tables]
}
hook postrouting {
-0000000225 apparmor_ip_postroute
+0000000100 nf_nat_ipv4_out [nf_nat]
+2147483647 **nf_confirm** [nf_conntrack] => **NEW event**
}

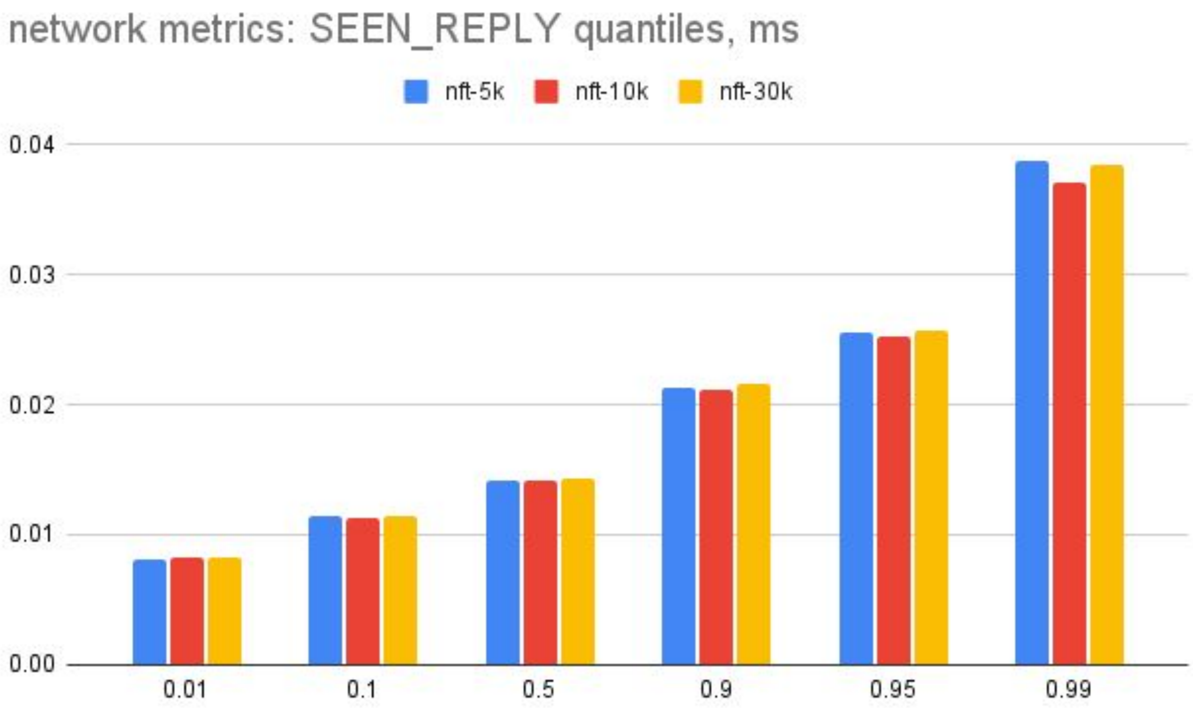
iptables latency: second (actually much more than second) try



that makes more sense!

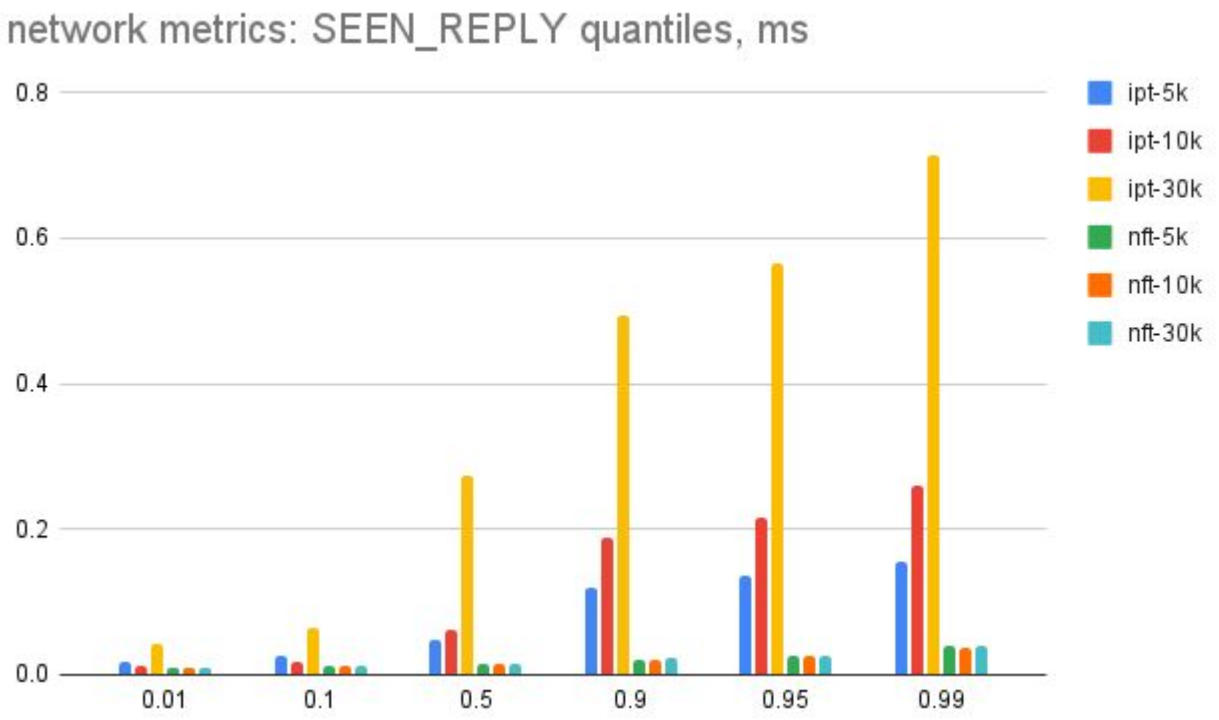
- latency grows with the number of services
- that is “linear search” effect

nftables first packet latency



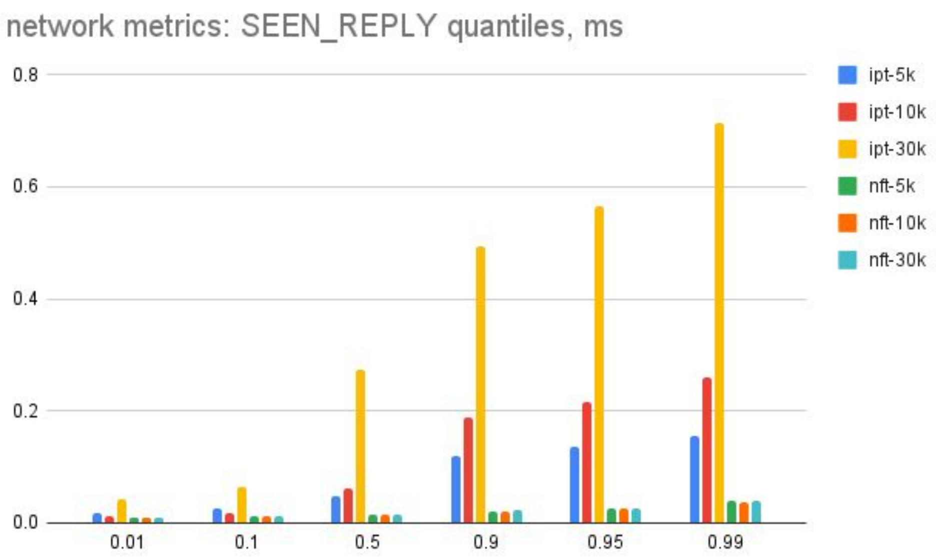
- latency doesn’t grow with the number of services
- that is no “linear search” effect

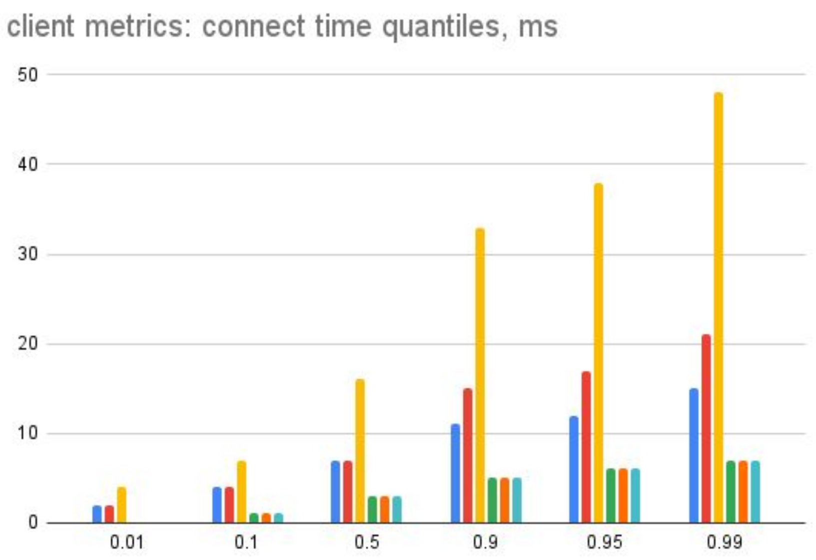
iptables vs nftables first packet latency



nftables' latency is much better

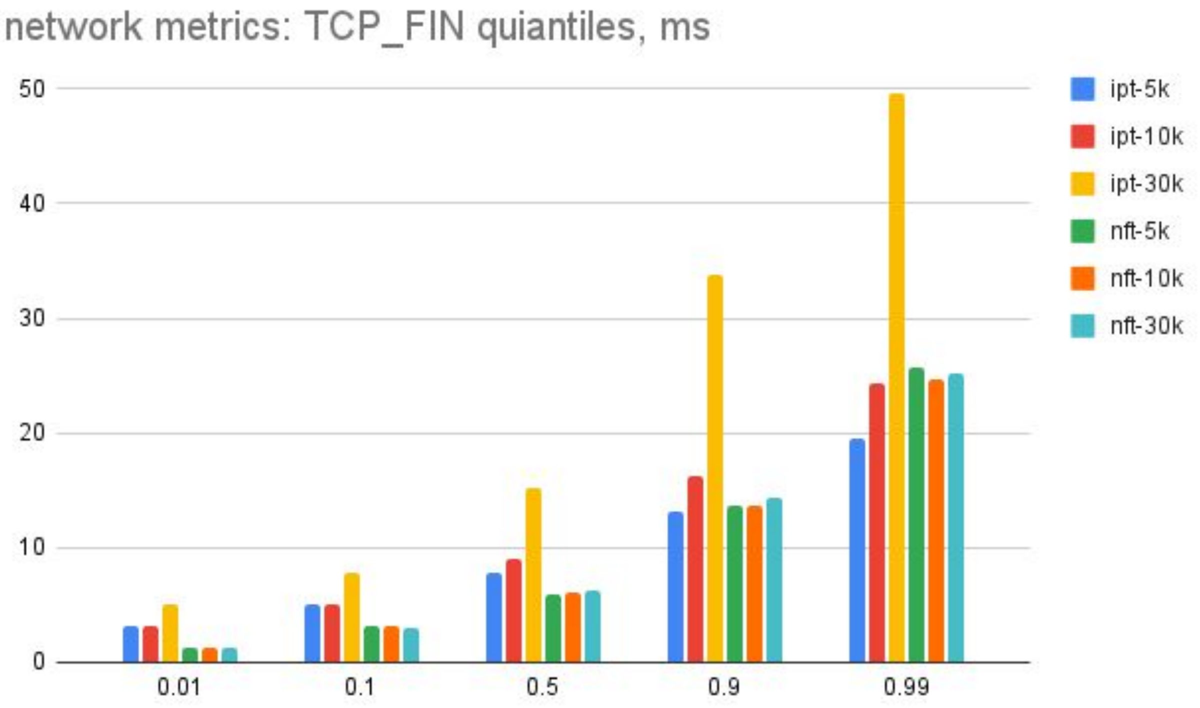
network vs client metrics: first packet latency





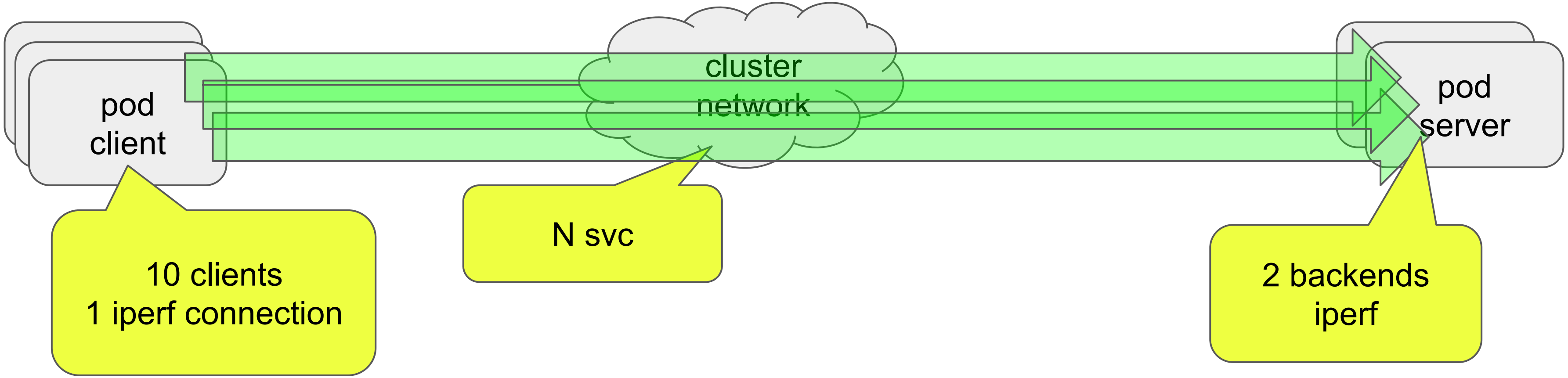
The pattern is very similar, but the absolute values are different.

iptables vs nftables connection time



For short connections, first packet latency is visible on connection time metric too.

iptables vs nftables throughput



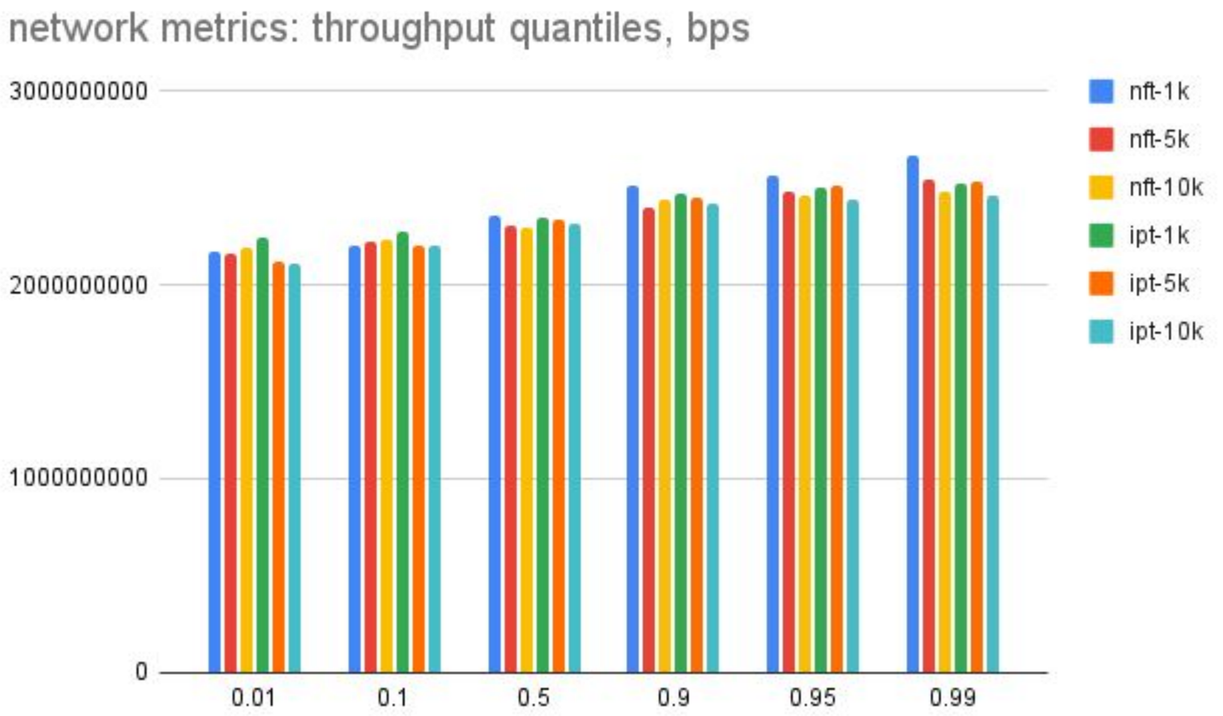
test variables:

- kube-proxy mode iptables vs nftables
- amount of services: 1K, 5K, 10K

measurements:

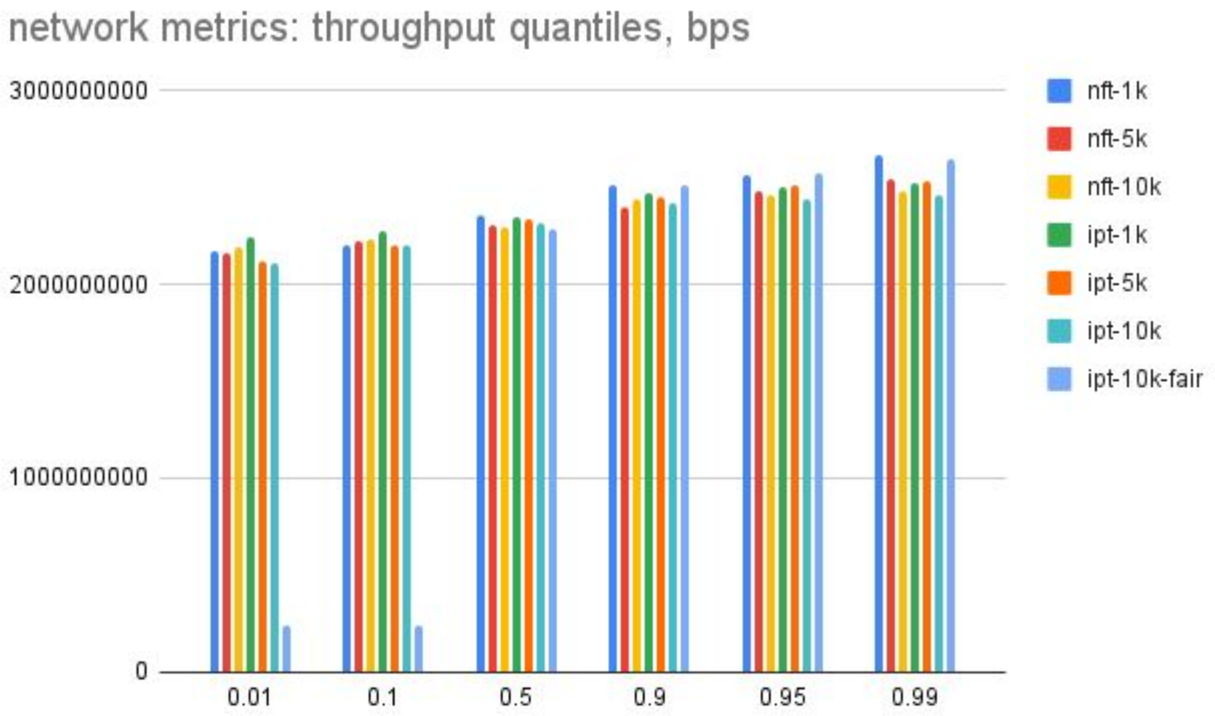
- client metrics and network metrics

iptables vs nftables throughput



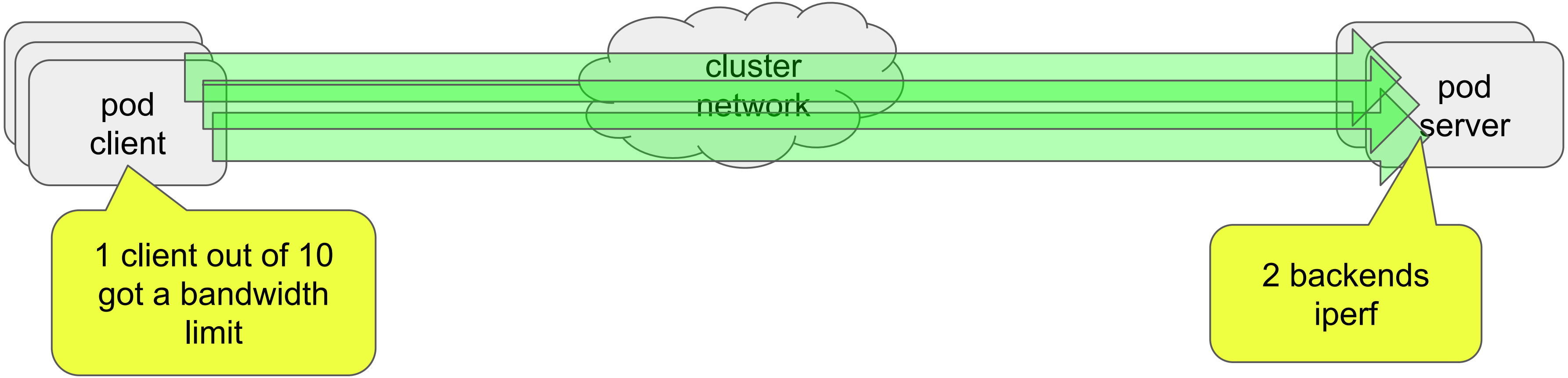
throughput for large connections sending a lot of data is almost the same for all test cases

iptables vs nftables throughput fairness



Take a look at the new test result: ipt-10k-fair

iptables vs nftables throughput fairness



sorted results for 10 clients look something like:

[0.2, 2, 2, 2, 2, 2, 2, 2, 2, 2] GBytes per second

0.1 quantile

Outcomes

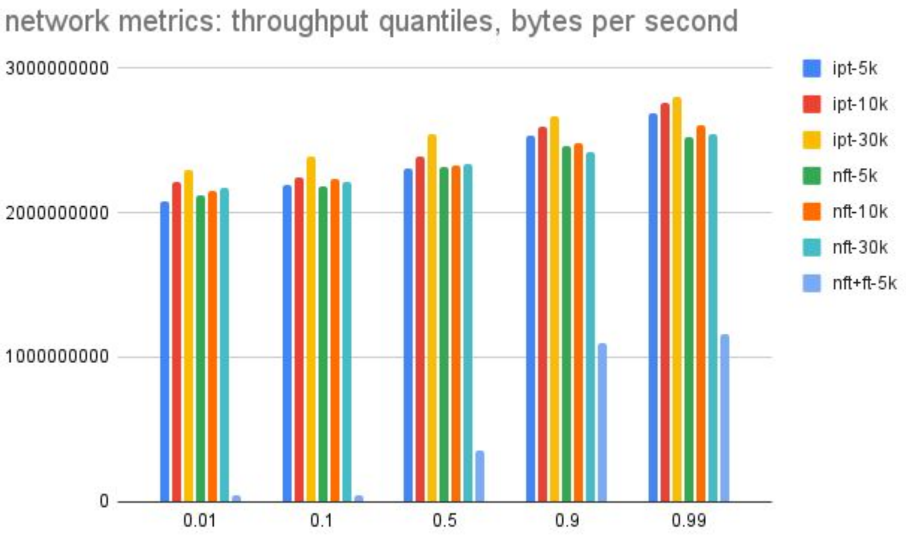
- every benchmarking/measuring tool has its limitations
- designing test workload is hard. Think about your workloads and stress your cluster with that instead of copy pasting from internet
- confirmation bias is real
- do not overgeneralize
- SLI-driven performance testing makes software better :)
- **nftables kube-proxy has better performance SLIs than iptables**

Outcomes

- every benchmarking/measuring tool has its limitations
- designing test workload is hard. Think about your workloads and stress your cluster with that instead of copy pasting from internet
- confirmation bias is real
- do not overgeneralize
- SLI-driven performance testing makes software better :)
- **nftables kube-proxy has better performance SLIs than iptables**
    - ... for all tested scenarios

nftables throughput: flowtables

nftables has a Flowtables feature, that allows you to accelerate packet forwarding by using a conntrack-based network stack bypass. One consequence of using it is that not every packet goes through conntrack, which makes bytes and packets counters not accurate.



Conclusion

- During the development of our monitoring tool and the kube-proxy improvements, we encountered several performance bugs that were not initially apparent through traditional benchmarking methods.

- By focusing on SLIs like network programming latency, connection round trip and throughput, we were able to identify and diagnose these bugs more effectively, as they directly impacted the user experience.

- The use of SLIs allowed us to pinpoint the root causes of these performance issues and develop targeted solutions, resulting in significant improvements in kube-proxy's efficiency and responsiveness.

- The discovery and resolution of these bugs served as a validation of our hypothesis that SLI-driven development is crucial for enhancing service performance and delivering a superior user experience.