# (P)sampling kubernetes network policies

Dumitru Ceară Nadia Pinaeva Adrián Moreno



Red Hat

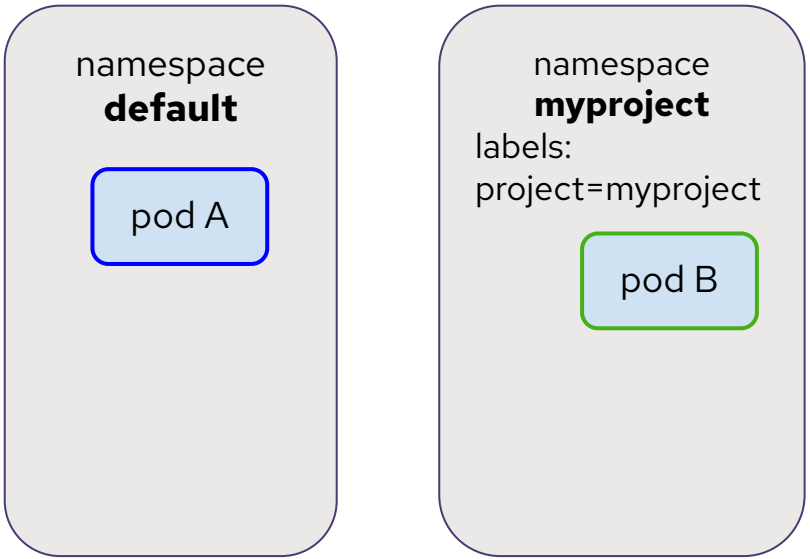
# Kubernetes

Kubernetes, also known as k8s, is a container orchestration engine.

It groups containers that make up an application into logical units called Pods.

Cluster resources, including Pods, are isolated using namespaces.

Labels are key/value pairs that are attached to objects such as Pods and Namespaces. Labels can be used to organize and to select subsets of objects.



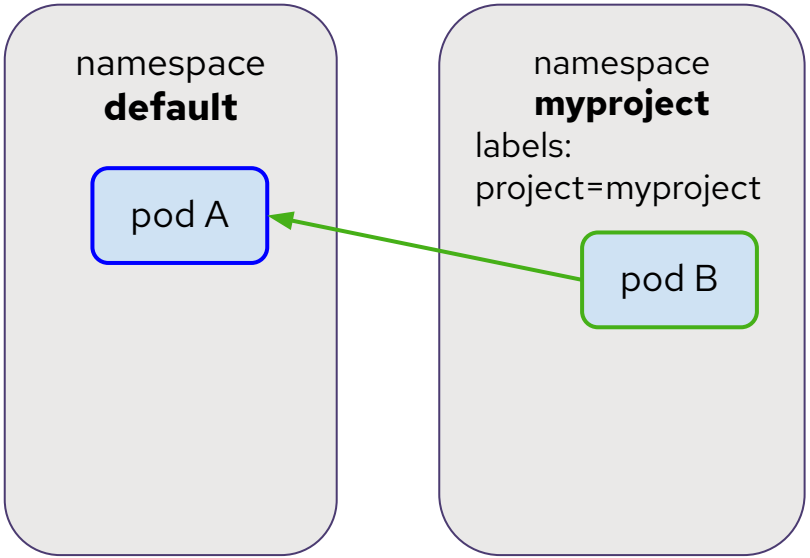
# Kubernetes Network Policy

NetworkPolicy is a kubernetes API that ensures network security, by specifying connections that should be allowed for a namespace. All the other connections for a namespace will be denied.

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
  namespace: default
spec:
  podSelector: {}
  ingress:
    - from:
      - namespaceSelector:
          matchLabels:
            project: myproject
```

pods that are isolated

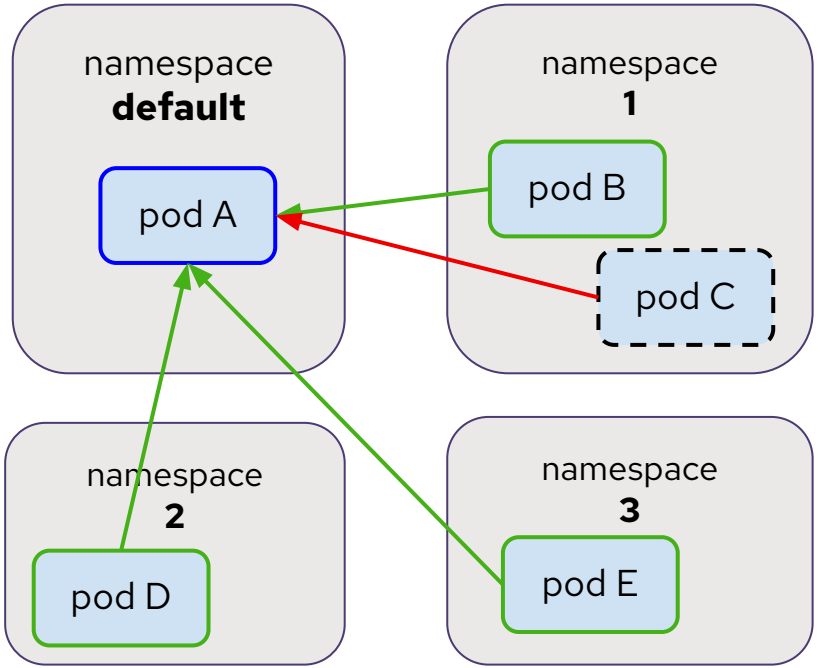
pods, from which connections are allowed



# Observability problem

In real clusters, the amount of Network Policies may go beyond thousands.

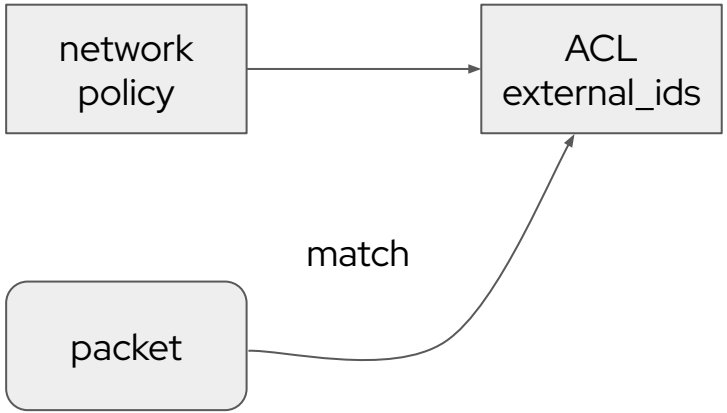
It gets difficult to understand, why exactly some connection is allowed or denied.



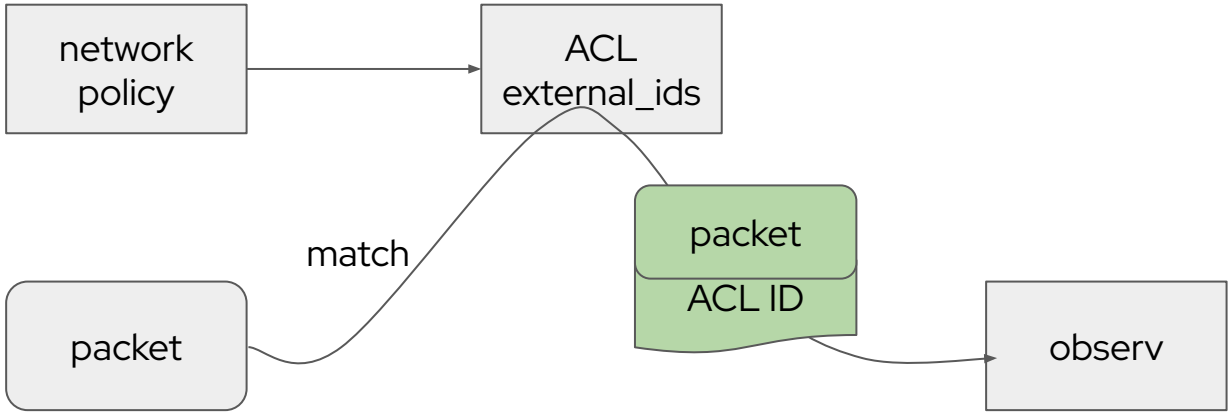

Red Hat

# OVN-Kubernetes

OVN-kubernetes uses OVN ACLs to implement k8s Network Policy.

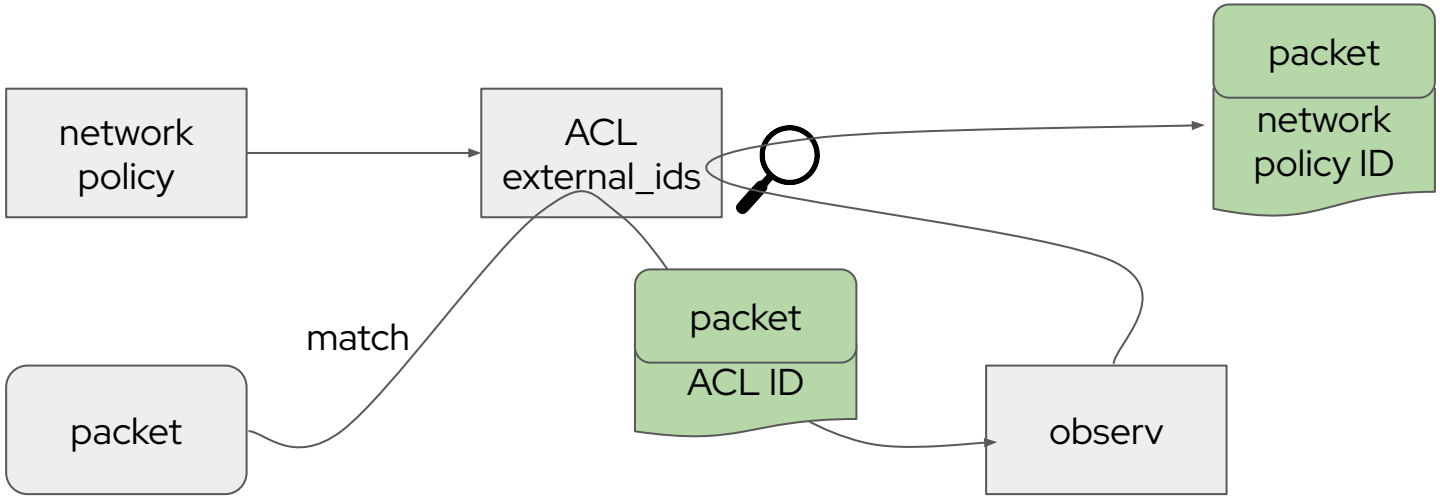


OVN-kubernetes uses ACLs to implement k8s Network Policy.

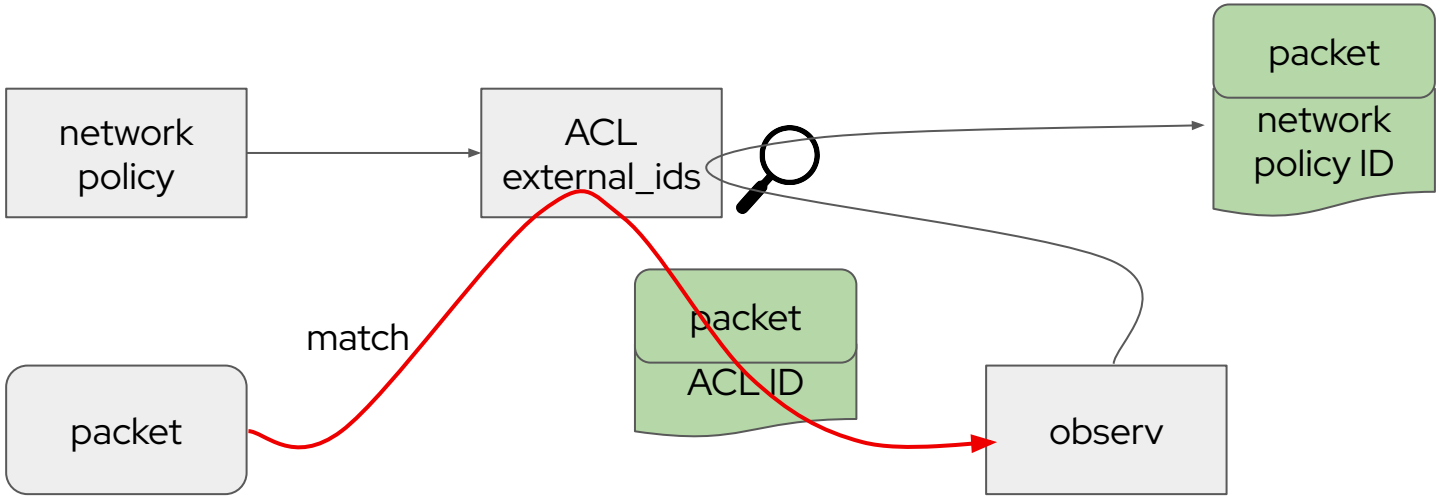



Red Hat

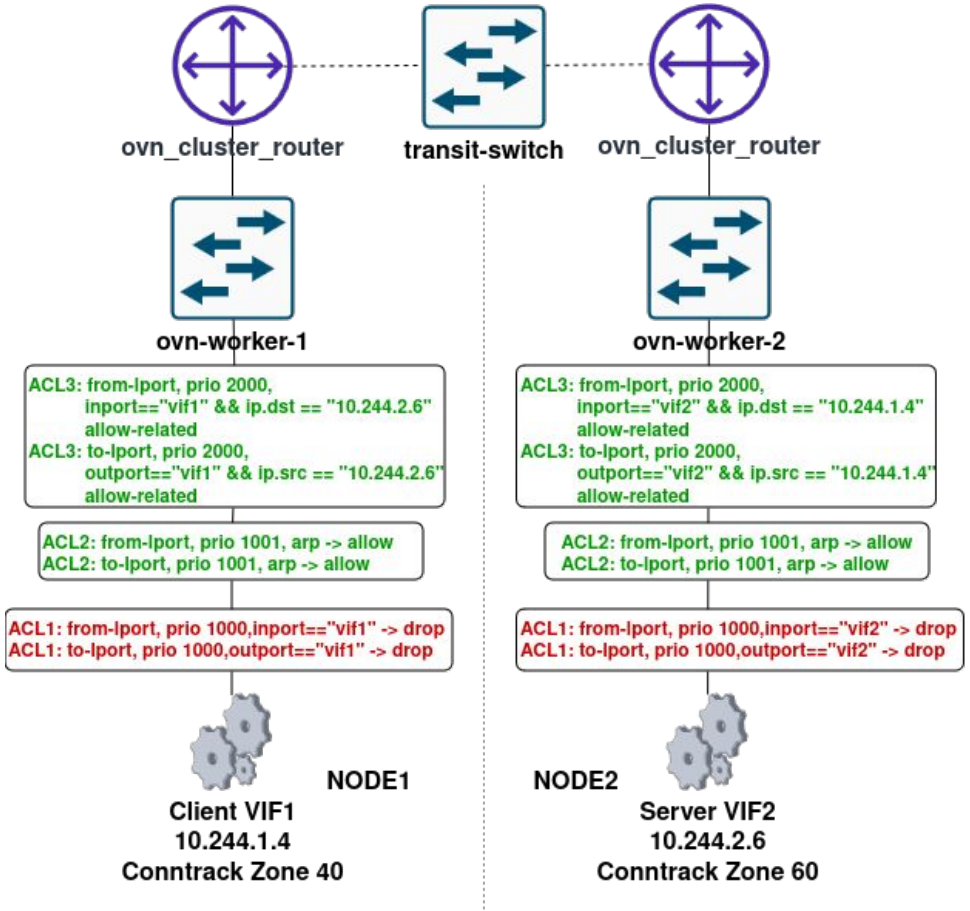
OVN-kubernetes uses ACLs to implement k8s Network Policy.



How can we do this?



# OVN ACL Sampling




- **ACL1** (*"default drop"*)
  - lower prio (1000)
  - drops all IP traffic originating from VIFs
- **ACL2** (*"allow ARPs from VIFs"*)
  - higher prio (1001)
- **ACL3** (*"allow IP traffic between VIFs"*)
  - highest prio (2000)
  - only allows IP traffic between VIF1 and VIF2
  - source is selected by logical ingress port
  - destination is selected by IP
- **OVN assigned conntrack zones**
  - each VIF gets one (ovn-controller)
  - used for stateful firewalling (ACL)

**Traffic pattern:** TCP session between VIF1 -> VIF2 (**10.244.1.4:4040 -> 10.244.2.6:5201**)

**Packet**: TCP SYN 10.244.1.4:4040 -> 10.244.2.6:5201

**Outcome:** explicitly allowed by ACL3, session **committed** to conntrack

| Stage + Logical Flow (ingress pipeline) | CT zone/state | CT zone 40 (VIF1) |
|---|---|---|
| **in_pre_stateful**:<br>**action**: ct_next(dnat) /* zone=vif1-zone */ | zone=**40**, state=**trk,new** | empty |

**Traffic pattern:** TCP session between VIF1 -> VIF2 (**10.244.1.4:4040 -> 10.244.2.6:5201**)

**Packet**: TCP SYN 10.244.1.4:4040 -> 10.244.2.6:5201

**Outcome:** explicitly allowed by ACL3, session **committed** to conntrack

| Stage + Logical Flow (ingress pipeline) | CT zone/state | CT zone 40 (VIF1) |
|---|---|---|
| **in_pre_stateful**:<br>**action**: ct_next(dnat) /* zone=vif1-zone */ | zone=**40**, state=**trk,new** | empty |
| **in_acl_eval**:<br>**match**: ct.new && (inport == "vif1" && ip4.dst == 10.244.2.6) (/* ACL3 */)<br>**action**: verdict: allow, set-commit, next | zone=40, state=trk,new | empty |

**Traffic pattern:** TCP session between VIF1 -> VIF2 (**10.244.1.4:4040 -> 10.244.2.6:5201**)

**Packet**: TCP SYN 10.244.1.4:4040 -> 10.244.2.6:5201

**Outcome:** explicitly allowed by ACL3, session **committed** to conntrack

| Stage + Logical Flow (ingress pipeline) | CT zone/state | CT zone 40 (VIF1) |
|---|---|---|
| **in_pre_stateful**:<br>**action**: ct_next(dnat) /* zone=vif1-zone */ | zone=**40**, state=**trk,new** | empty |
| **in_acl_eval**:<br>**match**: ct.new && (inport == "vif1" && ip4.dst == 10.244.2.6) (/* ACL3 */)<br>**action**: verdict: allow, set-commit, next | zone=40, state=trk,new | empty |
| **in_acl_action**:<br>**match**: verdict == allow<br>**action**: next (/* allow-related */) | zone=40, state=trk,new | empty |
| **in_stateful**:<br>**match**: commit?<br>**action**: commit(zone=vif1-zone), next (/* allow-related */) | zone=40, state=trk,new | 10.244.1.4:4040-><br>10.244.2.6:5201<br>**state=new** |

**Traffic pattern:** TCP session between VIF1 -> VIF2 (**10.244.1.4:4040 -> 10.244.2.6:5201**)

**Packet**: TCP SYN+ACK 10.244.2.6:5201 -> 10.244.1.4:4040 (reply traffic)

**Outcome:** implicitly allowed by ACL3, session **found** in conntrack

| Stage + Logical Flow (egress pipeline) | CT zone/state | CT zone 40 (VIF1) |
|---|---|---|
| **out_pre_stateful**:<br>**action**: ct_next(dnat) /* zone=vif1-zone */ | zone=**40**, state=**trk,est,rpl** | 10.244.1.4:4040-><br>10.244.2.6:5201<br>**state=est** |

**Traffic pattern:** TCP session between VIF1 -> VIF2 (**10.244.1.4:4040 -> 10.244.2.6:5201**)

**Packet**: TCP SYN+ACK 10.244.2.6:5201 -> 10.244.1.4:4040 (reply traffic)

**Outcome:** implicitly allowed by ACL3, session **found** in conntrack

| Stage + Logical Flow (egress pipeline) | CT zone/state | CT zone 40 (VIF1) |
|---|---|---|
| **out_pre_stateful**:<br>**action**: ct_next(dnat) /* zone=vif1-zone */ | zone=**40**, state=**trk,est,rpl** | 10.244.1.4:4040->10.244.2.6:5201<br>**state=est** |
| **out_acl_hint**:<br>**match**: ct.est && !ct.rel && !ct.new && !ct.inv && ct.rpl && ct_mark.blocked == 0 (/* allow-related */)<br>**action**: verdict: allow, next | zone=40, state=trk,est,rpl | 10.244.1.4:4040->10.244.2.6:5201<br>**state=est** |

**Traffic pattern:** TCP session between VIF1 -> VIF2 (**10.244.1.4:4040 -> 10.244.2.6:5201**)

**Packet**: TCP SYN+ACK 10.244.2.6:5201 -> 10.244.1.4:4040 (reply traffic)

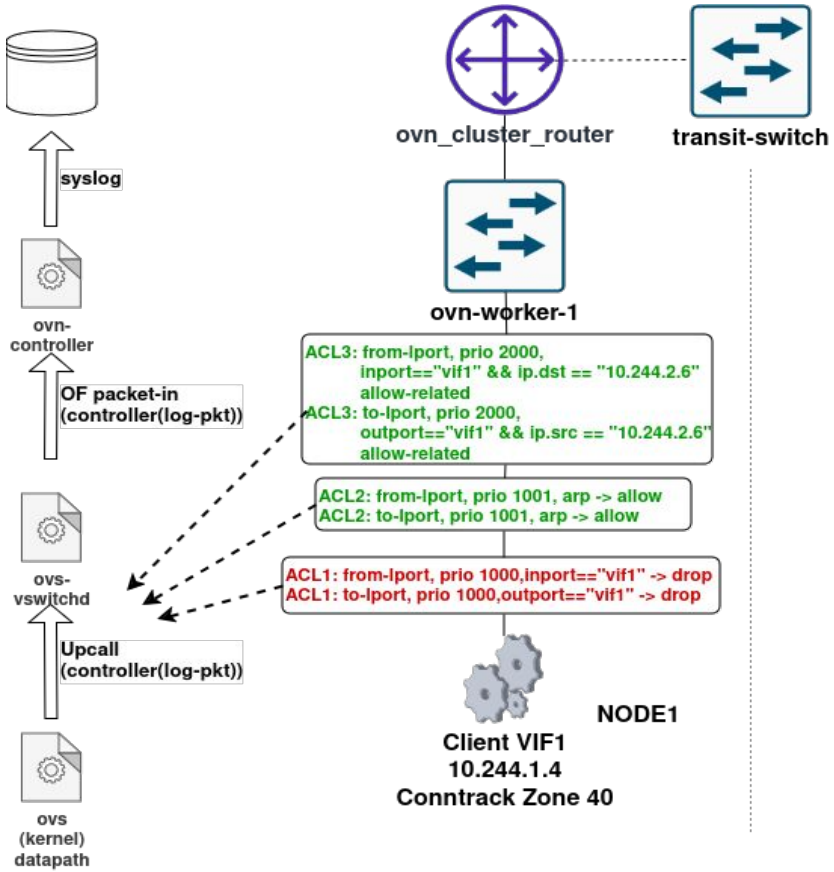
**Outcome:** implicitly allowed by ACL3, session **found** in conntrack

| Stage + Logical Flow (egress pipeline) | CT zone/state | CT zone 40 (VIF1) |
|---|---|---|
| **out_pre_stateful**:<br>**action**: ct_next(dnat) /* zone=vif1-zone */ | zone=**40**, state=**trk,est,rpl** | 10.244.1.4:4040->10.244.2.6:5201<br>**state=est** |
| **out_acl_hint**:<br>**match**: ct.est && !ct.rel && !ct.new && !ct.inv && ct.rpl && ct_mark.blocked == 0 (/* allow-related */)<br>**action**: verdict: allow, next | zone=40, state=trk,est,rpl | 10.244.1.4:4040->10.244.2.6:5201<br>**state=est** |
| **out_acl_action**:<br>**match**: verdict == allow<br>**action**: next (/* allow-related */) | zone=40, state=trk,est,rpl | 10.244.1.4:4040->10.244.2.6:5201<br>**state=est** |
| **out_stateful**:<br>**match**: 1<br>**action**: next | zone=40, state=trk,est,rpl | 10.244.1.4:4040->10.244.2.6:5201<br>**state=est** |

**OVN (pre-24.09.0)** already had features that provide visibility into ACL processing

- ACL logging
- ACL labeling

## Why not use those?

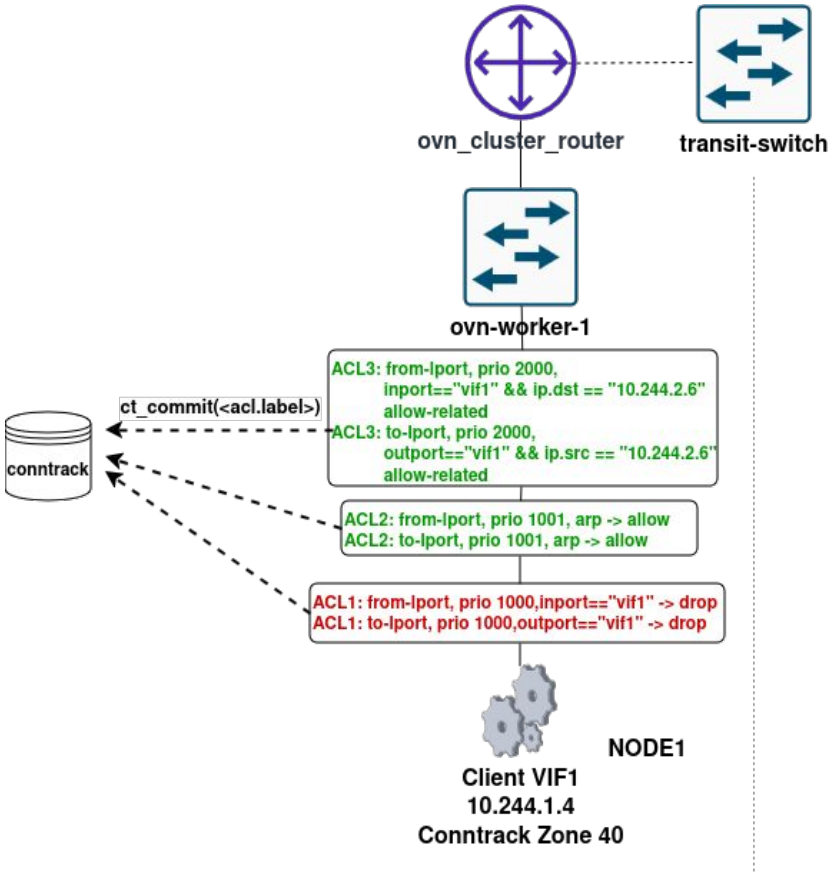



```
$ ovn-nbctl --name acl3 --log --label 42 acl-add ...
...
00035|acl_log(ovn_pinctrl0)|INFO|name="acl3",
verdict=allow, severity=info, direction=from-lport:
tcp,nw_src=10.244.1.4,nw_dst=10.244.2.6,tp_src=4040,
tp_dst=5201,tcp_flags=syn

00037|acl_log(ovn_pinctrl0)|INFO|name="acl3",
verdict=allow, severity=info, direction=from-lport:
tcp,nw_src=10.244.2.6,nw_dst=10.244.1.4,tp_src=5201,
tp_dst=4040,tcp_flags=syn|ack
```

## Issues

- (super) slow path all packets: upcall to ovs-vswitchd -> packet-in to ovn-controller -> syslog
- generates an additional OpenFlow rule for each ACL for reply traffic (matching on the ACL label)



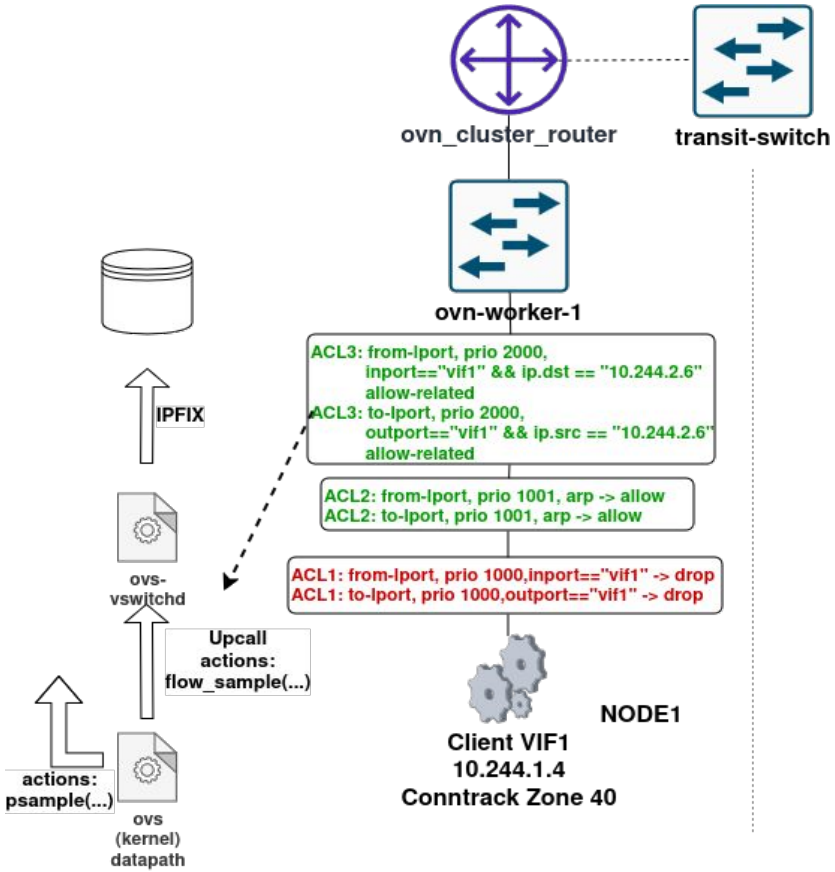

```
$ ovn-nbctl --label 42 acl-add ...
...
$ ovs-appctl dpctl/dump-conntrack | grep 5201
tcp,orig=(src=10.244.1.4,dst=10.244.2.6,sport=4040,dport=
5201),reply=(src=10.244.2.6,dst=10.244.1.4,sport=5201,dpo
rt=4040),zone=2,labels=0x2a00000000000000000000000,proto
info=(state=ESTABLISHED)

tcp,orig=(src=10.244.1.4,dst=10.244.2.6,sport=4040,dport=
5201),reply=(src=10.244.2.6,dst=10.244.1.4,sport=5201,dpo
rt=4040),zone=1,labels=0x2a00000000000000000000000,proto
info=(state=ESTABLISHED)
```

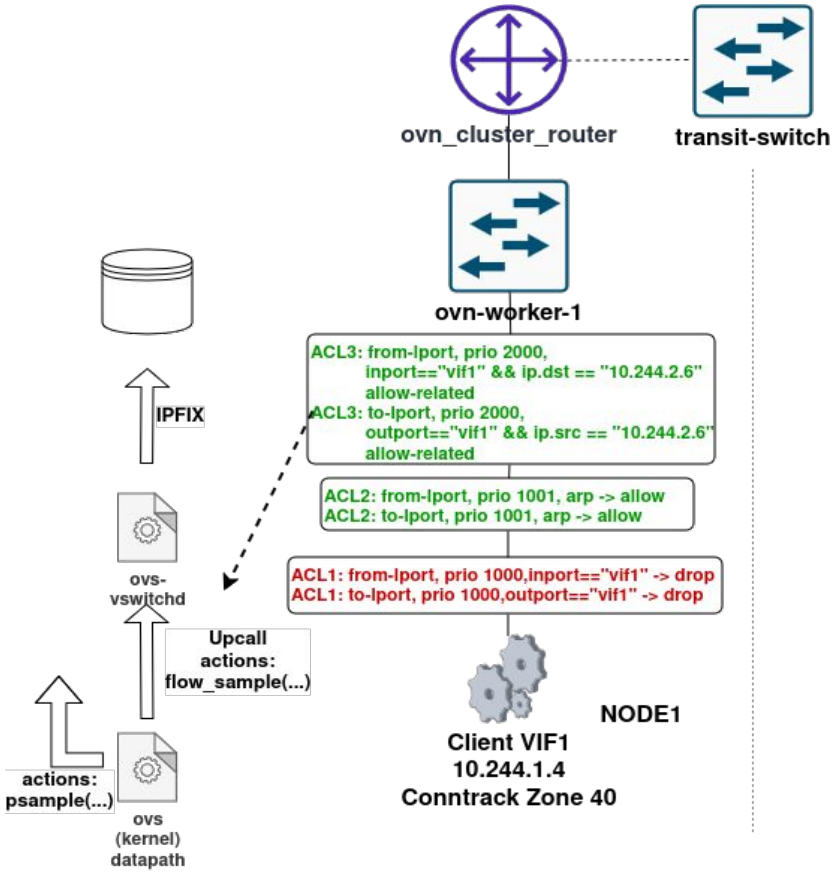
## Issues

- client applications must guess internal OVN ct_label representation (today 32 MSB)
- no per-packet information about ACLs that were hit

# OVN 24.09 supports ACL sampling!




```
$ ovn-nbctl list Sampling_App
id: 42, type: acl-new
id: 43, type: acl-est
$ ovn-nbctl list Sample_Collector
name: c1, probability:  100%, set_id: 100
name: c2, probability:  100%, set_id: 200
$ ovn-nbctl list Sample
uuid: s_new, collectors: [c1, c2], metadata:  1011
uuid: s_est, collectors: [c1, c2], metadata:  1012
$ ovn-nbctl list ACL
action: allow-related, direction: from-lport,
    match: "inport == \"vif2\" && ip4.dst == 10.244.1.4",
    priority: 2000,  sample_new: s_new, sample_est: s_est
action: allow-related, direction: to-lport,
    match: "outport == \"vif1\" && ip4.src == 10.244.2.6",
    priority: 2000,  sample_new: s_new, sample_est: s_est
```




```
$ ovn-nbctl list Sampling_App
id: 42, type: acl-new
id: 43, type: acl-est
$ ovn-nbctl list Sample_Collector
name: c1, probability:  100%, set_id: 100
name: c2, probability:  100%, set_id: 200
$ ovn-nbctl list Sample
uuid: s_new, collectors: [c1, c2], metadata:  1011
uuid: s_est, collectors: [c1, c2], metadata:  1012
$ ovn-nbctl list ACL
action: allow-related, direction: from-lport,
    match: "inport == \"vif2\" && ip4.dst == 10.244.1.4",
    priority: 2000,  sample_new: s_new, sample_est: s_est
action: allow-related, direction: to-lport,
    match: "outport == \"vif1\" && ip4.src == 10.244.2.6",
    priority: 2000,  sample_new: s_new, sample_est: s_est
-------------------------------------------------------

$ ovs-appctl dpctl/dump-flows | grep sample
flow_sample(probability=100%,collector_set_id= 100,
            obs_domain_id=0x 2b000003,obs_point_id= 1012),
flow_sample(probability= 100%,collector_set_id= 100,
            obs_domain_id=0x 2b000003,obs_point_id= 1012)

flow_sample(probability= 100%,collector_set_id= 100,
obs_domain_id=0x 2a000002,obs_point_id= 1011)
```



Red Hat

**Traffic pattern:** TCP session between VIF1 -> VIF2 (**10.244.1.4:4040 -> 10.244.2.6:5201**)

**Packet**: TCP SYN 10.244.1.4:4040 -> 10.244.2.6:5201, **Sampling** (new: **1011**, est: **1012**)

**Outcome:** explicitly allowed by ACL3, session **committed** to conntrack

| Stage + Logical Flow (ingress pipeline) | CT zone/state | CT zone 40 |
|---|---|---|
| **in_pre_stateful**: **action**: ct_next(dnat) /* zone=vif1-zone */ | zone=**40**, state=**trk,new** | empty |

**Traffic pattern:** TCP session between VIF1 -> VIF2 (**10.244.1.4:4040 -> 10.244.2.6:5201**)

**Packet**: TCP SYN 10.244.1.4:4040 -> 10.244.2.6:5201, **Sampling** (new: **1011**, est: **1012**)

**Outcome:** explicitly allowed by ACL3, session **committed** to conntrack

| Stage + Logical Flow (ingress pipeline) | CT zone/state | CT zone 40 |
|---|---|---|
| **in_pre_stateful**: **action**: ct_next(dnat) /* zone=vif1-zone */ | zone=**40**, state=**trk,new** | empty |
| **in_acl_eval**:<br>**match**: ct.new && (inport == "vif1" && ip4.dst == 10.244.2.6) (/* ACL3 */)<br>**action**: verdict: allow, set-commit, reg3<-1011, reg9<-1012, next | zone=40, state=trk,new | empty |

**Traffic pattern:** TCP session between VIF1 -> VIF2 (**10.244.1.4:4040 -> 10.244.2.6:5201**)

**Packet**: TCP SYN 10.244.1.4:4040 -> 10.244.2.6:5201, **Sampling** (new: **1011**, est: **1012**)

**Outcome:** explicitly allowed by ACL3, session **committed** to conntrack

| Stage + Logical Flow (ingress pipeline) | CT zone/state | CT zone 40 |
|---|---|---|
| **in_pre_stateful**: **action**: ct_next(dnat) /* zone=vif1-zone */ | zone=**40**, state=**trk,new** | empty |
| **in_acl_eval**:<br>**match**: ct.new && (inport == "vif1" && ip4.dst == 10.244.2.6) (/* ACL3 */)<br>**action**: verdict: allow, set-commit, reg3<-1011, reg9<-1012, next | zone=40, state=trk,new | empty |
| **in_acl_sample**:<br>**match**: ip && ct.new && reg3 == 1011<br>**action**:<br>sample(probability=100%,collector_set=200,obs_domain=**42**,obs_point=**1011**);<br>sample(probability=100%,collector_set=100,obs_domain=**42**,obs_point=**1011**); | zone=40, state=trk,new | empty |

**Traffic pattern:** TCP session between VIF1 -> VIF2 (**10.244.1.4:4040 -> 10.244.2.6:5201**)

**Packet**: TCP SYN 10.244.1.4:4040 -> 10.244.2.6:5201, **Sampling** (new: **1011**, est: **1012**)

**Outcome:** explicitly allowed by ACL3, session **committed** to conntrack

| Stage + Logical Flow (ingress pipeline) | CT zone/state | CT zone 40 |
|---|---|---|
| **in_pre_stateful**: **action**: ct_next(dnat) /* zone=vif1-zone */ | zone=**40**, state=**trk,new** | empty |
| **in_acl_eval**:<br>**match**: ct.new && (inport == "vif1" && ip4.dst == 10.244.2.6) (/* ACL3 */)<br>**action**: verdict: allow, set-commit, reg3<-1011, reg9<-1012, next | zone=40, state=trk,new | empty |
| **in_acl_sample**:<br>**match**: ip && ct.new && reg3 == 1011<br>**action**:<br>sample(probability=100%,collector_set=200,obs_domain=**42**,obs_point=**1011**);<br>sample(probability=100%,collector_set=100,obs_domain=**42**,obs_point=**1011**); | zone=40, state=trk,new | empty |
| **in_acl_action**: **match**: verdict == allow **action**: next (/* allow-related */) | zone=40, state=trk,new | empty |
| **in_stateful**: **match**: commit?<br>**action**: commit (zone=vif1-zone, ct_label.obs_point_id = reg9 /* 1012 */), next | zone=40, state=trk,new | 10.244.1.4:4040-><br>10.244.2.6:5201<br>**state=new** |

**Traffic pattern:** TCP session between VIF1 -> VIF2 (**10.244.1.4:4040 -> 10.244.2.6:5201**)

**Packet**: TCP SYN+ACK 10.244.2.6:5201 -> 10.244.1.4:4040 (reply traffic), **Sampling** (new: **1011**, est: **1012**)

**Outcome:** implicitly allowed by ACL3, session **found** in conntrack

| Stage + Logical Flow (egress pipeline) | CT zone/state | CT zone 40 |
|---|---|---|
| **out_pre_stateful**:<br>**action**: ct_next(dnat) /* zone=vif1-zone */ | zone=**40**, state=**trk,est,rpl**<br>**label.obs_point_id=1012** | **state=est** |

**Traffic pattern:** TCP session between VIF1 -> VIF2 (**10.244.1.4:4040 -> 10.244.2.6:5201**)

**Packet**: TCP SYN+ACK 10.244.2.6:5201 -> 10.244.1.4:4040 (reply traffic), **Sampling** (new: **1011**, est: **1012**)

**Outcome:** implicitly allowed by ACL3, session **found** in conntrack

| Stage + Logical Flow (egress pipeline) | CT zone/state | CT zone 40 |
|---|---|---|
| **out_pre_stateful**:<br>**action**: ct_next(dnat) /* zone=vif1-zone */ | zone=**40**, state=**trk,est,rpl**<br>**label.obs_point_id=1012** | **state=est** |
| **out_acl_sample**:<br>**match**: ip && ct.trk && (ct.est \|\| ct.rel) && ct.rpl && ct_label.obs_point_id == 1012<br>**action**:<br>sample(probability=65535,collector_set=200,obs_domain=**43**,obs_point=**1012**);<br>sample(probability=65535,collector_set=100,obs_domain=**43**,obs_point=**1012**); | zone=40, state=trk,est,rpl | **state=est** |

**Traffic pattern:** TCP session between VIF1 -> VIF2 (**10.244.1.4:4040 -> 10.244.2.6:5201**)

**Packet**: TCP SYN+ACK 10.244.2.6:5201 -> 10.244.1.4:4040 (reply traffic), **Sampling** (new: **1011**, est: **1012**)

**Outcome:** implicitly allowed by ACL3, session **found** in conntrack

| Stage + Logical Flow (egress pipeline) | CT zone/state | CT zone 40 |
|---|---|---|
| **out_pre_stateful**:<br>**action**: ct_next(dnat) /* zone=vif1-zone */ | zone=**40**, state=**trk,est,rpl**<br>**label.obs_point_id=1012** | **state=est** |
| **out_acl_sample**:<br>**match**: ip && ct.trk && (ct.est \|\| ct.rel) && ct.rpl && ct_label.obs_point_id == 1012<br>**action**:<br>sample(probability=65535,collector_set=200,obs_domain=**43**,obs_point=**1012**);<br>sample(probability=65535,collector_set=100,obs_domain=**43**,obs_point=**1012**); | zone=40, state=trk,est,rpl | **state=est** |
| **out_acl_action**:<br>**match**: verdict == allow<br>**action**: next (/* allow-related */) | zone=40, state=trk,est,rpl | **state=est** |
| **out_stateful**:<br>**match**: 1<br>**action**: next | zone=40, state=trk,est,rpl | **state=est** |

**The good:**

- can provide per packet ACL information
- can distinguish between packets that create a **new** connection
  OR are part of an **established** connection
- datapath agnostic: works fine with both **IPFIX** and **psample** collectors

**The not so good:**

- additional flows needed (one per *ACL - for sample_est -)* needed in the *reply direction*
  => O(number-of-ACLs) extra flows => **control plane cost!**

**OVS v3.4.0** includes *1aa9e137fe36 ("ofp-actions: Load data from fields in sample action.")*

OVN rewrites the logical flows in the reply direction (for *ACL Sample configurations with exactly one collector*):

- **From one flow per ACL:**

  out_acl_sample:

  match: ip && ct.trk && (ct.est || ct.rel) && ct.rpl && ct_label.obs_point_id == **1012**

  action: sample(probability=65535,collector_set=100,obs_domain=43,obs_point=**1012**);

- **To exactly two flows (regardless of number of ACLs):**

  out_acl_sample:

  match=ip && ct.trk && (ct.est || ct.rel) && !ct.rpl && ct_mark.obs_collector_id == 1

  action=sample(probability=100%,collector_set=100,obs_domain=43,obs_point=**ct_label.obs_point_id**);

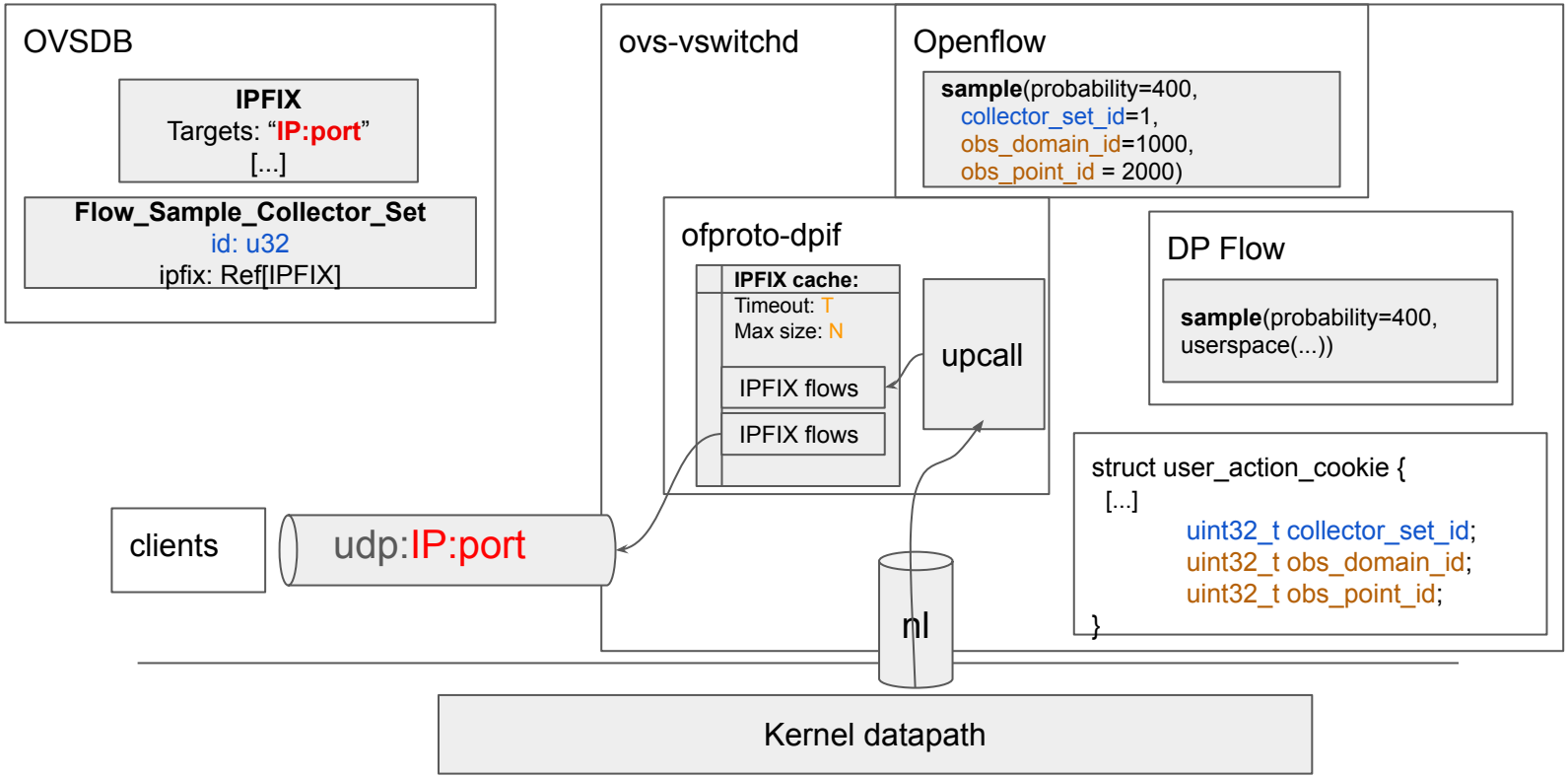
  out_acl_sample:

  match=ip && ct.trk && (ct.est || ct.rel) && ct.rpl && ct_mark.obs_collector_id == 1

  action=sample(probability=65535,collector_set=100,obs_domain=43,obs_point=**ct_label.obs_point_id**);

# OVS (p)sampling

# Existing sampling infrastructure: IPFIX



# Main problems

- Each sample goes through ovs-vswitchd: overload
- Samples share netlink socket with upcalls: packet loss!
- IPFIX cache accessed *handler* threads: contention
- IPFIX socket sent from *main* or *handler* threads

# Goal

- Send the sample to the end user directly (bypassing ovs-vswitchd)
- Have the sample contain metadata extracted from OpenFlow

# Introducing psample

✅ psample exists from v4.11, used by tc (see tc_sample(8))
✅ Creates a netlink multicast group where samples are sent
✅ A (u32) “group id” identifies “source” of the sample
✅ Includes sampling rate

❌ It does not have a cookie or any user-defined metadata
❌ The packet is copied even if there are no listeners

# Introducing psample

- ✅ psample exists from v4.11, used by tc (see tc_sample(8))
- ✅ Creates a netlink multicast group where samples are sent
- ✅ A (u32) “group id” identifies “source” of the sample
- ✅ Includes sampling rate


It does not have a cookie or any user-defined metadata

The packet is copied even if there are no listeners

# OVS datapath psample action

**psample(group=42, cookie=0x1234)**

- It integrates with other datapath actions. E.g:
  - sample(sample=10%, actions(psample(cookie=0x123)))
  - trunc(20), psample()
- Only implemented in kernel datapath *

# OpenFlow sample action optimization

```
sample(obs_domain_id=42,obs_point_id = 2000)
```

# OpenFlow sample action optimization

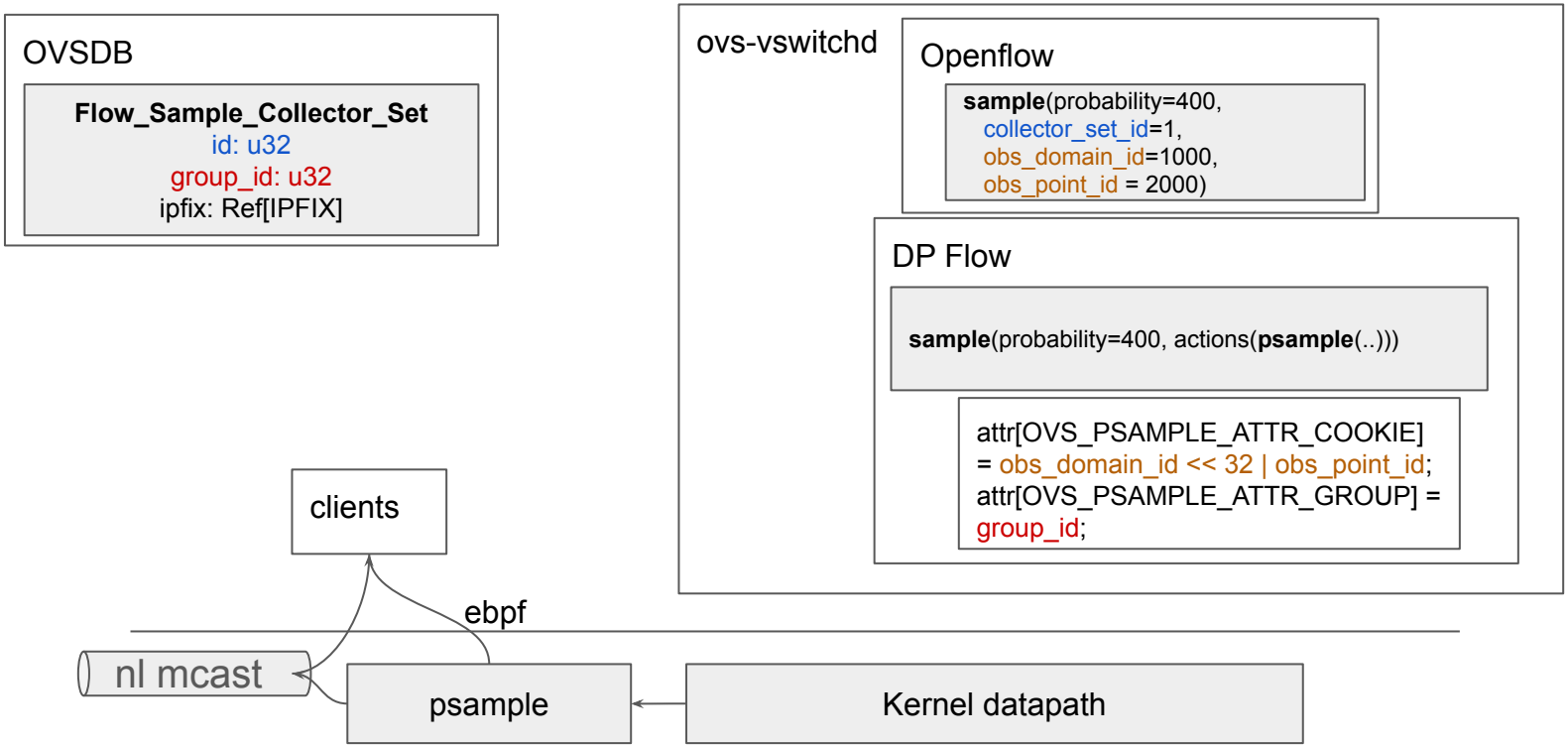
```
ct_label = 1000, actions=sample(obs_domain_id=42, obs_point_id=1000)
ct_label = 2000, actions=sample(obs_domain_id=42, obs_point_id=2000)
ct_label = 3000, actions=sample(obs_domain_id=42, obs_point_id=3000)
                                    …
```

# OpenFlow sample action optimization

```
ct_label = 1000, actions=sample(obs_domain_id=42, obs_point_id=1000)
ct_label = 2000, actions=sample(obs_domain_id=42, obs_point_id=2000)
ct_label = 3000, actions=sample(obs_domain_id=42, obs_point_id=3000)
…
```

```
actions=sample(obs_domain_id=42, obs_point_id=NXM_NX_CT_LABEL[32..63])
```

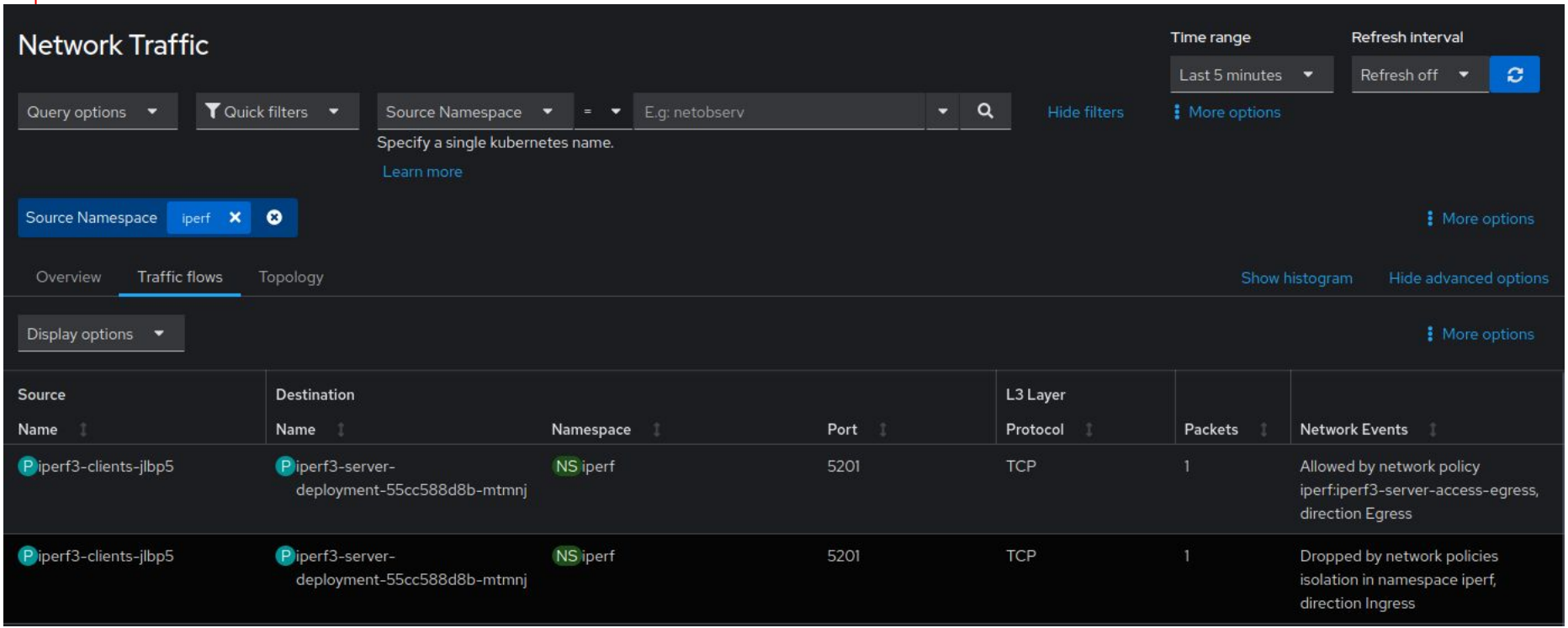
# New sampling infrastructure: psample



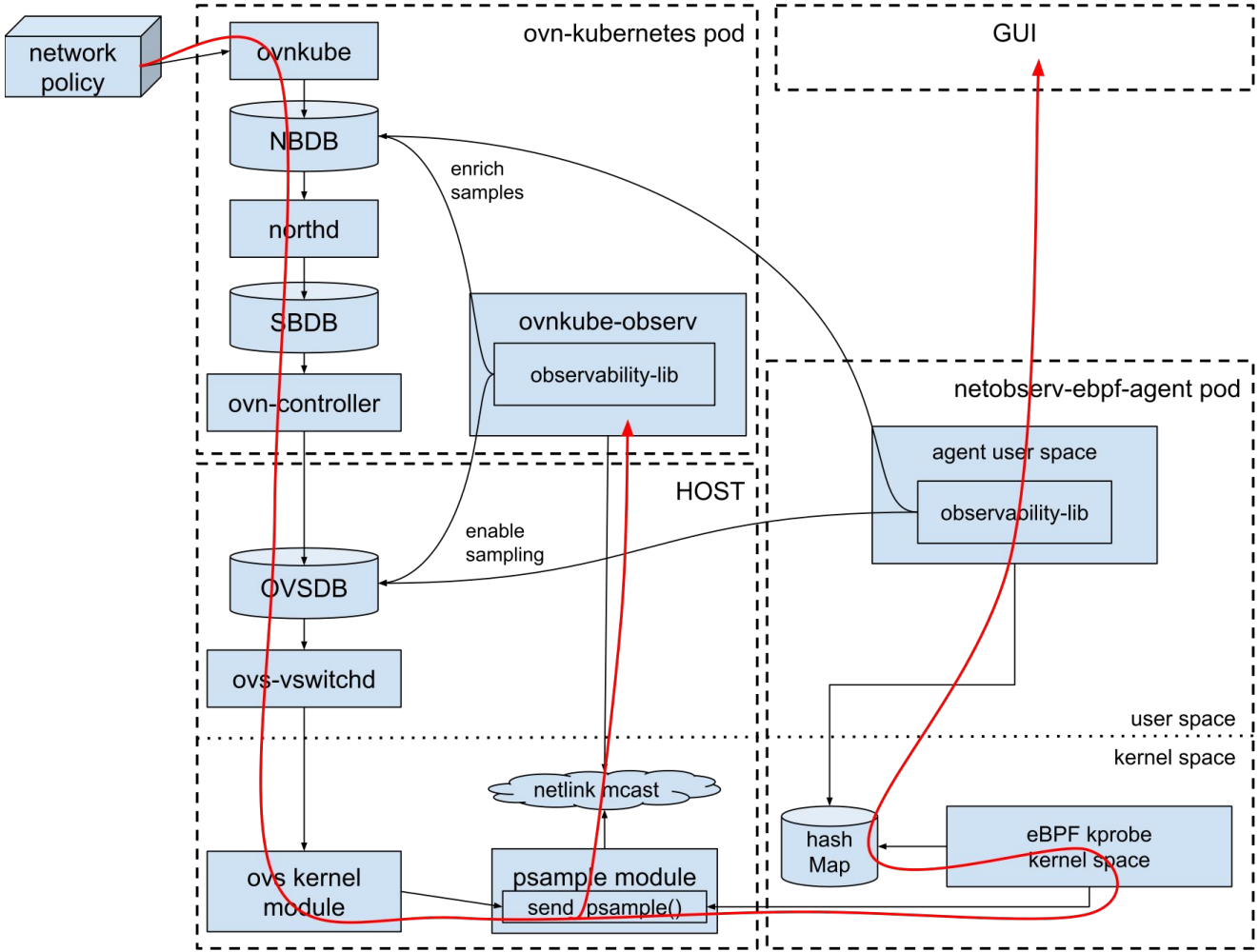
# Future work

- Finalize and publish performance tests. Stay tuned!
- Explore a userspace implementation
  - USDT probes? Userspace ebpf VM?
  - Static (hardcoded) behavior?
  - Ideas and requirements are welcome!

# Demo


Network Traffic
Time range
Last 5 minutes
Refresh interval
Refresh off
Query options
Quick filters
Source Namespace
=
E.g: netobserv
Hide filters
More options
Specify a single kubernetes name.
Learn more
Source Namespace
iperf
More options
Overview
Traffic flows
Topology
Show histogram
Hide advanced options
Display options
More options
Source
Name
Destination
Name
Namespace
Port
L3 Layer
Protocol
Packets
Network Events
iperf3-clients-jlbp5
iperf3-server-deployment-55cc588d8b-mtmnj
iperf
5201
TCP
1
Allowed by network policy iperf:iperf3-server-access-egress, direction Egress
iperf3-clients-jlbp5
iperf3-server-deployment-55cc588d8b-mtmnj
iperf
5201
TCP
1
Dropped by network policies isolation in namespace iperf, direction Ingress

Red Hat


network policy
ovnkube
ovn-kubernetes pod
GUI
NBDB
enrich samples
northd
SBDB
ovnkube-observ
observability-lib
ovn-controller
netobserv-ebpf-agent pod
agent user space
observability-lib
HOST
enable sampling
OVSDB
ovs-vswitchd
user space
kernel space
netlink mcast
hash Map
eBPF kprobe kernel space
ovs kernel module
psample module
send_psample()

Red Hat

# Thank you

Red Hat is the world's leading provider of enterprise open source software solutions. Award-winning support, training, and consulting services make
Red Hat a trusted adviser to the Fortune 500.


linkedin.com/company/red-hat


youtube.com/user/RedHatVideos


facebook.com/redhatinc


twitter.com/RedHat



Red Hat