ofproto/detrace - the missing link

Aleš Musil, Dumitru Ceară, 2024

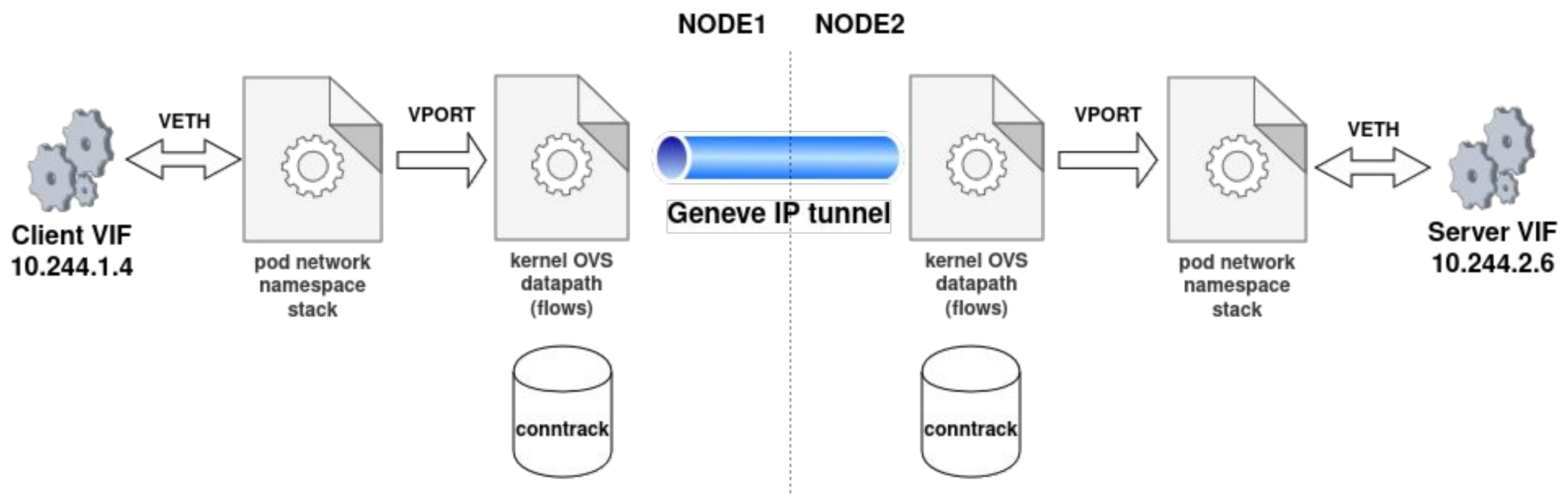
The network in the kernel..

```
ufid:afa73c50-ee58-4f65-8cd2-7fb5d0271ae4,
recirc_id(0),dp_hash(0/0),skb_priority(0/0),in_port(client),skb_mark(0/0),ct_state(0/0x21),ct_zone(0/0),ct_mark(0/0),ct_label(0/0),eth(src=00:00:00:00:31:10,dst=00:00:00:
00:31:00),eth_type(0x0800),ipv4(src=10.244.1.4,dst=10.244.1.42,proto=6,tos=0/0,ttl=64,frag=no),tcp(src=0/0,dst=0/0),tcp_flags(0/0), packets:61, bytes:4142, used:0.072s,
flags:P., dp:ovs, actions:ct(zone=2,nat),recirc(0x25)

ufid:2cc4a33c-c9fe-4cc0-aa37-adbc2494a856,
recirc_id(0x25),dp_hash(0/0),skb_priority(0/0),in_port(client),skb_mark(0/0),ct_state(0x22/0x27),ct_zone(0/0),ct_mark(0x2/0xe),ct_label(0/0),eth(src=00:00:00:00:31:10,dst
=00:00:00:00:31:00),eth_type(0x0800),ipv4(src=0.0.0.0/0.0.0.0,dst=10.244.2.0/255.255.255.0,proto=6,tos=0/0,ttl=64,frag=no),tcp(src=0/0,dst=0/0),tcp_flags(0/0),
packets:60, bytes:4076, used:0.072s, flags:P., dp:ovs, actions:set(eth(src=00:00:00:00:10:00,dst=00:00:00:00:20:00)),set(ipv4(ttl=63)),ct(zone=2,nat),recirc(0x28)

ufid:15b43126-bae2-48eb-bc05-f7580b46ff61,
recirc_id(0x28),dp_hash(0/0),skb_priority(0/0),in_port(client),skb_mark(0/0),ct_state(0x20/0x21),ct_zone(0/0),ct_mark(0/0),ct_label(0/0),eth(src=00:00:00:00:10:00,dst=00
:00:00:00:20:00),eth_type(0x0800),ipv4(src=0.0.0.0/0.0.0.0,dst=0.0.0.0/0.0.0.0,proto=0/0,tos=0/0x3,ttl=0/0,frag=no), packets:60, bytes:4076, used:0.072s, flags:P., dp:ovs,
actions:ct_clear,set(tunnel(tun_id=0xa,src=170.168.0.4,dst=170.168.0.5,ttl=64,tp_dst=6081,geneve({class=0x102,type=0x80,len=4,0xa0014}),flags(df|csum|key))),genev_sys_6081

ufid:6350dfed-cee1-4feb-9570-9c8ae3c50b80,
recirc_id(0),dp_hash(0/0),skb_priority(0/0),tunnel(tun_id=0xa,src=170.168.0.5,dst=170.168.0.4,ttl=0/0,geneve({class=0x102,type=0x80,len=4,0x14000a/0x7fffffff}),flags(-df+c
sum+key)),in_port(genev_sys_6081),skb_mark(0/0),ct_state(0/0x27),ct_zone(0/0),ct_mark(0/0xe),ct_label(0/0),eth(src=00:00:00:00:20:00,dst=00:00:00:00:10:00),eth_type
(0x0800),ipv4(src=10.244.2.0/255.255.254.0,dst=10.244.1.4,proto=6,tos=0/0,ttl=63,frag=no),tcp(src=0/0,dst=0/0),tcp_flags(0/0), packets:60, bytes:3962, used:0.072s, flags:P.,
dp:ovs, actions:set(eth(src=00:00:00:00:31:00,dst=00:00:00:00:31:10)),set(ipv4(ttl=62)),ct(zone=2,nat),recirc(0x29)

ufid:81838d8b-4046-49b0-b20f-08f2bf7249b4,
recirc_id(0x29),dp_hash(0/0),skb_priority(0/0),tunnel(tun_id=0xa,src=170.168.0.5,dst=170.168.0.4,ttl=0/0,geneve({class=0/0,type=0/0,len=0/0}{class=0/0,type=0/0,len=0/0}
),flags(-df+csum+key)),in_port(genev_sys_6081),skb_mark(0/0),ct_state(0x20/0x21),ct_zone(0/0),ct_mark(0/0),ct_label(0/0),eth(src=00:00:00:00:31:00,dst=00:00:00:00:31:
10),eth_type(0x0800),ipv4(src=0.0.0.0/0.0.0.0,dst=0.0.0.0/0.0.0.0,proto=0/0,tos=0/0,ttl=0/0,frag=no), packets:60, bytes:3962, used:0.072s, flags:P., dp:ovs,
actions:ct_clear,client
```

How do we understand and debug this?

- an overview of the topology we're trying to debug
- walk our way up the stack and connect the dots between abstraction layers
- take advantage of OVS/OVN tools already available

The “kernel view”



NODE1
NODE2
VETH
VPORT
VPORT
VETH
Client VIF
10.244.1.4
pod network namespace stack
kernel OVS datapath (flows)
Geneve IP tunnel
kernel OVS datapath (flows)
pod network namespace stack
Server VIF
10.244.2.6
conntrack
conntrack

- virtual networking through OVS (kernel) datapath flow table:
  - **match**: (masked) packet fields, conntrack state / label / mark, ingress port, recirc_id (if multiple passes required - conntrack), etc.
  - **actions**: drop, output to port, send to conntrack (and recirculate), set packet fields or metadata, etc.
  - **UFID:** *unique flow identifier*
    - quickly identify flows between userspace (ovs-vswitchd) / kernel (OVS kmod)
    - mapping between userspace context (e.g., OpenFlow rules) and flows

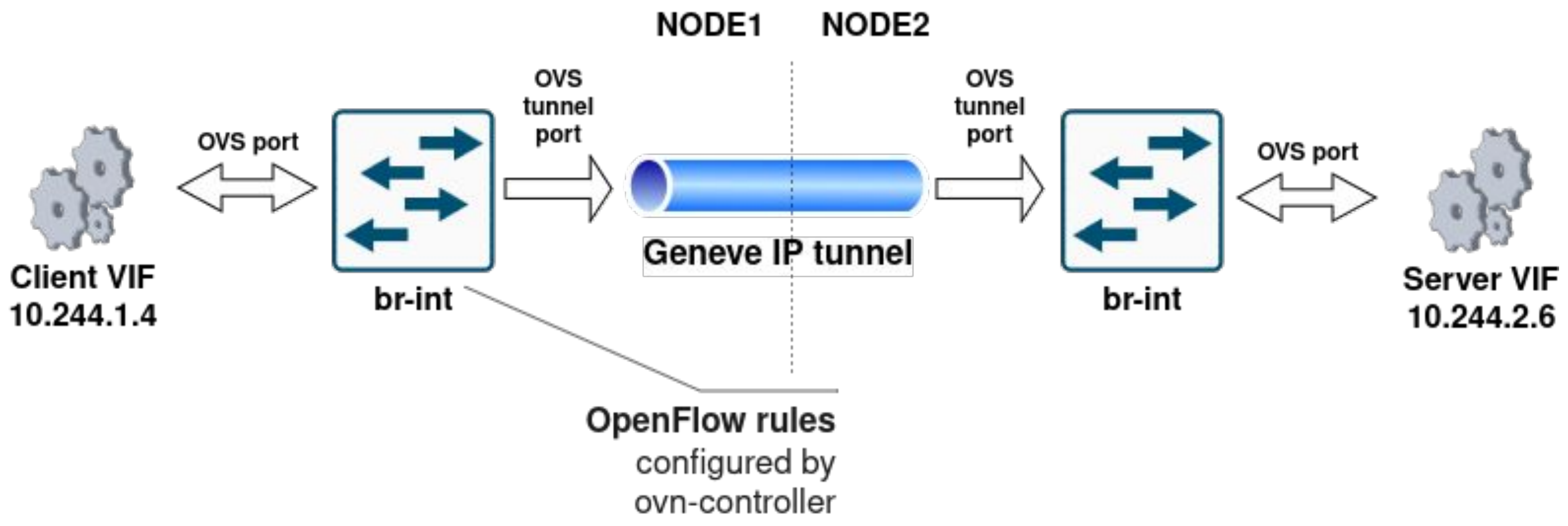
The “kernel view”

```
$ ovs-appctl dpctl/dump-flows -m


ufid:afa73c50-ee58-4f65-8cd2-7fb5d0271ae4
match:recirc_id(0),in_port(client),eth(src=00:00:00:00:31:10,dst=00:00:00:00:31:00),ipv4(src=10.244.1.4,dst=10.244.1.42), packets:61
actions:ct(zone=2,nat),recirc(0x25)

                                                  ct_state=+est+trk

ufid:2cc4a33c-c9fe-4cc0-aa37-adbc2494a856
match:recirc_id(0x25),in_port(client),ct_state(0x22/0x27),ct_mark(0x2/0xe),eth(src=00:00:00:00:31:10,dst=00:00:00:00:31:00),
ipv4(dst=10.244.2.0/255.255.255.0)
actions:set(eth(src=00:00:00:00:10:00,dst=00:00:00:00:20:00)),set(ipv4(ttl=63)),ct(zone=2,nat),recirc(0x28)
```

- traffic flowing from the client’s veth to the geneve tunnel
- conntrack lookup and a conntrack entry (established session) being matched
- packet destination IP changing (DNAT)
- packet MAC addresses changing and TTL decrementing (routing?)
- and more..

That’s useful but can we figure out why that’s happening?

The “OpenFlow view”



Rule format:

- table-id, priority, match, actions
- **cookie:** ovn-controller leaves “bread crumbs”, (IDs of OVN Southbound database records): *Logical_Flows*, *Port_Bindings*, *Load_Balancers*, etc.

The new “ofproto/detrace” appctl command

```
$  ovs-appctl list-commands | grep trace
ofproto/detrace               UFID    [pmd=PMD-ID]
ofproto/trace                 {[dp_name] odp_flow | bridge br_flow} [OPTIONS...] [-generate|packet]
ofproto/trace-packet-out [-consistent] {[dp_name] odp_flow | bridge br_flow} [OPTIONS...] [-generate|packet] actions
```

```
$ grep -A1 ofproto/detrace NEWS
   * Added 'ofproto/detrace' command that outputs the set of OpenFlow rules
     and groups that contributed to the creation of a specific datapath flow.
```

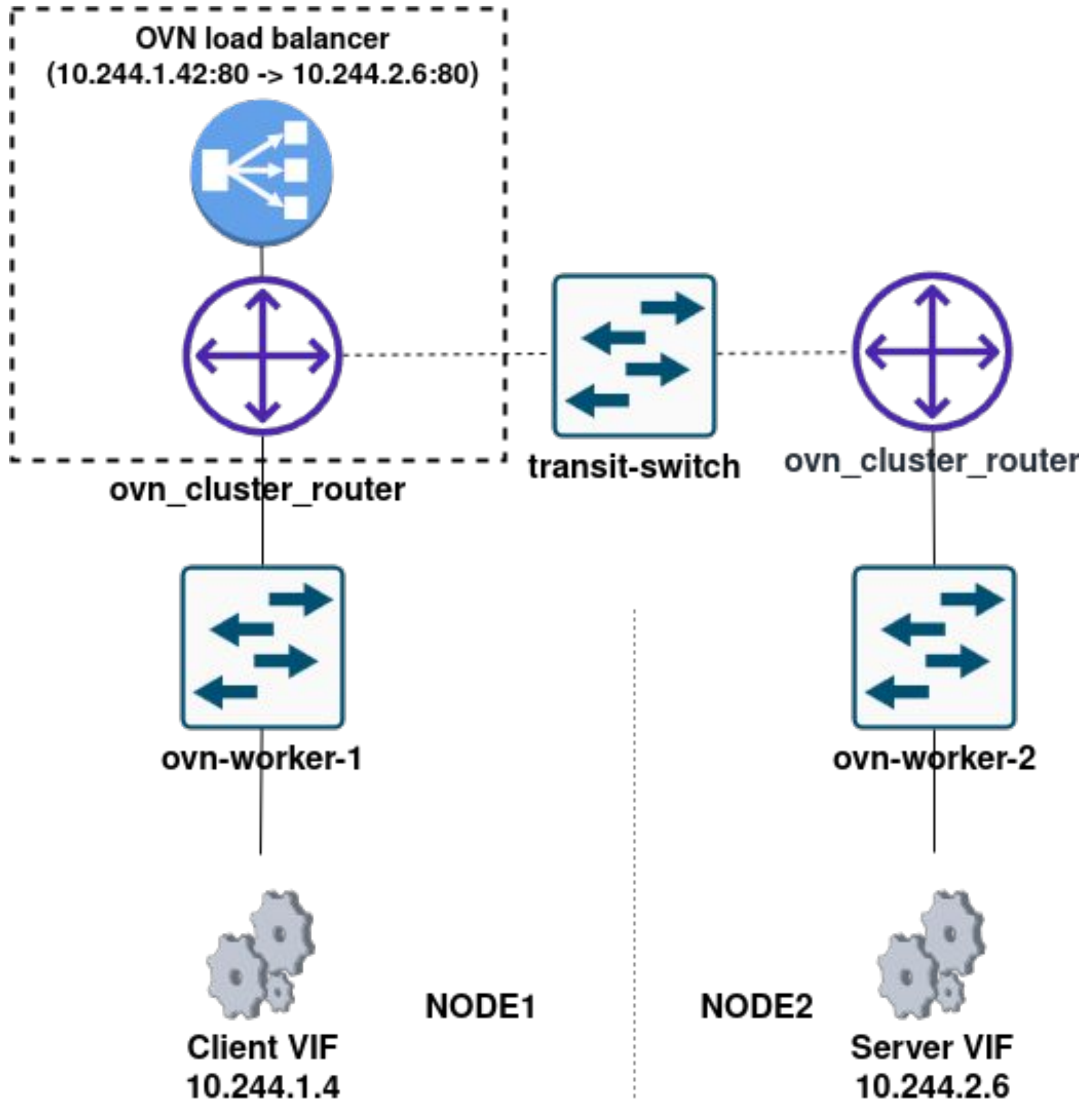
UFID: *unique flow identifier*

- quickly identify flows between userspace (ovs-vswitchd) / kernel (OVS kmod)
- mapping between userspace context (e.g., OpenFlow rules) and flows

pmd/PMD-ID: *(DPDK) poll mode driver / poll mode driver ID*

- *Core to which this queue should be pinned. OVS_CORE_UNSPEC if the queue doesn't need to be pinned to a particular core.*
- for the kernel (netlink) datapath: *PMD_ID_NULL (INT_MAX)*

The “OVN view”



For each node:

- separate OVN “cluster” connected to a distributed **transit Logical_Switch** (geneve)
- **node Logical_Switch** (*ovn-worker-x*)
- **Logical_Router** (*ovn_cluster_router*) for E-W vs N-S routing decisions
- …

For each pod, on the node where it runs:

- **Logical_Switch_Port** connected to the node logical switch

For each service, on each node:

- **Load_Balancer** attached to the node logical router

Red Hat

kernel -> OpenFlow -> OVN Southbound

OpenFlow:

```
$ ovs-appctl ofproto/detrace ufid:afa73c50-ee58-4f65-8cd2-7fb5d0271ae4
cookie=0xaed19b4d,table_id=0,priority=100,match=in_port=4,
actions=set_field:0xb/0xffff->reg13,set_field:0x9->reg11,set_field:0x8->reg12,set_field:0x2->metadata,set_field:0x2->reg14,set_field:0/0xffff0000->r
eg13,resubmit(,8)

cookie=0x1d826318, table_id=14,priority=100,match=ip,metadata=0x1,nw_dst=10.244.1.42,actions=ct(table=15,zone=NXM_NX_REG11[0..15],nat)
```

OVN Southbound DB:

```
$ ovn-sbctl list Port_Binding aed19b4d
_uuid               : aed19b4d-7f17-4654-b092-9c0d534a0c0f
logical_port        : client
mac                 : ["00:00:00:00:31:10 10.244.1.4"]
```

Corresponding to NB 'client' logical port

```
$ ovn-sbctl list Logical_Flow 1d826318
external_ids        : {source="northd.c:12397", stage-hint="501b7c9c", stage-name=lr_in_defrag}
priority            : 100
match               : "ip && ip4.dst == 10.244.1.42"
actions             : "ct_dnat;"
```

A load balancer related OpenFlow rule hit (ct_dnat)

OVN Southbound DB:

```
$ ovn-sbctl list Port_Binding aed19b4d
_uuid               : aed19b4d-7f17-4654-b092-9c0d534a0c0f
logical_port        : client
mac                 : ["0a:58:0a:f4:01:04 10.244.1.4"]

$ ovn-sbctl list Logical_Flow 1d826318
external_ids        : {source="northd.c:12397", stage-hint="501b7c9c", stage-name=lr_in_defrag}
priority            : 100
match               : "ip && ip4.dst == 10.244.1.42"
actions             : "ct_dnat;"
```

stage-hint by ovn-northd to NB objects

OVN Northbound DB:

```
$ ovn-nbctl list Logical_Switch_Port client
addresses           : ["00:00:00:00:31:10 10.244.1.4"]
name                : client
up                  : true

$ ovn-nbctl list Load_Balancer 501b7c9c
_uuid               : 501b7c9c-b219-4159-b256-1fe633dafc15
name                : "lb0"
protocol            : tcp
vips                : {"10.244.1.42:80"="10.244.2.6:80"}
```

Load balancer
VIP/backend mapping

The updated “ovn-detrace” ovn command

```
$  man ovn-detrace
ovn-detrace(1)                         OVN Manual                         ovn-detrace(1)

NAME
       ovn-detrace - convert ``ovs-appctl ofproto/trace'' (*) output to combine OVN logical flow information.

SYNOPSIS
       ovn-detrace < file

DESCRIPTION
       The  ovn-detrace  program reads ovs-appctl ofproto/trace output on stdin, looking for flow cookies, and expand
       each cookie with corresponding OVN logical flows. It expands logical flow further with the north-bound
       information e.g. the ACL that generated the logical flow, when relevant.
[...]
(*) works perfectly fine with ovs-appctl ofproto/detrace output too.
```

```
cookie=0x1d826318,table_id=14,priority=100,match=ip,metadata=0x1,nw_dst=10.244.1.42,
actions=ct(table=15,zone=NXM_NX_REG11[0..15],nat)
  * Logical datapaths:
  *      "gw" (0ed8f608-8736-4158-80b1-c7341f7659a4) [ingress]
  * Logical flow: table=6 (lr_in_defrag), priority=100, match=(ip && ip4.dst == 10.244.1.42), actions=(ct_dnat;)
    * Load Balancer: lb0 protocol ['tcp'] vips {'10.244.1.42:80': '10.244.2.6:80'} ip_port_mappings {}
```

Connecting the dots

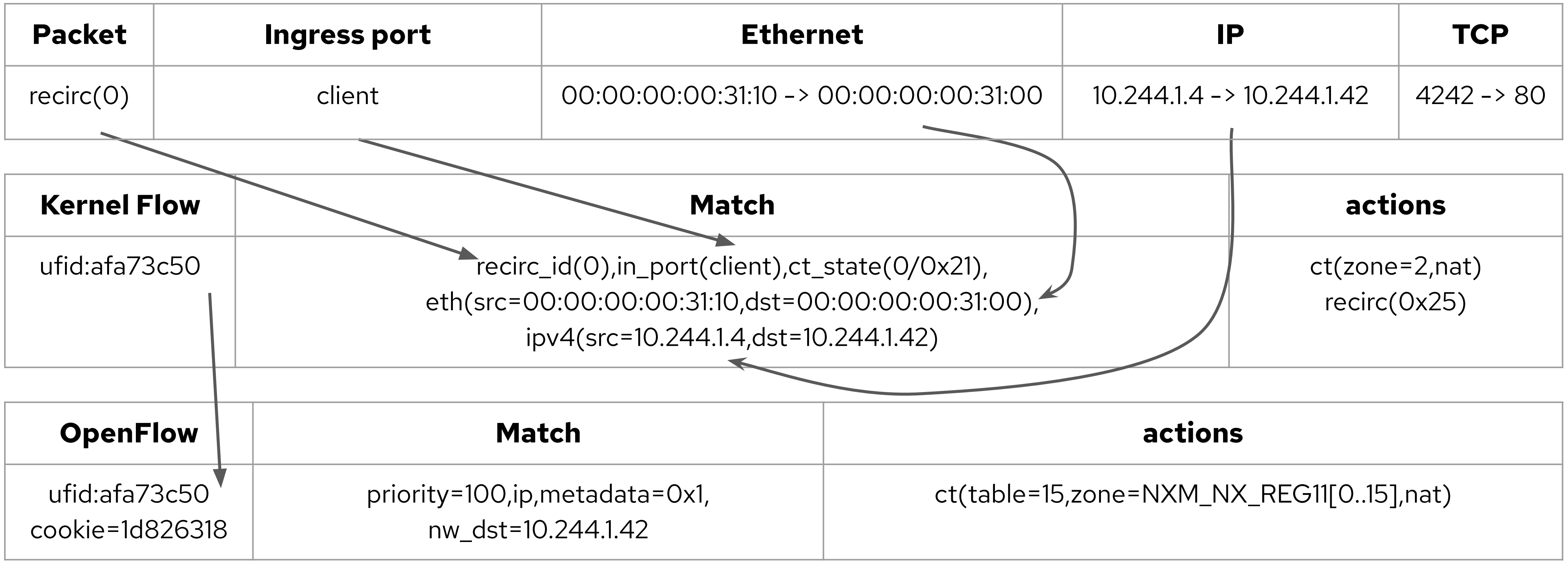
Packet	Ingress port	Ethernet	IP	TCP
recirc(0)	client	00:00:00:00:31:10 -> 00:00:00:00:31:00	10.244.1.4 -> 10.244.1.42	4242 -> 80

Connecting the dots

Packet	Ingress port	Ethernet	IP	TCP
recirc(0)	client	00:00:00:00:31:10 -> 00:00:00:00:31:00	10.244.1.4 -> 10.244.1.42	4242 -> 80

Kernel Flow	Match	actions
ufid:afa73c50	recirc_id(0),in_port(client),ct_state(0/0x21), eth(src=00:00:00:00:31:10,dst=00:00:00:00:31:00), ipv4(src=10.244.1.4,dst=10.244.1.42)	ct(zone=2,nat) recirc(0x25)

Connecting the dots



Packet	Ingress port	Ethernet	IP	TCP
recirc(0)	client	00:00:00:00:31:10 -> 00:00:00:00:31:00	10.244.1.4 -> 10.244.1.42	4242 -> 80

Kernel Flow	Match	actions
ufid:afa73c50	recirc_id(0),in_port(client),ct_state(0/0x21), eth(src=00:00:00:00:31:10,dst=00:00:00:00:31:00), ipv4(src=10.244.1.4,dst=10.244.1.42)	ct(zone=2,nat) recirc(0x25)

OpenFlow	Match	actions
ufid:afa73c50 cookie=1d826318	priority=100,ip,metadata=0x1, nw_dst=10.244.1.42	ct(table=15,zone=NXM_NX_REG11[0..15],nat)

Connecting the dots

Packet	Ingress port	Ethernet	IP	TCP
recirc(0)	client	00:00:00:00:31:10 -> 00:00:00:00:31:00	10.244.1.4 -> 10.244.1.42	4242 -> 80

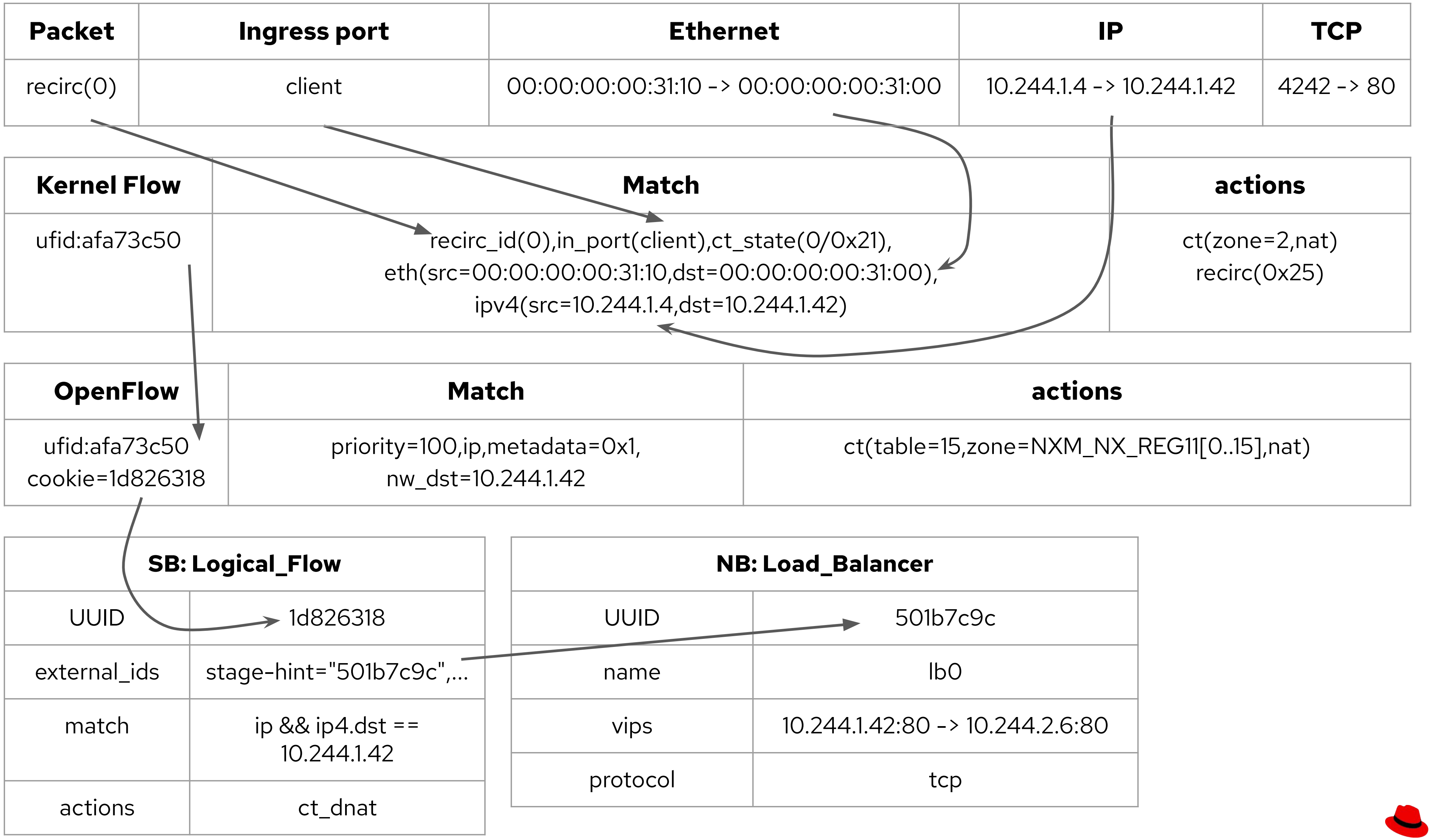
Kernel Flow	Match	actions
ufid:afa73c50	recirc_id(0),in_port(client),ct_state(0/0x21), eth(src=00:00:00:00:31:10,dst=00:00:00:00:31:00), ipv4(src=10.244.1.4,dst=10.244.1.42)	ct(zone=2,nat) recirc(0x25)

OpenFlow	Match	actions
ufid:afa73c50 cookie=1d826318	priority=100,ip,metadata=0x1, nw_dst=10.244.1.42	ct(table=15,zone=NXM_NX_REG11[0..15],nat)

SB: Logical_Flow	
UUID	1d826318
external_ids	stage-hint="501b7c9c",...
match	ip && ip4.dst == 10.244.1.42
actions	ct_dnat

Red Hat

Connecting the dots



Packet	Ingress port	Ethernet	IP	TCP
recirc(0)	client	00:00:00:00:31:10 -> 00:00:00:00:31:00	10.244.1.4 -> 10.244.1.42	4242 -> 80

Kernel Flow	Match	actions
ufid:afa73c50	recirc_id(0),in_port(client),ct_state(0/0x21), eth(src=00:00:00:00:31:10,dst=00:00:00:00:31:00), ipv4(src=10.244.1.4,dst=10.244.1.42)	ct(zone=2,nat) recirc(0x25)

OpenFlow	Match	actions
ufid:afa73c50 cookie=1d826318	priority=100,ip,metadata=0x1, nw_dst=10.244.1.42	ct(table=15,zone=NXM_NX_REG11[0..15],nat)

SB: Logical_Flow	
UUID	1d826318
external_ids	stage-hint="501b7c9c",...
match	ip && ip4.dst == 10.244.1.42
actions	ct_dnat

NB: Load_Balancer	
UUID	501b7c9c
name	lb0
vips	10.244.1.42:80 -> 10.244.2.6:80
protocol	tcp

Key takeaways?

- **there are no mysteries in networking**
  - it is complex and might look scary at first
  - **but** with the right tools it can become more **readable** and **easier** to understand
- **there's a link between abstractions at all the levels in the stack**
  - *from* highly optimized kernel datapath flows
  - *to* OVN (logical) routers and switches
- **use available tools** to better understand what the network is doing
- **when in doubt remember to check for:** UFIDs -> cookies -> stage-hints -> external-ids

Red Hat

Thank you!

Red Hat