Skydive

Analyzing Topology and Flows in OVS - DPDK and OVN OVS-DPDK Environments

Masco Yogananth

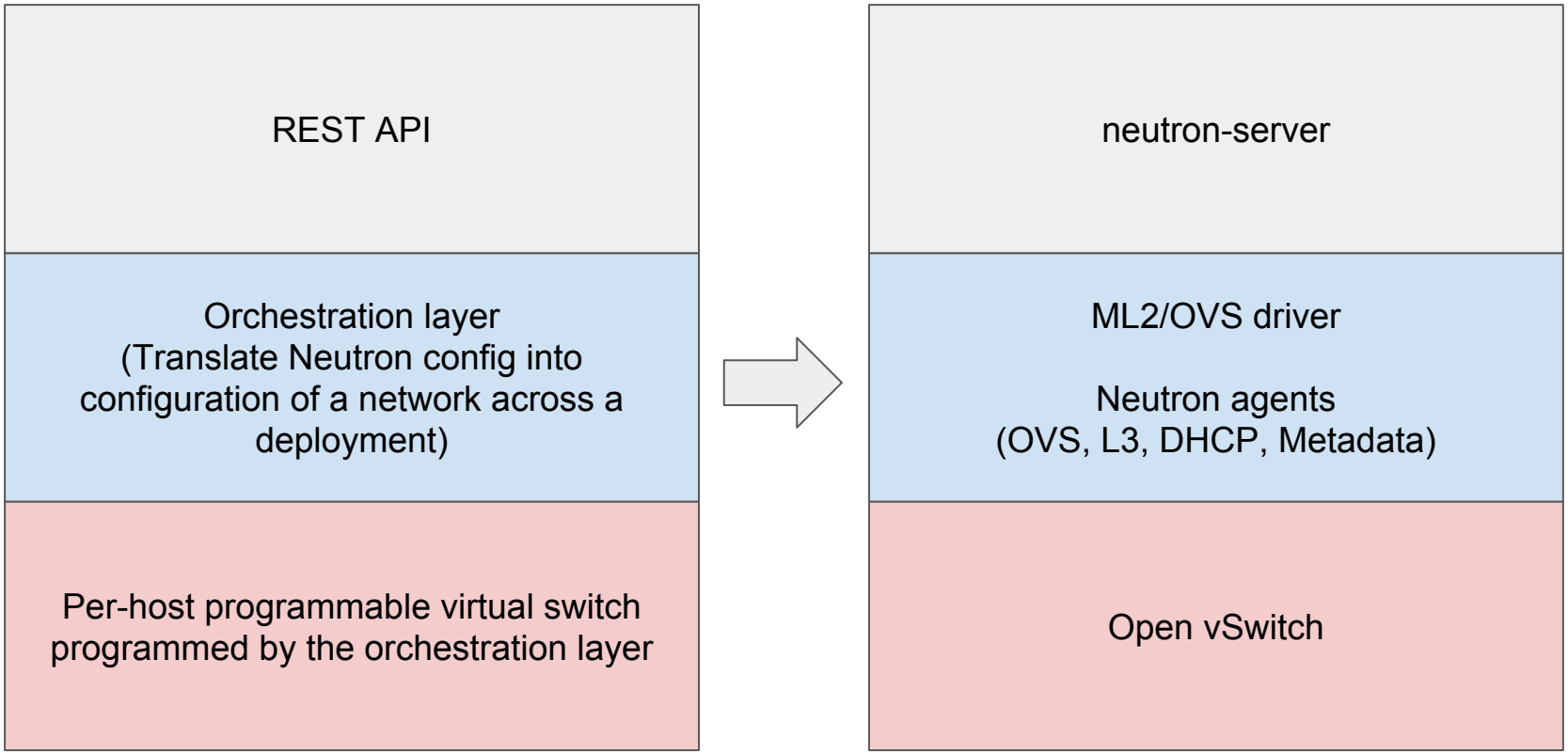
Numan Anil

redhat.

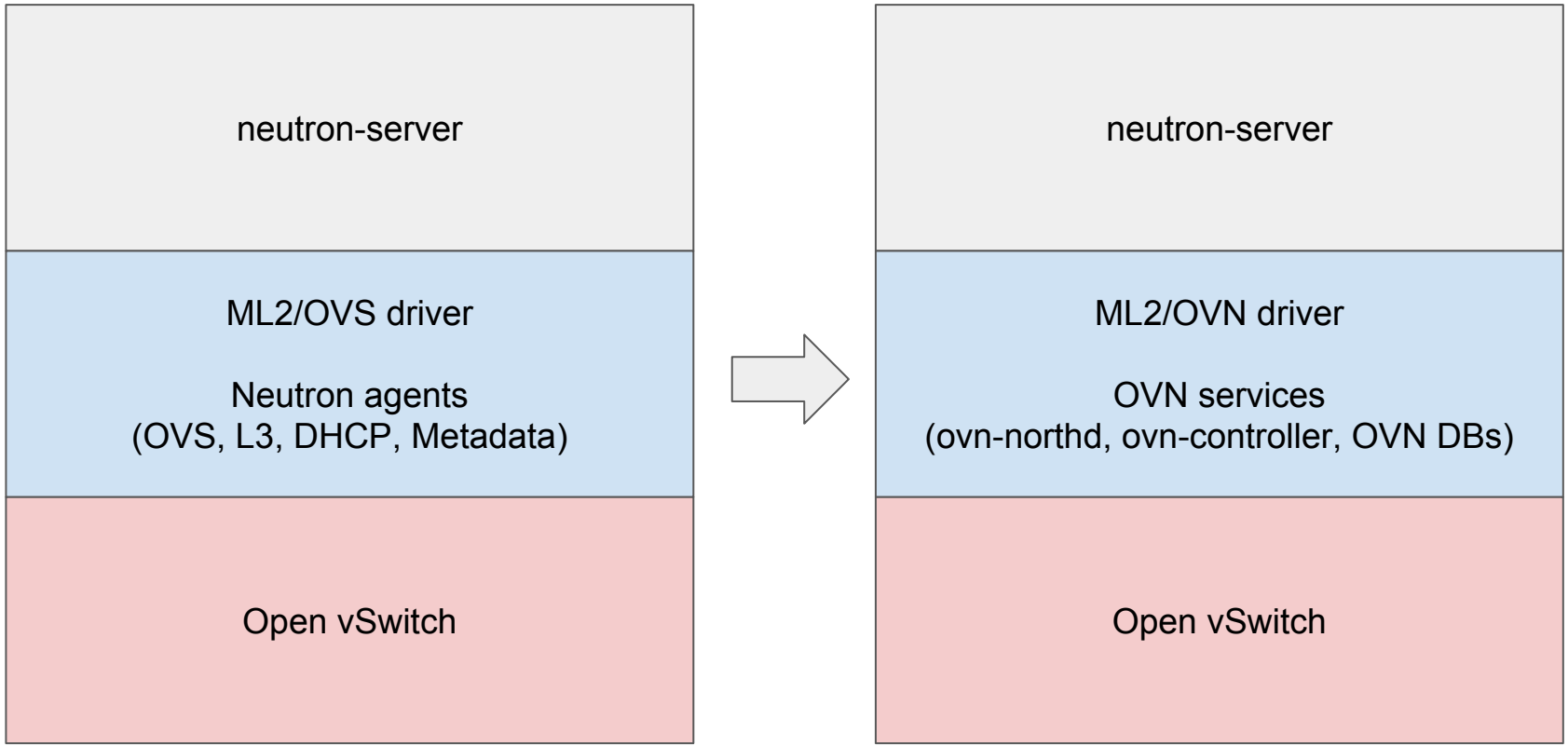
Introduction to OVN

- Launched early 2015 as part of the Open vSwitch project
- OVN is an evolution of the OVS project to move up the stack into OVS orchestration
- Provides l2 (switching) and l3 (routing) virtual network services.
- Has native dhcp v4/v6, internal dns support.
- Has native IPv6 support and other features.

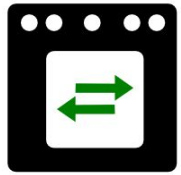
Openstack - Neutron



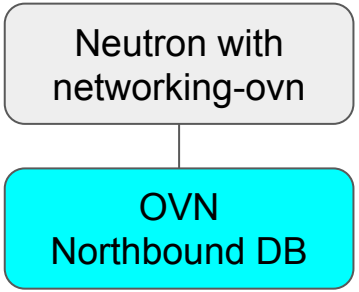
OpenStack - Neutron with OVN

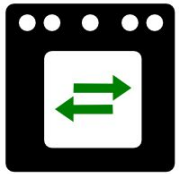


How does OVN work?

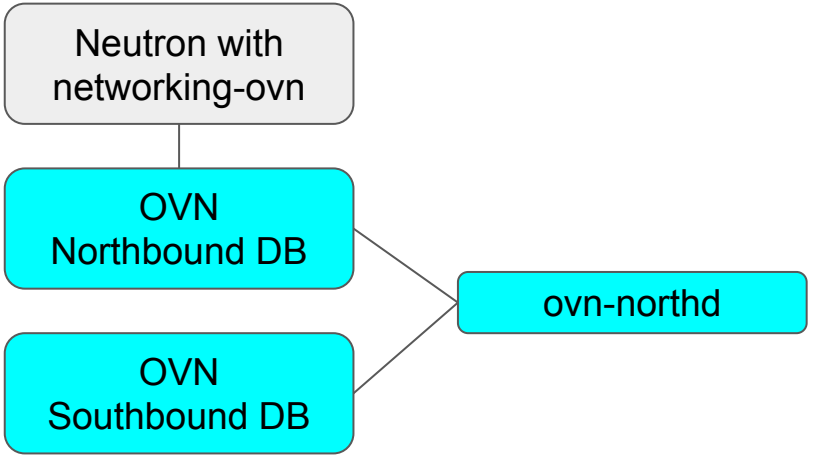


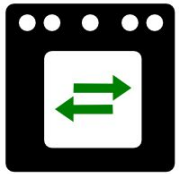
1. Logical Configuration in Northbound Database



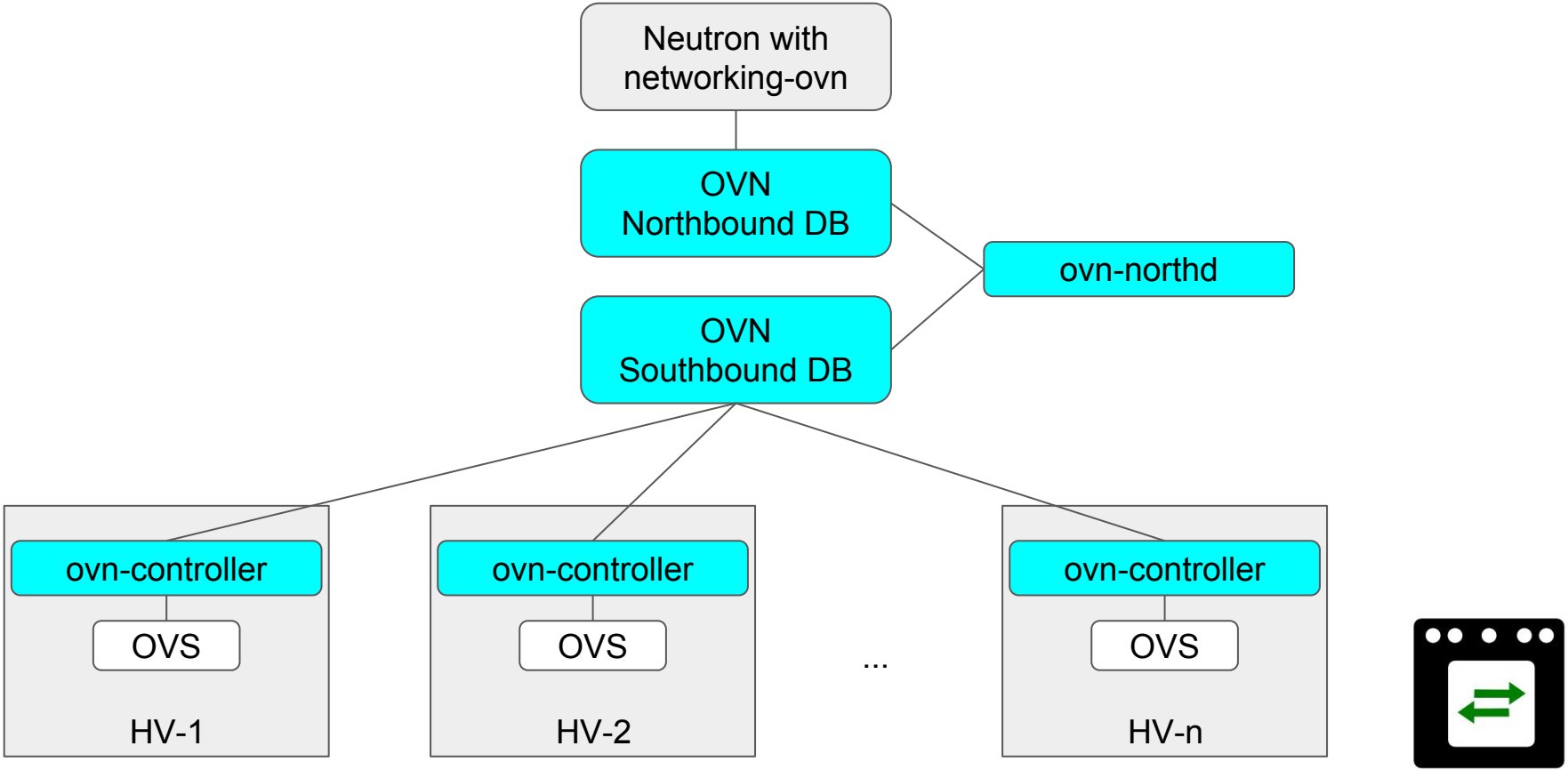


2. ovn-northd Populates Southbound Database

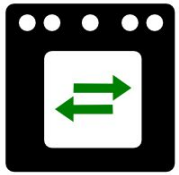




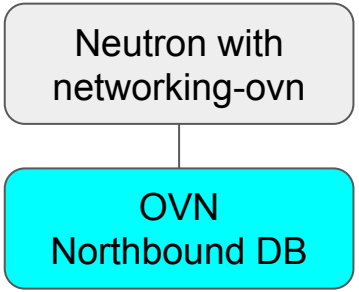
3. Hypervisors Generate Physical Flows

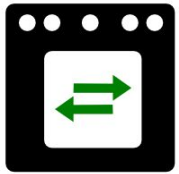


How does OVN work?
(with some examples this time)

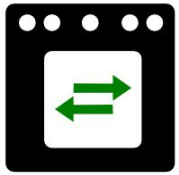


1. Logical Configuration in Northbound Database

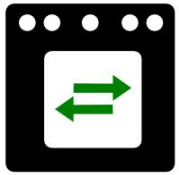




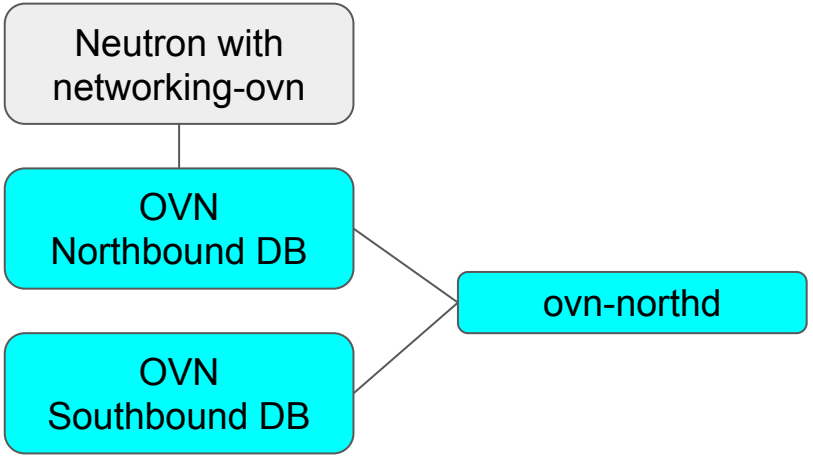
```
$ openstack network create demonet
+---------------------------+--------------------------------------+
| Field                     | Value                                |
+---------------------------+--------------------------------------+
...
| id                        | c6eb55a8-abee-463b-af26-297e8604bfc0 |
| mtu                       | 1442                                 |
| name                      | demonet                              |
| port_security_enabled     | True                                 |
...
+---------------------------+--------------------------------------+

$ ovn-nbctl show
    switch edd5a68b-09af-4b6e-bdbd-4320ca10e179 (neutron-c6eb55a8-abee-463b-af26-297e8604bfc0)

$ ovn-nbctl list Logical_Switch
_uuid               : edd5a68b-09af-4b6e-bdbd-4320ca10e179
acls                : []
external_ids        : {"neutron:network_name"=demonet}
load_balancer       : []
name                : "neutron-c6eb55a8-abee-463b-af26-297e8604bfc0"
other_config        : {}
ports               : []
qos_rules           : []
```

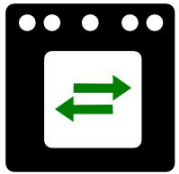


```
$ openstack subnet create demosubnet --network demonet --subnet-range 10.0.1.0/24

$ openstack port create --network demonet demoport1
+-----------------------+--------------------------------------------------------------------------+
| Field                 | Value                                                                    |
+-----------------------+--------------------------------------------------------------------------+
| fixed_ips             | ip_address='10.0.1.4', subnet_id='22e24e02-fa66-4820-b41d-95ab01a7e63b' |
| id                    | 9a0b5db4-064f-4fd9-9bb2-1cb2fd04f20b                                     |
| mac_address           | fa:16:3e:ce:8a:5b                                                        |
| name                  | demoport1                                                                |
| network_id            | c6eb55a8-abee-463b-af26-297e8604bfc0                                     |
…

$ openstack port create --network demonet demoport2
+-----------------------+--------------------------------------------------------------------------+
| Field                 | Value                                                                    |
+-----------------------+--------------------------------------------------------------------------+
| fixed_ips             | ip_address='10.0.1.12', subnet_id='22e24e02-fa66-4820-b41d-95ab01a7e63b' |
| id                    | 5889797e-25e2-4343-9b1b-92b3a47dbcdf                                     |
| mac_address           | fa:16:3e:9f:1a:65                                                        |
| name                  | demoport2                                                                |
| network_id            | c6eb55a8-abee-463b-af26-297e8604bfc0                                     |
…

$ ovn-nbctl show neutron-c6eb55a8-abee-463b-af26-297e8604bfc0
    switch edd5a68b-09af-4b6e-bdbd-4320ca10e179 (neutron-c6eb55a8-abee-463b-af26-297e8604bfc0)
        port 5889797e-25e2-4343-9b1b-92b3a47dbcdf
            addresses: ["fa:16:3e:9f:1a:65 10.0.1.12"]
        port 9a0b5db4-064f-4fd9-9bb2-1cb2fd04f20b
            addresses: ["fa:16:3e:ce:8a:5b 10.0.1.4"]
```



2. ovn-northd Populates Southbound Database



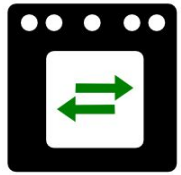


The Southbound database has physical runtime state, such as host details and physical locations of OVN ports.

```
$ ovn-sbctl show
Chassis "ddc8991a-d838-4758-8d15-71032da9d062"
    hostname: "centos7-ovn-devstack.os1.phx2.redhat.com"
    Encap vxlan
        ip: "172.16.189.6"
        options: {csum="true"}
    Encap geneve
        ip: "172.16.189.6"
        options: {csum="true"}
Chassis "b194d07e-0733-4405-b795-63b172b722fd"
    hostname: "centos7-ovn-devstack-2.os1.phx2.redhat.com"
    Encap geneve
        ip: "172.16.189.30"
        options: {csum="true"}
    Encap vxlan
        ip: "172.16.189.30"
        options: {csum="true"}
```

A port not yet bound to a physical host:

```
$ ovn-sbctl list Port_Binding ${DEMOPORT1_ID}
_uuid               : 2855b7d6-085a-47d3-a293-053fd6fc800b
chassis             : []
datapath            : a9e9fea0-c965-4fbb-af0e-5665ead9120d
logical_port        : "9a0b5db4-064f-4fd9-9bb2-1cb2fd04f20b"
mac                 : ["fa:16:3e:ce:8a:5b 10.0.1.4"]
options             : {}
parent_port         : []
tag                 : []
tunnel_key          : 1
type                : ""
```



Logical Flows

For more reading on the topic:
https://blog.russellbryant.net/2016/11/11/ovn-logical-flows-and-ovn-trace/

Logical Flows

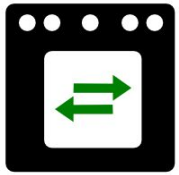
OpenFlow -- a table based match-action packet processing pipeline for a switch

Take the expressiveness of OpenFlow, but allow you to program entire networks:

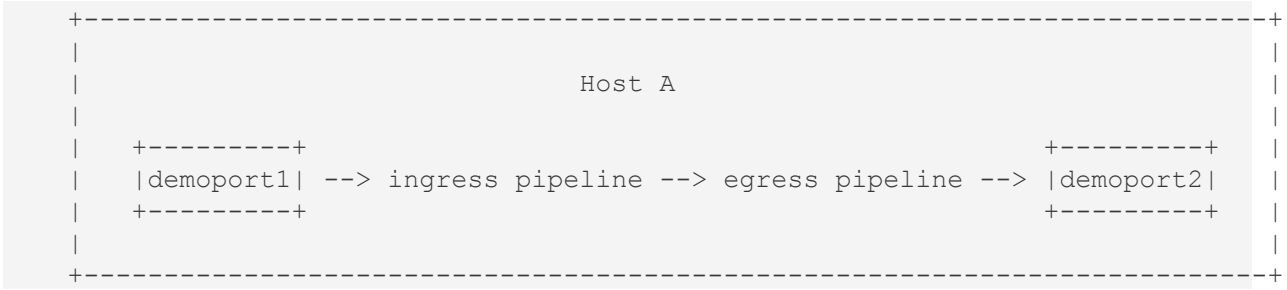
Logical Flows -- a table based match-action packet processing pipeline for logical networks, without regard for physical hosts or port locations.

In OVN, a network is programmed as an ingress and egress pipeline of logical flows.

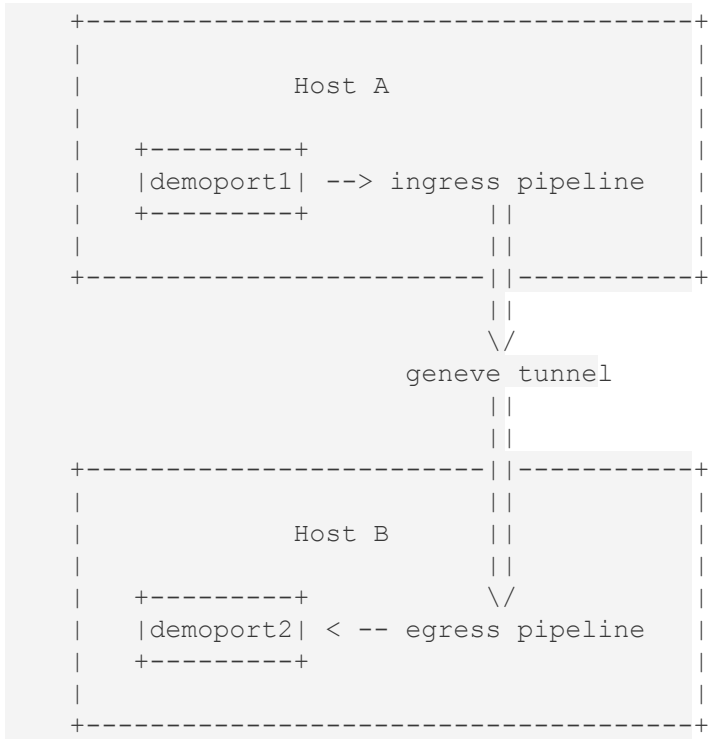
Includes the ability to program higher level concepts, like DHCP!



demoport1 and demoport2 on the same host:



demoport1 and demoport2 on separate hosts:



```
$ ovn-sbctl lflow-list ${DEMONET_ID}
… verbose output …
```

Set some more variables so we can do some tracing ...

```
$ DEMOPORT1_ID=9a0b5db4-064f-4fd9-9bb2-1cb2fd04f20b   $ DEMONET_ID=neutron-c6eb55a8-abee-463b-af26-297e8604bfc0
$ DEMOPORT1_MAC=fa:16:3e:ce:8a:5b                     $ DEMOPORT1_IP=10.0.1.4
$ DEMOPORT2_ID=5889797e-25e2-4343-9b1b-92b3a47dbcdf   $ DEMOPORT2_MAC=fa:16:3e:9f:1a:65
$ DEMOPORT2_IP=10.0.1.12
```

Use ovn-trace to see OVN logical flows in action. (Packet from demoport1 to demoport2, minimal output)

```
$ ovn-trace --minimal ${DEMONET_ID} "inport == \"${DEMOPORT1_ID}\" && eth.src == ${DEMOPORT1_MAC}
> && eth.dst == ${DEMOPORT2_MAC} && ip4.src == ${DEMOPORT1_IP} && ip4.dst == ${DEMOPORT2_IP}"
output("5889797e-25e2-4343-9b1b-92b3a47dbcdf");← Output to demoport2
```

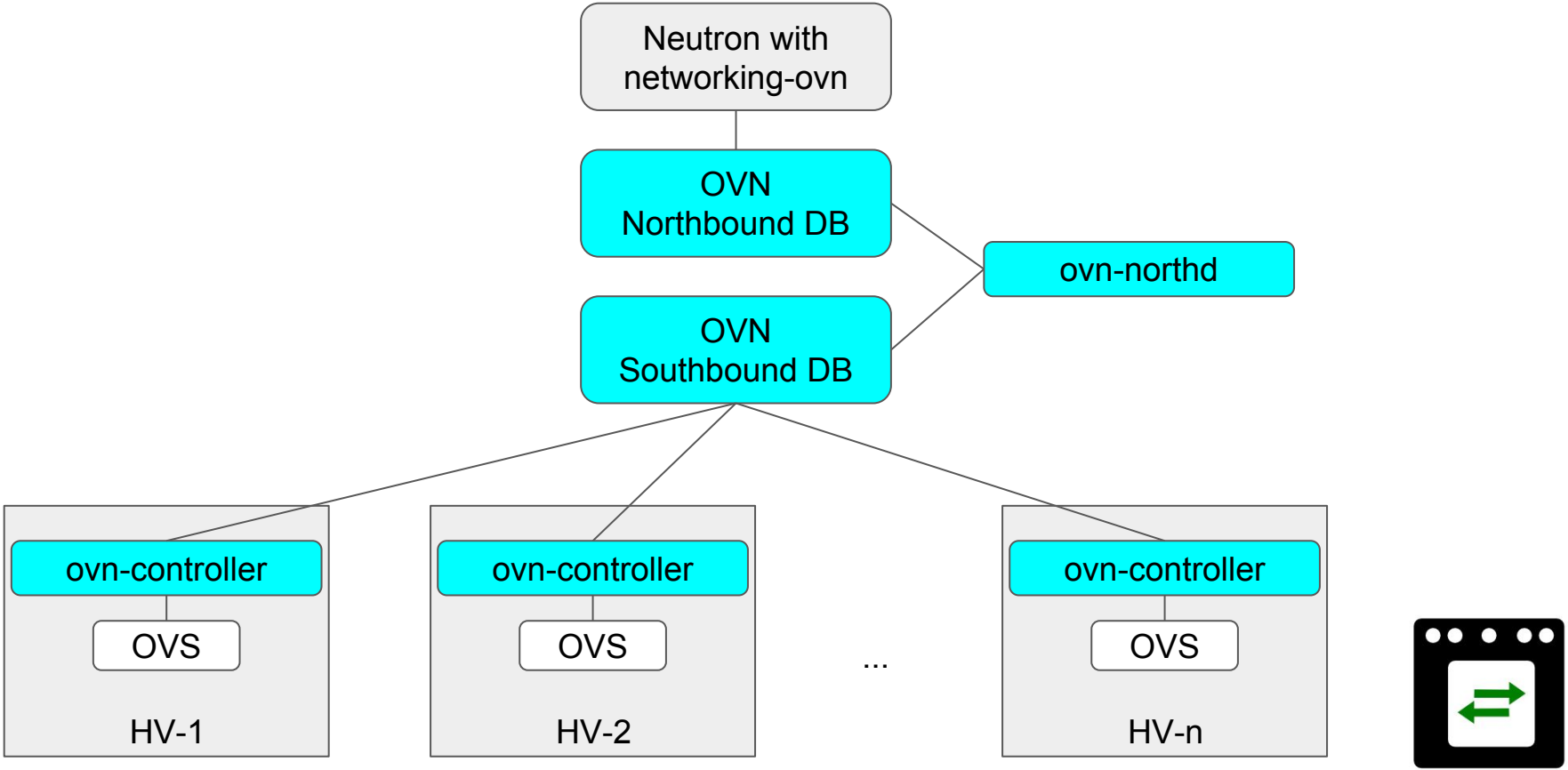
Now with more output showing the logical flow actions executed through the ingress and egress pipelines:

```
$ ovn-trace --summary ${DEMONET_ID} "inport == \"${DEMOPORT1_ID}\" && eth.src == ${DEMOPORT1_MAC}
> && eth.dst == ${DEMOPORT2_MAC} && ip4.src == ${DEMOPORT1_IP} && ip4.dst == ${DEMOPORT2_IP}"
ingress(dp="neutron-c6eb55a8-abee-463b-af26-297e8604bfc0", inport="9a0b5db4-064f-4fd9-9bb2-1cb2fd04f20b") {
    next;
    next;
    outport = "5889797e-25e2-4343-9b1b-92b3a47dbcdf";
    output;
    egress(dp="neutron-c6eb55a8-abee-463b-af26-297e8604bfc0", inport="9a0b5db4-064f-4fd9-9bb2-1cb2fd04f20b",
outport="5889797e-25e2-4343-9b1b-92b3a47dbcdf") {
        next;
        output;
        /* output to "5889797e-25e2-4343-9b1b-92b3a47dbcdf", type "" */;
    };
};
```

Even more flow detail, including source code references for what created those flows ...

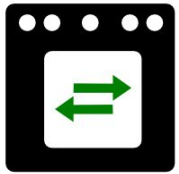
```
$ ovn-trace --detailed ...
```

3. Hypervisors Generate Physical Flows



ovn-controller

- High level view: a logical flows to host-specific OpenFlow compiler
- Watch for physical ports to come and go
- Update local OpenFlow programming to reflect logical flows and current physical port locations throughout the environment



A physical port is created on a hypervisor … (using a net namespace instead of a Nova VM here)

Instantiate demoport1 on hypervisor 1:

```
$ sudo ip netns add demo-ns1
$ sudo ovs-vsctl add-port br-int demo-ns1 -- set Interface demo-ns1 type=internal
$ sudo ip link set demo-ns1 netns demo-ns1
$ sudo ip netns exec demo-ns1 ip link set demo-ns1 address ${DEMOPORT1 MAC}
$ sudo ip netns exec demo-ns1 ip addr add ${DEMOPORT1_IP}/24 dev demo-ns1
$ sudo ip netns exec demo-ns1 ip link set demo-ns1 up
$ sudo ovs-vsctl set Interface demo-ns1 external_ids:iface-id=${DEMOPORT1_ID}
```

Now instantiate demoport2 on hypervisor 2:

```
[centos@centos7-ovn-devstack-2 ~]$ sudo ip netns add demo-ns2
$ sudo ovs-vsctl add-port br-int demo-ns2 -- set Interface demo-ns2 type=internal
$ sudo ip link set demo-ns2 netns demo-ns2
$ sudo ip netns exec demo-ns2 ip link set demo-ns2 address ${DEMOPORT2 MAC}
$ sudo ip netns exec demo-ns2 ip addr add ${DEMOPORT2_IP}/24 dev demo-ns2
$ sudo ip netns exec demo-ns2 ip link set demo-ns2 up
$ sudo ovs-vsctl set Interface demo-ns2 external_ids:iface-id=${DEMOPORT2_ID}
```

OVN now sees demoport1 and demoport2 on a chassis.

```
$ ovn-sbctl show
Chassis "ddc8991a-d838-4758-8d15-71032da9d062"
    hostname: "centos7-ovn-devstack"
    Encap vxlan
        ip: "172.16.189.6"
        options: {csum="true"}
    Encap geneve
        ip: "172.16.189.6"
        options: {csum="true"}
    Port_Binding "9a0b5db4-064f-4fd9-9bb2-1cb2fd04f20b"     ← demoport1
Chassis "b194d07e-0733-4405-b795-63b172b722fd"
    hostname: "centos7-ovn-devstack-2.os1.phx2.redhat.com"
    Encap geneve
        ip: "172.16.189.30"
        options: {csum="true"}
    Encap vxlan
        ip: "172.16.189.30"
        options: {csum="true"}
    Port_Binding "5889797e-25e2-4343-9b1b-92b3a47dbcdf"     ← demoport2
```

ovn-controller has now programmed OpenFlow flows on each chassis based on the desired network configuration and location of the ports.

… and it works! These pings traverse a geneve tunnel between the two hosts.

```
[centos@centos7-ovn-devstack ~]$ sudo ip netns exec demo-ns1 ping -c 1 ${DEMOPORT2_IP}
PING 10.0.1.12 (10.0.1.12) 56(84) bytes of data.
64 bytes from 10.0.1.12: icmp_seq=1 ttl=64 time=0.898 ms

--- 10.0.1.12 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.898/0.898/0.898/0.000 ms
```

```
[centos@centos7-ovn-devstack-2 ~]$ sudo ip netns exec demo-ns2 ping -c 1 ${DEMOPORT1_IP}
PING 10.0.1.4 (10.0.1.4) 56(84) bytes of data.
64 bytes from 10.0.1.4: icmp_seq=1 ttl=64 time=1.16 ms

--- 10.0.1.4 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1.163/1.163/1.163/0.000 ms
```

OVN with ovs-dpdk

- Nothing special to be done for OVN
- Configuring the dpdk parameters in ovs database (http://docs.openvswitch.org/en/latest/intro/install/dpdk/)
  - **dpdk-init**
  - **dpdk-lcore-mask**
  - **dpdk-socket-mem**
  - **dpdk-hugepage-dir**
  - **vhost-sock-dir**
- Create VMs using vhost-user or vhost-user-client ports.

Skydive

Why ?

- SDN is complex
- Highly dynamic
- Lack of open source tooling for troubleshooting

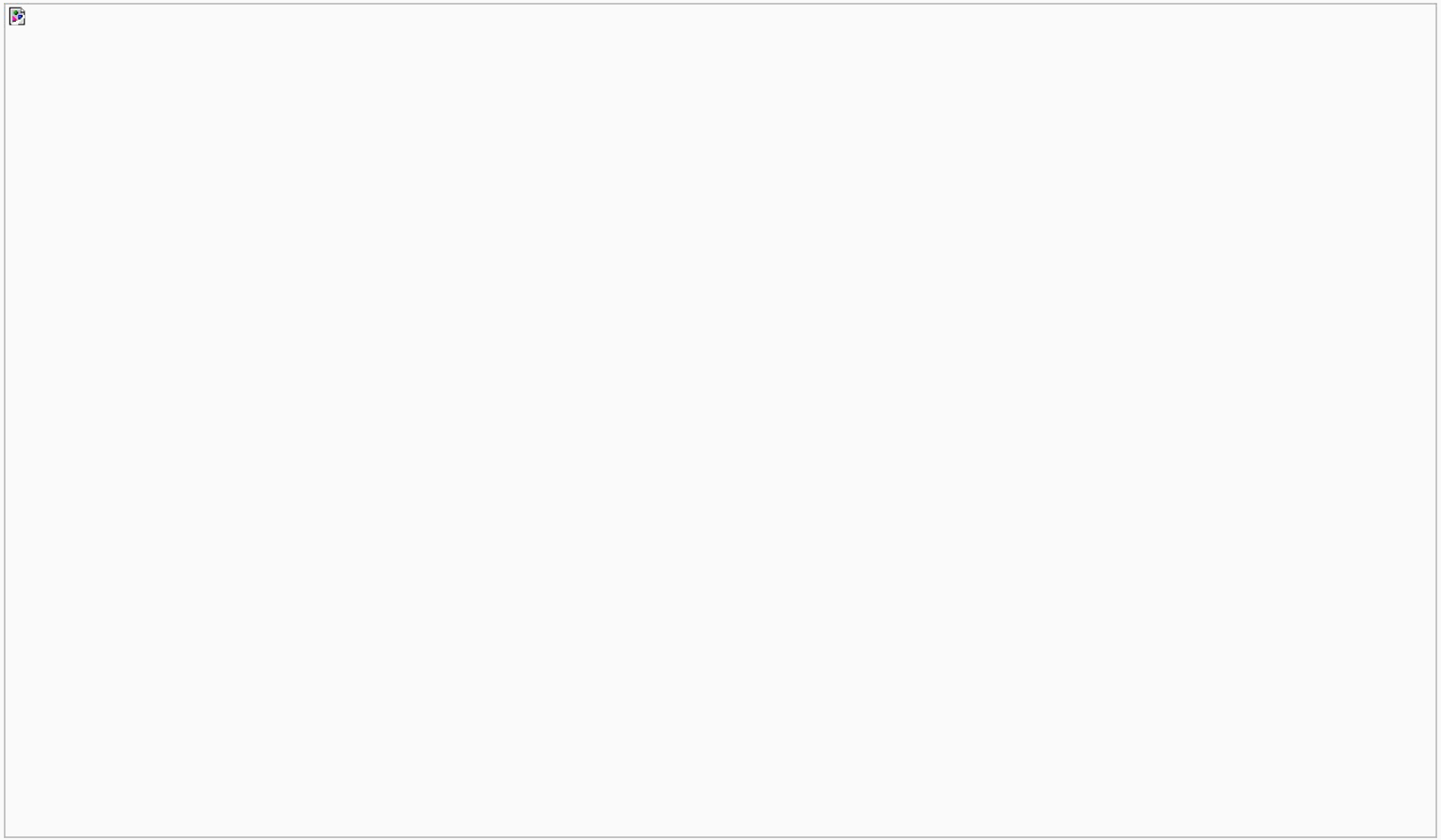
Goals

- SDN agnostic
- Real-time / post-mortem network analysis framework
- Lightweight, easy to deploy, single binary

Overview

- Agents
  - Capture topology and flows
  - Forwards to the analyzers
- Analyzers
  - Aggregate and store topology and flows
  - Serve API

Architecture



Topology probes

Topology probes:

- OVSDB
- NetLINK, ethtool
- NetNS

Topology connectors:

- Neutron
- Docker
- Opencontrail
- Openshift

Topology query

- Graph engine
  - Create a graph :
    - Nodes : interfaces, network objects with metadata
    - Links : L2, ownership, ...
- Event based
  - Graph listener through WebSocket (agents, Web UI, your software)
- Gremlin like query language
- Full history (elasticsearch, orientdb)

Topology query

```
    $ skydive client topology query -q 'G.V().Has("Type",
"ovsbridge").Out().Out().Has("Name", Without("br-int"))
    [ { "Host": "localhost.localdomain",
        "ID": "a190409e-f76e-4c8f-55b9-985e662a37c0",
        "Metadata": {
            "Driver": "veth",
            "IfIndex": 168,
            "MAC": "3e:88:b9:65:04:7e",
            "MTU": 1500,
            "Name": "vm1-eth0",
            "State": "UP",
            "Type": "veth",
            "UUID": "b6e9bf79-9b58-4b65-800e-1ddf9909d9dc" }}]
```

What we call a flow

- Layers :
  - Link, Network, Transport
- Metrics (packets, bytes)
- Source, destination, capture point
- ID, Tracking ID, L3Tracking ID
- Encapsulation support (GRE, VXLAN, MPLS, Geneve)

Flows

- Defined capture using the Skydive API
- Capture Types: sflow, afpackets, libpcap, ovsmirror, eBPF
- Traffic is captured on the agent
- BPF for all kind of capture
- Stored into a local flow table
- Push flow metrics to the analyzer
- Map endpoints to known interfaces
- Stored into database

Flows

- Still the same Gremlin language
- … and the history

- Examples of Gremlin queries
  - g.Flows().Has(‘TrackingID’, ‘123’).Hops()
  - g.Flows().Has(‘Network.A’, ‘192.168.0.1’).Hops()
  - g.Context(“-1h”).V().Has(‘Name’, ‘br-int’).Flows().Count()

Features to Note

- Alerts

```
skydive client alert create --expression "G.V().Has('Name', 'eth0', 'State', 'DOWN')"
{
  "UUID": "185c49ba-341d-41a0-6f96-f3224140b2fa",
  "Expression": "G.V().Has('Name', 'eth0', 'State', 'DOWN')",
  "CreateTime": "2016-12-29T13:29:05.273620179+01:00"
}
```

- Packet/Traffic generator
  ICMP, TCP, UDP
- Python client
- High Availability

DEMO

Q&A

Thanks :)