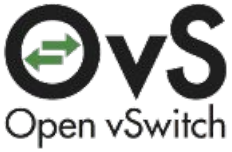
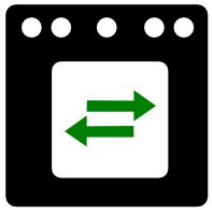


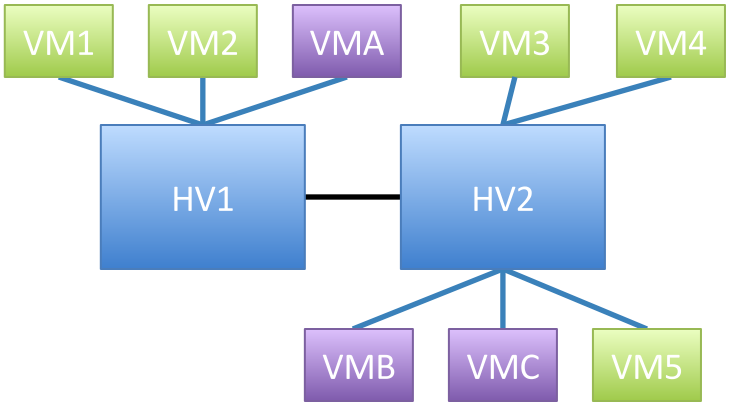


OVN: An SDN System for Virtual Networking

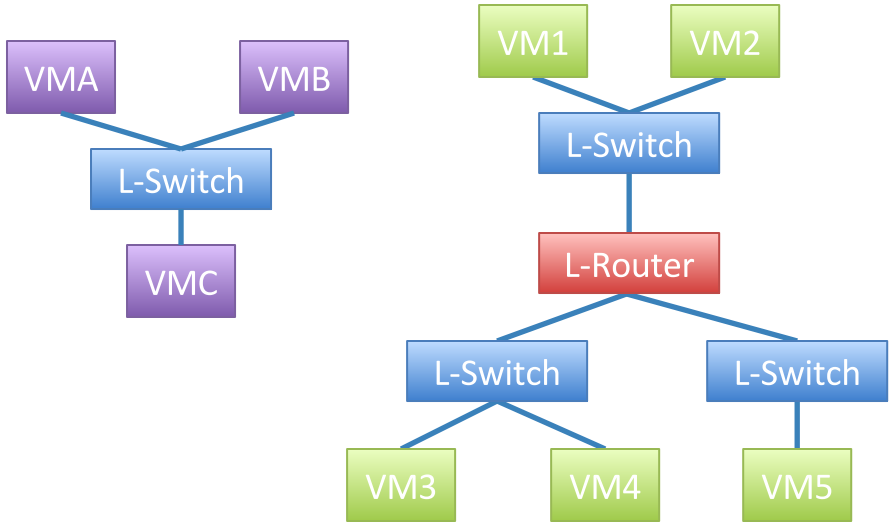
Ben Pfaff - @Ben_Pfaff
Justin Pettit - @Justin_D_Pettit

Virtual Networking Overview

Provides a logical network abstraction on top of a physical network



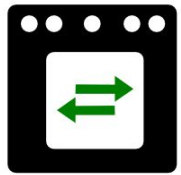
Physical



Logical

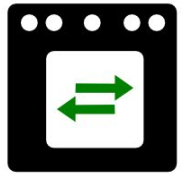
What is OVN?

- Virtual Networking for Open vSwitch (OVS)
- Developed within the OVS project
- Linux Foundation Collaborative Project
- First release of OVN comes with OVS 2.6
- First release of Neutron integration (networking-ovn) available in OpenStack Newton



Feature Overview

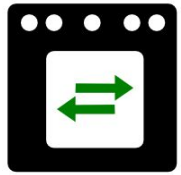
- Manages overlays and physical network connectivity
- Flexible security policies (ACLs)
- Distributed L3 routing, IPv4 and IPv6
- Native support for NAT, load balancing, DHCP
- Works with Linux, DPDK, and Hyper-V
- L2 and L3 gateways
- Designed to be integrated into another system
  - OpenStack, Kubernetes, Docker, Mesos, oVirt



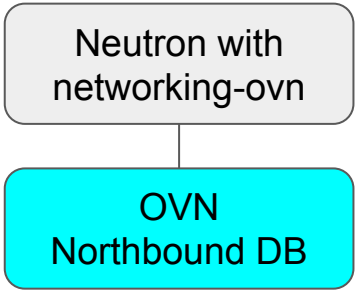
Designed to Scale

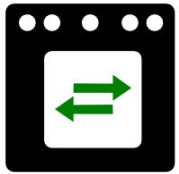
- Configuration coordinated through databases
- Local controller converts logical flow state into physical flow state
  - Centrally creating each hypervisor's view is expensive
  - Identical state sent to each hypervisor
- Desired state clearly separated from run-time state
  - Easier to reason about the system
  - Replication story clear
- Grouping techniques reduce Cartesian Product issues
  - High-level grouping constructs in database
  - Use of conjunctive match in switch

How does OVN work?

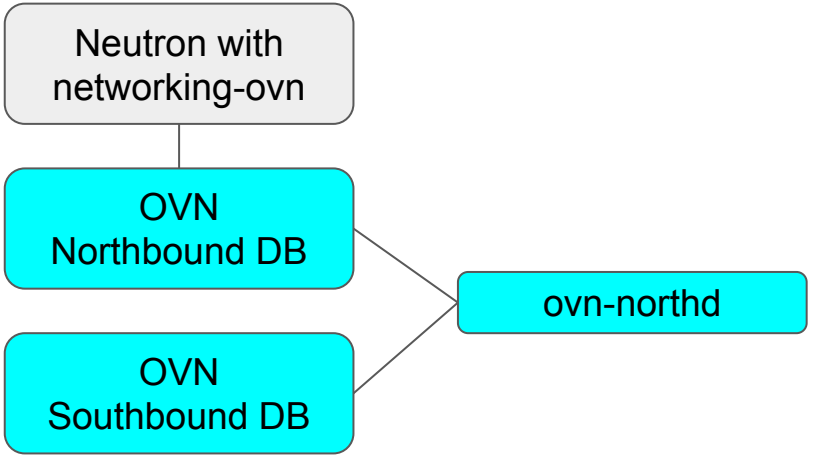


1. Logical Configuration in Northbound Database

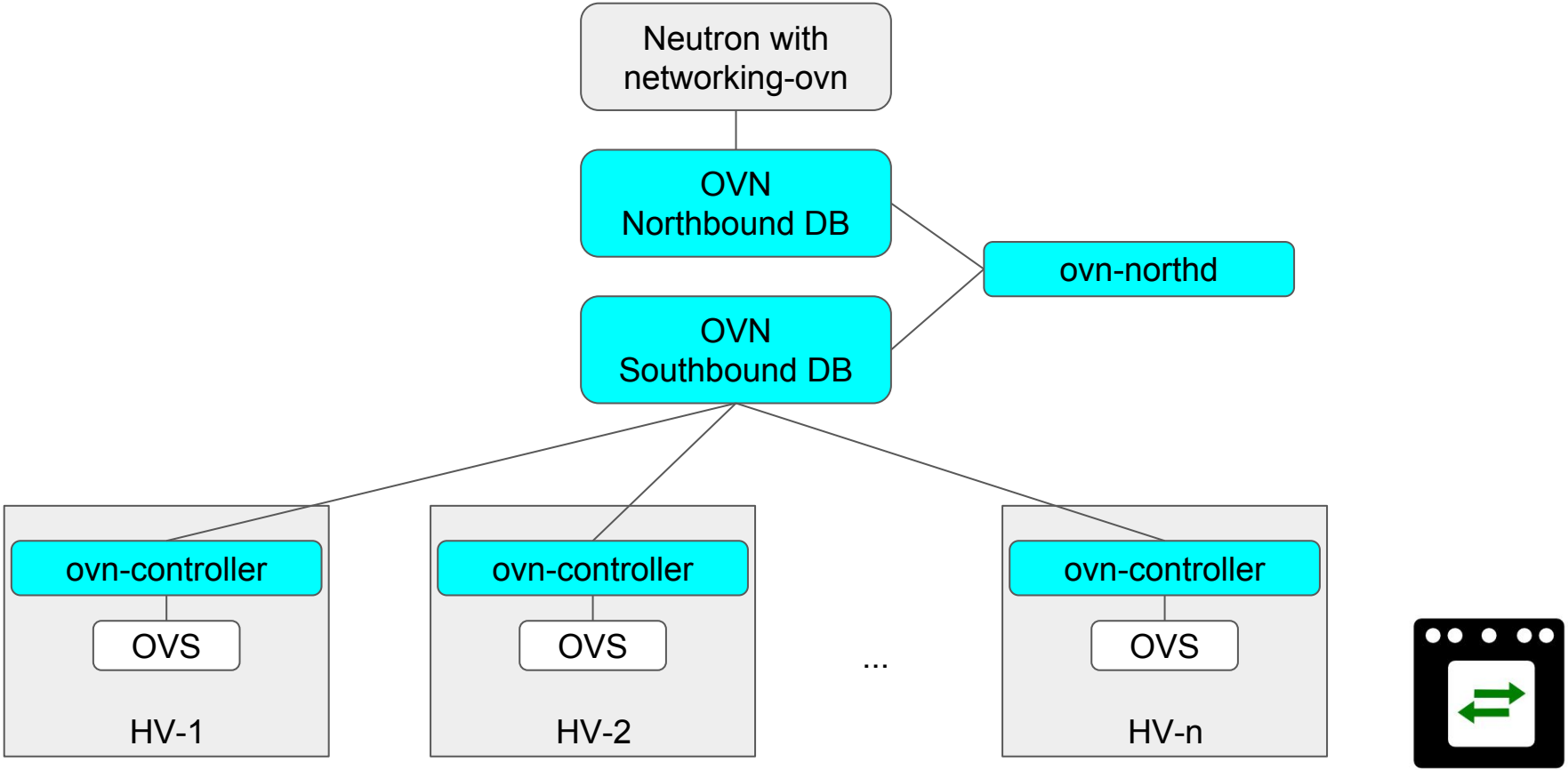




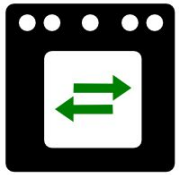
2. ovn-northd Populates Southbound Logical Flows



3. Hypervisors Generate Physical Flows



Logical Flows

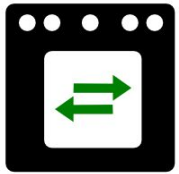


Strategy

Flow-based modeling works pretty well for network virtualization, but raw OpenFlow has undesirable properties:

- Uses physical names, that change on every hypervisor
- Non-composable flow matching model
- Naming doesn’t scale
- Difficult to simulate a chain of network elements

To solve these problems, we added some primitives to Open vSwitch and then added a layer of indirection.



Physical Names

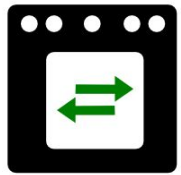
OpenFlow uses physical names. On HV 1:

- VM X might be on OpenFlow port 2 (or whatever) or it might require sending a packet on a tunnel to HV 2 with tunnel key 12345.
- “eth0” might be OpenFlow port 1 (or whatever).

OVN uses logical names. On HV 1:

- The name for VM X is always “X” regardless of its current location.
- “eth0” probably doesn’t have a name at all.

This single change makes a huge difference: OVN can distribute a single logical flow table to every hypervisor without change.



Non-Composable Flow Matching

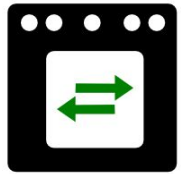
It's cheap in OpenFlow to match on IP source addresses:

ip_src=*a*
ip_src=*b*
ip_src=*c*

or on IP destination addresses:

ip_dst=*d*
ip_dst=*e*
ip_dst=*f*
ip_dst=*g*

That's 3 + 4 flows.



Non-Composable Flow Matching

The effect of those 3 + 4 flows is logical “or”:

$$\text{ip\_src} \in \{a,b,c\} \vee \text{ip\_dst} \in \{d,e,f,g\}$$

What if you want “and” instead of “or”, like this?

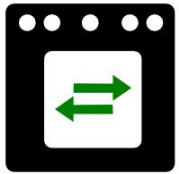
$$\text{ip\_src} \in \{a,b,c\} \wedge \text{ip\_dst} \in \{d,e,f,g\}$$

OpenFlow falls apart: this requires 3 × 4 flows. This is not just surprising, it is difficult to explain to users.

In OVN, such expressions have similar syntax and compose at higher levels:

```
ip4.src == {a,b,c} || ip4.dst == {d,e,f,g}
ip4.src == {a,b,c} && ip4.dst == {d,e,f,g}
```

The implementation is novel but out of scope for general presentation.



Naming Doesn’t Scale

ACLs often cover huge groups of VMs. If the ACLs are written the obvious way using expressions, e.g.

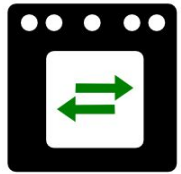
inport == {VM1,VM2,VM3,...,VMn} && …

then individual ACLs will be large and any change to a group will cause a lot of churn.

OVN introduces a grouping construct called an “address set” whose membership can be managed in a straightforward way. Then the ACL becomes:

inport == $vmgroup && …

and scaling improves dramatically.

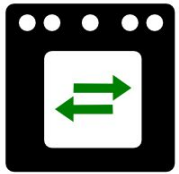


Simulating Networks

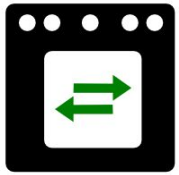
OpenFlow doesn’t include support simulating hops through multiple devices, but this is important for network virtualization. A packet might travel through multiple logical switches and routers before it actually gets sent to VM(s) on the same or remote HVs.

OVN simulates networks using Open vSwitch features:

- OVS 2.6 and earlier have “patch ports”.
- OVS 2.7 and later add a “clone” action.



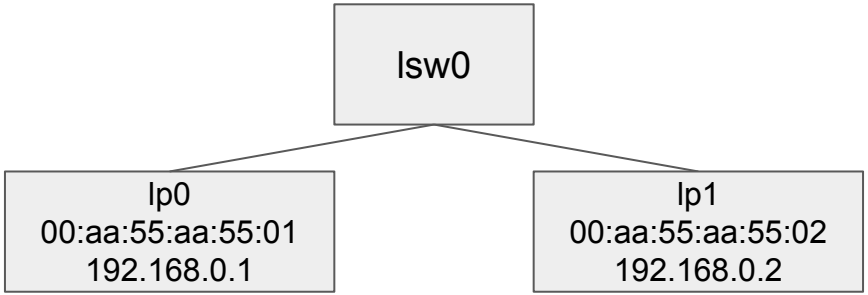
Understanding OVN

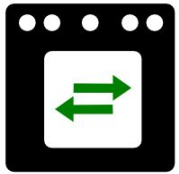


Using ovn-trace to understand OVN

"What's happening to these packets?"
"What if…?"
ovn-trace answers these questions and more.

- Only requires access to southbound database.
- Multiple output formats with varying levels of detail.
- Physical network layout independent.
- Provides references back to source code, to make debugging easier for developers
- Omits trivial match-action tables that would otherwise clutter output.

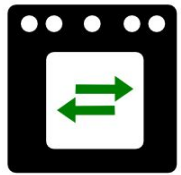




ovn-trace example #1: detailed output

$ ovn-trace lsw0 ‘inport == "lp0" && eth.src == 00:aa:55:aa:55:01 && eth.dst == ff:ff:ff:ff:ff:ff’

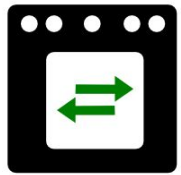
```
ingress(dp="lsw0", inport="lp0")
 0. ls_in_port_sec_l2 (ovn-northd.c:2826): inport == "lp0" && eth.src == {00:aa:55:aa:55:01}, priority 50
        next(1);
13. ls_in_l2_lkup (ovn-northd.c:3071): eth.mcast, priority 100
        outport = "_MC_flood";
        output;

multicast(dp="lsw0", mcgroup="_MC_flood")
        egress(dp="lsw0", inport="lp0", outport="lp0")
        /* omitting output because inport == outport && !flags.loopback */

        egress(dp="lsw0", inport="lp0", outport="lp1")
        8. ls_out_port_sec_l2 (ovn-northd.c:3146): eth.mcast, priority 100
        output;
        /* output to "lp1", type "" */
```



ovn-trace example #1: minimal output

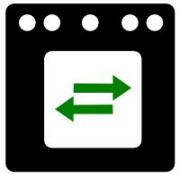
$ ovn-trace lsw0 ‘inport == "lp0" && eth.src == 00:aa:55:aa:55:01 && eth.dst == ff:ff:ff:ff:ff:ff’

output("lp1");



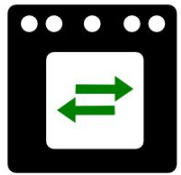
ovn-trace example #2: DHCP request processing

```
$ ovn-trace lsw0 'inport == "lp0" && eth.src == 00:aa:55:aa:55:01 && eth.dst == ff:ff:ff:ff:ff:ff
&& ip4.src == 0.0.0.0 && ip4.dst == 255.255.255.255 && udp.src == 68 && udp.dst == 67'

/* We assume that this packet is DHCPDISCOVER or DHCPREQUEST. */;
put_dhcp_opts(offerip = 192.168.0.1, netmask = 255.255.255.0,
                      router = 192.168.0.254, server_id = 192.168.0.253,
                      lease_time = 3600);
eth.dst = 00:aa:55:aa:55:01;
eth.src = 00:aa:55:aa:55:fd;
ip4.dst = 192.168.0.1;
ip4.src = 192.168.0.253;
udp.src = 67;
udp.dst = 68;
output("lp0");
```



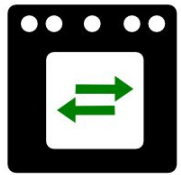
OVN in an Academic Setting

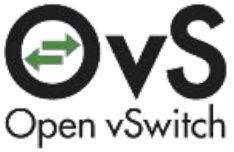
- Create arbitrarily complex virtual networks on one or more systems
- Small code base that is easily hackable for new features
- Use logical flows to more easily program OVS (ie, use ovn-controller without northbound components)

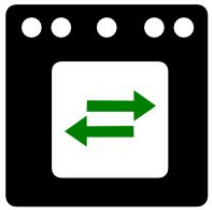


Other Resources

- OVS/OVN Repository
  - https://github.com/openvswitch/ovs
- Kubernetes OVN Plugin
  - https://github.com/openvswitch/ovn-kubernetes
- OVS Orbit Podcast
  - https://ovsorbit.org/
- OVS Conference, 7-8 November 2016 in San Jose, CA
  - http://openvswitch.org/







Thank you!

Ben Pfaff - @Ben_Pfaff
Justin Pettit - @Justin_D_Pettit