Open vSwitch

From: OpenStack: OVS Deep Dive

Justin Pettit Eric Lopez 07 November 2013

Features

- Open vSwitch supports the following features:
- Visibility into inter-VM communication via NetFlow, sFlow(R), IPFIX, SPAN, RSPAN, and GRE-tunneled mirrors
- LACP (IEEE 802.1AX-2008)
- Standard 802.1Q VLAN model with trunking
- Multicast snooping
- IETF Auto-Attach SPBM and rudimentary required LLDP support
- BFD and 802.1ag link monitoring
- STP (IEEE 802.1D-1998) and RSTP (IEEE 802.1D-2004)
- Fine-grained QoS control
- Support for HFSC qdisc
- Per VM interface traffic policing
- NIC bonding with source-MAC load balancing, active backup, and L4 hashing
- OpenFlow protocol support (including many extensions for virtualization)
- IPv6 support
- Multiple tunneling protocols (GRE, VXLAN, STT, and Geneve, with IPsec support)
- Remote configuration protocol with C and Python bindings
- Kernel and user-space forwarding engine options
- Multi-table forwarding pipeline with flow-caching engine
- Forwarding layer abstraction to ease porting to new software and hardware platforms

Status: check with Claude (4.5)

- Open vSwitch (OVS) remains widely used and relevant, particularly in:
  - **OpenStack deployments** - still the default virtual switch
  - **Kubernetes networking** - used by CNI plugins like OVN-Kubernetes, Antrea
  - **NFV/SDN environments** - for programmable networking
- **Current landscape includes:**
- **eBPF-based alternatives gaining traction:**
  - **Cilium** - increasingly popular for Kubernetes, offers better performance
  - **Calico** (eBPF mode) - competing in container networking
- **Cloud provider solutions:**
  - AWS VPC networking (proprietary)
  - Azure VFP (Virtual Filtering Platform)
  - GCP Andromeda
- **Hardware offload:**
  - DPDK-accelerated OVS
  - SmartNICs with OVS offload
- **Status:** OVS isn't being replaced wholesale, but:
  - For new Kubernetes deployments, many choose Cilium or Calico for better observability and performance
  - Legacy virtualization and OpenStack still heavily use OVS
  - OVN (Open Virtual Network) extends OVS with L3 capabilities and remains active
  - The shift is toward eBPF-based solutions for containers while OVS maintains its position in traditional virtualization and specific SDN use cases.

Status: check with Meta AI (llama-4)

- Open vSwitch (OVS) remains a popular and widely-used open-source virtual switch in the world of networking and cloud networking. Its status can be summarized as follows:
  - **Still widely used and maintained**: OVS is actively maintained by a large community of developers and is widely used in various cloud and networking projects, such as OpenStack, Kubernetes, and NFV (Network Function Virtualization) environments.
- **Key features and strengths**:
  - **High-performance switching**: OVS provides high-performance switching capabilities, with support for various hardware acceleration technologies like DPDK (Data Plane Development Kit).
  - **Flexibility and programmability**: OVS is highly configurable and programmable, with support for OpenFlow, OVSDB (Open vSwitch Database), and other protocols.
  - **Virtualization and containerization**: OVS is widely used in virtualized and containerized environments, providing connectivity and networking services for VMs and containers.

- **Other popular alternatives**:
- While OVS remains a popular choice, other alternatives have gained traction in recent years:
- **FD.io (Vector Packet Processing)**: FD.io is an open-source project providing a high-performance, scalable, and modular packet processing framework. It's used in various projects, including Cisco's VPP (Vector Packet Processing).
- **Contiv-VPP**: Contiv-VPP is a Kubernetes networking plugin that leverages VPP (Vector Packet Processing) technology to provide high-performance networking for containerized workloads.
- **Calico**: Calico is a popular open-source networking project that provides a scalable and secure networking solution for cloud-native applications. While not a traditional virtual switch, Calico provides a networking fabric that can be used in conjunction with OVS or other switching technologies.
- **Cilium**: Cilium is another popular open-source project that provides a networking and security solution for containerized workloads. It uses eBPF (Extended Berkeley Packet Filter) technology to provide high-performance networking and security features.

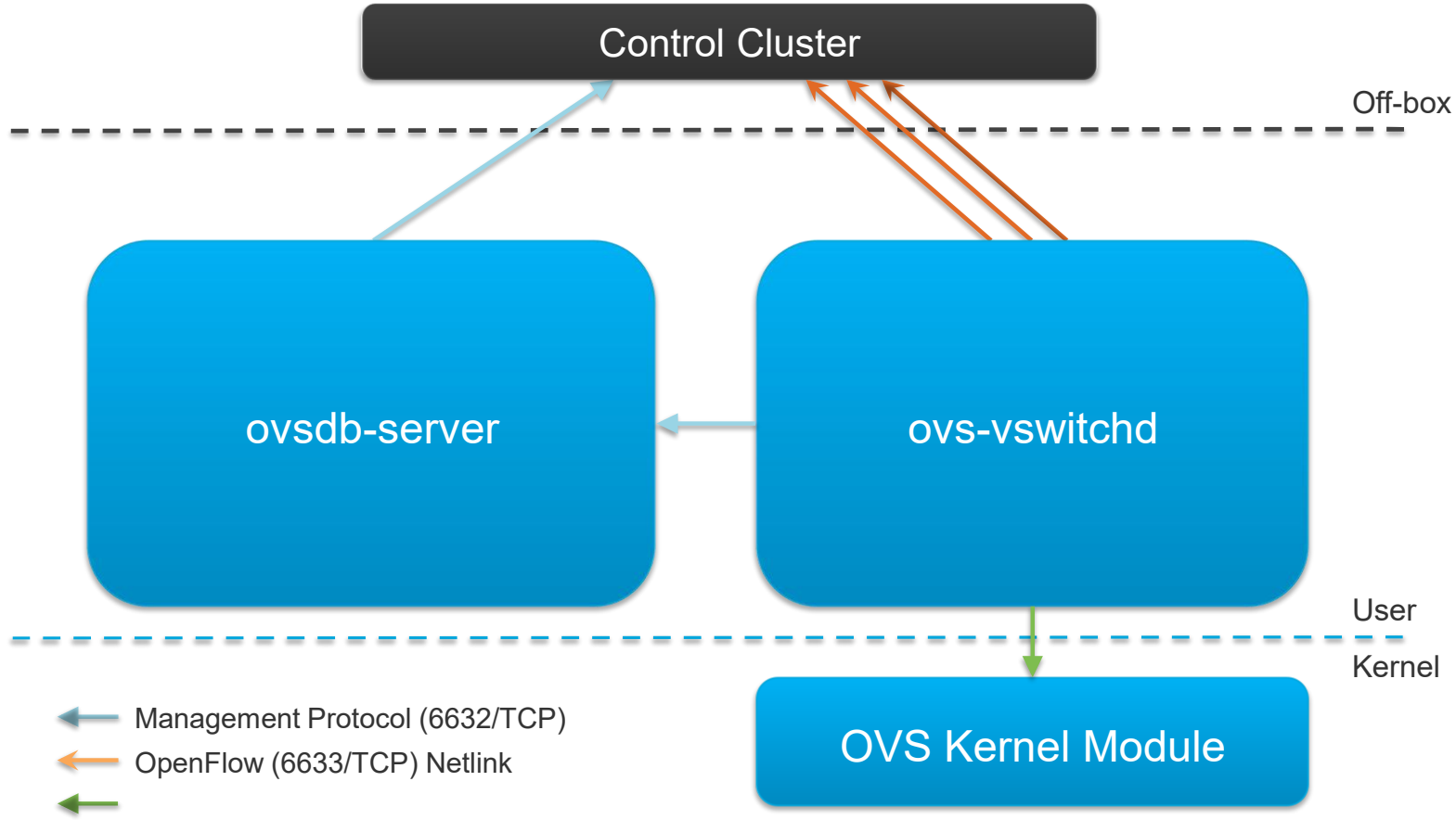
Cloud and networking trends:

The networking landscape is evolving, with trends like cloud-native networking, NFV, and SDN (Software-Defined Networking) driving the development of new networking technologies. OVS remains a key player in this ecosystem, but other technologies like those mentioned above are gaining traction.

In summary, Open vSwitch is still a widely-used and maintained open-source virtual switch, but other alternatives are gaining popularity, driven by the evolving needs of cloud-native networking, NFV, and SDN.

(Partial) List of Contributors

Main Components



OVS Components

- **Open vSwitch Database (OVSDB)** contains switch configuration and keeps track of created and modified interfaces. All the configuration is stored on persistent storage and survives a reboot. The OVSDB-server communicates with ovs-vswitchd and the controller using the OVSDB management protocol.
- **Open vSwitch Daemon:** The Open vSwitch daemon (ovs-vswitchd) is one of the major components of Open vSwitch. It communicates with the controller using OpenFlow and with the OVSDB-server through the OVSDB management protocol. The ovs-vswitchd communicates with the kernel module over netlink (a Linux kernel interface used for creating a connection between userspace processes and the kernel). It supports multiple independent data paths, known as bridges.

CONFIGURATION DATABASE

ovsdb-server

Database that holds switch-level configuration

- Bridge, interface, tunnel definitions
  OVSDB and OpenFlow controller addresses

Configuration is stored on disk and survives a reboot Custom database with nice properties:

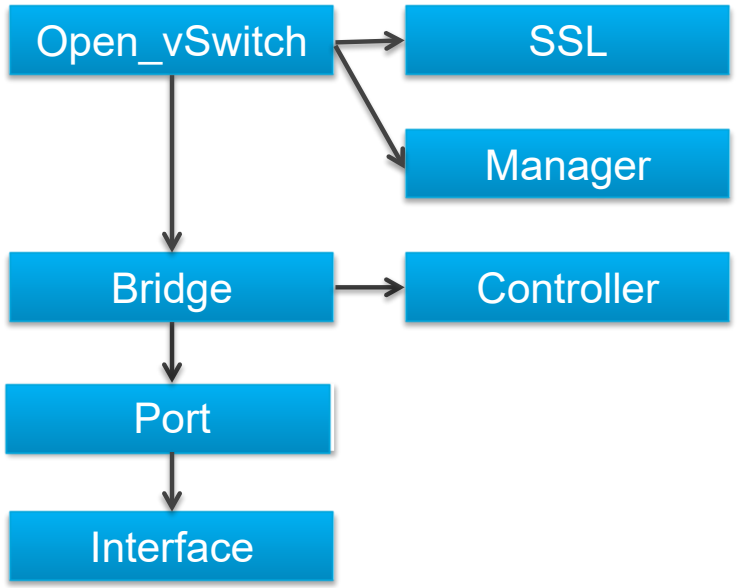
Value constraints
Weak references
Garbage collection

Log-based

Speaks OVSDB protocol to manager and ovs-vswitchd

Tools: ovs-vsctl, ovsdb-tool, ovsdb-client, ovs-appctl

Core Tables



“Open_vSwitch” is the root table and there is always only a single row. The tables here are the ones most commonly used; a full entity- relationship diagram is available in the ovs-vswitchd.conf.db man page.

Debugging the Database

ovs-vsctl: Configures ovs-vswitchd, but really a high-level interface for database

ovs-vsctl add-br <bridge>

ovs-vsctl list-br ovs-vsctl add-port <bridge> <port>

ovs-vsctl list-ports <bridge>

ovs-vsctl get-manager <bridge>

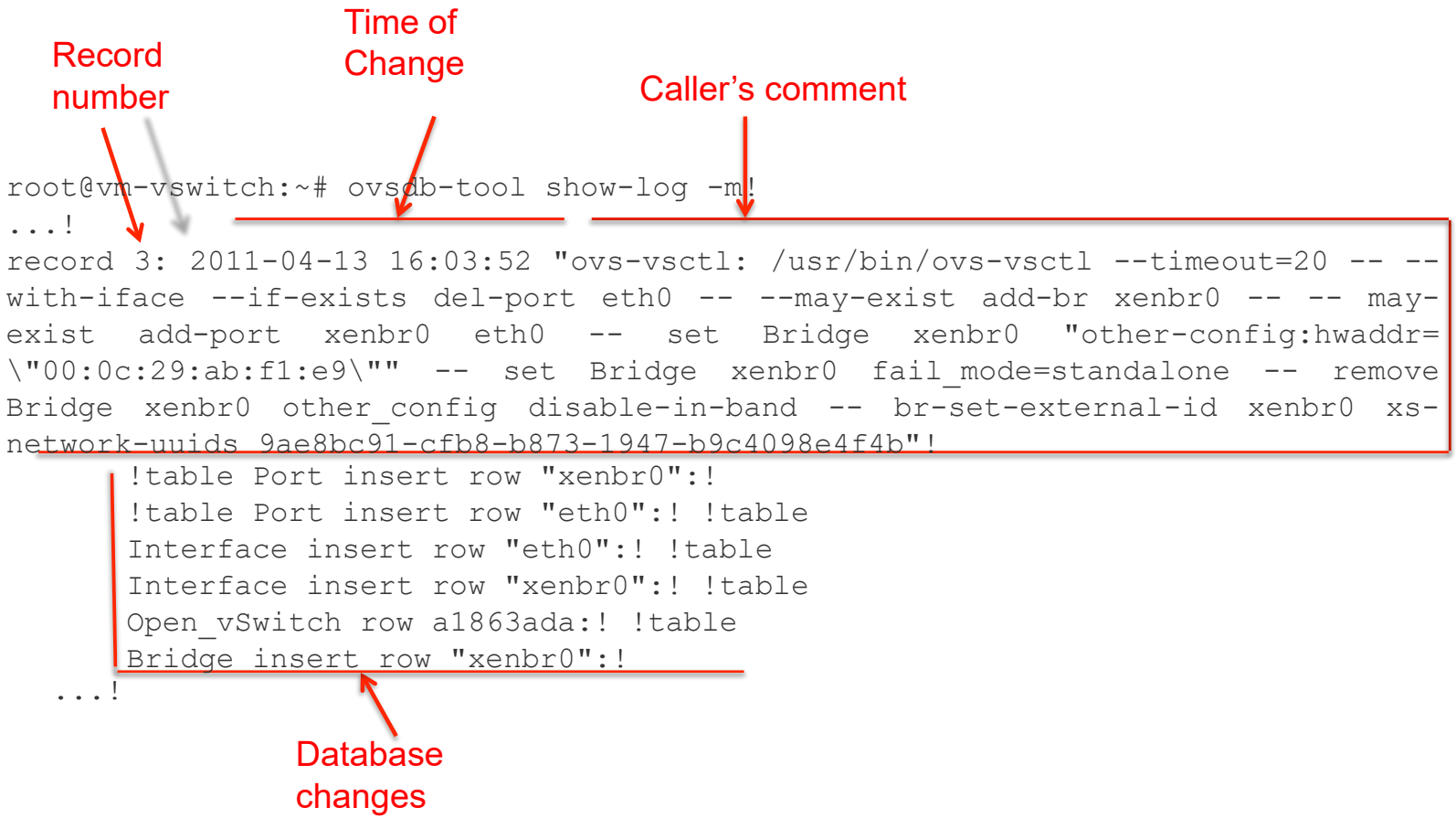
ovs-vsctl get-controller <bridge>

ovs-vsctl list <table>

ovsdb-tool: Command-line tool for managing database file

- ovsdb-tool show-log[-mmm]<file>

ovsdb-tool show-log



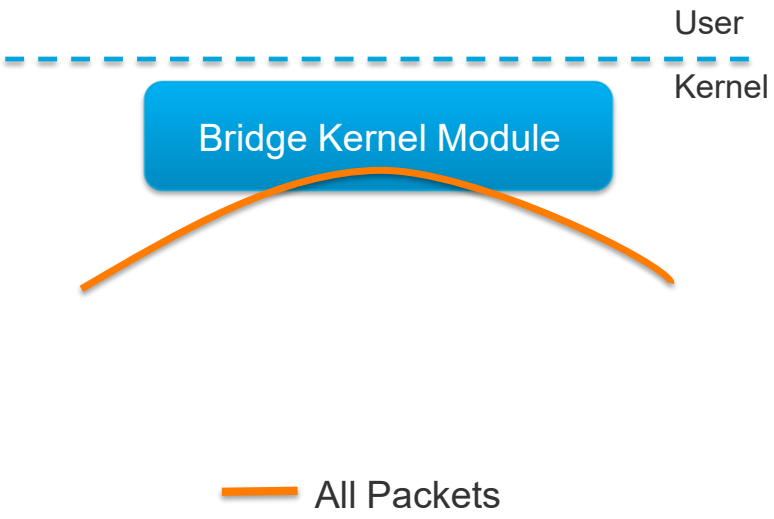
FORWARDING PATH

Linux Bridge Design

Simple forwarding

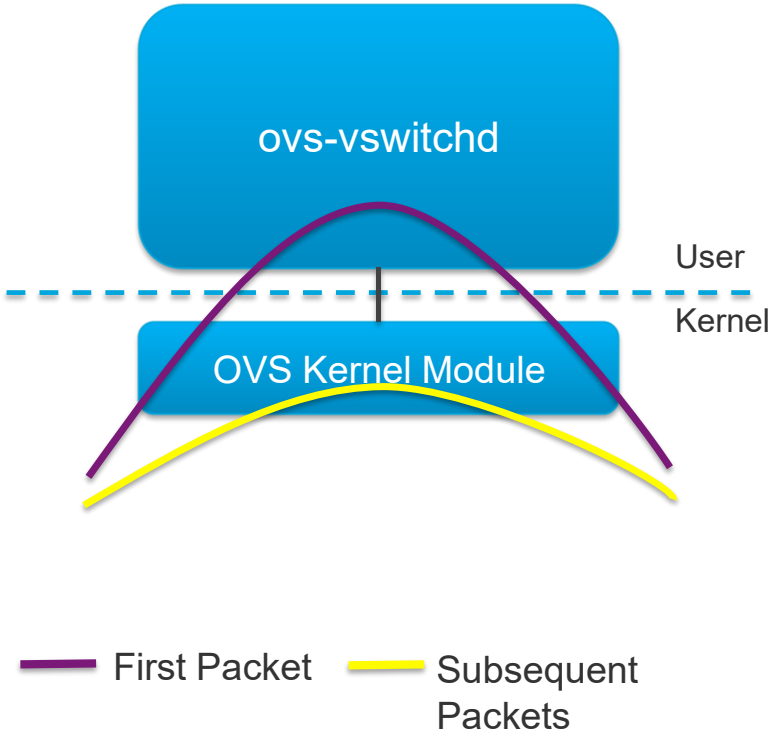
Matches destination MAC address and forwards

Packet never leaves kernel



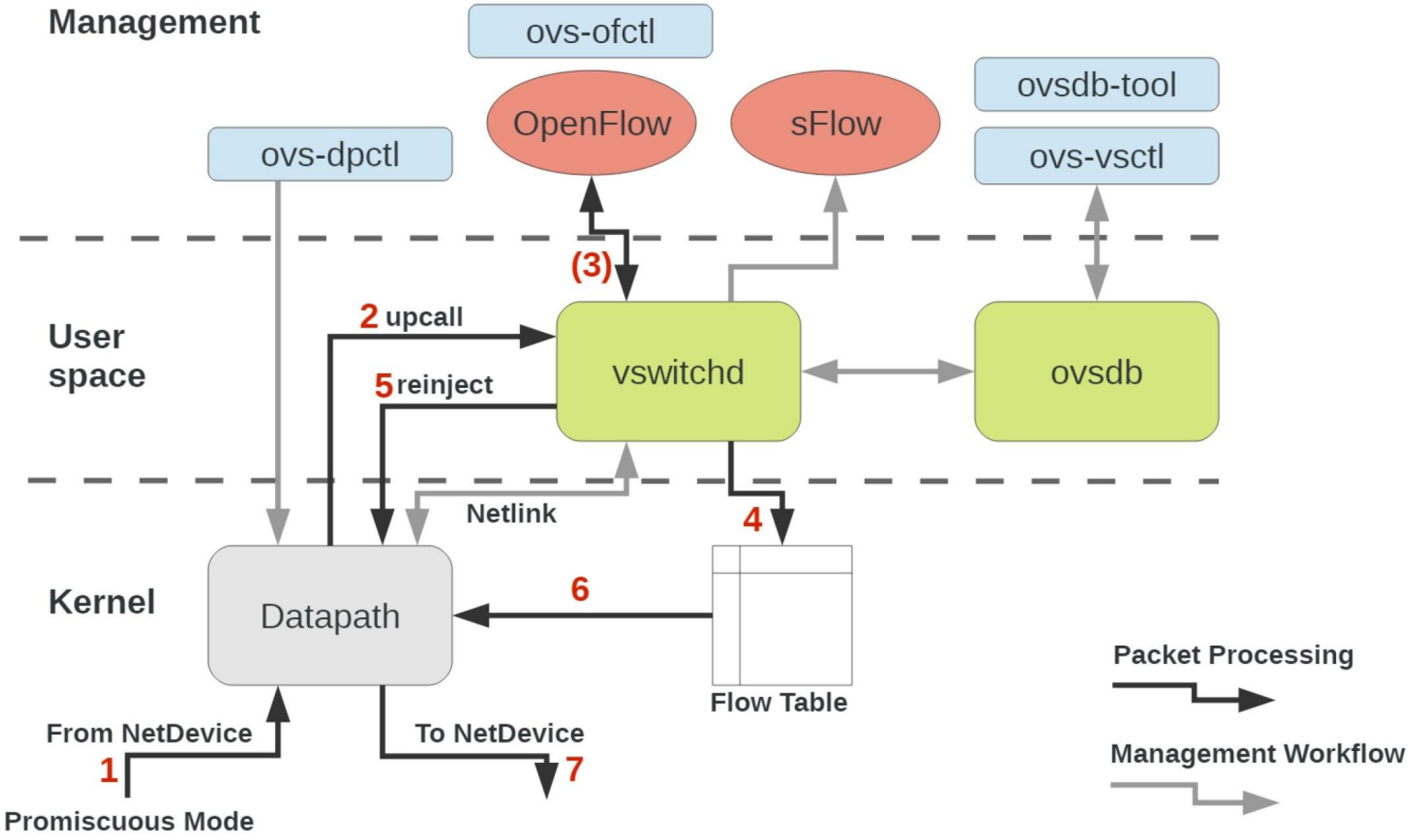
Open vSwitch Design

- **Decision about how to process packet made in userspace**
- **First packet of new flow goes to ovs-vswitchd, following packets hit cached entry in kernel**



Flow entry	match field	counter	Action (Instruction)	priority	Timeout
1					
n	…	…	…		

Inside OVS


Management
ovs-ofctl
ovsdb-tool
OpenFlow
sFlow
ovs-dpctl
ovs-vsctl
(3)
User space
2 upcall
vswitchd
ovsdb
5 reinject
Netlink
4
Kernel
Datapath
6
Flow Table
From NetDevice
1
Promiscuous Mode
To NetDevice
7
Packet Processing
Management Workflow

ovs-vswitchd

Core component in the system:

Communicates with outsideworld using OpenFlow

Communicat with ovsdb-server using OVSDB protocol

Communicates with kernel module over netlink

Communicates with the system through netdev abstract interface

- **Supports multiple independent datapaths (bridges)**
- **Packet classifier supports efficient flow lookup with wildcards and “explodes” these (possibly) wildcard rules for fast processing by the datapath**
- **Implements mirroring, bonding, and VLANs through modifications of the same flow table exposed through OpenFlow**
- **Checks datapath flow counters to handle flow expiration and stats requests**

Tools: ovs-ofctl, ovs-appctl

OVS Kernel Module

- **Kernel module that handles switching and tunneling**
- **Fast cache of non-overlapping flows**
- **Designed to be fast and simple**
  - Packet comes in, if found, associated actions executed and counters updated. Otherwise, sent to userspace
  - Does no flow expiration
  - Knows nothing of OpenFlow
- **Implements tunnels**

Tools: ovs-dpctl

Userspace Processing

- Packet received from kernel
- Given to the classifier to look for matching flows accumulates
- If “normal” action included, accumulates actions from “normal” processing, such as L2 forwarding and bonding
- Actions accumulated from configured modules, such as mirroring Prior, an exact match flow is generated with the accumulated actions and pushed down to the kernel module (along with the packet)

Kernel Processing

- Packet arrives and header fields extracted Header fields are hashed and used as an index into a set of large hash tables
- If entry found, actions applied to packet and counters are updated
- If entry is not found, packet sent to userspace and miss counter incremented

Megaflows

- **Version 1.11 added support for wildcarding in the datapath**
- **ovs-vswitchd dynamically determines how much wildcarding can done**
  - Flow table
  - Actions from matching flow
  - General switch configuration (e.g. bonding)
- **With megaflows, “normal” performance close to Linux bridge**

Tunnels

- Tunnels in OVS are just virtual ports with own OpenFlow port number
- Keys set statically at creation time or dynamically through OpenFlow action
- Types:
  - GRE
  - VxLAN
  - LISP
- Visible in Kernal datapath:
  - ovs-dpctl show

UTILITIES

ovs-ofctl speaks to OpenFlow module

ovs-ofctl show<bridge>

ovs-ofctl dump-flows <bridge>

ovs-ofctl add-flow <bridge> <flow>

ovs-ofctl del-flows <bridge> [flow]

ovs-ofctl snoop <bridge>

OpenFlow plus extensions

Resubmit Action: Simulate multiple tables in a single table

NXM: Extensible match

Registers: Eight 32-bit metadata registers

Fine-grained control over multiple controllers

See “hidden” flows (in-band, fail-open, etc):

ovs-appctlbridge/dump-flows<bridge>

ovs-ofctl show

```
root@vm-vswitch:~# ovs-ofctl show br0! OFPT_FEATURES_REPLY (xid=0x2):
dpid:0000505400000005! n_tables:254, n_buffers:256! capabilities: FLOW_STATS
TABLE_STATS PORT_STATS QUEUE_STATS ARP_MATCH_IP!
actions: OUTPUT SET_VLAN_VID SET_VLAN_PCP STRIP_VLAN SET_DL_SRC SET_DL_DST SET_NW_SRC
SET_NW_DST SET_NW_TOS SET_TP_SRC SET_TP_DST ENQUEUE!
 1(eth0): addr:50:54:00:00:00:05!
     config:     0!
     state:      0!
     current:    1GB-FD COPPER AUTO_NEG!
      advertised: 10MB-HD 10MB-FD 100MB-HD 100MB-FD 1GB-FD COPPER AUTO_NEG!
     supported:   10MB-HD 10MB-FD 100MB-HD 100MB-FD 1GB-FD COPPER AUTO_NEG!
     speed: 1000 Mbps now, 1000 Mbps max!
 2(eth1): addr:50:54:00:00:00:06!
     config:     0!
     state:      0!
     current:    1GB-FD COPPER AUTO_NEG!
      dvertised: 10MB-HD 10MB-FD 100MB-HD 100MB-FD 1GB-FD COPPER AUTO_NEG!
     supported:   10MB-HD 10MB-FD 100MB-HD 100MB-FD 1GB-FD COPPER AUTO_NEG!
     speed: 1000 Mbps now, 1000 Mbps max!
 LOCAL(br0): addr:50:54:00:00:00:05!
     config:    0! 0!
     state:
     speed: 0 Mbps now, 0 Mbps max!
OFPT_GET_CONFIG_REPLY (xid=0x4): frags=normal miss_send_len=0!
```

OpenFlow port number

Interface name

ovs-ofctl dump-flows

The default flow table includes a single entry that does “normal” processing:

```
root@vm-vswitch:~# ovs-ofctl dump-flows br0! NXST_FLOW reply
(xid=0x4):!  cookie=0x0, duration=4.05s, table=0, n_packets=8,
n_bytes=784, idle_age=0, priority=0 actions=NORMAL! !
```

Hidden Flows

There are flows that OVS uses for its own purpose that are higher priority than can be configured from outside

Types

In-bandcontrol (priority>=180000): Allow controltr affic to pass regardless of configured flows

Fail-open (priority = 0xf0f0f0): Allow all traffic to pass when a connection to the controller fails

They are hidden from controllers and commands like “ovs-ofctl dump-flows” due to being higher priority than OpenFlow allows (>65535)

Only visible with “ovs-appctl bridge/dump-flows <bridge>”

Kernel Datapath

ovs-dpctl speaks to kernel module See datapaths and their attached interfaces:

ovs-dpctlshow

See flows cached in datapath:

ovs-dpctldump-flows

ovs-dpctl show

ovs-dpctl dump-flows

Flows used to be exact-match:

in_port(2),eth(src=50:54:00:00:00:01,dst=50:54:00:00:00:03),eth_type(0x0800),ipv4(src=192.168.0.1,dst=192.168.0.2,proto=1,tos=0,ttl=64,frag=no),icmp(type=8,code=0), packets:3, bytes:294, used:0.185s, actions:3

in_port(3),eth(src=50:54:00:00:00:03,dst=50:54:00:00:00:01),eth_type(0x0800),ipv4(src=192.168.0.2,dst=192.168.0.1,proto=1,tos=0,ttl=64,frag=no),icmp(type=0,code=0),packets:3,bytes:294,used:0.205s,actions:2

Starting in OVS 1.11, may contain wildcards:

in_port(3),eth(src=50:54:00:00:00:03,dst=50:54:00:00:00:01),eth_type(0x0800),ipv4(src=192.168.0.2/0.0.0.0,dst=192.168.0.1/0.0.0.0,proto=1/0,tos=0/0,ttl=64/0,frag=no/0x2),icmp(type=0/0,code=0/0), packets:95, bytes:9310, used:0.425s, actions:2

in_port(2),eth(src=50:54:00:00:00:01,dst=50:54:00:00:00:03),eth_type(0x0800),ipv4(src=192.168.0.1/0.0.0.0,dst=192.168.0.2/0.0.0.0,proto=1/0,tos=0/0,ttl=64/0,frag=no/0x2),icmp(type=8/0,code=0/0), packets:95, bytes:9310, used:0.525s, actions:3

ovs-appctl

Utility to invoke runtime control and query facilities in most OVS daemons

The “-t <target>” option specifies the daemon name (default is ovs-vswitchd)

All daemons support the following commands:

help– Lists the commands supported by the target

version – Displays the version and compilation date of the target

vlog/list – List the known logging modules and their current levels

vlog/set [spec] – Sets logging levels

Many interesting features supported, which are defined in thetargets’ man pages

Flow Debugging

Flow tables can become incredibly complex, but OVS has tools to make it easier to debug

Here is a set of rules to (poorly) implement a firewall (with an unnecessary resubmit) to block all TCP traffic except port 80:

```
# Move TCP traffic arriving on port 1 to next stage of “pipeline”!
 priority=100,tcp,in_port=1 actions=resubmit:4000! !


# Allow port TCP port 80 traffic (and implicitly drop all others)!
priority=100,tcp,in_port=4000,tp_dst=80 actions=NORMAL! !

# Allow all non-TCP traffic arriving on port 1!
priority=90,in_port=1 actions=NORMAL! !


# Allow all traffic arriving on port 2!
priority=100,in_port=2 actions=NORMAL! ! !
```

Tracing Flow (ICMP Allowed)

```
root@vm-vswitch:~# ovs-appctl ofproto/trace
"skb_priority(0),in_port(2),skb_mark(0),eth(src=50:54:00:00:00:01,dst=50:54
:00:00:00:03),eth_type(0x0800),ipv4(src=192.168.0.1,dst=192.168.0.2,proto=1
,tos=0,ttl=64,frag=no),icmp(type=8,code=0)"
Bridge: br0
Flow:
icmp,metadata=0,in_port=1,vlan_tci=0x0000,dl_src=50:54:00:00:00:01,dl_dst=5
0:54:00:00:00:03,nw_src=192.168.0.1,nw_dst=192.168.0.2,nw_tos=0,nw_ecn=0,nw
_ttl=64,icmp_type=8,icmp_code=0
Rule: table=0 cookie=0 priority=90,in_port=1
OpenFlow actions=NORMAL
forwarding to learned port

Final flow: unchanged
Relevant fields:
skb_priority=0,icmp,in_port=1,vlan_tci=0x0000/0x1fff,dl_src=50:54:00:00:00:
01,dl_dst=50:54:00:00:00:03,nw_frag=no,icmp_code=0
Datapath actions: 3
```

Applied OpenFlow rule

Datapath flow description

Datapath action

Tracing Flow (TCP allowed)

```
root@vm-vswitch:~# ovs-appctl ofproto/trace
"skb_priority(0),in_port(2),skb_mark(0),eth(src=50:54:00:00:00:01,dst=50:5
4:00:00:00:03),eth_type(0x0800),ipv4(src=192.168.0.1,dst=192.168.0.2,proto
=6,tos=0x10,ttl=64,frag=no),tcp(src=56176,dst=80),tcp_flags(0x002)"
Bridge: br0
Flow:
tcp,metadata=0,in_port=1,vlan_tci=0x0000,dl_src=50:54:00:00:00:01,dl_dst=5
0:54:00:00:00:03,nw_src=192.168.0.1,nw_dst=192.168.0.2,nw_tos=16,nw_ecn=0,
nw_ttl=64,tp_src=56176,tp_dst=80,tcp_flags=0x002
Rule: table=0 cookie=0 priority=100,tcp,in_port=1
OpenFlow actions=resubmit:4000

    Resubmitted flow: unchanged
    Resubmitted regs: reg0=0x0 reg1=0x0 reg2=0x0 reg3=0x0 reg4=0x0
reg5=0x0 reg6=0x0 reg7=0x0
    Resubmitted  odp: drop
    Rule: table=0 cookie=0 priority=100,tcp,in_port=4000,tp_dst=80
    OpenFlow actions=NORMAL
    forwarding to learned port

Final flow: unchanged
Relevant fields:
skb_priority=0,tcp,in_port=1,vlan_tci=0x0000/0x1fff,dl_src=50:54:00:00:00:
01,dl_dst=50:54:00:00:00:03,nw_frag=no,tp_dst=80
Datapath actions: 3
```

First applied OpenFlow rule

Second applied OpenFlow rule

Datapath flow description

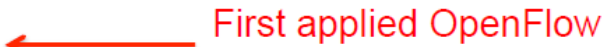
Datapath action

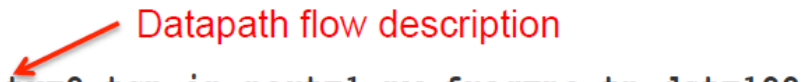
Tracing Flow (TCP denied)

```
root@vm-vswitch:~# ovs-appctl ofproto/trace
"skb_priority(0),in_port(2),skb_mark(0),eth(src=50:54:00:00:00:01,dst=50:54:00:
00:00:03),eth_type(0x0800),ipv4(src=192.168.0.1,dst=192.168.0.2,proto=6,tos=0x1
0,ttl=64,frag=no),tcp(src=56177,dst=100),tcp_flags(0x002)"
Bridge: br0
Flow:
tcp,metadata=0,in_port=1,vlan_tci=0x0000,dl_src=50:54:00:00:00:01,dl_dst=50:54:
00:00:00:03,nw_src=192.168.0.1,nw_dst=192.168.0.2,nw_tos=16,nw_ecn=0,nw_ttl=64,
tp_src=56177,tp_dst=100,tcp_flags=0x002
Rule: table=0 cookie=0 priority=100,tcp,in_port=1
OpenFlow actions=resubmit:4000

    Resubmitted flow: unchanged
    Resubmitted regs: reg0=0x0 reg1=0x0 reg2=0x0 reg3=0x0 reg4=0x0 reg5=0x0
reg6=0x0 reg7=0x0
    Resubmitted  odp: drop
    No match

Final flow: unchanged
Relevant fields: skb_priority=0,tcp,in_port=1,nw_frag=no,tp_dst=100
Datapath actions: drop
```





Logging

- ovs-appctl configures running OVS daemons
- Most common use is to modify logging levels
- By default configures ovs-vswitchd, but “-t” option changes target
- Default level for log files is “info”, only thing lower is “dbg”

```
root@vm-vswitch:~# ovs-appctl vlog/list
                  console    syslog    file
                  -------    ------    ------
bridge             EMER        ERR       INFO
vswitchd           EMER        ERR       INFO
...
root@vm-vswitch:~# ovs-appctl vlog/set ofproto:file:dbg
```

Log Files

- **Open vSwitch logs: /var/log/openvswitch/***
  - ovs-vswitchd.log
  - ovsdb-server.log
- **System: /var/log/messages**
- **Configuration Database: /etc/openvswitch/conf.d**

Install OVS

- Install openVswitch

  #sudo apt-get install openvswitch-switch

- Install the ovs-docker utility
  - #cd /usr/bin
  - wget https://raw.githubusercontent.com/openvswitch/ovs/master/utilities/ovs-docker
  - #chmod a+rwx ovs-docker

Config bridge

- Config bridge

  #ovs-vsctl add-br ovs-br1

  #ifconfig ovs-br1 173.16.1.1 netmask 255.255.255.0 up

  (if no ifconfig, install it)

  #apt install net-tools

- Show bridge

  #ovs-vsctl show

Add docker

- Add container

  #docker run -dt --name c3 alpine sleep 1d

  #docker run -dt --name c4 alpine sleep 1d

- Connect the container to the OVS

  #ovs-docker add-port ovs-br1 eth1 c3 --ipaddress=173.16.1.20/24

  #ovs-docker add-port ovs-br1 eth1 c4 --ipaddress=173.16.1.30/24

- Check container

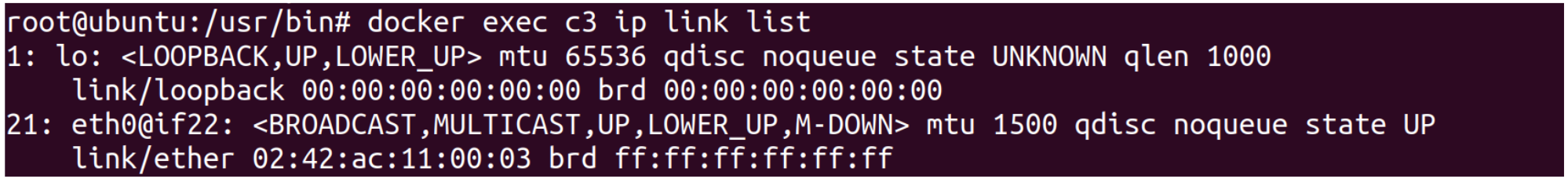
  #docker exec c3 ip link list

  #docker exec c4 ip link list

root@ubuntu:/usr/bin# docker exec c3 ip link list

1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

21: eth0@if22: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue state UP link/ether 02:42:ac:11:00:03 brd ff:ff:ff:ff:ff:ff



Set IP Address

```
root@ubuntu:/usr/bin# ovs-docker add-port ovs-br1 eth1 c3 --ipaddress=173.16.1.20/24
root@ubuntu:/usr/bin# docker exec c3 ip address list
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
21: eth0@if22: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue state UP
    link/ether 02:42:ac:11:00:03 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.3/16 brd 172.17.255.255 scope global eth0
       valid_lft forever preferred_lft forever
23: eth1@if24: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue state UP qlen 1000
    link/ether 8e:52:06:9a:7c:9d brd ff:ff:ff:ff:ff:ff
    inet 173.16.1.20/24 scope global eth1
       valid_lft forever preferred_lft forever
```

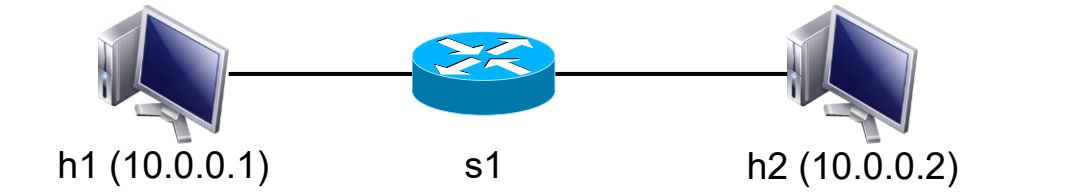
```
root@ubuntu:/usr/bin# docker exec c3 ip link list
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
21: eth0@if22: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500
qdisc noqueue state UP
    link/ether 02:42:ac:11:00:03 brd ff:ff:ff:ff:ff:ff
23: eth1@if24: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500
qdisc noqueue state UP qlen 1000
    link/ether 8e:52:06:9a:7c:9d brd ff:ff:ff:ff:ff:ff
```

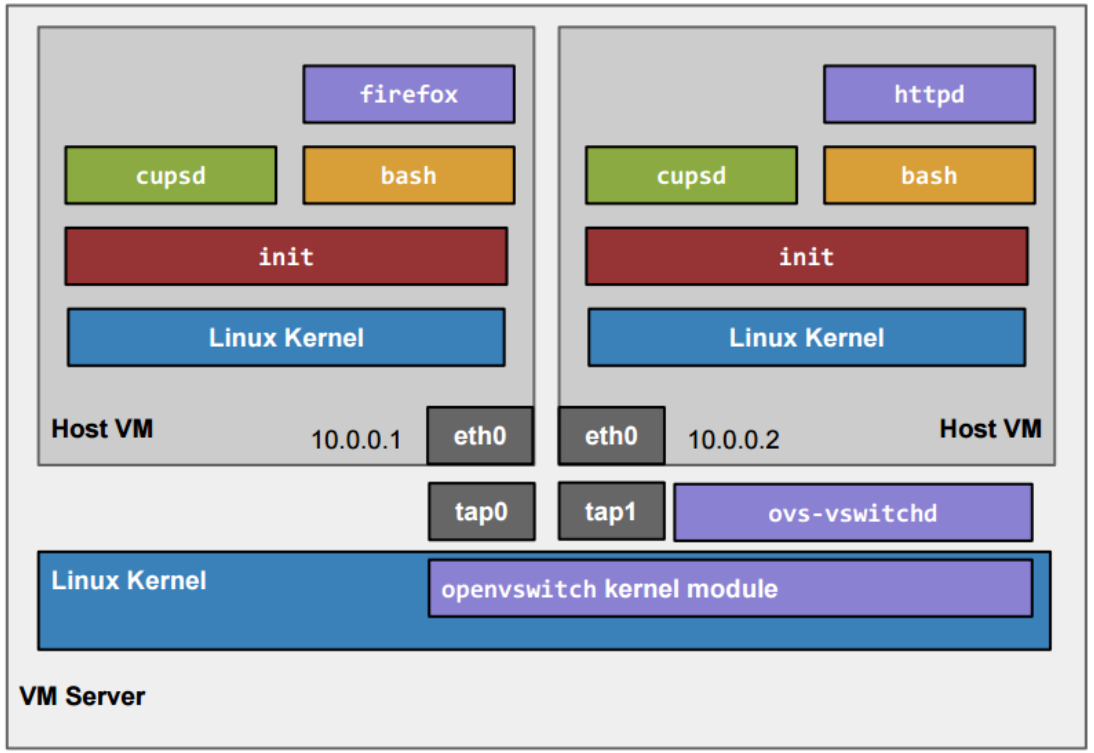
```
root@ubuntu:/usr/bin# docker exec c4 ping 173.16.1.20
PING 173.16.1.20 (173.16.1.20): 56 data bytes
64 bytes from 173.16.1.20: seq=0 ttl=64 time=0.551 ms
64 bytes from 173.16.1.20: seq=1 ttl=64 time=0.102 ms
64 bytes from 173.16.1.20: seq=2 ttl=64 time=0.059 ms
```

```
root@ubuntu:/usr/bin# ovs-vsctl show
cd18c9a3-49a1-4608-999a-5fd3da3c9e1b
    Bridge "ovs-br1"
        Port "fcf399d5e1cb4_l"
            Interface "fcf399d5e1cb4_l"
        Port "ovs-br1"
            Interface "ovs-br1"
                type: internal
        Port "76fec545fdf74_l"
            Interface "76fec545fdf74_l"
        Port "960d648383a84_l"
            Interface "960d648383a84_l"
        Port "d5607674b29d4_l"
            Interface "d5607674b29d4_l"
        Port "cc09edcb865d4_l"
            Interface "cc09edcb865d4_l"
    ovs_version: "2.9.5"
root@ubuntu:/usr/bin#
```

Network Using Namespace

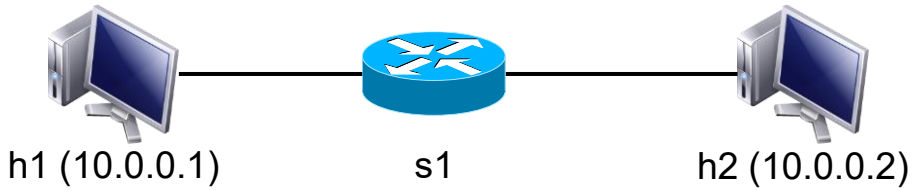
- Very Simple Network using Full System Virtualization





Network using Namespace

- Problems
  - Too much work even for creating such a simple network topology
  - Not programmable

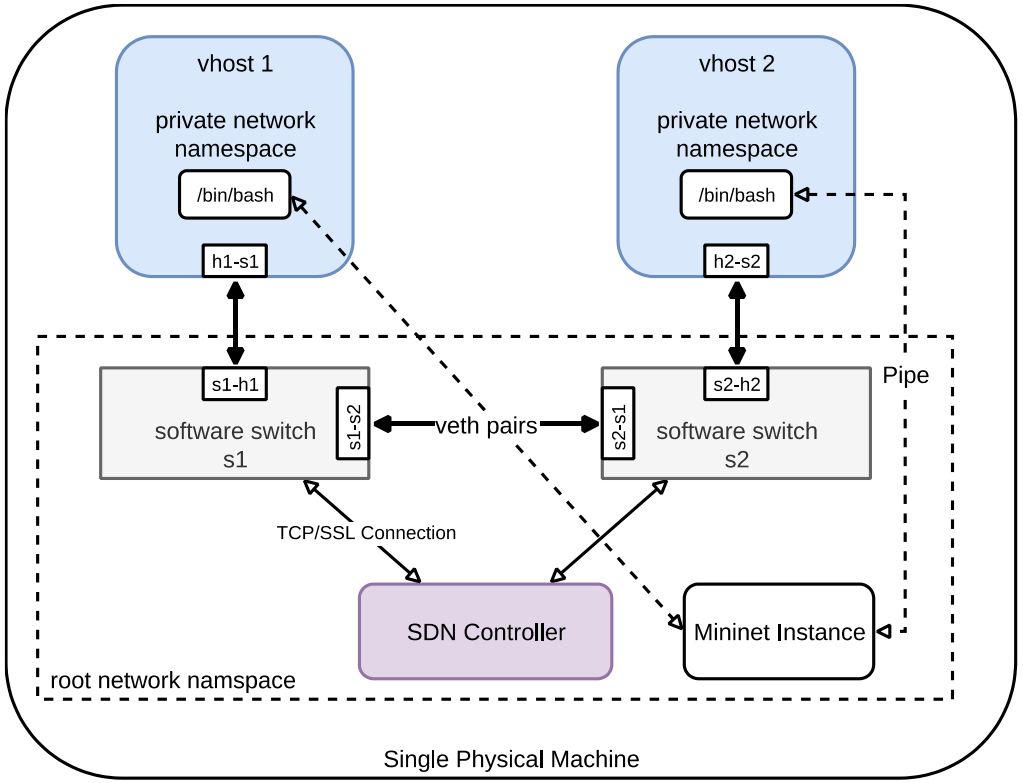



```
sudo bash
# Create host namespaces
ip netns add h1
ip netns add h2
# Create switch
ovs-vsctl add-br s1
# Create links
ip link add h1-eth0 type veth peer name s1-eth1
ip link add h2-eth0 type veth peer name s1-eth2
ip link show
# Move host ports into namespaces
ip link set h1-eth0 netns h1
ip link set h2-eth0 netns h2
ip netns exec h1 ip link show
ip netns exec h2 ip link show
```

```
# Connect switch ports to OVS
ovs-vsctl add-port s1 s1-eth1
ovs-vsctl add-port s1 s1-eth2
ovs-vsctl show
# Set up OpenFlow controller
ovs-vsctl set-controller s1 tcp:127.0.0.1
ovs-controller ptcp: &
ovs-vsctl show
# Configure network
ip netns exec h1 ifconfig h1-eth0 10.1
ip netns exec h1 ifconfig lo up
ip netns exec h2 ifconfig h2-eth0 10.2
ip netns exec h1 ifconfig lo up
ifconfig s1-eth1 up
ifconfig s1-eth2 up
# Test network
ip netns exec h1 ping -c1 10.2
```

Mininet

- A simple command-line tool / API which can ease the work
- The solution should allow us to easily create topologies for varying size, up to hundreds and thousand of nodes



Lab

- ทอสอบหัวข้อ 7.6 การทดสอบการทำงานของโอเพนวีสวิตซ์เบื้องต้น
  - Docker 4 ตัว พร้อม VLAN
- ทอสอบการใช้ OVS บน Mininet (บทที่ 8)
- mn --topo tree,2 --mac --arp --switch ovsk
  - ทดสอบการ Ping สามารถ ping ได้ทั้งหมดหรือไม่
- mn --topo tree, 2 --mac --arp --switch ovsk --controller
  - ทดสอบการ Ping สามารถ ping ได้ทั้งหมดหรือไม่ แก้ไข อย่างไร