**Open vSwitch <http://openvswitch.org>**

**Frequently Asked Questions**

**==========================**

General

-------

Q: What is Open vSwitch?

A: Open vSwitch is a production quality open source software switch

designed to be used as a vswitch in virtualized server

environments. A vswitch forwards traffic between different VMs on

the same physical host and also forwards traffic between VMs and

the physical network. Open vSwitch supports standard management

interfaces (e.g. sFlow, NetFlow, IPFIX, RSPAN, CLI), and is open to

programmatic extension and control using OpenFlow and the OVSDB

management protocol.

Open vSwitch as designed to be compatible with modern switching

chipsets. This means that it can be ported to existing high-fanout

switches allowing the same flexible control of the physical

infrastructure as the virtual infrastructure. It also means that

Open vSwitch will be able to take advantage of on-NIC switching

chipsets as their functionality matures.

Q: What virtualization platforms can use Open vSwitch?

A: Open vSwitch can currently run on any Linux-based virtualization

platform (kernel 2.6.32 and newer), including: KVM, VirtualBox, Xen,

Xen Cloud Platform, XenServer. As of Linux 3.3 it is part of the

mainline kernel. The bulk of the code is written in platform-

independent C and is easily ported to other environments. We welcome

inquires about integrating Open vSwitch with other virtualization

platforms.

Q: How can I try Open vSwitch?

A: The Open vSwitch source code can be built on a Linux system. You can

build and experiment with Open vSwitch on any Linux machine.

Packages for various Linux distributions are available on many

platforms, including: Debian, Ubuntu, Fedora.

You may also download and run a virtualization platform that already

has Open vSwitch integrated. For example, download a recent ISO for

XenServer or Xen Cloud Platform. Be aware that the version

integrated with a particular platform may not be the most recent Open

vSwitch release.

Q: Does Open vSwitch only work on Linux?

A: No, Open vSwitch has been ported to a number of different operating

systems and hardware platforms. Most of the development work occurs

on Linux, but the code should be portable to any POSIX system. We've

seen Open vSwitch ported to a number of different platforms,

including FreeBSD, Windows, and even non-POSIX embedded systems.

By definition, the Open vSwitch Linux kernel module only works on

Linux and will provide the highest performance. However, a userspace

datapath is available that should be very portable.

Q: What's involved with porting Open vSwitch to a new platform or

switching ASIC?

A: The PORTING document describes how one would go about porting Open

vSwitch to a new operating system or hardware platform.

Q: Why would I use Open vSwitch instead of the Linux bridge?

A: Open vSwitch is specially designed to make it easier to manage VM

network configuration and monitor state spread across many physical

hosts in dynamic virtualized environments. Please see WHY-OVS for a

more detailed description of how Open vSwitch relates to the Linux

Bridge.

Q: How is Open vSwitch related to distributed virtual switches like the

VMware vNetwork distributed switch or the Cisco Nexus 1000V?

A: Distributed vswitch applications (e.g., VMware vNetwork distributed

switch, Cisco Nexus 1000V) provide a centralized way to configure and

monitor the network state of VMs that are spread across many physical

hosts. Open vSwitch is not a distributed vswitch itself, rather it

runs on each physical host and supports remote management in a way

that makes it easier for developers of virtualization/cloud

management platforms to offer distributed vswitch capabilities.

To aid in distribution, Open vSwitch provides two open protocols that

are specially designed for remote management in virtualized network

environments: OpenFlow, which exposes flow-based forwarding state,

and the OVSDB management protocol, which exposes switch port state.

In addition to the switch implementation itself, Open vSwitch

includes tools (ovs-ofctl, ovs-vsctl) that developers can script and

extend to provide distributed vswitch capabilities that are closely

integrated with their virtualization management platform.

Q: Why doesn't Open vSwitch support distribution?

A: Open vSwitch is intended to be a useful component for building

flexible network infrastructure. There are many different approaches

to distribution which balance trade-offs between simplicity,

scalability, hardware compatibility, convergence times, logical

forwarding model, etc. The goal of Open vSwitch is to be able to

support all as a primitive building block rather than choose a

particular point in the distributed design space.

Q: How can I contribute to the Open vSwitch Community?

A: You can start by joining the mailing lists and helping to answer

questions. You can also suggest improvements to documentation. If

you have a feature or bug you would like to work on, send a mail to

one of the mailing lists:

http://openvswitch.org/mlists/

Releases

--------

Q: What does it mean for an Open vSwitch release to be LTS (long-term

support)?

A: All official releases have been through a comprehensive testing

process and are suitable for production use. Planned releases will

occur several times a year. If a significant bug is identified in an

LTS release, we will provide an updated release that includes the

fix. Releases that are not LTS may not be fixed and may just be

supplanted by the next major release. The current LTS release is

1.9.x.

Q: What Linux kernel versions does each Open vSwitch release work with?

A: The following table lists the Linux kernel versions against which the

given versions of the Open vSwitch kernel module will successfully

build. The Linux kernel versions are upstream kernel versions, so

Linux kernels modified from the upstream sources may not build in

some cases even if they are based on a supported version. This is

most notably true of Red Hat Enterprise Linux (RHEL) kernels, which

are extensively modified from upstream.

Open vSwitch Linux kernel

------------ -------------

1.4.x 2.6.18 to 3.2

1.5.x 2.6.18 to 3.2

1.6.x 2.6.18 to 3.2

1.7.x 2.6.18 to 3.3

1.8.x 2.6.18 to 3.4

1.9.x 2.6.18 to 3.8

1.10.x 2.6.18 to 3.8

1.11.x 2.6.18 to 3.8

2.0.x 2.6.32 to 3.10

2.1.x 2.6.32 to 3.11

Open vSwitch userspace should also work with the Linux kernel module

built into Linux 3.3 and later.

Open vSwitch userspace is not sensitive to the Linux kernel version.

It should build against almost any kernel, certainly against 2.6.32

and later.

Q: What Linux kernel versions does IPFIX flow monitoring work with?

A: IPFIX flow monitoring requires the Linux kernel module from Open

vSwitch version 1.10.90 or later.

Q: Should userspace or kernel be upgraded first to minimize downtime?

In general, the Open vSwitch userspace should be used with the

kernel version included in the same release or with the version

from upstream Linux. However, when upgrading between two releases

of Open vSwitch it is best to migrate userspace first to reduce

the possibility of incompatibilities.

Q: What features are not available in the Open vSwitch kernel datapath

that ships as part of the upstream Linux kernel?

A: The kernel module in upstream Linux 3.3 and later does not include

tunnel virtual ports, that is, interfaces with type "gre",

"ipsec\_gre", "gre64", "ipsec\_gre64", "vxlan", or "lisp". It is

possible to create tunnels in Linux and attach them to Open vSwitch

as system devices. However, they cannot be dynamically created

through the OVSDB protocol or set the tunnel ids as a flow action.

Work is in progress in adding tunnel virtual ports to the upstream

Linux version of the Open vSwitch kernel module. For now, if you

need these features, use the kernel module from the Open vSwitch

distribution instead of the upstream Linux kernel module.

The upstream kernel module does not include patch ports, but this

only matters for Open vSwitch 1.9 and earlier, because Open vSwitch

1.10 and later implement patch ports without using this kernel

feature.

Q: What features are not available when using the userspace datapath?

A: Tunnel virtual ports are not supported, as described in the

previous answer. It is also not possible to use queue-related

actions. On Linux kernels before 2.6.39, maximum-sized VLAN packets

may not be transmitted.

Q: What happened to the bridge compatibility feature?

A: Bridge compatibility was a feature of Open vSwitch 1.9 and earlier.

When it was enabled, Open vSwitch imitated the interface of the

Linux kernel "bridge" module. This allowed users to drop Open

vSwitch into environments designed to use the Linux kernel bridge

module without adapting the environment to use Open vSwitch.

Open vSwitch 1.10 and later do not support bridge compatibility.

The feature was dropped because version 1.10 adopted a new internal

architecture that made bridge compatibility difficult to maintain.

Now that many environments use OVS directly, it would be rarely

useful in any case.

To use bridge compatibility, install OVS 1.9 or earlier, including

the accompanying kernel modules (both the main and bridge

compatibility modules), following the instructions that come with

the release. Be sure to start the ovs-brcompatd daemon.

Terminology

-----------

Q: I thought Open vSwitch was a virtual Ethernet switch, but the

documentation keeps talking about bridges. What's a bridge?

A: In networking, the terms "bridge" and "switch" are synonyms. Open

vSwitch implements an Ethernet switch, which means that it is also

an Ethernet bridge.

Q: What's a VLAN?

A: See the "VLAN" section below.

Basic Configuration

-------------------

Q: How do I configure a port as an access port?

A: Add "tag=VLAN" to your "ovs-vsctl add-port" command. For example,

the following commands configure br0 with eth0 as a trunk port (the

default) and tap0 as an access port for VLAN 9:

ovs-vsctl add-br br0

ovs-vsctl add-port br0 eth0

ovs-vsctl add-port br0 tap0 tag=9

If you want to configure an already added port as an access port,

use "ovs-vsctl set", e.g.:

ovs-vsctl set port tap0 tag=9

Q: How do I configure a port as a SPAN port, that is, enable mirroring

of all traffic to that port?

A: The following commands configure br0 with eth0 and tap0 as trunk

ports. All traffic coming in or going out on eth0 or tap0 is also

mirrored to tap1; any traffic arriving on tap1 is dropped:

ovs-vsctl add-br br0

ovs-vsctl add-port br0 eth0

ovs-vsctl add-port br0 tap0

ovs-vsctl add-port br0 tap1 \

-- --id=@p get port tap1 \

-- --id=@m create mirror name=m0 select-all=true output-port=@p \

-- set bridge br0 mirrors=@m

To later disable mirroring, run:

ovs-vsctl clear bridge br0 mirrors

Q: Does Open vSwitch support configuring a port in promiscuous mode?

A: Yes. How you configure it depends on what you mean by "promiscuous

mode":

- Conventionally, "promiscuous mode" is a feature of a network

interface card. Ordinarily, a NIC passes to the CPU only the

packets actually destined to its host machine. It discards

the rest to avoid wasting memory and CPU cycles. When

promiscuous mode is enabled, however, it passes every packet

to the CPU. On an old-style shared-media or hub-based

network, this allows the host to spy on all packets on the

network. But in the switched networks that are almost

everywhere these days, promiscuous mode doesn't have much

effect, because few packets not destined to a host are

delivered to the host's NIC.

This form of promiscuous mode is configured in the guest OS of

the VMs on your bridge, e.g. with "ifconfig".

- The VMware vSwitch uses a different definition of "promiscuous

mode". When you configure promiscuous mode on a VMware vNIC,

the vSwitch sends a copy of every packet received by the

vSwitch to that vNIC. That has a much bigger effect than just

enabling promiscuous mode in a guest OS. Rather than getting

a few stray packets for which the switch does not yet know the

correct destination, the vNIC gets every packet. The effect

is similar to replacing the vSwitch by a virtual hub.

This "promiscuous mode" is what switches normally call "port

mirroring" or "SPAN". For information on how to configure

SPAN, see "How do I configure a port as a SPAN port, that is,

enable mirroring of all traffic to that port?"

Q: How do I configure a VLAN as an RSPAN VLAN, that is, enable

mirroring of all traffic to that VLAN?

A: The following commands configure br0 with eth0 as a trunk port and

tap0 as an access port for VLAN 10. All traffic coming in or going

out on tap0, as well as traffic coming in or going out on eth0 in

VLAN 10, is also mirrored to VLAN 15 on eth0. The original tag for

VLAN 10, in cases where one is present, is dropped as part of

mirroring:

ovs-vsctl add-br br0

ovs-vsctl add-port br0 eth0

ovs-vsctl add-port br0 tap0 tag=10

ovs-vsctl \

-- --id=@m create mirror name=m0 select-all=true select-vlan=10 \

output-vlan=15 \

-- set bridge br0 mirrors=@m

To later disable mirroring, run:

ovs-vsctl clear bridge br0 mirrors

Mirroring to a VLAN can disrupt a network that contains unmanaged

switches. See ovs-vswitchd.conf.db(5) for details. Mirroring to a

GRE tunnel has fewer caveats than mirroring to a VLAN and should

generally be preferred.

Q: Can I mirror more than one input VLAN to an RSPAN VLAN?

A: Yes, but mirroring to a VLAN strips the original VLAN tag in favor

of the specified output-vlan. This loss of information may make

the mirrored traffic too hard to interpret.

To mirror multiple VLANs, use the commands above, but specify a

comma-separated list of VLANs as the value for select-vlan. To

mirror every VLAN, use the commands above, but omit select-vlan and

its value entirely.

When a packet arrives on a VLAN that is used as a mirror output

VLAN, the mirror is disregarded. Instead, in standalone mode, OVS

floods the packet across all the ports for which the mirror output

VLAN is configured. (If an OpenFlow controller is in use, then it

can override this behavior through the flow table.) If OVS is used

as an intermediate switch, rather than an edge switch, this ensures

that the RSPAN traffic is distributed through the network.

Mirroring to a VLAN can disrupt a network that contains unmanaged

switches. See ovs-vswitchd.conf.db(5) for details. Mirroring to a

GRE tunnel has fewer caveats than mirroring to a VLAN and should

generally be preferred.

Q: How do I configure mirroring of all traffic to a GRE tunnel?

A: The following commands configure br0 with eth0 and tap0 as trunk

ports. All traffic coming in or going out on eth0 or tap0 is also

mirrored to gre0, a GRE tunnel to the remote host 192.168.1.10; any

traffic arriving on gre0 is dropped:

ovs-vsctl add-br br0

ovs-vsctl add-port br0 eth0

ovs-vsctl add-port br0 tap0

ovs-vsctl add-port br0 gre0 \

-- set interface gre0 type=gre options:remote\_ip=192.168.1.10 \

-- --id=@p get port gre0 \

-- --id=@m create mirror name=m0 select-all=true output-port=@p \

-- set bridge br0 mirrors=@m

To later disable mirroring and destroy the GRE tunnel:

ovs-vsctl clear bridge br0 mirrors

ovs-vcstl del-port br0 gre0

Q: Does Open vSwitch support ERSPAN?

A: No. ERSPAN is an undocumented proprietary protocol. As an

alternative, Open vSwitch supports mirroring to a GRE tunnel (see

above).

Q: How do I connect two bridges?

A: First, why do you want to do this? Two connected bridges are not

much different from a single bridge, so you might as well just have

a single bridge with all your ports on it.

If you still want to connect two bridges, you can use a pair of

patch ports. The following example creates bridges br0 and br1,

adds eth0 and tap0 to br0, adds tap1 to br1, and then connects br0

and br1 with a pair of patch ports.

ovs-vsctl add-br br0

ovs-vsctl add-port br0 eth0

ovs-vsctl add-port br0 tap0

ovs-vsctl add-br br1

ovs-vsctl add-port br1 tap1

ovs-vsctl \

-- add-port br0 patch0 \

-- set interface patch0 type=patch options:peer=patch1 \

-- add-port br1 patch1 \

-- set interface patch1 type=patch options:peer=patch0

Bridges connected with patch ports are much like a single bridge.

For instance, if the example above also added eth1 to br1, and both

eth0 and eth1 happened to be connected to the same next-hop switch,

then you could loop your network just as you would if you added

eth0 and eth1 to the same bridge (see the "Configuration Problems"

section below for more information).

If you are using Open vSwitch 1.9 or an earlier version, then you

need to be using the kernel module bundled with Open vSwitch rather

than the one that is integrated into Linux 3.3 and later, because

Open vSwitch 1.9 and earlier versions need kernel support for patch

ports. This also means that in Open vSwitch 1.9 and earlier, patch

ports will not work with the userspace datapath, only with the

kernel module.

Implementation Details

----------------------

Q: I hear OVS has a couple of kinds of flows. Can you tell me about them?

A: Open vSwitch uses different kinds of flows for different purposes:

- OpenFlow flows are the most important kind of flow. OpenFlow

controllers use these flows to define a switch's policy.

OpenFlow flows support wildcards, priorities, and multiple

tables.

When in-band control is in use, Open vSwitch sets up a few

"hidden" flows, with priority higher than a controller or the

user can configure, that are not visible via OpenFlow. (See

the "Controller" section of the FAQ for more information

about hidden flows.)

- The Open vSwitch software switch implementation uses a second

kind of flow internally. These flows, called "datapath" or

"kernel" flows, do not support priorities and comprise only a

single table, which makes them suitable for caching. (Like

OpenFlow flows, datapath flows do support wildcarding, in Open

vSwitch 1.11 and later.) OpenFlow flows and datapath flows

also support different actions and number ports differently.

Datapath flows are an implementation detail that is subject to

change in future versions of Open vSwitch. Even with the

current version of Open vSwitch, hardware switch

implementations do not necessarily use this architecture.

Users and controllers directly control only the OpenFlow flow

table. Open vSwitch manages the datapath flow table itself, so

users should not normally be concerned with it.

Q: Why are there so many different ways to dump flows?

A: Open vSwitch has two kinds of flows (see the previous question), so

it has commands with different purposes for dumping each kind of

flow:

- "ovs-ofctl dump-flows <br>" dumps OpenFlow flows, excluding

hidden flows. This is the most commonly useful form of flow

dump. (Unlike the other commands, this should work with any

OpenFlow switch, not just Open vSwitch.)

- "ovs-appctl bridge/dump-flows <br>" dumps OpenFlow flows,

including hidden flows. This is occasionally useful for

troubleshooting suspected issues with in-band control.

- "ovs-dpctl dump-flows [dp]" dumps the datapath flow table

entries for a Linux kernel-based datapath. In Open vSwitch

1.10 and later, ovs-vswitchd merges multiple switches into a

single datapath, so it will show all the flows on all your

kernel-based switches. This command can occasionally be

useful for debugging.

- "ovs-appctl dpif/dump-flows <br>", new in Open vSwitch 1.10,

dumps datapath flows for only the specified bridge, regardless

of the type.

Performance

-----------

Q: I just upgraded and I see a performance drop. Why?

A: The OVS kernel datapath may have been updated to a newer version than

the OVS userspace components. Sometimes new versions of OVS kernel

module add functionality that is backwards compatible with older

userspace components but may cause a drop in performance with them.

Especially, if a kernel module from OVS 2.1 or newer is paired with

OVS userspace 1.10 or older, there will be a performance drop for

TCP traffic.

Updating the OVS userspace components to the latest released

version should fix the performance degradation.

To get the best possible performance and functionality, it is

recommended to pair the same versions of the kernel module and OVS

userspace.

Configuration Problems

----------------------

Q: I created a bridge and added my Ethernet port to it, using commands

like these:

ovs-vsctl add-br br0

ovs-vsctl add-port br0 eth0

and as soon as I ran the "add-port" command I lost all connectivity

through eth0. Help!

A: A physical Ethernet device that is part of an Open vSwitch bridge

should not have an IP address. If one does, then that IP address

will not be fully functional.

You can restore functionality by moving the IP address to an Open

vSwitch "internal" device, such as the network device named after

the bridge itself. For example, assuming that eth0's IP address is

192.168.128.5, you could run the commands below to fix up the

situation:

ifconfig eth0 0.0.0.0

ifconfig br0 192.168.128.5

(If your only connection to the machine running OVS is through the

IP address in question, then you would want to run all of these

commands on a single command line, or put them into a script.) If

there were any additional routes assigned to eth0, then you would

also want to use commands to adjust these routes to go through br0.

If you use DHCP to obtain an IP address, then you should kill the

DHCP client that was listening on the physical Ethernet interface

(e.g. eth0) and start one listening on the internal interface

(e.g. br0). You might still need to manually clear the IP address

from the physical interface (e.g. with "ifconfig eth0 0.0.0.0").

There is no compelling reason why Open vSwitch must work this way.

However, this is the way that the Linux kernel bridge module has

always worked, so it's a model that those accustomed to Linux

bridging are already used to. Also, the model that most people

expect is not implementable without kernel changes on all the

versions of Linux that Open vSwitch supports.

By the way, this issue is not specific to physical Ethernet

devices. It applies to all network devices except Open vswitch

"internal" devices.

Q: I created a bridge and added a couple of Ethernet ports to it,

using commands like these:

ovs-vsctl add-br br0

ovs-vsctl add-port br0 eth0

ovs-vsctl add-port br0 eth1

and now my network seems to have melted: connectivity is unreliable

(even connectivity that doesn't go through Open vSwitch), all the

LEDs on my physical switches are blinking, wireshark shows

duplicated packets, and CPU usage is very high.

A: More than likely, you've looped your network. Probably, eth0 and

eth1 are connected to the same physical Ethernet switch. This

yields a scenario where OVS receives a broadcast packet on eth0 and

sends it out on eth1, then the physical switch connected to eth1

sends the packet back on eth0, and so on forever. More complicated

scenarios, involving a loop through multiple switches, are possible

too.

The solution depends on what you are trying to do:

- If you added eth0 and eth1 to get higher bandwidth or higher

reliability between OVS and your physical Ethernet switch,

use a bond. The following commands create br0 and then add

eth0 and eth1 as a bond:

ovs-vsctl add-br br0

ovs-vsctl add-bond br0 bond0 eth0 eth1

Bonds have tons of configuration options. Please read the

documentation on the Port table in ovs-vswitchd.conf.db(5)

for all the details.

- Perhaps you don't actually need eth0 and eth1 to be on the

same bridge. For example, if you simply want to be able to

connect each of them to virtual machines, then you can put

each of them on a bridge of its own:

ovs-vsctl add-br br0

ovs-vsctl add-port br0 eth0

ovs-vsctl add-br br1

ovs-vsctl add-port br1 eth1

and then connect VMs to br0 and br1. (A potential

disadvantage is that traffic cannot directly pass between br0

and br1. Instead, it will go out eth0 and come back in eth1,

or vice versa.)

- If you have a redundant or complex network topology and you

want to prevent loops, turn on spanning tree protocol (STP).

The following commands create br0, enable STP, and add eth0

and eth1 to the bridge. The order is important because you

don't want have to have a loop in your network even

transiently:

ovs-vsctl add-br br0

ovs-vsctl set bridge br0 stp\_enable=true

ovs-vsctl add-port br0 eth0

ovs-vsctl add-port br0 eth1

The Open vSwitch implementation of STP is not well tested.

Please report any bugs you observe, but if you'd rather avoid

acting as a beta tester then another option might be your

best shot.

Q: I can't seem to use Open vSwitch in a wireless network.

A: Wireless base stations generally only allow packets with the source

MAC address of NIC that completed the initial handshake.

Therefore, without MAC rewriting, only a single device can

communicate over a single wireless link.

This isn't specific to Open vSwitch, it's enforced by the access

point, so the same problems will show up with the Linux bridge or

any other way to do bridging.

Q: I can't seem to add my PPP interface to an Open vSwitch bridge.

A: PPP most commonly carries IP packets, but Open vSwitch works only

with Ethernet frames. The correct way to interface PPP to an

Ethernet network is usually to use routing instead of switching.

Q: Is there any documentation on the database tables and fields?

A: Yes. ovs-vswitchd.conf.db(5) is a comprehensive reference.

Q: When I run ovs-dpctl I no longer see the bridges I created. Instead,

I only see a datapath called "ovs-system". How can I see datapath

information about a particular bridge?

A: In version 1.9.0, OVS switched to using a single datapath that is

shared by all bridges of that type. The "ovs-appctl dpif/\*"

commands provide similar functionality that is scoped by the bridge.

Quality of Service (QoS)

------------------------

Q: How do I configure Quality of Service (QoS)?

A: Suppose that you want to set up bridge br0 connected to physical

Ethernet port eth0 (a 1 Gbps device) and virtual machine interfaces

vif1.0 and vif2.0, and that you want to limit traffic from vif1.0

to eth0 to 10 Mbps and from vif2.0 to eth0 to 20 Mbps. Then, you

could configure the bridge this way:

ovs-vsctl -- \

add-br br0 -- \

add-port br0 eth0 -- \

add-port br0 vif1.0 -- set interface vif1.0 ofport\_request=5 -- \

add-port br0 vif2.0 -- set interface vif2.0 ofport\_request=6 -- \

set port eth0 qos=@newqos -- \

--id=@newqos create qos type=linux-htb \

other-config:max-rate=1000000000 \

queues:123=@vif10queue \

queues:234=@vif20queue -- \

--id=@vif10queue create queue other-config:max-rate=10000000 -- \

--id=@vif20queue create queue other-config:max-rate=20000000

At this point, bridge br0 is configured with the ports and eth0 is

configured with the queues that you need for QoS, but nothing is

actually directing packets from vif1.0 or vif2.0 to the queues that

we have set up for them. That means that all of the packets to

eth0 are going to the "default queue", which is not what we want.

We use OpenFlow to direct packets from vif1.0 and vif2.0 to the

queues reserved for them:

ovs-ofctl add-flow br0 in\_port=5,actions=set\_queue:123,normal

ovs-ofctl add-flow br0 in\_port=6,actions=set\_queue:234,normal

Each of the above flows matches on the input port, sets up the

appropriate queue (123 for vif1.0, 234 for vif2.0), and then

executes the "normal" action, which performs the same switching

that Open vSwitch would have done without any OpenFlow flows being

present. (We know that vif1.0 and vif2.0 have OpenFlow port

numbers 5 and 6, respectively, because we set their ofport\_request

columns above. If we had not done that, then we would have needed

to find out their port numbers before setting up these flows.)

Now traffic going from vif1.0 or vif2.0 to eth0 should be

rate-limited.

By the way, if you delete the bridge created by the above commands,

with:

ovs-vsctl del-br br0

then that will leave one unreferenced QoS record and two

unreferenced Queue records in the Open vSwich database. One way to

clear them out, assuming you don't have other QoS or Queue records

that you want to keep, is:

ovs-vsctl -- --all destroy QoS -- --all destroy Queue

If you do want to keep some QoS or Queue records, or the Open

vSwitch you are using is older than version 1.8 (which added the

--all option), then you will have to destroy QoS and Queue records

individually.

Q: I configured Quality of Service (QoS) in my OpenFlow network by

adding records to the QoS and Queue table, but the results aren't

what I expect.

A: Did you install OpenFlow flows that use your queues? This is the

primary way to tell Open vSwitch which queues you want to use. If

you don't do this, then the default queue will be used, which will

probably not have the effect you want.

Refer to the previous question for an example.

Q: I configured QoS, correctly, but my measurements show that it isn't

working as well as I expect.

A: With the Linux kernel, the Open vSwitch implementation of QoS has

two aspects:

- Open vSwitch configures a subset of Linux kernel QoS

features, according to what is in OVSDB. It is possible that

this code has bugs. If you believe that this is so, then you

can configure the Linux traffic control (QoS) stack directly

with the "tc" program. If you get better results that way,

you can send a detailed bug report to bugs@openvswitch.org.

It is certain that Open vSwitch cannot configure every Linux

kernel QoS feature. If you need some feature that OVS cannot

configure, then you can also use "tc" directly (or add that

feature to OVS).

- The Open vSwitch implementation of OpenFlow allows flows to

be directed to particular queues. This is pretty simple and

unlikely to have serious bugs at this point.

However, most problems with QoS on Linux are not bugs in Open

vSwitch at all. They tend to be either configuration errors

(please see the earlier questions in this section) or issues with

the traffic control (QoS) stack in Linux. The Open vSwitch

developers are not experts on Linux traffic control. We suggest

that, if you believe you are encountering a problem with Linux

traffic control, that you consult the tc manpages (e.g. tc(8),

tc-htb(8), tc-hfsc(8)), web resources (e.g. http://lartc.org/), or

mailing lists (e.g. http://vger.kernel.org/vger-lists.html#netdev).

VLANs

-----

Q: What's a VLAN?

A: At the simplest level, a VLAN (short for "virtual LAN") is a way to

partition a single switch into multiple switches. Suppose, for

example, that you have two groups of machines, group A and group B.

You want the machines in group A to be able to talk to each other,

and you want the machine in group B to be able to talk to each

other, but you don't want the machines in group A to be able to

talk to the machines in group B. You can do this with two

switches, by plugging the machines in group A into one switch and

the machines in group B into the other switch.

If you only have one switch, then you can use VLANs to do the same

thing, by configuring the ports for machines in group A as VLAN

"access ports" for one VLAN and the ports for group B as "access

ports" for a different VLAN. The switch will only forward packets

between ports that are assigned to the same VLAN, so this

effectively subdivides your single switch into two independent

switches, one for each group of machines.

So far we haven't said anything about VLAN headers. With access

ports, like we've described so far, no VLAN header is present in

the Ethernet frame. This means that the machines (or switches)

connected to access ports need not be aware that VLANs are

involved, just like in the case where we use two different physical

switches.

Now suppose that you have a whole bunch of switches in your

network, instead of just one, and that some machines in group A are

connected directly to both switches 1 and 2. To allow these

machines to talk to each other, you could add an access port for

group A's VLAN to switch 1 and another to switch 2, and then

connect an Ethernet cable between those ports. That works fine,

but it doesn't scale well as the number of switches and the number

of VLANs increases, because you use up a lot of valuable switch

ports just connecting together your VLANs.

This is where VLAN headers come in. Instead of using one cable and

two ports per VLAN to connect a pair of switches, we configure a

port on each switch as a VLAN "trunk port". Packets sent and

received on a trunk port carry a VLAN header that says what VLAN

the packet belongs to, so that only two ports total are required to

connect the switches, regardless of the number of VLANs in use.

Normally, only switches (either physical or virtual) are connected

to a trunk port, not individual hosts, because individual hosts

don't expect to see a VLAN header in the traffic that they receive.

None of the above discussion says anything about particular VLAN

numbers. This is because VLAN numbers are completely arbitrary.

One must only ensure that a given VLAN is numbered consistently

throughout a network and that different VLANs are given different

numbers. (That said, VLAN 0 is usually synonymous with a packet

that has no VLAN header, and VLAN 4095 is reserved.)

Q: VLANs don't work.

A: Many drivers in Linux kernels before version 3.3 had VLAN-related

bugs. If you are having problems with VLANs that you suspect to be

driver related, then you have several options:

- Upgrade to Linux 3.3 or later.

- Build and install a fixed version of the particular driver

that is causing trouble, if one is available.

- Use a NIC whose driver does not have VLAN problems.

- Use "VLAN splinters", a feature in Open vSwitch 1.4 and later

that works around bugs in kernel drivers. To enable VLAN

splinters on interface eth0, use the command:

ovs-vsctl set interface eth0 other-config:enable-vlan-splinters=true

For VLAN splinters to be effective, Open vSwitch must know

which VLANs are in use. See the "VLAN splinters" section in

the Interface table in ovs-vswitchd.conf.db(5) for details on

how Open vSwitch infers in-use VLANs.

VLAN splinters increase memory use and reduce performance, so

use them only if needed.

- Apply the "vlan workaround" patch from the XenServer kernel

patch queue, build Open vSwitch against this patched kernel,

and then use ovs-vlan-bug-workaround(8) to enable the VLAN

workaround for each interface whose driver is buggy.

(This is a nontrivial exercise, so this option is included

only for completeness.)

It is not always easy to tell whether a Linux kernel driver has

buggy VLAN support. The ovs-vlan-test(8) and ovs-test(8) utilities

can help you test. See their manpages for details. Of the two

utilities, ovs-test(8) is newer and more thorough, but

ovs-vlan-test(8) may be easier to use.

Q: VLANs still don't work. I've tested the driver so I know that it's OK.

A: Do you have VLANs enabled on the physical switch that OVS is

attached to? Make sure that the port is configured to trunk the

VLAN or VLANs that you are using with OVS.

Q: Outgoing VLAN-tagged traffic goes through OVS to my physical switch

and to its destination host, but OVS seems to drop incoming return

traffic.

A: It's possible that you have the VLAN configured on your physical

switch as the "native" VLAN. In this mode, the switch treats

incoming packets either tagged with the native VLAN or untagged as

part of the native VLAN. It may also send outgoing packets in the

native VLAN without a VLAN tag.

If this is the case, you have two choices:

- Change the physical switch port configuration to tag packets

it forwards to OVS with the native VLAN instead of forwarding

them untagged.

- Change the OVS configuration for the physical port to a

native VLAN mode. For example, the following sets up a

bridge with port eth0 in "native-tagged" mode in VLAN 9:

ovs-vsctl add-br br0

ovs-vsctl add-port br0 eth0 tag=9 vlan\_mode=native-tagged

In this situation, "native-untagged" mode will probably work

equally well. Refer to the documentation for the Port table

in ovs-vswitchd.conf.db(5) for more information.

Q: I added a pair of VMs on different VLANs, like this:

ovs-vsctl add-br br0

ovs-vsctl add-port br0 eth0

ovs-vsctl add-port br0 tap0 tag=9

ovs-vsctl add-port br0 tap1 tag=10

but the VMs can't access each other, the external network, or the

Internet.

A: It is to be expected that the VMs can't access each other. VLANs

are a means to partition a network. When you configured tap0 and

tap1 as access ports for different VLANs, you indicated that they

should be isolated from each other.

As for the external network and the Internet, it seems likely that

the machines you are trying to access are not on VLAN 9 (or 10) and

that the Internet is not available on VLAN 9 (or 10).

Q: I added a pair of VMs on the same VLAN, like this:

ovs-vsctl add-br br0

ovs-vsctl add-port br0 eth0

ovs-vsctl add-port br0 tap0 tag=9

ovs-vsctl add-port br0 tap1 tag=9

The VMs can access each other, but not the external network or the

Internet.

A: It seems likely that the machines you are trying to access in the

external network are not on VLAN 9 and that the Internet is not

available on VLAN 9. Also, ensure VLAN 9 is set up as an allowed

trunk VLAN on the upstream switch port to which eth0 is connected.

Q: Can I configure an IP address on a VLAN?

A: Yes. Use an "internal port" configured as an access port. For

example, the following configures IP address 192.168.0.7 on VLAN 9.

That is, OVS will forward packets from eth0 to 192.168.0.7 only if

they have an 802.1Q header with VLAN 9. Conversely, traffic

forwarded from 192.168.0.7 to eth0 will be tagged with an 802.1Q

header with VLAN 9:

ovs-vsctl add-br br0

ovs-vsctl add-port br0 eth0

ovs-vsctl add-port br0 vlan9 tag=9 -- set interface vlan9 type=internal

ifconfig vlan9 192.168.0.7

Q: My OpenFlow controller doesn't see the VLANs that I expect.

A: The configuration for VLANs in the Open vSwitch database (e.g. via

ovs-vsctl) only affects traffic that goes through Open vSwitch's

implementation of the OpenFlow "normal switching" action. By

default, when Open vSwitch isn't connected to a controller and

nothing has been manually configured in the flow table, all traffic

goes through the "normal switching" action. But, if you set up

OpenFlow flows on your own, through a controller or using ovs-ofctl

or through other means, then you have to implement VLAN handling

yourself.

You can use "normal switching" as a component of your OpenFlow

actions, e.g. by putting "normal" into the lists of actions on

ovs-ofctl or by outputting to OFPP\_NORMAL from an OpenFlow

controller. In situations where this is not suitable, you can

implement VLAN handling yourself, e.g.:

- If a packet comes in on an access port, and the flow table

needs to send it out on a trunk port, then the flow can add

the appropriate VLAN tag with the "mod\_vlan\_vid" action.

- If a packet comes in on a trunk port, and the flow table

needs to send it out on an access port, then the flow can

strip the VLAN tag with the "strip\_vlan" action.

Q: I configured ports on a bridge as access ports with different VLAN

tags, like this:

ovs-vsctl add-br br0

ovs-vsctl set-controller br0 tcp:192.168.0.10:6633

ovs-vsctl add-port br0 eth0

ovs-vsctl add-port br0 tap0 tag=9

ovs-vsctl add-port br0 tap1 tag=10

but the VMs running behind tap0 and tap1 can still communicate,

that is, they are not isolated from each other even though they are

on different VLANs.

A: Do you have a controller configured on br0 (as the commands above

do)? If so, then this is a variant on the previous question, "My

OpenFlow controller doesn't see the VLANs that I expect," and you

can refer to the answer there for more information.

VXLANs

-----

Q: What's a VXLAN?

A: VXLAN stands for Virtual eXtensible Local Area Network, and is a means

to solve the scaling challenges of VLAN networks in a multi-tenant

environment. VXLAN is an overlay network which transports an L2 network

over an existing L3 network. For more information on VXLAN, please see

the IETF draft available here:

http://tools.ietf.org/html/draft-mahalingam-dutt-dcops-vxlan-03

Q: How much of the VXLAN protocol does Open vSwitch currently support?

A: Open vSwitch currently supports the framing format for packets on the

wire. There is currently no support for the multicast aspects of VXLAN.

To get around the lack of multicast support, it is possible to

pre-provision MAC to IP address mappings either manually or from a

controller.

Q: What destination UDP port does the VXLAN implementation in Open vSwitch

use?

A: By default, Open vSwitch will use the assigned IANA port for VXLAN, which

is 4789. However, it is possible to configure the destination UDP port

manually on a per-VXLAN tunnel basis. An example of this configuration is

provided below.

ovs-vsctl add-br br0

ovs-vsctl add-port br0 vxlan1 -- set interface vxlan1

type=vxlan options:remote\_ip=192.168.1.2 options:key=flow

options:dst\_port=8472

Using OpenFlow (Manually or Via Controller)

-------------------------------------------

Q: What versions of OpenFlow does Open vSwitch support?

A: Open vSwitch 1.9 and earlier support only OpenFlow 1.0 (plus

extensions that bring in many of the features from later versions

of OpenFlow).

Open vSwitch 1.10 and later have experimental support for OpenFlow

1.2 and 1.3. On these versions of Open vSwitch, the following

command enables OpenFlow 1.0, 1.2, and 1.3 on bridge br0:

ovs-vsctl set bridge br0 protocols=OpenFlow10,OpenFlow12,OpenFlow13

Open vSwitch version 1.12 and later will have experimental support

for OpenFlow 1.1, 1.2, and 1.3. On these versions of Open vSwitch,

the following command enables OpenFlow 1.0, 1.1, 1.2, and 1.3 on

bridge br0:

ovs-vsctl set bridge br0 protocols=OpenFlow10,OpenFlow11,OpenFlow12,OpenFlow13

Use the -O option to enable support for later versions of OpenFlow

in ovs-ofctl. For example:

ovs-ofctl -O OpenFlow13 dump-flows br0

Support for OpenFlow 1.1, 1.2, and 1.3 is still incomplete. Work

to be done is tracked in OPENFLOW-1.1+ in the Open vSwitch sources

(also via http://openvswitch.org/development/openflow-1-x-plan/).

When support for a given OpenFlow version is solidly implemented,

Open vSwitch will enable that version by default.

Q: I'm getting "error type 45250 code 0". What's that?

A: This is a Open vSwitch extension to OpenFlow error codes. Open

vSwitch uses this extension when it must report an error to an

OpenFlow controller but no standard OpenFlow error code is

suitable.

Open vSwitch logs the errors that it sends to controllers, so the

easiest thing to do is probably to look at the ovs-vswitchd log to

find out what the error was.

If you want to dissect the extended error message yourself, the

format is documented in include/openflow/nicira-ext.h in the Open

vSwitch source distribution. The extended error codes are

documented in lib/ofp-errors.h.

Q1: Some of the traffic that I'd expect my OpenFlow controller to see

doesn't actually appear through the OpenFlow connection, even

though I know that it's going through.

Q2: Some of the OpenFlow flows that my controller sets up don't seem

to apply to certain traffic, especially traffic between OVS and

the controller itself.

A: By default, Open vSwitch assumes that OpenFlow controllers are

connected "in-band", that is, that the controllers are actually

part of the network that is being controlled. In in-band mode,

Open vSwitch sets up special "hidden" flows to make sure that

traffic can make it back and forth between OVS and the controllers.

These hidden flows are higher priority than any flows that can be

set up through OpenFlow, and they are not visible through normal

OpenFlow flow table dumps.

Usually, the hidden flows are desirable and helpful, but

occasionally they can cause unexpected behavior. You can view the

full OpenFlow flow table, including hidden flows, on bridge br0

with the command:

ovs-appctl bridge/dump-flows br0

to help you debug. The hidden flows are those with priorities

greater than 65535 (the maximum priority that can be set with

OpenFlow).

The DESIGN file at the top level of the Open vSwitch source

distribution describes the in-band model in detail.

If your controllers are not actually in-band (e.g. they are on

localhost via 127.0.0.1, or on a separate network), then you should

configure your controllers in "out-of-band" mode. If you have one

controller on bridge br0, then you can configure out-of-band mode

on it with:

ovs-vsctl set controller br0 connection-mode=out-of-band

Q: I configured all my controllers for out-of-band control mode but

"ovs-appctl bridge/dump-flows" still shows some hidden flows.

A: You probably have a remote manager configured (e.g. with "ovs-vsctl

set-manager"). By default, Open vSwitch assumes that managers need

in-band rules set up on every bridge. You can disable these rules

on bridge br0 with:

ovs-vsctl set bridge br0 other-config:disable-in-band=true

This actually disables in-band control entirely for the bridge, as

if all the bridge's controllers were configured for out-of-band

control.

Q: My OpenFlow controller doesn't see the VLANs that I expect.

A: See answer under "VLANs", above.

Q: I ran "ovs-ofctl add-flow br0 nw\_dst=192.168.0.1,actions=drop"

but I got a funny message like this:

ofp\_util|INFO|normalization changed ofp\_match, details:

ofp\_util|INFO| pre: nw\_dst=192.168.0.1

ofp\_util|INFO|post:

and when I ran "ovs-ofctl dump-flows br0" I saw that my nw\_dst

match had disappeared, so that the flow ends up matching every

packet.

A: The term "normalization" in the log message means that a flow

cannot match on an L3 field without saying what L3 protocol is in

use. The "ovs-ofctl" command above didn't specify an L3 protocol,

so the L3 field match was dropped.

In this case, the L3 protocol could be IP or ARP. A correct

command for each possibility is, respectively:

ovs-ofctl add-flow br0 ip,nw\_dst=192.168.0.1,actions=drop

and

ovs-ofctl add-flow br0 arp,nw\_dst=192.168.0.1,actions=drop

Similarly, a flow cannot match on an L4 field without saying what

L4 protocol is in use. For example, the flow match "tp\_src=1234"

is, by itself, meaningless and will be ignored. Instead, to match

TCP source port 1234, write "tcp,tp\_src=1234", or to match UDP

source port 1234, write "udp,tp\_src=1234".

Q: How can I figure out the OpenFlow port number for a given port?

A: The OFPT\_FEATURES\_REQUEST message requests an OpenFlow switch to

respond with an OFPT\_FEATURES\_REPLY that, among other information,

includes a mapping between OpenFlow port names and numbers. From a

command prompt, "ovs-ofctl show br0" makes such a request and

prints the response for switch br0.

The Interface table in the Open vSwitch database also maps OpenFlow

port names to numbers. To print the OpenFlow port number

associated with interface eth0, run:

ovs-vsctl get Interface eth0 ofport

You can print the entire mapping with:

ovs-vsctl -- --columns=name,ofport list Interface

but the output mixes together interfaces from all bridges in the

database, so it may be confusing if more than one bridge exists.

In the Open vSwitch database, ofport value -1 means that the

interface could not be created due to an error. (The Open vSwitch

log should indicate the reason.) ofport value [] (the empty set)

means that the interface hasn't been created yet. The latter is

normally an intermittent condition (unless ovs-vswitchd is not

running).

Q: I added some flows with my controller or with ovs-ofctl, but when I

run "ovs-dpctl dump-flows" I don't see them.

A: ovs-dpctl queries a kernel datapath, not an OpenFlow switch. It

won't display the information that you want. You want to use

"ovs-ofctl dump-flows" instead.

Q: It looks like each of the interfaces in my bonded port shows up

as an individual OpenFlow port. Is that right?

A: Yes, Open vSwitch makes individual bond interfaces visible as

OpenFlow ports, rather than the bond as a whole. The interfaces

are treated together as a bond for only a few purposes:

- Sending a packet to the OFPP\_NORMAL port. (When an OpenFlow

controller is not configured, this happens implicitly to

every packet.)

- Mirrors configured for output to a bonded port.

It would make a lot of sense for Open vSwitch to present a bond as

a single OpenFlow port. If you want to contribute an

implementation of such a feature, please bring it up on the Open

vSwitch development mailing list at dev@openvswitch.org.

Q: I have a sophisticated network setup involving Open vSwitch, VMs or

multiple hosts, and other components. The behavior isn't what I

expect. Help!

A: To debug network behavior problems, trace the path of a packet,

hop-by-hop, from its origin in one host to a remote host. If

that's correct, then trace the path of the response packet back to

the origin.

Usually a simple ICMP echo request and reply ("ping") packet is

good enough. Start by initiating an ongoing "ping" from the origin

host to a remote host. If you are tracking down a connectivity

problem, the "ping" will not display any successful output, but

packets are still being sent. (In this case the packets being sent

are likely ARP rather than ICMP.)

Tools available for tracing include the following:

- "tcpdump" and "wireshark" for observing hops across network

devices, such as Open vSwitch internal devices and physical

wires.

- "ovs-appctl dpif/dump-flows <br>" in Open vSwitch 1.10 and

later or "ovs-dpctl dump-flows <br>" in earlier versions.

These tools allow one to observe the actions being taken on

packets in ongoing flows.

See ovs-vswitchd(8) for "ovs-appctl dpif/dump-flows"

documentation, ovs-dpctl(8) for "ovs-dpctl dump-flows"

documentation, and "Why are there so many different ways to

dump flows?" above for some background.

- "ovs-appctl ofproto/trace" to observe the logic behind how

ovs-vswitchd treats packets. See ovs-vswitchd(8) for

documentation. You can out more details about a given flow

that "ovs-dpctl dump-flows" displays, by cutting and pasting

a flow from the output into an "ovs-appctl ofproto/trace"

command.

- SPAN, RSPAN, and ERSPAN features of physical switches, to

observe what goes on at these physical hops.

Starting at the origin of a given packet, observe the packet at

each hop in turn. For example, in one plausible scenario, you

might:

1. "tcpdump" the "eth" interface through which an ARP egresses

a VM, from inside the VM.

2. "tcpdump" the "vif" or "tap" interface through which the ARP

ingresses the host machine.

3. Use "ovs-dpctl dump-flows" to spot the ARP flow and observe

the host interface through which the ARP egresses the

physical machine. You may need to use "ovs-dpctl show" to

interpret the port numbers. If the output seems surprising,

you can use "ovs-appctl ofproto/trace" to observe details of

how ovs-vswitchd determined the actions in the "ovs-dpctl

dump-flows" output.

4. "tcpdump" the "eth" interface through which the ARP egresses

the physical machine.

5. "tcpdump" the "eth" interface through which the ARP

ingresses the physical machine, at the remote host that

receives the ARP.

6. Use "ovs-dpctl dump-flows" to spot the ARP flow on the

remote host that receives the ARP and observe the VM "vif"

or "tap" interface to which the flow is directed. Again,

"ovs-dpctl show" and "ovs-appctl ofproto/trace" might help.

7. "tcpdump" the "vif" or "tap" interface to which the ARP is

directed.

8. "tcpdump" the "eth" interface through which the ARP

ingresses a VM, from inside the VM.

It is likely that during one of these steps you will figure out the

problem. If not, then follow the ARP reply back to the origin, in

reverse.

Q: How do I make a flow drop packets?

A: To drop a packet is to receive it without forwarding it. OpenFlow

explicitly specifies forwarding actions. Thus, a flow with an

empty set of actions does not forward packets anywhere, causing

them to be dropped. You can specify an empty set of actions with

"actions=" on the ovs-ofctl command line. For example:

ovs-ofctl add-flow br0 priority=65535,actions=

would cause every packet entering switch br0 to be dropped.

You can write "drop" explicitly if you like. The effect is the

same. Thus, the following command also causes every packet

entering switch br0 to be dropped:

ovs-ofctl add-flow br0 priority=65535,actions=drop

"drop" is not an action, either in OpenFlow or Open vSwitch.

Rather, it is only a way to say that there are no actions.

Q: I added a flow to send packets out the ingress port, like this:

ovs-ofctl add-flow br0 in\_port=2,actions=2

but OVS drops the packets instead.

A: Yes, OpenFlow requires a switch to ignore attempts to send a packet

out its ingress port. The rationale is that dropping these packets

makes it harder to loop the network. Sometimes this behavior can

even be convenient, e.g. it is often the desired behavior in a flow

that forwards a packet to several ports ("floods" the packet).

Sometimes one really needs to send a packet out its ingress port.

In this case, output to OFPP\_IN\_PORT, which in ovs-ofctl syntax is

expressed as just "in\_port", e.g.:

ovs-ofctl add-flow br0 in\_port=2,actions=in\_port

This also works in some circumstances where the flow doesn't match

on the input port. For example, if you know that your switch has

five ports numbered 2 through 6, then the following will send every

received packet out every port, even its ingress port:

ovs-ofctl add-flow br0 actions=2,3,4,5,6,in\_port

or, equivalently:

ovs-ofctl add-flow br0 actions=all,in\_port

Sometimes, in complicated flow tables with multiple levels of

"resubmit" actions, a flow needs to output to a particular port

that may or may not be the ingress port. It's difficult to take

advantage of OFPP\_IN\_PORT in this situation. To help, Open vSwitch

provides, as an OpenFlow extension, the ability to modify the

in\_port field. Whatever value is currently in the in\_port field is

the port to which outputs will be dropped, as well as the

destination for OFPP\_IN\_PORT. This means that the following will

reliably output to port 2 or to ports 2 through 6, respectively:

ovs-ofctl add-flow br0 in\_port=2,actions=load:0->NXM\_OF\_IN\_PORT[],2

ovs-ofctl add-flow br0 actions=load:0->NXM\_OF\_IN\_PORT[],2,3,4,5,6

If the input port is important, then one may save and restore it on

the stack:

ovs-ofctl add-flow br0 actions=push:NXM\_OF\_IN\_PORT[],\

load:0->NXM\_OF\_IN\_PORT[],\

2,3,4,5,6,\

pop:NXM\_OF\_IN\_PORT[]

Q: My bridge br0 has host 192.168.0.1 on port 1 and host 192.168.0.2

on port 2. I set up flows to forward only traffic destined to the

other host and drop other traffic, like this:

priority=5,in\_port=1,ip,nw\_dst=192.168.0.2,actions=2

priority=5,in\_port=2,ip,nw\_dst=192.168.0.1,actions=1

priority=0,actions=drop

But it doesn't work--I don't get any connectivity when I do this.

Why?

A: These flows drop the ARP packets that IP hosts use to establish IP

connectivity over Ethernet. To solve the problem, add flows to

allow ARP to pass between the hosts:

priority=5,in\_port=1,arp,actions=2

priority=5,in\_port=2,arp,actions=1

This issue can manifest other ways, too. The following flows that

match on Ethernet addresses instead of IP addresses will also drop

ARP packets, because ARP requests are broadcast instead of being

directed to a specific host:

priority=5,in\_port=1,dl\_dst=54:00:00:00:00:02,actions=2

priority=5,in\_port=2,dl\_dst=54:00:00:00:00:01,actions=1

priority=0,actions=drop

The solution already described above will also work in this case.

It may be better to add flows to allow all multicast and broadcast

traffic:

priority=5,in\_port=1,dl\_dst=01:00:00:00:00:00/01:00:00:00:00:00,actions=2

priority=5,in\_port=2,dl\_dst=01:00:00:00:00:00/01:00:00:00:00:00,actions=1

Contact

-------

bugs@openvswitch.org

http://openvswitch.org/