

**NAME**

ovs-vswitchd – Open vSwitch daemon

**SYNOPSIS**

**ovs-vswitchd** [*database*]

**DESCRIPTION**

A daemon that manages and controls any number of Open vSwitch switches on the local machine.

The *database* argument specifies how **ovs-vswitchd** connects to **ovsdb-server**. The default is **unix:/usr/local/var/run/openvswitch/db.sock**. The following forms are accepted:

**ssl:***ip:port*

**tcp:***ip:port*

The given SSL or plain TCP *port* on the host at the given *ip*, which must be expressed as an IP address (not a DNS name) in IPv4 or IPv6 address format. If *ip* is an IPv6 address, then wrap *ip* with square brackets, e.g.: **ssl:::1:6640**. On Linux, use *%device* to designate a scope for IPv6 link-level addresses, e.g. **ssl:[fe80::1234%eth0]:6653**. For **ssl**, the **—private-key**, **—certificate**, and **—ca-cert** options are mandatory.

**unix:***file*

On POSIX, connect to the Unix domain server socket named *file*.

On Windows, connect to a local named pipe that is represented by a file created in the path *file* to mimic the behavior of a Unix domain socket.

**pssl:***port[:ip]*

**ptcp:***port[:ip]*

Listen on the given SSL or TCP *port* for a connection. By default, connections are not bound to a particular local IP address and it listens only on IPv4 (but not IPv6) addresses, but specifying *ip* limits connections to those from the given *ip*, either IPv4 or IPv6 address. If *ip* is an IPv6 address, then wrap *ip* with square brackets, e.g.: **pssl:6640:::1**. On Linux, use *%device* to designate a scope for IPv6 link-level addresses, e.g. **pssl:6653:[fe80::1234%eth0]**. For **pssl**, the **—private-key**, **—certificate**, and **—ca-cert** options are mandatory.

**punix:***file*

On POSIX, listen on the Unix domain server socket named *file* for a connection.

On Windows, listen on a local named pipe. A file is created in the path *file* to mimic the behavior of a Unix domain socket.

**ovs-vswitchd** retrieves its configuration from *database* at startup. It sets up Open vSwitch datapaths and then operates switching across each bridge described in its configuration files. As the database changes, **ovs-vswitchd** automatically updates its configuration to match.

**ovs-vswitchd** switches may be configured with any of the following features:

- L2 switching with MAC learning.
- NIC bonding with automatic fail-over and source MAC-based TX load balancing ("SLB").
- 802.1Q VLAN support.
- Port mirroring, with optional VLAN tagging.
- NetFlow v5 flow logging.
- sFlow(R) monitoring.
- Connectivity to an external OpenFlow controller, such as NOX.

Only a single instance of **ovs-vswitchd** is intended to run at a time. A single **ovs-vswitchd** can manage any number of switch instances, up to the maximum number of supported Open vSwitch datapaths.

**ovs-vswitchd** does all the necessary management of Open vSwitch datapaths itself. Thus, external tools, such **ovs-dpctl**(8), are not needed for managing datapaths in conjunction with **ovs-vswitchd**, and their use

to modify datapaths when **ovs-vswitchd** is running can interfere with its operation. (**ovs-dpctl** may still be useful for diagnostics.)

An Open vSwitch datapath kernel module must be loaded for **ovs-vswitchd** to be useful. Refer to the documentation for instructions on how to build and load the Open vSwitch kernel module.

## OPTIONS

### —mlockall

Causes **ovs-vswitchd** to call the **mlockall()** function, to attempt to lock all of its process memory into physical RAM, preventing the kernel from paging any of its memory to disk. This helps to avoid networking interruptions due to system memory pressure.

Some systems do not support **mlockall()** at all, and other systems only allow privileged users, such as the superuser, to use it. **ovs-vswitchd** emits a log message if **mlockall()** is unavailable or unsuccessful.

### DPDK Options

For details on initializing the **ovs-vswitchd** DPDK datapath, refer to the documentation or **ovs-vswitchd.conf.db(5)** for details.

### Daemon Options

The following options are valid on POSIX based platforms.

#### —pidfile[=*pidfile*]

Causes a file (by default, **ovs-vswitchd.pid**) to be created indicating the PID of the running process. If the *pidfile* argument is not specified, or if it does not begin with /, then it is created in **/usr/local/var/run/openvswitch**.

If **—pidfile** is not specified, no pidfile is created.

#### —overwrite-pidfile

By default, when **—pidfile** is specified and the specified pidfile already exists and is locked by a running process, **ovs-vswitchd** refuses to start. Specify **—overwrite-pidfile** to cause it to instead overwrite the pidfile.

When **—pidfile** is not specified, this option has no effect.

#### —detach

Runs **ovs-vswitchd** as a background process. The process forks, and in the child it starts a new session, closes the standard file descriptors (which has the side effect of disabling logging to the console), and changes its current directory to the root (unless **—no-chdir** is specified). After the child completes its initialization, the parent exits. **ovs-vswitchd** detaches only after it has connected to the database, retrieved the initial configuration, and set up that configuration.

#### —monitor

Creates an additional process to monitor the **ovs-vswitchd** daemon. If the daemon dies due to a signal that indicates a programming error (**SIGABRT**, **SIGALRM**, **SIGBUS**, **SIGFPE**, **SIGILL**, **SIGPIPE**, **SIGSEGV**, **SIGXCPU**, or **SIGXFSZ**) then the monitor process starts a new copy of it. If the daemon dies or exits for another reason, the monitor process exits.

This option is normally used with **—detach**, but it also functions without it.

#### —no-chdir

By default, when **—detach** is specified, **ovs-vswitchd** changes its current working directory to the root directory after it detaches. Otherwise, invoking **ovs-vswitchd** from a carelessly chosen directory would prevent the administrator from unmounting the file system that holds that directory.

Specifying **—no-chdir** suppresses this behavior, preventing **ovs-vswitchd** from changing its current working directory. This may be useful for collecting core files, since it is common behavior to write core dumps into the current working directory and the root directory is not a good directory to use.

This option has no effect when **—detach** is not specified.

**—no-self-confinement**

By default daemon will try to self-confine itself to work with files under well-know, at build-time whitelisted directories. It is better to stick with this default behavior and not to use this flag unless some other Access Control is used to confine daemon. Note that in contrast to other access control implementations that are typically enforced from kernel-space (e.g. DAC or MAC), self-confinement is imposed from the user-space daemon itself and hence should not be considered as a full confinement strategy, but instead should be viewed as an additional layer of security.

**—user** Causes **ovs-vswitchd** to run as a different user specified in "user:group", thus dropping most of the root privileges. Short forms "user" and ":group" are also allowed, with current user or group are assumed respectively. Only daemons started by the root user accepts this argument.

On Linux, daemons will be granted CAP\_IPC\_LOCK and CAP\_NET\_BIND\_SERVICES before dropping root privileges. Daemons that interact with a datapath, such as **ovs-vswitchd**, will be granted two additional capabilities, namely CAP\_NET\_ADMIN and CAP\_NET\_RAW. The capability change will apply even if new user is "root".

On Windows, this option is not currently supported. For security reasons, specifying this option will cause the daemon process not to start.

### Service Options

The following options are valid only on Windows platform.

**—service**

Causes **ovs-vswitchd** to run as a service in the background. The service should already have been created through external tools like **SC.exe**.

**—service-monitor**

Causes the **ovs-vswitchd** service to be automatically restarted by the Windows services manager if the service dies or exits for unexpected reasons.

When **—service** is not specified, this option has no effect.

### Public Key Infrastructure Options

**—p** *privkey.pem*

**—private-key=***privkey.pem*

Specifies a PEM file containing the private key used as **ovs-vswitchd**'s identity for outgoing SSL connections.

**—c** *cert.pem*

**—certificate=***cert.pem*

Specifies a PEM file containing a certificate that certifies the private key specified on **—p** or **—private-key** to be trustworthy. The certificate must be signed by the certificate authority (CA) that the peer in SSL connections will use to verify it.

**—C** *cacert.pem*

**—ca-cert=***cacert.pem*

Specifies a PEM file containing the CA certificate that **ovs-vswitchd** should use to verify certificates presented to it by SSL peers. (This may be the same certificate that SSL peers use to verify the certificate specified on **—c** or **—certificate**, or it may be a different one, depending on the PKI design in use.)

**—C** *none*

**—ca-cert=***none*

Disables verification of certificates presented by SSL peers. This introduces a security risk, because it means that certificates cannot be verified to be those of known trusted hosts.

**—bootstrap-ca-cert=***cacert.pem*

When *cacert.pem* exists, this option has the same effect as **—C** or **—ca-cert**. If it does not exist, then **ovs-vswitchd** will attempt to obtain the CA certificate from the SSL peer on its first SSL

connection and save it to the named PEM file. If it is successful, it will immediately drop the connection and reconnect, and from then on all SSL connections must be authenticated by a certificate signed by the CA certificate thus obtained.

**This option exposes the SSL connection to a man-in-the-middle attack obtaining the initial CA certificate**, but it may be useful for bootstrapping.

This option is only useful if the SSL peer sends its CA certificate as part of the SSL certificate chain. The SSL protocol does not require the server to send the CA certificate.

This option is mutually exclusive with `-C` and `—ca-cert`.

## Logging Options

`-v[spec]`

`—verbose=[spec]`

Sets logging levels. Without any *spec*, sets the log level for every module and destination to **dbg**. Otherwise, *spec* is a list of words separated by spaces or commas or colons, up to one from each category below:

- A valid module name, as displayed by the **vlog/list** command on **ovs-appctl(8)**, limits the log level change to the specified module.
- **syslog**, **console**, or **file**, to limit the log level change to only to the system log, to the console, or to a file, respectively. (If `—detach` is specified, **ovs-vswitchd** closes its standard file descriptors, so logging to the console will have no effect.)

On Windows platform, **syslog** is accepted as a word and is only useful along with the `—syslog-target` option (the word has no effect otherwise).

- **off**, **emer**, **err**, **warn**, **info**, or **dbg**, to control the log level. Messages of the given severity or higher will be logged, and messages of lower severity will be filtered out. **off** filters out all messages. See **ovs-appctl(8)** for a definition of each log level.

Case is not significant within *spec*.

Regardless of the log levels set for **file**, logging to a file will not take place unless `—log-file` is also specified (see below).

For compatibility with older versions of OVS, **any** is accepted as a word but has no effect.

`-v`

`—verbose`

Sets the maximum logging verbosity level, equivalent to `—verbose=dbg`.

`-vPATTERN:destination:pattern`

`—verbose=PATTERN:destination:pattern`

Sets the log pattern for *destination* to *pattern*. Refer to **ovs-appctl(8)** for a description of the valid syntax for *pattern*.

`-vFACILITY:facility`

`—verbose=FACILITY:facility`

Sets the RFC5424 facility of the log message. *facility* can be one of **kern**, **user**, **mail**, **daemon**, **auth**, **syslog**, **lpr**, **news**, **uucp**, **clock**, **ftp**, **ntp**, **audit**, **alert**, **clock2**, **local0**, **local1**, **local2**, **local3**, **local4**, **local5**, **local6** or **local7**. If this option is not specified, **daemon** is used as the default for the local system syslog and **local0** is used while sending a message to the target provided via the `—syslog-target` option.

`—log-file[=file]`

Enables logging to a file. If *file* is specified, then it is used as the exact name for the log file. The default log file name used if *file* is omitted is **/usr/local/var/log/openvswitch/ovs-vswitchd.log**.

`—syslog-target=host:port`

Send syslog messages to UDP *port* on *host*, in addition to the system syslog. The *host* must be a numerical IP address, not a hostname.

**—syslog=method**

Specify *method* how syslog messages should be sent to syslog daemon. Following forms are supported:

- **libc**, use libc **syslog()** function. This is the default behavior. Downside of using this options is that libc adds fixed prefix to every message before it is actually sent to the syslog daemon over **/dev/log** UNIX domain socket.
- **unix:file**, use UNIX domain socket directly. It is possible to specify arbitrary message format with this option. However, **rsyslogd 8.9** and older versions use hard coded parser function anyway that limits UNIX domain socket use. If you want to use arbitrary message format with older **rsyslogd** versions, then use UDP socket to localhost IP address instead.
- **udp:ip:port**, use UDP socket. With this method it is possible to use arbitrary message format also with older **rsyslogd**. When sending syslog messages over UDP socket extra precaution needs to be taken into account, for example, syslog daemon needs to be configured to listen on the specified UDP port, accidental iptables rules could be interfering with local syslog traffic and there are some security considerations that apply to UDP sockets, but do not apply to UNIX domain sockets.

**Other Options****—unixctl=socket**

Sets the name of the control socket on which **ovs-vswitchd** listens for runtime management commands (see **RUNTIME MANAGEMENT COMMANDS**, below). If *socket* does not begin with */*, it is interpreted as relative to **/usr/local/var/run/openvswitch**. If **—unixctl** is not used at all, the default socket is **/usr/local/var/run/openvswitch/ovs-vswitchd.pid.ctl**, where *pid* is **ovs-vswitchd**'s process ID.

On Windows a local named pipe is used to listen for runtime management commands. A file is created in the absolute path as pointed by *socket* or if **—unixctl** is not used at all, a file is created as **ovs-vswitchd.ctl** in the configured **OVS\_RUNDIR** directory. The file exists just to mimic the behavior of a Unix domain socket.

Specifying **none** for *socket* disables the control socket feature.

**-h**

**—help** Prints a brief help message to the console.

**-V****—version**

Prints version information to the console.

**RUNTIME MANAGEMENT COMMANDS**

**ovs-appctl(8)** can send commands to a running **ovs-vswitchd** process. The currently supported commands are described below. The command descriptions assume an understanding of how to configure Open vSwitch.

**GENERAL COMMANDS****exit --cleanup**

Causes **ovs-vswitchd** to gracefully terminate. If **--cleanup** is specified, release datapath resources configured by **ovs-vswitchd**. Otherwise, datapath flows and other resources remains undeleted.

**qos/show-types interface**

Queries the interface for a list of Quality of Service types that are configurable via Open vSwitch for the given *interface*.

**qos/show interface**

Queries the kernel for Quality of Service configuration and statistics associated with the given *interface*.

**bfd/show** [*interface*]

Displays detailed information about Bidirectional Forwarding Detection configured on *interface*. If *interface* is not specified, then displays detailed information about all interfaces with BFD enabled.

**bfd/set-forwarding** [*interface*] *status*

Force the fault status of the BFD module on *interface* (or all interfaces if none is given) to be *status*. *status* can be "true", "false", or "normal" which reverts to the standard behavior.

**cfm/show** [*interface*]

Displays detailed information about Connectivity Fault Management configured on *interface*. If *interface* is not specified, then displays detailed information about all interfaces with CFM enabled.

**cfm/set-fault** [*interface*] *status*

Force the fault status of the CFM module on *interface* (or all interfaces if none is given) to be *status*. *status* can be "true", "false", or "normal" which reverts to the standard behavior.

**stp/tcn** [*bridge*]

Forces a topology change event on *bridge* if it's running STP. This may cause it to send Topology Change Notifications to its peers and flush its MAC table. If no *bridge* is given, forces a topology change event on all bridges.

**stp/show** [*bridge*]

Displays detailed information about spanning tree on the *bridge*. If *bridge* is not specified, then displays detailed information about all bridges with STP enabled.

**rstp/tcn** [*bridge*]

Forces a topology change event on *bridge* if it's running RSTP. This may cause it to send Topology Change Notifications to its peers and flush its MAC table. If no *bridge* is given, forces a topology change event on all bridges.

**rstp/show** [*bridge*]

Displays detailed information about rapid spanning tree on the *bridge*. If *bridge* is not specified, then displays detailed information about all bridges with RSTP enabled.

**BRIDGE COMMANDS**

These commands manage bridges.

**fdb/flush** [*bridge*]

Flushes *bridge* MAC address learning table, or all learning tables if no *bridge* is given.

**fdb/show** *bridge*

Lists each MAC address/VLAN pair learned by the specified *bridge*, along with the port on which it was learned and the age of the entry, in seconds.

**mdb/flush** [*bridge*]

Flushes *bridge* multicast snooping table, or all snooping tables if no *bridge* is given.

**mdb/show** *bridge*

Lists each multicast group/VLAN pair learned by the specified *bridge*, along with the port on which it was learned and the age of the entry, in seconds.

**bridge/reconnect** [*bridge*]

Makes *bridge* drop all of its OpenFlow controller connections and reconnect. If *bridge* is not specified, then all bridges drop their controller connections and reconnect.

This command might be useful for debugging OpenFlow controller issues.

**bridge/dump-flows** *bridge*

Lists all flows in *bridge*, including those normally hidden to commands such as **ovs-ofctl dump-flows**. Flows set up by mechanisms such as in-band control and fail-open are hidden from the controller since it is not allowed to modify or override them.

## BOND COMMANDS

These commands manage bonded ports on an Open vSwitch's bridges. To understand some of these commands, it is important to understand a detail of the bonding implementation called "source load balancing" (SLB). Instead of directly assigning Ethernet source addresses to slaves, the bonding implementation computes a function that maps an 48-bit Ethernet source addresses into an 8-bit value (a "MAC hash" value). All of the Ethernet addresses that map to a single 8-bit value are then assigned to a single slave.

### **bond/list**

Lists all of the bonds, and their slaves, on each bridge.

### **bond/show** [*port*]

Lists all of the bond-specific information (updelay, downdelay, time until the next rebalance) about the given bonded *port*, or all bonded ports if no *port* is given. Also lists information about each slave: whether it is enabled or disabled, the time to completion of an updelay or downdelay if one is in progress, whether it is the active slave, the hashes assigned to the slave. Any LACP information related to this bond may be found using the **lACP/show** command.

### **bond/migrate** *port hash slave*

Only valid for SLB bonds. Assigns a given MAC hash to a new slave. *port* specifies the bond port, *hash* the MAC hash to be migrated (as a decimal number between 0 and 255), and *slave* the new slave to be assigned.

The reassignment is not permanent: rebalancing or fail-over will cause the MAC hash to be shifted to a new slave in the usual manner.

A MAC hash cannot be migrated to a disabled slave.

### **bond/set-active-slave** *port slave*

Sets *slave* as the active slave on *port*. *slave* must currently be enabled.

The setting is not permanent: a new active slave will be selected if *slave* becomes disabled.

### **bond/enable-slave** *port slave*

### **bond/disable-slave** *port slave*

Enables (or disables) *slave* on the given bond *port*, skipping any updelay (or downdelay).

This setting is not permanent: it persists only until the carrier status of *slave* changes.

### **bond/hash** *mac* [*vlan*] [*basis*]

Returns the hash value which would be used for *mac* with *vlan* and *basis* if specified.

### **lACP/show** [*port*]

Lists all of the LACP related information about the given *port*: active or passive, aggregation key, system id, and system priority. Also lists information about each slave: whether it is enabled or disabled, whether it is attached or detached, port id and priority, actor information, and partner information. If *port* is not specified, then displays detailed information about all interfaces with CFM enabled.

## DPCTL DATAPATH DEBUGGING COMMANDS

The primary way to configure **ovs-vswitchd** is through the Open vSwitch database, e.g. using **ovs-vsctl**(8). These commands provide a debugging interface for managing datapaths. They implement the same features (and syntax) as **ovs-dpctl**(8). Unlike **ovs-dpctl**(8), these commands work with datapaths that are integrated into **ovs-vswitchd** (e.g. the **netdev** datapath type).

### **dpctl/add-dp** *dp* [*netdev*[,*option*]...]

Creates datapath *dp*, with a local port also named *dp*. This will fail if a network device *dp* already exists.

If *netdevs* are specified, **ovs-vswitchd** adds them to the new datapath, just as if **add-if** was specified.

**dpctl/del-dp *dp***

Deletes datapath *dp*. If *dp* is associated with any network devices, they are automatically removed.

**dpctl/add-if *dp netdev*[,*option*]...**

Adds each *netdev* to the set of network devices datapath *dp* monitors, where *dp* is the name of an existing datapath, and *netdev* is the name of one of the host's network devices, e.g. **eth0**. Once a network device has been added to a datapath, the datapath has complete ownership of the network device's traffic and the network device appears silent to the rest of the system.

A *netdev* may be followed by a comma-separated list of options. The following options are currently supported:

**type=*type***

Specifies the type of port to add. The default type is **system**.

**port\_no=*port***

Requests a specific port number within the datapath. If this option is not specified then one will be automatically assigned.

**key=*value***

Adds an arbitrary key-value option to the port's configuration.

**ovs-vswitchd.conf.db**(5) documents the available port types and options.

**dpctl/set-if *dp port*[,*option*]...**

Reconfigures each *port* in *dp* as specified. An *option* of the form *key=value* adds the specified key-value option to the port or overrides an existing key's value. An *option* of the form *key=*, that is, without a value, deletes the key-value named *key*. The type and port number of a port cannot be changed, so **type** and **port\_no** are only allowed if they match the existing configuration.

**dpctl/del-if *dp netdev*...**

Removes each *netdev* from the list of network devices datapath *dp* monitors.

**dpctl/dump-dps**

Prints the name of each configured datapath on a separate line.

**dpctl/show [-s | --statistics] [*dp*...]**

Prints a summary of configured datapaths, including their datapath numbers and a list of ports connected to each datapath. (The local port is identified as port 0.) If **-s** or **--statistics** is specified, then packet and byte counters are also printed for each port.

The datapath numbers consists of flow stats and mega flow mask stats.

The "lookups" row displays three stats related to flow lookup triggered by processing incoming packets in the datapath. "hit" displays number of packets matches existing flows. "missed" displays the number of packets not matching any existing flow and require user space processing. "lost" displays number of packets destined for user space process but subsequently dropped before reaching userspace. The sum of "hit" and "miss" equals to the total number of packets datapath processed.

The "flows" row displays the number of flows in datapath.

The "masks" row displays the mega flow mask stats. This row is omitted for datapath not implementing mega flow. "hit" displays the total number of masks visited for matching incoming packets. "total" displays number of masks in the datapath. "hit/pkt" displays the average number of masks visited per packet; the ratio between "hit" and total number of packets processed by the datapath.

If one or more datapaths are specified, information on only those datapaths are displayed. Otherwise, **ovs-vswitchd** displays information about all configured datapaths.



## DATAPATH FLOW TABLE DEBUGGING COMMANDS

The following commands are primarily useful for debugging Open vSwitch. The flow table entries (both matches and actions) that they work with are not OpenFlow flow entries. Instead, they are different and considerably simpler flows maintained by the Open vSwitch kernel module. Use **ovs-ofctl(8)**, instead, to work with OpenFlow flow entries.

The *dp* argument to each of these commands is optional when exactly one datapath exists, in which case that datapath is the default. When multiple datapaths exist, then a datapath name is required.

**dpctl/dump-flows** [**-m** | **---more**] [**---names** | **---no-names**] [*dp*] [**filter=filter**] [**type=type**]

Prints to the console all flow entries in datapath *dp*'s flow table. Without **-m** or **---more**, output omits match fields that a flow wildcards entirely; with **-m** or **---more**, output includes all wildcarded fields.

If **filter=filter** is specified, only displays the flows that match the *filter*. *filter* is a flow in the form similar to that accepted by **ovs-ofctl(8)**'s **add-flow** command. (This is not an OpenFlow flow: besides other differences, it never contains wildcards.) The *filter* is also useful to match wildcarded fields in the datapath flow. As an example, **filter='tcp,tp\_src=100'** will match the datapath flow containing **'tcp(src=80/0xff00,dst=8080/0xff)'**.

If **type=type** is specified, only displays flows of a specific type. *type* can be **offloaded** to display only offloaded rules or **OVS** to display only non-offloaded rules. By default both offloaded and non-offloaded rules are displayed.

**dpctl/add-flow** [*dp*] *flow actions*

**dpctl/mod-flow** [**---clear**] [**---may-create**] [**-s** | **---statistics**] [*dp*] *flow actions*

Adds or modifies a flow in *dp*'s flow table that, when a packet matching *flow* arrives, causes *actions* to be executed.

The **add-flow** command succeeds only if *flow* does not already exist in *dp*. Contrariwise, **mod-flow** without **---may-create** only modifies the actions for an existing flow. With **---may-create**, **mod-flow** will add a new flow or modify an existing one.

If **-s** or **---statistics** is specified, then **mod-flow** prints the modified flow's statistics. A flow's statistics are the number of packets and bytes that have passed through the flow, the elapsed time since the flow last processed a packet (if ever), and (for TCP flows) the union of the TCP flags processed through the flow.

With **---clear**, **mod-flow** zeros out the flow's statistics. The statistics printed if **-s** or **---statistics** is also specified are those from just before clearing the statistics.

NOTE: *flow* and *actions* do not match the syntax used with **ovs-ofctl(8)**'s **add-flow** command.

### Usage Examples

Forward ARP between ports 1 and 2 on datapath myDP:

```
ovs-dpctl add-flow myDP \
    "in_port(1),eth(),eth_type(0x0806),arp()" 2
ovs-dpctl add-flow myDP \
    "in_port(2),eth(),eth_type(0x0806),arp()" 1
```

Forward all IPv4 traffic between two addresses on ports 1 and 2:

```
ovs-dpctl add-flow myDP \
    "in_port(1),eth(),eth_type(0x800),\
    ipv4(src=172.31.110.4,dst=172.31.110.5)" 2
ovs-dpctl add-flow myDP \
    "in_port(2),eth(),eth_type(0x800),\
    ipv4(src=172.31.110.5,dst=172.31.110.4)" 1
```

**dpctl/del-flow** [-s | --statistics] [dp] flow

Deletes the flow from *dp*'s flow table that matches *flow*. If **-s** or **--statistics** is specified, then **del-flow** prints the deleted flow's statistics.

**dpctl/get-flow** [dp] ufid:ufid [-m | --more] [--names | --no-names]

Fetches the flow from *dp*'s flow table with unique identifier *ufid*. *ufid* must be specified as a string of 32 hexadecimal characters.

**dpctl/del-flows** [dp]

Deletes all flow entries from datapath *dp*'s flow table.

## CONNECTION TRACKING TABLE DEBUGGING COMMANDS

The following commands are primarily useful for debugging the connection tracking entries in the datapath.

The *dp* argument to each of these commands is optional when exactly one datapath exists, in which case that datapath is the default. When multiple datapaths exist, then a datapath name is required.

**N.B.**(Linux specific): the *system* datapaths (i.e. the Linux kernel module Open vSwitch datapaths) share a single connection tracking table (which is also used by other kernel subsystems, such as iptables, nftables and the regular host stack). Therefore, the following commands do not apply specifically to one datapath.

**dpctl/dump-conntack** [-m | --more] [-s | --statistics] [dp] [zone=zone]

Prints to the console all the connection entries in the tracker used by *dp*. If **zone=zone** is specified, only shows the connections in **zone**. With **--more**, some implementation specific details are included. With **--statistics** timeouts and timestamps are added to the output.

**dpctl/flush-conntack** [dp] [zone=zone]

Flushes all the connection entries in the tracker used by *dp*. If **zone=zone** is specified, only flushes the connections in **zone**.

**dpctl/ct-stats-show** [dp] [zone=zone] [verbose]

Displays the number of connections grouped by protocol used by *dp*. If **zone=zone** is specified, numbers refer to the connections in **zone**. The **verbose** option allows to group by connection state for each protocol.

**dpctl/ct-bkts** [dp] [gt=Threshold]

For each ConnTracker bucket, displays the number of connections used by *dp*. If **gt=Threshold** is specified, bucket numbers are displayed when the number of connections in a bucket is greater than *Threshold*.

## DPIF-NETDEV COMMANDS

These commands are used to expose internal information (mostly statistics) about the “dpif-netdev” userspace datapath. If there is only one datapath (as is often the case, unless **dpctl/** commands are used), the *dp* argument can be omitted.

**dpif-netdev/pmd-stats-show** [dp]

Shows performance statistics for each pmd thread of the datapath *dp*. The special thread “main” sums up the statistics of every non pmd thread. The sum of “emc hits”, “masked hits” and “miss” is the number of packets received by the datapath. Cycles are counted using the TSC or similar facilities (when available on the platform). To reset these counters use **dpif-netdev/pmd-stats-clear**. The duration of one cycle depends on the measuring infrastructure. “idle cycles” refers to cycles spent polling devices but not receiving any packets. “processing cycles” refers to cycles spent polling devices and successfully receiving packets, plus the cycles spent processing said packets.

**dpif-netdev/pmd-stats-clear** [dp]

Resets to zero the per pmd thread performance numbers shown by the **dpif-netdev/pmd-stats-show** command. It will NOT reset datapath or bridge statistics, only the values shown by the above command.

**dpif-netdev/pmd-rxq-show** [*dp*]

For each pmd thread of the datapath *dp* shows list of queue-ids with port names, which this thread polls.

**dpif-netdev/pmd-rxq-rebalance** [*dp*]

Reassigns rxqs to pmds in the datapath *dp* based on their current usage.

**DATAPATH DEBUGGING COMMANDS**

These commands query and modify datapaths. They are similar to **ovs-dpctl**(8) commands. **dpif/show** has the additional functionality, beyond **dpctl/show** of printing OpenFlow port numbers. The other commands are redundant and will be removed in a future release.

**dpif/dump-dps**

Prints the name of each configured datapath on a separate line.

**dpif/show**

Prints a summary of configured datapaths, including statistics and a list of connected ports. The port information includes the OpenFlow port number, datapath port number, and the type. (The local port is identified as OpenFlow port 65534.)

**dpif/dump-flows** [**-m**] *dp*

Prints to the console all flow entries in datapath *dp*'s flow table. Without **-m**, output omits match fields that a flow wilcards entirely; with **-m** output includes all wildcarded fields.

This command is primarily useful for debugging Open vSwitch. The flow table entries that it displays are not OpenFlow flow entries. Instead, they are different and considerably simpler flows maintained by the datapath module. If you wish to see the OpenFlow flow entries, use **ovs-ofctl dump-flows**.

**dpif/del-flows** *dp*

Deletes all flow entries from datapath *dp*'s flow table and underlying datapath implementation (e.g., kernel datapath module).

This command is primarily useful for debugging Open vSwitch. As discussed in **dpif/dump-flows**, these entries are not OpenFlow flow entries.

**OFPROTO COMMANDS**

These commands manage the core OpenFlow switch implementation (called **ofproto**).

**ofproto/list**

Lists the names of the running ofproto instances. These are the names that may be used on **ofproto/trace**.

**ofproto/trace** [*dpname*] *odp\_flow* [*OPTIONS*] [**-generate** | *packet*]**ofproto/trace bridge br\_flow** [*OPTIONS*] [**-generate** | *packet*]**ofproto/trace-packet-out** [**-consistent**] [*dpname*] *odp\_flow* [*OPTIONS*] [**-generate** | *packet*] *actions***ofproto/trace-packet-out** [**-consistent**] *bridge br\_flow* [*OPTIONS*] [**-generate** | *packet*] *actions*

Traces the path of an imaginary packet through *switch* and reports the path that it took. The initial treatment of the packet varies based on the command:

- **ofproto/trace** looks the packet up in the OpenFlow flow table, as if the packet had arrived on an OpenFlow port.
- **ofproto/trace-packet-out** applies the specified OpenFlow *actions*, as if the packet, flow, and actions had been specified in an OpenFlow “packet-out” request.

The packet's headers (e.g. source and destination) and metadata (e.g. input port), together called its “flow,” are usually all that matter for the purpose of tracing a packet. You can specify the flow in the following ways:

*dpname odp\_flow*

*odp\_flow* is a flow in the form printed by **ovs-dpctl**(8)'s **dump-flows** command. If all of your bridges have the same type, which is the common case, then you can omit *dpname*,

but if you have bridges of different types (say, both **ovs-netdev** and **ovs-system**), then you need to specify a *dpname* to disambiguate.

#### *bridge br\_flow*

*br\_flow* is a flow in the form similar to that accepted by **ovs-ofctl(8)**'s **add-flow** command. (This is not an OpenFlow flow: besides other differences, it never contains wildcards.) *bridge* names of the bridge through which *br\_flow* should be traced.

**ofproto/trace** supports the following options:

#### **--ct-next** *flags*

When the traced flow triggers conntrack actions, **ofproto/trace** will automatically trace the forked packet processing pipeline with user specified *ct\_state*. This option sets the *ct\_state* flags that the conntrack module will report. The *flags* must be a comma- or space-separated list of the following connection tracking flags:

- **trk**: Include to indicate connection tracking has taken place.
- **new**: Include to indicate a new flow.
- **est**: Include to indicate an established flow.
- **rel**: Include to indicate a related flow.
- **rpl**: Include to indicate a reply flow.
- **inv**: Include to indicate a connection entry in a bad state.
- **dnat**: Include to indicate a packet whose destination IP address has been changed.
- **snat**: Include to indicate a packet whose source IP address has been changed.

When **--ct-next** is unspecified, or when there are fewer **--ct-next** options than *ct* actions, the *flags* default to *trk,new*.

Most commonly, one specifies only a flow, using one of the forms above, but sometimes one might need to specify an actual packet instead of just a flow:

#### Side effects.

Some actions have side effects. For example, the **normal** action can update the MAC learning table, and the **learn** action can change OpenFlow tables. The trace commands only perform side effects when a packet is specified. If you want side effects to take place, then you must supply a packet.

(Output actions are obviously side effects too, but the trace commands never execute them, even when one specifies a packet.)

#### Incomplete information.

Most of the time, Open vSwitch can figure out everything about the path of a packet using just the flow, but in some special circumstances it needs to look at parts of the packet that are not included in the flow. When this is the case, and you do not supply a packet, then a trace command will tell you it needs a packet.

If you wish to include a packet as part of a trace operation, there are two ways to do it:

#### **--generate**

This option, added to one of the ways to specify a flow already described, causes Open vSwitch to internally generate a packet with the flow described and then to use that packet. If your goal is to execute side effects, then **--generate** is the easiest way to do it, but **--generate** is not a good way to fill in incomplete information, because it generates packets based on only the flow information, which means that the packets really do not have any more information than the flow.

*packet* This form supplies an explicit *packet* as a sequence of hex digits. An Ethernet frame is at least 14 bytes long, so there must be at least 28 hex digits. Obviously, it is inconvenient

to type in the hex digits by hand, so the **ovs-pcap(1)** and **ovs-tcpdump(1)** utilities provide easier ways.

With this form, packet headers are extracted directly from *packet*, so the *odp\_flow* or *br\_flow* should specify only metadata. The metadata can be:

*skb\_priority*

Packet QoS priority.

*pkt\_mark*

Mark of the packet.

*ct\_state*

Connection state of the packet.

*ct\_zone* Connection tracking zone for packet.

*ct\_mark*

Connection mark of the packet.

*ct\_label*

Connection label of the packet.

*tun\_id* The tunnel ID on which the packet arrived.

*in\_port* The port on which the packet arrived.

The *in\_port* value is kernel datapath port number for the first format and OpenFlow port number for the second format. The numbering of these two types of port usually differs and there is no relationship.

**ofproto-trace-packet-out** accepts an additional **-consistent** option. With this option specified, the command rejects *actions* that are inconsistent with the specified packet. (An example of an inconsistency is attempting to strip the VLAN tag from a packet that does not have a VLAN tag.) Open vSwitch ignores most forms of inconsistency in OpenFlow 1.0 and rejects inconsistencies in later versions of OpenFlow. The option is necessary because the command does not ordinarily imply a particular OpenFlow version. One exception is that, when *actions* includes an action that only OpenFlow 1.1 and later supports (such as **push\_vlan**), **-consistent** is automatically enabled.

Usage examples:

**Trace an unicast ICMP echo request on ingress port 1 to destination MAC 00:00:5E:00:53:01**

```
ofproto/trace br in_port=1,icmp,icmp_type=8,\
dl_dst=00:00:5E:00:53:01
```

**Trace an unicast ICMP echo reply on ingress port 1 to destination MAC 00:00:5E:00:53:01**

```
ofproto/trace br in_port=1,icmp,icmp_type=0,\
dl_dst=00:00:5E:00:53:01
```

**Trace an ARP request on ingress port 1**

```
ofproto/trace br in_port=1,arp,arp_op=1
```

**Trace an ARP reply on ingress port 1**

```
ofproto/trace br in_port=1,arp,arp_op=2
```

## VLOG COMMANDS

These commands manage **ovs-vswitchd**'s logging settings.

**vlog/set** [*spec*]

Sets logging levels. Without any *spec*, sets the log level for every module and destination to **dbg**. Otherwise, *spec* is a list of words separated by spaces or commas or colons, up to one from each category below:

- A valid module name, as displayed by the **vlog/list** command on **ovs-appctl(8)**, limits the log level change to the specified module.

- **syslog**, **console**, or **file**, to limit the log level change to only to the system log, to the console, or to a file, respectively.

On Windows platform, **syslog** is accepted as a word and is only useful along with the **—syslog-target** option (the word has no effect otherwise).

- **off**, **emer**, **err**, **warn**, **info**, or **dbg**, to control the log level. Messages of the given severity or higher will be logged, and messages of lower severity will be filtered out. **off** filters out all messages. See **ovs-appctl(8)** for a definition of each log level.

Case is not significant within *spec*.

Regardless of the log levels set for **file**, logging to a file will not take place unless **ovs-vswitchd** was invoked with the **—log-file** option.

For compatibility with older versions of OVS, **any** is accepted as a word but has no effect.

#### **vlog/set PATTERN:destination:pattern**

Sets the log pattern for *destination* to *pattern*. Refer to **ovs-appctl(8)** for a description of the valid syntax for *pattern*.

#### **vlog/list**

Lists the supported logging modules and their current levels.

#### **vlog/list-pattern**

Lists logging patterns used for each destination.

#### **vlog/close**

Causes **ovs-vswitchd** to close its log file, if it is open. (Use **vlog/reopen** to reopen it later.)

#### **vlog/reopen**

Causes **ovs-vswitchd** to close its log file, if it is open, and then reopen it. (This is useful after rotating log files, to cause a new log file to be used.)

This has no effect unless **ovs-vswitchd** was invoked with the **—log-file** option.

#### **vlog/disable-rate-limit [module]...**

#### **vlog/enable-rate-limit [module]...**

By default, **ovs-vswitchd** limits the rate at which certain messages can be logged. When a message would appear more frequently than the limit, it is suppressed. This saves disk space, makes logs easier to read, and speeds up execution, but occasionally troubleshooting requires more detail. Therefore, **vlog/disable-rate-limit** allows rate limits to be disabled at the level of an individual log module. Specify one or more module names, as displayed by the **vlog/list** command. Specifying either no module names at all or the keyword **any** disables rate limits for every log module.

The **vlog/enable-rate-limit** command, whose syntax is the same as **vlog/disable-rate-limit**, can be used to re-enable a rate limit that was previously disabled.

### MEMORY COMMANDS

These commands report memory usage.

#### **memory/show**

Displays some basic statistics about **ovs-vswitchd**'s memory usage. **ovs-vswitchd** also logs this information soon after startup and periodically as its memory consumption grows.

### COVERAGE COMMANDS

These commands manage **ovs-vswitchd**'s "coverage counters," which count the number of times particular events occur during a daemon's runtime. In addition to these commands, **ovs-vswitchd** automatically logs coverage counter values, at **INFO** level, when it detects that the daemon's main loop takes unusually long to run.

Coverage counters are useful mainly for performance analysis and debugging.

**coverage/show**

Displays the averaged per-second rates for the last few seconds, the last minute and the last hour, and the total counts of all of the coverage counters.

**OPENVSWITCH TUNNELING COMMANDS**

These commands query and modify OVS tunnel components.

**ovs/route/add ipv4\_address/plen output\_bridge [GW]**

Adds ipv4\_address/plen route to vswitchd routing table. output\_bridge needs to be OVS bridge name. This command is useful if OVS cached routes does not look right.

**ovs/route/show**

Print all routes in OVS routing table, This includes routes cached from system routing table and user configured routes.

**ovs/route/del ipv4\_address/plen**

Delete ipv4\_address/plen route from OVS routing table.

**tnl/neigh/show****tnl/arp/show**

OVS builds ARP cache by snooping ARP messages. This command shows ARP cache table.

**tnl/neigh/set bridge ip mac****tnl/arp/set bridge ip mac**

Adds or modifies an ARP cache entry in bridge, mapping ip to mac.

**tnl/neigh/flush****tnl/arp/flush**

Flush ARP table.

**tnl/egress\_port\_range [num1] [num2]**

Set range for UDP source port used for UDP based Tunnels. For example VxLAN. If case of zero arguments this command prints current range in use.

**OPENFLOW IMPLEMENTATION**

This section documents aspects of OpenFlow for which the OpenFlow specification requires documentation.

**Packet buffering.**

The OpenFlow specification, version 1.2, says:

Switches that implement buffering are expected to expose, through documentation, both the amount of available buffering, and the length of time before buffers may be reused.

Open vSwitch does not maintain any packet buffers.

**Bundle lifetime**

The OpenFlow specification, version 1.4, says:

If the switch does not receive any OFPT\_BUNDLE\_CONTROL or OFPT\_BUNDLE\_ADD\_MESSAGE message for an opened bundle\_id for a switch defined time greater than 1s, it may send an ofp\_error\_msg with OFPET\_BUNDLE\_FAILED type and OFPBFC\_TIMEOUT code. If the switch does not receive any new message in a bundle apart from echo request and replies for a switch defined time greater than 1s, it may send an ofp\_error\_msg with OFPET\_BUNDLE\_FAILED type and OFPBFC\_TIMEOUT code.

Open vSwitch implements idle bundle lifetime of 10 seconds.

**LIMITS**

We believe these limits to be accurate as of this writing. These limits assume the use of the Linux kernel datapath.

- **ovs-vswitchd** started through **ovs-ctl(8)** provides a limit of 65535 file descriptors. The limits on the number of bridges and ports is decided by the availability of file descriptors. With the Linux kernel datapath, creation of a single bridge consumes three file descriptors and adding a port consumes "n-handler-threads" file descriptors per bridge port. Performance will degrade beyond 1,024 ports per bridge due to fixed hash table sizing. Other platforms may have different limitations.
- 2,048 MAC learning entries per bridge, by default. (This is configurable via **other-config:mac-table-size** in the **Bridge** table. See **ovs-vswitchd.conf.db(5)** for details.)
- Kernel flows are limited only by memory available to the kernel. Performance will degrade beyond 1,048,576 kernel flows per bridge with a 32-bit kernel, beyond 262,144 with a 64-bit kernel. (**ovs-vswitchd** should never install anywhere near that many flows.)
- OpenFlow flows are limited only by available memory. Performance is linear in the number of unique wildcard patterns. That is, an OpenFlow table that contains many flows that all match on the same fields in the same way has a constant-time lookup, but a table that contains many flows that match on different fields requires lookup time linear in the number of flows.
- 255 ports per bridge participating in 802.1D Spanning Tree Protocol.
- 32 mirrors per bridge.
- 15 bytes for the name of a port. (This is a Linux kernel limitation.)

**SEE ALSO**

**ovs-appctl(8)**, **ovsdb-server(1)**.