NAME

ovs−ofctl − administer OpenFlow switches

SYNOPSIS

ovs−ofctl [*options*] *command* [*switch*] [*args...*]

DESCRIPTION

The **ovs−ofctl** program is a command line tool for monitoring and administering OpenFlow switches. It can also show the current state of an OpenFlow switch, including features, configuration, and table entries. It should work with any OpenFlow switch, not just Open vSwitch.

OpenFlow Switch Management Commands

These commands allow **ovs−ofctl** to monitor and administer an OpenFlow switch. It is able to show the current state of a switch, including features, configuration, and table entries.

Most of these commands take an argument that specifies the method for connecting to an OpenFlow switch. The following connection methods are supported:

ssl:*ip*[**:***port*]
tcp:*ip*[**:***port*]
: The specified *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. Wrap IPv6 addresses in square brackets, e.g. **tcp:[::1]:6653**. For **ssl**, the **−−private−key**, **−−certificate**, and **−−ca−cert** options are mandatory.

  If *port* is not specified, it defaults to 6653.

unix:*file*
: On POSIX, a 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.

file
: This is short for **unix:***file*, as long as *file* does not contain a colon.

bridge
: This is short for **unix:/usr/local/var/run/openvswitch/***bridge***.mgmt**, as long as *bridge* does not contain a colon.

[*type***@**]*dp*
: Attempts to look up the bridge associated with *dp* and open as above. If *type* is given, it specifies the datapath provider of *dp*, otherwise the default provider **system** is assumed.

show *switch*
: Prints to the console information on *switch*, including information on its flow tables and ports.

dump−tables *switch*
: Prints to the console statistics for each of the flow tables used by *switch*.

dump−table−features *switch*
: Prints to the console features for each of the flow tables used by *switch*.

dump−table−desc *switch*
: Prints to the console configuration for each of the flow tables used by *switch* for OpenFlow 1.4+.

mod−table *switch table_id setting*
: This command configures flow table settings for OpenFlow table *table_id* within *switch*. The available settings depend on the OpenFlow version in use. In OpenFlow 1.1 and 1.2 (which must be enabled with the **−O** option) only, **mod−table** configures behavior when no flow is found when a packet is looked up in a flow table. The following *setting* values are available:

  drop Drop the packet.

  continue
  : Continue to the next table in the pipeline. (This is how an OpenFlow 1.0 switch always handles packets that do not match any flow, in tables other than the last one.)

controller
Send to controller. (This is how an OpenFlow 1.0 switch always handles packets that do not match any flow in the last table.)

In OpenFlow 1.4 and later (which must be enabled with the **−O** option) only, **mod−table** configures the behavior when a controller attempts to add a flow to a flow table that is full. The following *setting* values are available:

evict Delete some existing flow from the flow table, according to the algorithm described for the **Flow_Table** table in **ovs-vswitchd.conf.db**(5).

noevict Refuse to add the new flow. (Eviction might still be enabled through the **overflow_policy** column in the **Flow_Table** table documented in **ovs-vswitchd.conf.db**(5).)

vacancy:*low***,***high*
Enables sending vacancy events to controllers using **TABLE_STATUS** messages, based on percentage thresholds *low* and *high*.

novacancy
Disables vacancy events.

dump−ports *switch* [*netdev*]
Prints to the console statistics for network devices associated with *switch*. If *netdev* is specified, only the statistics associated with that device will be printed. *netdev* can be an OpenFlow assigned port number or device name, e.g. **eth0**.

dump−ports−desc *switch* [*port*]
Prints to the console detailed information about network devices associated with *switch*. To dump only a specific port, specify its number as *port*. Otherwise, if *port* is omitted, or if it is specified as **ANY**, then all ports are printed. This is a subset of the information provided by the **show** command.

If the connection to *switch* negotiates OpenFlow 1.0, 1.2, or 1.2, this command uses an OpenFlow extension only implemented in Open vSwitch (version 1.7 and later).

Only OpenFlow 1.5 and later support dumping a specific port. Earlier versions of OpenFlow always dump all ports.

mod−port *switch port action*
Modify characteristics of port **port** in *switch*. *port* may be an OpenFlow port number or name or the keyword **LOCAL** (the preferred way to refer to the OpenFlow local port). The *action* may be any one of the following:

up
down Enable or disable the interface. This is equivalent to **ifconfig up** or **ifconfig down** on a Unix system.

stp
no−stp Enable or disable 802.1D spanning tree protocol (STP) on the interface. OpenFlow implementations that don't support STP will refuse to enable it.

receive
no−receive
receive−stp
no−receive−stp
Enable or disable OpenFlow processing of packets received on this interface. When packet processing is disabled, packets will be dropped instead of being processed through the OpenFlow table. The **receive** or **no−receive** setting applies to all packets except 802.1D spanning tree packets, which are separately controlled by **receive−stp** or **no−receive−stp**.

forward
no−forward
Allow or disallow forwarding of traffic to this interface. By default, forwarding is enabled.

flood
no−flood
Controls whether an OpenFlow **flood** action will send traffic out this interface. By default, flooding is enabled. Disabling flooding is primarily useful to prevent loops when a spanning tree protocol is not in use.

packet−in
no−packet−in
Controls whether packets received on this interface that do not match a flow table entry generate a ‘‘packet in’’ message to the OpenFlow controller. By default, ‘‘packet in’’ messages are enabled.

The **show** command displays (among other information) the configuration that **mod−port** changes.

get−frags *switch*
Prints *switch*’s fragment handling mode. See **set−frags**, below, for a description of each fragment handling mode.

The **show** command also prints the fragment handling mode among its other output.

set−frags *switch frag_mode*
Configures *switch*’s treatment of IPv4 and IPv6 fragments. The choices for *frag_mode* are:

normal
Fragments pass through the flow table like non-fragmented packets. The TCP ports, UDP ports, and ICMP type and code fields are always set to 0, even for fragments where that information would otherwise be available (fragments with offset 0). This is the default fragment handling mode for an OpenFlow switch.

drop Fragments are dropped without passing through the flow table.

reassemble
The switch reassembles fragments into full IP packets before passing them through the flow table. Open vSwitch does not implement this fragment handling mode.

nx−match
Fragments pass through the flow table like non-fragmented packets. The TCP ports, UDP ports, and ICMP type and code fields are available for matching for fragments with offset 0, and set to 0 in fragments with nonzero offset. This mode is a Nicira extension.

See the description of **ip_frag**, below, for a way to match on whether a packet is a fragment and on its fragment offset.

dump−flows *switch* [*flows*]
Prints to the console all flow entries in *switch*’s tables that match *flows*. If *flows* is omitted, all flows in the switch are retrieved. See **Flow Syntax**, below, for the syntax of *flows*. The output format is described in **Table Entry Output**.

By default, **ovs−ofctl** prints flow entries in the same order that the switch sends them, which is unlikely to be intuitive or consistent. See the description of **−−sort** and **−−rsort**, under **OPTIONS** below, to influence the display order.

dump−aggregate *switch* [*flows*]
Prints to the console aggregate statistics for flows in *switch*’s tables that match *flows*. If *flows* is omitted, the statistics are aggregated across all flows in the switch’s flow tables. See **Flow Syntax**, below, for the syntax of *flows*. The output format is described in **Table Entry Output**.

queue−stats *switch* [*port* [*queue*]]

Prints to the console statistics for the specified *queue* on *port* within *switch*. *port* can be an OpenFlow port number or name, the keyword **LOCAL** (the preferred way to refer to the OpenFlow local port), or the keyword **ALL**. Either of *port* or *queue* or both may be omitted (or equivalently the keyword **ALL**). If both are omitted, statistics are printed for all queues on all ports. If only *queue* is omitted, then statistics are printed for all queues on *port*; if only *port* is omitted, then statistics are printed for *queue* on every port where it exists.

queue−get−config *switch [port* [*queue*]]

Prints to the console the configuration of *queue* on *port* in *switch*. If *port* is omitted or **ANY**, reports queues for all port. If *queue* is omitted or **ANY**, reports all queues. For OpenFlow 1.3 and earlier, the output always includes all queues, ignoring *queue* if specified.

This command has limited usefulness, because ports often have no configured queues and because the OpenFlow protocol provides only very limited information about the configuration of a queue.

dump−ipfix−bridge *switch*

Prints to the console the statistics of bridge IPFIX for *switch*. If bridge IPFIX is configured on the *switch*, IPFIX statistics can be retrieved. Otherwise, error message will be printed.

This command uses an Open vSwitch extension that is only in Open vSwitch 2.6 and later.

dump−ipfix−flow *switch*

Prints to the console the statistics of flow-based IPFIX for *switch*. If flow-based IPFIX is configured on the *switch*, statistics of all the collector set ids on the *switch* will be printed. Otherwise, print error message.

Refer to **ovs−vswitchd.conf.db**(5) for more details on configuring flow based IPFIX and collector set ids.

This command uses an Open vSwitch extension that is only in Open vSwitch 2.6 and later.

ct−flush−zone *switch zone*

Flushes the connection tracking entries in *zone* on *switch*.

This command uses an Open vSwitch extension that is only in Open vSwitch 2.6 and later.

OpenFlow 1.1+ Group Table Commands

The following commands work only with switches that support OpenFlow 1.1 or later. Because support for OpenFlow 1.1 and later is still experimental in Open vSwitch, it is necessary to explicitly enable these protocol versions in **ovs−ofctl** (using **−O**) and in the switch itself (with the **protocols** column in the **Bridge** table). For more information, see ‘‘Q: What versions of OpenFlow does Open vSwitch support?’’ in the Open vSwitch FAQ.

dump−groups *switch* [*group*]

Prints group entries in *switch*’s tables to console. To dump only a specific group, specify its number as *group*. Otherwise, if *group* is omitted, or if it is specified as **ALL**, then all groups are printed. Each line of output is a group entry as described in **Group Syntax** below.

Only OpenFlow 1.5 and later support dumping a specific group. Earlier versions of OpenFlow always dump all groups.

dump−group−features *switch*

Prints to the console the group features of the *switch*.

dump−group-stats *switch* [*groups*]

Prints to the console statistics for the specified *groups in the switch*’s tables. If *groups* is omitted then statistics for all groups are printed. See **Group Syntax**, below, for the syntax of *groups*.

OpenFlow 1.3+ Switch Meter Table Commands

These commands manage the meter table in an OpenFlow switch. In each case, *meter* specifies a meter entry in the format described in **Meter Syntax**, below.

OpenFlow 1.3 introduced support for meters, so these commands only work with switches that support

OpenFlow 1.3 or later. The caveats described for groups in the previous section also apply to meters.

add−meter *switch meter*
Add a meter entry to *switch*'s tables. The *meter* syntax is described in section **Meter Syntax**, below.

mod−meter *switch meter*
Modify an existing meter.

del−meters *switch*
del−meter *switch* [*meter*]
Delete entries from *switch*'s meter table. *meter* can specify a single meter with syntax **meter=***id*, or all meters with syntax **meter=all**.

dump−meters *switch*
dump−meter *switch* [*meter*]
Print meter configuration. *meter* can specify a single meter with syntax **meter=***id*, or all meters with syntax **meter=all**.

meter−stats *switch* [*meter*]
Print meter statistics. *meter* can specify a single meter with syntax **meter=***id*, or all meters with syntax **meter=all**.

meter−features *switch*
Print meter features.

OpenFlow Switch Flow Table Commands

These commands manage the flow table in an OpenFlow switch. In each case, *flow* specifies a flow entry in the format described in **Flow Syntax**, below, *file* is a text file that contains zero or more flows in the same syntax, one per line, and the optional **−−bundle** option operates the command as a single atomic transation, see option **−−bundle**, below.

[**−−bundle**] **add−flow** *switch flow*
[**−−bundle**] **add−flow** *switch* **−** < *file*
[**−−bundle**] **add−flows** *switch file*
Add each flow entry to *switch*'s tables. Each flow specification (e.g., each line in *file*) may start with **add**, **modify**, **delete**, **modify_strict**, or **delete_strict** keyword to specify whether a flow is to be added, modified, or deleted, and whether the modify or delete is strict or not. For backwards compatibility a flow specification without one of these keywords is treated as a flow add. All flow mods are executed in the order specified.

[**−−bundle**] [**−−strict**] **mod−flows** *switch flow*
[**−−bundle**] [**−−strict**] **mod−flows** *switch* **−** < *file*
Modify the actions in entries from *switch*'s tables that match the specified flows. With **−−strict**, wildcards are not treated as active for matching purposes.

[**−−bundle**] **del−flows** *switch*
[**−−bundle**] [**−−strict**] **del−flows** *switch* [*flow*]
[**−−bundle**] [**−−strict**] **del−flows** *switch* **−** < *file*
Deletes entries from *switch*'s flow table. With only a *switch* argument, deletes all flows. Otherwise, deletes flow entries that match the specified flows. With **−−strict**, wildcards are not treated as active for matching purposes.

[**−−bundle**] [**−−readd**] **replace−flows** *switch file*
Reads flow entries from *file* (or **stdin** if *file* is **−**) and queries the flow table from *switch*. Then it fixes up any differences, adding flows from *flow* that are missing on *switch*, deleting flows from *switch* that are not in *file*, and updating flows in *switch* whose actions, cookie, or timeouts differ in *file*.

With **−−readd**, **ovs−ofctl** adds all the flows from *file*, even those that exist with the same actions, cookie, and timeout in *switch*. This resets all the flow packet and byte counters to 0, which can be useful for debugging.

diff−flows *source1 source2*

Reads flow entries from *source1* and *source2* and prints the differences. A flow that is in *source1* but not in *source2* is printed preceded by a −, and a flow that is in *source2* but not in *source1* is printed preceded by a +. If a flow exists in both *source1* and *source2* with different actions, cookie, or timeouts, then both versions are printed preceded by − and +, respectively.

source1 and *source2* may each name a file or a switch. If a name begins with **/** or **.**, then it is considered to be a file name. A name that contains **:** is considered to be a switch. Otherwise, it is a file if a file by that name exists, a switch if not.

For this command, an exit status of 0 means that no differences were found, 1 means that an error occurred, and 2 means that some differences were found.

packet−out *switch packet-out*

Connects to *switch* and instructs it to execute the *packet-out* OpenFlow message, specified as defined in **Packet−Out Syntax** section.

OpenFlow Switch Group Table Commands

These commands manage the group table in an OpenFlow switch. In each case, *group* specifies a group entry in the format described in **Group Syntax**, below, and *file* is a text file that contains zero or more groups in the same syntax, one per line, and the optional **−−bundle** option operates the command as a single atomic transation, see option **−−bundle**, below.

[**−−bundle**] **add−group** *switch group*
[**−−bundle**] **add−group** *switch* **−** < *file*
[**−−bundle**] **add−groups** *switch file*

Add each group entry to *switch*'s tables. Each group specification (e.g., each line in *file*) may start with **add**, **modify**, **add_or_mod**, **delete**, **insert_bucket**, or **remove_bucket** keyword to specify whether a flow is to be added, modified, or deleted, or whether a group bucket is to be added or removed. For backwards compatibility a group specification without one of these keywords is treated as a group add. All group mods are executed in the order specified.

[**−−bundle**] [**−−may−create**] **mod−group** *switch group*
[**−−bundle**] [**−−may−create**] **mod−group** *switch* **−** < *file*

Modify the action buckets in entries from *switch*'s tables for each group entry. If a specified group does not already exist, then without **−−may−create**, this command has no effect; with **−−may−create**, it creates a new group. The **−−may−create** option uses an Open vSwitch extension to OpenFlow only implemented in Open vSwitch 2.6 and later.

[**−−bundle**] **del−groups** *switch*
[**−−bundle**] **del−groups** *switch* [*group*]
[**−−bundle**] **del−groups** *switch* **−** < *file*

Deletes entries from *switch*'s group table. With only a *switch* argument, deletes all groups. Otherwise, deletes the group for each group entry.

[**−−bundle**] **insert−buckets** *switch group*
[**−−bundle**] **insert−buckets** *switch* **−** < *file*

Add buckets to an existing group present in the *switch*'s group table. If no *command_bucket_id* is present in the group specification then all buckets of the group are removed.

[**−−bundle**] **remove−buckets** *switch group*
[**−−bundle**] **remove−buckets** *switch* **−** < *file*

Remove buckets to an existing group present in the *switch*'s group table. If no *command_bucket_id* is present in the group specification then all buckets of the group are removed.

OpenFlow Switch Bundle Command

Transactional updates to both flow and group tables can be made with the **bundle** command. *file* is a text file that contains zero or more flow mods, group mods, or packet-outs in **Flow Syntax**, **Group Syntax**, or **Packet−Out Syntax**, each line preceded by **flow**, **group**, or **packet−out** keyword, correspondingly. The

flow keyword may be optionally followed by one of the keywords **add**, **modify**, **modify_strict**, **delete**, or **delete_strict**, of which the **add** is assumed if a bare **flow** is given. Similarly, the **group** keyword may be optionally followed by one of the keywords **add**, **modify**, **add_or_mod**, **delete**, **insert_bucket**, or **remove_bucket**, of which the **add** is assumed if a bare **group** is given.

bundle *switch file*
Execute all flow and group mods in *file* as a single atomic transaction against *switch*'s tables. All bundled mods are executed in the order specified.

OpenFlow Switch Tunnel TLV Table Commands

Open vSwitch maintains a mapping table between tunnel option TLVs (defined by <class, type, length>) and NXM fields **tun_metadata***n*, where *n* ranges from 0 to 63, that can be operated on for the purposes of matches, actions, etc. This TLV table can be used for Geneve option TLVs or other protocols with options in same TLV format as Geneve options. This mapping must be explicitly specified by the user through the following commands.

A TLV mapping is specified with the syntax **{class=***class***,type=***type***,len=***length***}->tun_metadata***n*. When an option mapping exists for a given **tun_metadata***n*, matching on the defined field becomes possible, e.g.:

```
ovs-ofctl add-tlv-map br0 "{class=0xffff,type=0,len=4}->tun_metadata0"
ovs-ofctl add-flow br0 tun_metadata0=1234,actions=controller
```

A mapping should not be changed while it is in active use by a flow. The result of doing so is undefined.

These commands are Nicira extensions to OpenFlow and require Open vSwitch 2.5 or later.

add−tlv−map *switch option*[**,***option*]...
Add each *option* to *switch*'s tables. Duplicate fields are rejected.

del−tlv−map *switch* [*option*[**,***option*]]...
Delete each *option* from *switch*'s table, or all option TLV mapping if no *option* is specified. Fields that aren't mapped are ignored.

dump−tlv−map *switch*
Show the currently mapped fields in the switch's option table as well as switch capabilities.

OpenFlow Switch Monitoring Commands

snoop *switch*
Connects to *switch* and prints to the console all OpenFlow messages received. Unlike other **ovs−ofctl** commands, if *switch* is the name of a bridge, then the **snoop** command connects to a Unix domain socket named **/usr/local/var/run/openvswitch/***switch***.snoop**. **ovs−vswitchd** listens on such a socket for each bridge and sends to it all of the OpenFlow messages sent to or received from its configured OpenFlow controller. Thus, this command can be used to view OpenFlow protocol activity between a switch and its controller.

When a switch has more than one controller configured, only the traffic to and from a single controller is output. If none of the controllers is configured as a master or a slave (using a Nicira extension to OpenFlow 1.0 or 1.1, or a standard request in OpenFlow 1.2 or later), then a controller is chosen arbitrarily among them. If there is a master controller, it is chosen; otherwise, if there are any controllers that are not masters or slaves, one is chosen arbitrarily; otherwise, a slave controller is chosen arbitrarily. This choice is made once at connection time and does not change as controllers reconfigure their roles.

If a switch has no controller configured, or if the configured controller is disconnected, no traffic is sent, so monitoring will not show any traffic.

monitor *switch* [*miss-len*] [**invalid_ttl**] [**watch:**[*spec*...]]

Connects to *switch* and prints to the console all OpenFlow messages received. Usually, *switch* should specify the name of a bridge in the **ovs−vswitchd** database.

If *miss-len* is provided, **ovs−ofctl** sends an OpenFlow ''set configuration'' message at connection setup time that requests *miss-len* bytes of each packet that misses the flow table. Open vSwitch does not send these and other asynchronous messages to an **ovs−ofctl monitor** client connection unless a nonzero value is specified on this argument. (Thus, if *miss−len* is not specified, very little traffic will ordinarily be printed.)

If **invalid_ttl** is passed, **ovs−ofctl** sends an OpenFlow ''set configuration'' message at connection setup time that requests **INVALID_TTL_TO_CONTROLLER**, so that **ovs−ofctl monitor** can receive ''packet-in'' messages when TTL reaches zero on **dec_ttl** action. Only OpenFlow 1.1 and 1.2 support **invalid_ttl**; Open vSwitch also implements it for OpenFlow 1.0 as an extension.

watch:[*spec*...] causes **ovs−ofctl** to send a ''monitor request'' Nicira extension message to the switch at connection setup time. This message causes the switch to send information about flow table changes as they occur. The following comma-separated *spec* syntax is available:

!initial Do not report the switch's initial flow table contents.

!add Do not report newly added flows.

!delete Do not report deleted flows.

!modify
Do not report modifications to existing flows.

!own Abbreviate changes made to the flow table by **ovs−ofctl**'s own connection to the switch. (These could only occur using the **ofctl/send** command described below under **RUN-TIME MANAGEMENT COMMANDS**.)

!actions
Do not report actions as part of flow updates.

table=*number*
Limits the monitoring to the table with the given *number* between 0 and 254. By default, all tables are monitored.

out_port=*port*
If set, only flows that output to *port* are monitored. The *port* may be an OpenFlow port number or keyword (e.g. **LOCAL**).

field=value
Monitors only flows that have *field* specified as the given *value*. Any syntax valid for matching on **dump−flows** may be used.

This command may be useful for debugging switch or controller implementations. With **watch:**, it is particularly useful for observing how a controller updates flow tables.

OpenFlow Switch and Controller Commands

The following commands, like those in the previous section, may be applied to OpenFlow switches, using any of the connection methods described in that section. Unlike those commands, these may also be applied to OpenFlow controllers.

probe *target*
Sends a single OpenFlow echo-request message to *target* and waits for the response. With the **−t** or **−−timeout** option, this command can test whether an OpenFlow switch or controller is up and running.

ping *target* [*n*]
Sends a series of 10 echo request packets to *target* and times each reply. The echo request packets consist of an OpenFlow header plus *n* bytes (default: 64) of randomly generated payload. This measures the latency of individual requests.

benchmark *target n count*

Sends *count* echo request packets that each consist of an OpenFlow header plus *n* bytes of payload and waits for each response. Reports the total time required. This is a measure of the maximum bandwidth to *target* for round-trips of *n*-byte messages.

Other Commands

ofp−parse *file*

Reads *file* (or **stdin** if *file* is −) as a series of OpenFlow messages in the binary format used on an OpenFlow connection, and prints them to the console. This can be useful for printing OpenFlow messages captured from a TCP stream.

ofp−parse−pcap *file* [*port*...]

Reads *file*, which must be in the PCAP format used by network capture tools such as **tcpdump** or **wireshark**, extracts all the TCP streams for OpenFlow connections, and prints the OpenFlow messages in those connections in human-readable format on **stdout**.

OpenFlow connections are distinguished by TCP port number. Non-OpenFlow packets are ignored. By default, data on TCP ports 6633 and 6653 are considered to be OpenFlow. Specify one or more *port* arguments to override the default.

This command cannot usefully print SSL encrypted traffic. It does not understand IPv6.

Flow Syntax

Some **ovs−ofctl** commands accept an argument that describes a flow or flows. Such flow descriptions comprise a series of *field=value* assignments, separated by commas or white space. (Embedding spaces into a flow description normally requires quoting to prevent the shell from breaking the description into multiple arguments.)

Flow descriptions should be in **normal form**. This means that a flow may only specify a value for an L3 field if it also specifies a particular L2 protocol, and that a flow may only specify an L4 field if it also specifies particular L2 and L3 protocol types. For example, if the L2 protocol type **dl_type** is wildcarded, then L3 fields **nw_src**, **nw_dst**, and **nw_proto** must also be wildcarded. Similarly, if **dl_type** or **nw_proto** (the L3 protocol type) is wildcarded, so must be the L4 fields **tcp_dst** and **tcp_src**. **ovs−ofctl** will warn about flows not in normal form.

ovs−fields(7) describes the supported fields and how to match them. In addition to match fields, commands that operate on flows accept a few additional key-value pairs:

table=*number*

For flow dump commands, limits the flows dumped to those in the table with the given *number* between 0 and 254. If not specified (or if 255 is specified as *number*), then flows in all tables are dumped.

For flow table modification commands, behavior varies based on the OpenFlow version used to connect to the switch:

OpenFlow 1.0

OpenFlow 1.0 does not support **table** for modifying flows. **ovs−ofctl** will exit with an error if **table** (other than **table=255**) is specified for a switch that only supports OpenFlow 1.0.

In OpenFlow 1.0, the switch chooses the table into which to insert a new flow. The Open vSwitch software switch always chooses table 0. Other Open vSwitch datapaths and other OpenFlow implementations may choose different tables.

The OpenFlow 1.0 behavior in Open vSwitch for modifying or removing flows depends on whether **−−strict** is used. Without **−−strict**, the command applies to matching flows in all tables. With **−−strict**, the command will operate on any single matching flow in any table; it will do nothing if there are matches in more than one table. (The distinction between these behaviors only matters if non-OpenFlow 1.0 commands were also used, because OpenFlow 1.0 alone cannot add flows with the same matching criteria to multiple tables.)

OpenFlow 1.0 with table_id extension

Open vSwitch implements an OpenFlow extension that allows the controller to specify the table on which to operate. **ovs−ofctl** automatically enables the extension when **table** is specified and OpenFlow 1.0 is used. **ovs−ofctl** automatically detects whether the switch supports the extension. As of this writing, this extension is only known to be implemented by Open vSwitch.

With this extension, **ovs−ofctl** operates on the requested table when **table** is specified, and acts as described for OpenFlow 1.0 above when no **table** is specified (or for **table=255**).

OpenFlow 1.1

OpenFlow 1.1 requires flow table modification commands to specify a table. When **table** is not specified (or **table=255** is specified), **ovs−ofctl** defaults to table 0.

OpenFlow 1.2 and later

OpenFlow 1.2 and later allow flow deletion commands, but not other flow table modification commands, to operate on all flow tables, with the behavior described above for OpenFlow 1.0.

duration=...
n_packet=...
n_bytes=...

ovs−ofctl ignores assignments to these ''fields'' to allow output from the **dump−flows** command to be used as input for other commands that parse flows.

The **add−flow**, **add−flows**, and **mod−flows** commands require an additional field, which must be the final field specified:

actions=[*action*][**,***action*...]

Specifies a comma-separated list of actions to take on a packet when the flow entry matches. If no *action* is specified, then packets matching the flow are dropped. The following forms of *action* are supported:

port
output:*port*

Outputs the packet to OpenFlow port number *port*. If *port* is the packet's input port, the packet is not output.

output:*src***[***start***..***end***]**

Outputs the packet to the OpenFlow port number read from *src*, which may be an NXM field name, as described above, or a match field name. **output:reg0[16..31]** outputs to the OpenFlow port number written in the upper half of register 0. If the port number is the packet's input port, the packet is not output.

This form of **output** was added in Open vSwitch 1.3.0. This form of **output** uses an OpenFlow extension that is not supported by standard OpenFlow switches.

output(port=*port***,max_len=***nbytes*)

Outputs the packet to the OpenFlow port number read from *port*, with maximum packet size set to *nbytes*. *port* may be OpenFlow port number, **local**, or **in_port**. Patch port is not supported. Packets larger than *nbytes* will be trimmed to *nbytes* while packets smaller than *nbytes* remains the original size.

group:*group_id*

Outputs the packet to the OpenFlow group *group_id*. OpenFlow 1.1 introduced support for groups; Open vSwitch 2.6 and later also supports output to groups as an extension to OpenFlow 1.0. See **Group Syntax** for more details.

normal

Subjects the packet to the device's normal L2/L3 processing. (This action is not implemented by all OpenFlow switches.)

flood Outputs the packet on all switch physical ports other than the port on which it was received and any ports on which flooding is disabled (typically, these would be ports disabled by the IEEE 802.1D spanning tree protocol).

all Outputs the packet on all switch physical ports other than the port on which it was received.

local Outputs the packet on the ‘‘local port,’’ which corresponds to the network device that has the same name as the bridge.

in_port
Outputs the packet on the port from which it was received.

controller(*key***=***value***...)**
Sends the packet and its metadata to the OpenFlow controller as a ‘‘packet in’’ message. The supported key-value pairs are:

max_len=*nbytes*
Limit to *nbytes* the number of bytes of the packet to send to the controller. By default the entire packet is sent.

reason=*reason*
Specify *reason* as the reason for sending the message in the ‘‘packet in’’ message. The supported reasons are **action** (the default), **no_match**, and **invalid_ttl**.

id=*controller-id*
Specify *controller-id*, a 16-bit integer, as the connection ID of the OpenFlow controller or controllers to which the ‘‘packet in’’ message should be sent. The default is zero. Zero is also the default connection ID for each controller connection, and a given controller connection will only have a nonzero connection ID if its controller uses the **NXT_SET_CONTROLLER_ID** Nicira extension to OpenFlow.

userdata=*hh*...
Supplies the bytes represented as hex digits *hh* as additional data to the controller in the packet-in message. Pairs of hex digits may be separated by periods for readability.

pause Causes the switch to freeze the packet’s trip through Open vSwitch flow tables and serializes that state into the packet-in message as a ‘‘continuation,’’ an additional property in the **NXT_PACKET_IN2** message. The controller can later send the continuation back to the switch in an **NXT_RESUME** message, which will restart the packet’s traversal from the point where it was interrupted. This permits an OpenFlow controller to interpose on a packet midway through processing in Open vSwitch.

If any *reason* other than **action** or any nonzero *controller-id* is supplied, Open vSwitch extension **NXAST_CONTROLLER**, supported by Open vSwitch 1.6 and later, is used. If **userdata** is supplied, then **NXAST_CONTROLLER2**, supported by Open vSwitch 2.6 and later, is used.

controller
controller[**:***nbytes*]
Shorthand for **controller()** or **controller(max_len=***nbytes***)**, respectively.

enqueue(*port***,***queue***)**
Enqueues the packet on the specified *queue* within port *port*, which must be an OpenFlow port number or keyword (e.g. **LOCAL**). The number of supported queues depends on the switch; some OpenFlow implementations do not support queuing at all.

drop Discards the packet, so no further processing or forwarding takes place. If a drop action is used, no other actions may be specified.

mod_vlan_vid:*vlan_vid*
Modifies the VLAN id on a packet. The VLAN tag is added or modified as necessary to match the value specified. If the VLAN tag is added, a priority of zero is used (see the **mod_vlan_pcp** action to set this).

mod_vlan_pcp:*vlan_pcp*
Modifies the VLAN priority on a packet. The VLAN tag is added or modified as necessary to match the value specified. Valid values are between 0 (lowest) and 7 (highest). If the VLAN tag is added, a vid of zero is used (see the **mod_vlan_vid** action to set this).

strip_vlan
Strips the VLAN tag from a packet if it is present.

push_vlan:*ethertype*
Push a new VLAN tag onto the packet. Ethertype is used as the Ethertype for the tag. Only ethertype 0x8100 should be used. (0x88a8 which the spec allows isn't supported at the moment.) A priority of zero and the tag of zero are used for the new tag.

push_mpls:*ethertype*
Changes the packet's Ethertype to *ethertype*, which must be either **0x8847** or **0x8848**, and pushes an MPLS LSE.

If the packet does not already contain any MPLS labels then an initial label stack entry is pushed. The label stack entry's label is 2 if the packet contains IPv6 and 0 otherwise, its default traffic control value is the low 3 bits of the packet's DSCP value (0 if the packet is not IP), and its TTL is copied from the IP TTL (64 if the packet is not IP).

If the packet does already contain an MPLS label, pushes a new outermost label as a copy of the existing outermost label.

A limitation of the implementation is that processing of actions will stop if **push_mpls** follows another **push_mpls** unless there is a **pop_mpls** in between.

pop_mpls:*ethertype*
Strips the outermost MPLS label stack entry. Currently the implementation restricts *ethertype* to a non-MPLS Ethertype and thus **pop_mpls** should only be applied to packets with an MPLS label stack depth of one. A further limitation is that processing of actions will stop if **pop_mpls** follows another **pop_mpls** unless there is a **push_mpls** in between.

mod_dl_src:*mac*
Sets the source Ethernet address to *mac*.

mod_dl_dst:*mac*
Sets the destination Ethernet address to *mac*.

mod_nw_src:*ip*
Sets the IPv4 source address to *ip*.

mod_nw_dst:*ip*
Sets the IPv4 destination address to *ip*.

mod_tp_src:*port*
Sets the TCP or UDP or SCTP source port to *port*.

mod_tp_dst:*port*
Sets the TCP or UDP or SCTP destination port to *port*.

mod_nw_tos:*tos*
Sets the DSCP bits in the IPv4 ToS/DSCP or IPv6 traffic class field to *tos*, which must be a multiple of 4 between 0 and 255. This action does not modify the two least significant bits of the ToS field (the ECN bits).

mod_nw_ecn:*ecn*
Sets the ECN bits in the IPv4 ToS or IPv6 traffic class field to *ecn*, which must be a value between 0 and 3, inclusive. This action does not modify the six most significant bits of the field (the DSCP bits).

Requires OpenFlow 1.1 or later.

mod_nw_ttl:*ttl*
Sets the IPv4 TTL or IPv6 hop limit field to *ttl*, which is specified as a decimal number between 0 and 255, inclusive. Switch behavior when setting *ttl* to zero is not well specified, though.

Requires OpenFlow 1.1 or later.

The following actions are Nicira vendor extensions that, as of this writing, are only known to be implemented by Open vSwitch:

resubmit:*port*
resubmit([*port*]**,**[*table*]**)**
resubmit([*port*]**,**[*table*]**,ct)**
Re-searches this OpenFlow flow table (or the table whose number is specified by *table*) with the **in_port** field replaced by *port* (if *port* is specified) and the packet 5-tuple fields swapped with the corresponding conntrack original direction tuple fields (if **ct** is specified, see **ct_nw_src** above), and executes the actions found, if any, in addition to any other actions in this flow entry. The **in_port** and swapped 5-tuple fields are restored immediately after the search, before any actions are executed.

The **ct** option requires a valid connection tracking state as a match prerequisite in the flow where this action is placed. Examples of valid connection tracking state matches include **ct_state=+new**, **ct_state=+est**, **ct_state=+rel**, and **ct_state=+trk-inv**.

Recursive **resubmit** actions are obeyed up to implementation-defined limits:

- Open vSwitch 1.0.1 and earlier did not support recursion.
- Open vSwitch 1.0.2 and 1.0.3 limited recursion to 8 levels.
- Open vSwitch 1.1 and 1.2 limited recursion to 16 levels.
- Open vSwitch 1.2 through 1.8 limited recursion to 32 levels.
- Open vSwitch 1.9 through 2.0 limited recursion to 64 levels.
- Open vSwitch 2.1 through 2.5 limited recursion to 64 levels and impose a total limit of 4,096 resubmits per flow translation (earlier versions did not impose any total limit).
- Open vSwitch 2.6 and later imposes the same limits as 2.5, with one exception: **resubmit** from table x to any table $y > x$ does not count against the recursion limit.

Open vSwitch before 1.2.90 did not support *table*. Open vSwitch before 2.7 did not support **ct**.

set_tunnel:*id*
set_tunnel64:*id*
If outputting to a port that encapsulates the packet in a tunnel and supports an identifier (such as GRE), sets the identifier to *id*. If the **set_tunnel** form is used and *id* fits in 32 bits, then this uses an action extension that is supported by Open vSwitch 1.0 and later. Otherwise, if *id* is a 64-bit value, it requires Open vSwitch 1.1 or later.

set_queue:*queue*
Sets the queue that should be used to *queue* when packets are output. The number of supported queues depends on the switch; some OpenFlow implementations do not support queuing at all.

pop_queue
Restores the queue to the value it was before any **set_queue** actions were applied.

ct
ct([*argument*][**,***argument*...]**)**
Send the packet through the connection tracker. Refer to the **ct_state** documentation above for possible packet and connection states. The following arguments are supported:

commit
Commit the connection to the connection tracking module. Information about the connection will be stored beyond the lifetime of the packet in the pipeline. Some **ct_state** flags are only available for committed connections.

force
A committed connection always has the directionality of the packet that caused the connection to be committed in the first place. This is the ''original direction'' of the connection, and the opposite direction is the ''reply direction''. If a connection is already committed, but it is in the wrong direction, **force** flag may be used in addition to **commit** flag to effectively terminate the existing connection and start a new one in the current direction. This flag has no effect if the original direction of the connection is already the same as that of the current packet.

table=*number*
Fork pipeline processing in two. The original instance of the packet will continue processing the current actions list as an untracked packet. An additional instance of the packet will be sent to the connection tracker, which will be re-injected into the OpenFlow pipeline to resume processing in table *number*, with the **ct_state** and other ct match fields set. If the **table** is not specified, then the packet which is submitted to the connection tracker is not re-injected into the OpenFlow pipeline. It is strongly recommended to specify a table later than the current table to prevent loops.

zone=*value*
zone=*src***[***start***..***end***]**
A 16-bit context id that can be used to isolate connections into separate domains, allowing overlapping network addresses in different zones. If a zone is not provided, then the default is to use zone zero. The **zone** may be specified either as an immediate 16-bit *value*, or may be provided from an NXM field *src*. The *start* and *end* pair are inclusive, and must specify a 16-bit range within the field. This value is copied to the **ct_zone** match field for packets which are re-injected into the pipeline using the **table** option.

exec([*action*][**,***action*...]**)**
Perform actions within the context of connection tracking. This is a restricted set of actions which are in the same format as their specifications as part of a flow. Only actions which modify the **ct_mark** or **ct_label** fields are accepted within the **exec** action, and these fields may only be modified with this option. For example:

set_field:*value*[/*mask*]->ct_mark
Store a 32-bit metadata value with the connection. Subsequent lookups for packets in this connection will populate the **ct_mark** flow field when the packet is sent to the connection tracker with the **table** specified.

set_field:*value*[/*mask*]->ct_label
Store a 128-bit metadata value with the connection. Subsequent lookups for packets in this connection will populate the **ct_label** flow field when the packet is sent to the connection tracker with the **table** specified.

The **commit** parameter must be specified to use **exec(...)**.

alg=*alg*
Specify application layer gateway *alg* to track specific connection types. If subsequent related connections are sent through the **ct** action, then the **rel** flag in the **ct_state** field will be set. Supported types include:

ftp Look for negotiation of FTP data connections. Specify this option for FTP control connections to detect related data connections and populate the **rel** flag for the data connections.

tftp Look for negotiation of TFTP data connections. Specify this option for TFTP control connections to detect related data connections and populate the **rel** flag for the data connections.

The **commit** parameter must be specified to use **alg=***alg*.

When committing related connections, the **ct_mark** for that connection is inherited from the current **ct_mark** stored with the original connection (ie, the connection created by **ct(alg=...)**).

Note that with the Linux datapath, global sysctl options affect the usage of the **ct** action. In particular, if **net.netfilter.nf_conntrack_helper** is enabled then application layer gateway helpers may be executed even if the **alg** option is not specified. This is the default setting until Linux 4.7. For security reasons, the netfilter team recommends users to disable this option. See this blog post for further details: http://www.netfilter.org/news.html#2012-04-03

nat[**(**(**src**|**dst**)=*addr1*[-*addr2*][**:***port1*[-*port2*]][**,***flags*]**)**]
Specify address and port translation for the connection being tracked. For new connections either **src** or **dst** argument must be provided to set up either source address/port translation (SNAT) or destination address/port translation (DNAT), respectively. Setting up address translation for a new connection takes effect only if the **commit** flag is also provided for the enclosing **ct** action. A bare **nat** action will only translate the packet being processed in the way the connection has been set up with an earlier **ct** action. Also a **nat** action with **src** or **dst**, when applied to a packet belonging to an established (rather than new) connection, will behave the same as a bare **nat**.

src and **dst** options take the following arguments:

addr1[-*addr2*]
The address range from which the translated address should be selected. If only one address is given, then that address will always be selected, otherwise the address selection can be informed by the optional **persistent** flag as described below. Either IPv4 or IPv6 addresses can be provided, but both addresses must be of the same type, and the datapath behavior is undefined in case of providing IPv4 address range for an IPv6 packet, or IPv6 address range for an IPv4 packet. IPv6 addresses must be bracketed with '[' and ']' if a port range is also given.

port1[-*port2*]
The port range from which the translated port should be selected. If only one port number is provided, then that should be selected. In case

of a mapping conflict the datapath may choose any other non-conflicting port number instead, even when no port range is specified. The port number selection can be informed by the optional **random** and **hash** flags as described below.

The optional flags are:

random
The selection of the port from the given range should be done using a fresh random number. This flag is mutually exclusive with **hash**.

hash The selection of the port from the given range should be done using a datapath specific hash of the packet's IP addresses and the other, non-mapped port number. This flag is mutually exclusive with **random**.

persistent
The selection of the IP address from the given range should be done so that the same mapping can be provided after the system restarts.

If an **alg** is specified for the committing **ct** action that also includes **nat** with a **src** or **dst** attribute, then the datapath tries to set up the helper to be NAT aware. This functionality is datapath specific and may not be supported by all datapaths.

nat was introduced in Open vSwitch 2.6. The first datapath that implements **ct nat** support is the one that ships with Linux 4.6.

The **ct** action may be used as a primitive to construct stateful firewalls by selectively committing some traffic, then matching the **ct_state** to allow established connections while denying new connections. The following flows provide an example of how to implement a simple firewall that allows new connections from port 1 to port 2, and only allows established connections to send traffic from port 2 to port 1:

table=0,priority=1,action=drop
table=0,priority=10,arp,action=normal
table=0,priority=100,ip,ct_state=-trk,action=ct(table=1)
table=1,in_port=1,ip,ct_state=+trk+new,action=ct(commit),2
table=1,in_port=1,ip,ct_state=+trk+est,action=2
table=1,in_port=2,ip,ct_state=+trk+new,action=drop
table=1,in_port=2,ip,ct_state=+trk+est,action=1

If **ct** is executed on IP (or IPv6) fragments, then the message is implicitly reassembled before sending to the connection tracker and refragmented upon **output**, to the original maximum received fragment size. Reassembly occurs within the context of the **zone**, meaning that IP fragments in different zones are not assembled together. Pipeline processing for the initial fragments is halted; When the final fragment is received, the message is assembled and pipeline processing will continue for that flow. Because packet ordering is not guaranteed by IP protocols, it is not possible to determine which IP fragment will cause message reassembly (and therefore continue pipeline processing). As such, it is strongly recommended that multiple flows should not execute **ct** to reassemble fragments from the same IP message.

Currently, connection tracking is only available on Linux kernels with the nf_conntrack module loaded. The **ct** action was introduced in Open vSwitch 2.5.

ct_clear
Clears connection tracking state from the flow, zeroing **ct_state**, **ct_zone**, **ct_mark**, and **ct_label**.

This action was introduced in Open vSwitch 2.6.90.

dec_ttl

dec_ttl(*id1*[**,***id2*]...**)**

Decrement TTL of IPv4 packet or hop limit of IPv6 packet. If the TTL or hop limit is initially zero or decrementing would make it so, no decrement occurs, as packets reaching TTL zero must be rejected. Instead, a ‘‘packet-in’’ message with reason code **OFPR_INVALID_TTL** is sent to each connected controller that has enabled receiving them, if any. Processing the current set of actions then stops. However, if the current set of actions was reached through ‘‘resubmit’’ then remaining actions in outer levels resume processing.

This action also optionally supports the ability to specify a list of valid controller ids. Each of the controllers in the list will receive the ‘‘packet_in’’ message only if they have registered to receive the invalid ttl packets. If controller ids are not specified, the ‘‘packet_in’’ message will be sent only to the controllers having controller id zero which have registered for the invalid ttl packets.

set_mpls_label:*label*

Set the label of the outer MPLS label stack entry of a packet. *label* should be a 20-bit value that is decimal by default; use a **0x** prefix to specify them in hexadecimal.

set_mpls_tc:*tc*

Set the traffic-class of the outer MPLS label stack entry of a packet. *tc* should be a in the range 0 to 7 inclusive.

set_mpls_ttl:*ttl*

Set the TTL of the outer MPLS label stack entry of a packet. *ttl* should be in the range 0 to 255 inclusive.

dec_mpls_ttl

Decrement TTL of the outer MPLS label stack entry of a packet. If the TTL is initially zero or decrementing would make it so, no decrement occurs. Instead, a ‘‘packet-in’’ message with reason code **OFPR_INVALID_TTL** is sent to the main controller (id zero), if it has enabled receiving them. Processing the current set of actions then stops. However, if the current set of actions was reached through ‘‘resubmit’’ then remaining actions in outer levels resume processing.

note:[*hh*]...

Does nothing at all. Any number of bytes represented as hex digits *hh* may be included. Pairs of hex digits may be separated by periods for readability. The **note** action’s format doesn’t include an exact length for its payload, so the provided bytes will be padded on the right by enough bytes with value 0 to make the total number 6 more than a multiple of 8.

move:*src***[***start***..***end***]−>***dst***[***start***..***end***]**

Copies the named bits from field *src* to field *dst*. *src* and *dst* may be NXM field names as defined in **nicira−ext.h**, e.g. **NXM_OF_UDP_SRC** or **NXM_NX_REG0**, or a match field name, e.g. **reg0**. Each *start* and *end* pair, which are inclusive, must specify the same number of bits and must fit within its respective field. Shorthands for **[***start***..***end***]** exist: use **[***bit***]** to specify a single bit or **[]** to specify an entire field (in the latter case the brackets can also be left off).

Examples: **move:NXM_NX_REG0[0..5]−>NXM_NX_REG1[26..31]** copies the six bits numbered 0 through 5, inclusive, in register 0 into bits 26 through 31, inclusive; **move:reg0[0..15]−>vlan_tci** copies the least significant 16 bits of register 0 into the VLAN TCI field.

In OpenFlow 1.0 through 1.4, **move** ordinarily uses an Open vSwitch extension to OpenFlow. In OpenFlow 1.5, **move** uses the OpenFlow 1.5 standard **copy_field** action. The ONF has also made **copy_field** available as an extension to OpenFlow 1.3. Open vSwitch 2.4 and later understands this extension and uses it if a controller uses it, but for backward compatibility with older versions of Open vSwitch, **ovs−ofctl** does not use it.

set_field:*value*[/*mask*]**−>***dst*
load:*value***−>***dst***[***start***..***end***]**
Loads a literal value into a field or part of a field. With **set_field**, **value** and the optional **mask** are given in the customary syntax for field *dst*, which is expressed as a field name. For example, **set_field:00:11:22:33:44:55->eth_src** sets the Ethernet source address to 00:11:22:33:44:55. With **load**, *value* must be an integer value (in decimal or prefixed by **0x** for hexadecimal) and *dst* can also be the NXM or OXM name for the field. For example, **load:0x001122334455->OXM_OF_ETH_SRC[]** has the same effect as the prior **set_field** example.

The two forms exist for historical reasons. Open vSwitch 1.1 introduced **NXAST_REG_LOAD** as a Nicira extension to OpenFlow 1.0 and used **load** to express it. Later, OpenFlow 1.2 introduced a standard **OFPAT_SET_FIELD** action that was restricted to loading entire fields, so Open vSwitch added the form **set_field** with this restriction. OpenFlow 1.5 extended **OFPAT_SET_FIELD** to the point that it became a superset of **NXAST_REG_LOAD**. Open vSwitch translates either syntax as necessary for the OpenFlow version in use: in OpenFlow 1.0 and 1.1, **NXAST_REG_LOAD**; in OpenFlow 1.2, 1.3, and 1.4, **NXAST_REG_LOAD** for **load** or for loading a subfield, **OFPAT_SET_FIELD** otherwise; and OpenFlow 1.5 and later, **OFPAT_SET_FIELD**.

push:*src***[***start***..***end***]**
Pushes *start* to *end* bits inclusive, in fields on top of the stack.

Example: **push:NXM_NX_REG2[0..5]** or **push:reg2[0..5]** push the value stored in register 2 bits 0 through 5, inclusive, on to the internal stack.

pop:*dst***[***start***..***end***]**
Pops from the top of the stack, retrieves the *start* to *end* bits inclusive, from the value popped and store them into the corresponding bits in *dst*.

Example: **pop:NXM_NX_REG2[0..5]** or **pop:reg2[0..5]** pops the value from top of the stack. Set register 2 bits 0 through 5, inclusive, based on bits 0 through 5 from the value just popped.

multipath(*fields***,** *basis***,** *algorithm***,** *n_links***,** *arg***,** *dst***[***start***..***end***])**
Hashes *fields* using *basis* as a universal hash parameter, then the applies multipath link selection *algorithm* (with parameter *arg*) to choose one of *n_links* output links numbered 0 through *n_links* minus 1, and stores the link into *dst***[***start***..***end***]**, which must be an NXM field as described above.

fields must be one of the following:

eth_src Hashes Ethernet source address only.

symmetric_l4
Hashes Ethernet source, destination, and type, VLAN ID, IPv4/IPv6 source, destination, and protocol, and TCP or SCTP (but not UDP) ports. The hash is computed so that pairs of corresponding flows in each direction hash to the same value, in environments where L2 paths are the same in each direction. UDP ports are not included in the hash to support protocols such as VXLAN that use asymmetric ports in each direction.

symmetric_l3l4
Hashes IPv4/IPv6 source, destination, and protocol, and TCP or SCTP (but not UDP) ports. Like **symmetric_l4**, this is a symmetric hash, but by excluding L2 headers it is more effective in environments with asymmetric L2 paths (e.g. paths involving VRRP IP addresses on a router). Not an effective hash function for protocols other than IPv4 and IPv6, which hash to a constant zero.

symmetric_l3l4+udp
Like **symmetric_l3l4+udp**, but UDP ports are included in the hash. This is a more effective hash when asymmetric UDP protocols such as VXLAN are not a consideration.

algorithm must be one of **modulo_n**, **hash_threshold**, **hrw**, and **iter_hash**. Only the **iter_hash** algorithm uses *arg*.

Refer to **nicira−ext.h** for more details.

bundle(*fields***,** *basis***,** *algorithm***,** *slave_type***, slaves:[***s1***,** *s2***, ...])**
Hashes *fields* using *basis* as a universal hash parameter, then applies the bundle link selection *algorithm* to choose one of the listed slaves represented as *slave_type*. Currently the only supported *slave_type* is **ofport**. Thus, each *s1* through *sN* should be an OpenFlow port number. Outputs to the selected slave.

Currently, *fields* must be either **eth_src**, **symmetric_l4**, **symmetric_l3l4**, or **symmetric_l3l4+udp**, and *algorithm* must be one of **hrw** and **active_backup**.

Example: **bundle(eth_src,0,hrw,ofport,slaves:4,8)** uses an Ethernet source hash with basis 0, to select between OpenFlow ports 4 and 8 using the Highest Random Weight algorithm.

Refer to **nicira−ext.h** for more details.

bundle_load(*fields***,** *basis***,** *algorithm***,** *slave_type***,** *dst***[***start***..***end***], slaves:[***s1***,** *s2***, ...])**
Has the same behavior as the **bundle** action, with one exception. Instead of outputting to the selected slave, it writes its selection to *dst***[***start***..***end***]**, which must be an NXM field as described above.

Example: **bundle_load(eth_src, 0, hrw, ofport, NXM_NX_REG0[], slaves:4, 8)** uses an Ethernet source hash with basis 0, to select between OpenFlow ports 4 and 8 using the Highest Random Weight algorithm, and writes the selection to **NXM_NX_REG0[]**. Also the match field name can be used, for example, instead of 'NXM_NX_REG0' the name 'reg0' can be used. When the while field is indicated the empty brackets can also be left off.

Refer to **nicira−ext.h** for more details.

learn(*argument***[,***argument***]...)**
This action adds or modifies a flow in an OpenFlow table, similar to **ovs−ofctl −−strict mod−flows**. The arguments specify the flow's match fields, actions, and other properties, as follows. At least one match criterion and one action argument should ordinarily be specified.

idle_timeout=*seconds*
hard_timeout=*seconds*
priority=*value*
cookie=*value*
send_flow_rem
These arguments have the same meaning as in the usual **ovs−ofctl** flow syntax.

fin_idle_timeout=*seconds*
fin_hard_timeout=*seconds*
Adds a **fin_timeout** action with the specified arguments to the new flow. This feature was added in Open vSwitch 1.5.90.

table=*number*
The table in which the new flow should be inserted. Specify a decimal number between 0 and 254. The default, if **table** is unspecified, is table 1.

delete_learned
This flag enables deletion of the learned flows when the flow with the **learn** action is removed. Specifically, when the last **learn** action with this flag and particular **table** and **cookie** values is removed, the switch deletes all of the flows in the specified table with the specified cookie.

This flag was added in Open vSwitch 2.4.

limit=*number*
If the number of flows in table **table** with cookie id **cookie** exceeds *number*, a new flow will not be learned by this action. By default there's no limit.

This flag was added in Open vSwitch 2.8.

result_dst=*field***[***bit***]**
If learning failed (because the number of flows exceeds **limit**), the action sets *field***[***bit***]** to 0, otherwise it will be set to 1. *field***[***bit***]** must be a single bit.

This flag was added in Open vSwitch 2.8.

*field***=***value*
*field***[***start***..***end***]=***src***[***start***..***end***]**
*field***[***start***..***end***]**
Adds a match criterion to the new flow.

The first form specifies that *field* must match the literal *value*, e.g. **dl_type=0x0800**. All of the fields and values for **ovs−ofctl** flow syntax are available with their usual meanings. Shorthand notation matchers (e.g. **ip** in place of **dl_type=0x0800**) are not currently implemented.

The second form specifies that *field***[***start***..***end***]** in the new flow must match *src***[***start***..***end***]** taken from the flow currently being processed. For example, *NXM_OF_UDP_DST***[]**=*NXM_OF_UDP_SRC***[]** on a TCP packet for which the UDP src port is **53**, creates a flow which matches *NXM_OF_UDP_DST***[]**=**53**.

The third form is a shorthand for the second form. It specifies that *field***[***start***..***end***]** in the new flow must match the same *field***[***start***..***end***]** taken from the flow currently being processed. For example, *NXM_OF_TCP_DST***[]** on a TCP packet for which the TCP dst port is **80**, creates a flow which matches *NXM_OF_TCP_DST***[]**=**80**.

load:*value***−>***dst***[***start***..***end***]**
load:*src***[***start***..***end***]−>***dst***[***start***..***end***]**
Adds a **load** action to the new flow.

The first form loads the literal *value* into bits *start* through *end*, inclusive, in field *dst*. Its syntax is the same as the **load** action described earlier in this section.

The second form loads *src***[***start***..***end***]**, a value from the flow currently being processed, into bits *start* through *end*, inclusive, in field *dst*.

output:*field***[***start***..***end***]**
Add an **output** action to the new flow's actions, that outputs to the OpenFlow port taken from *field***[***start***..***end***]**, which must be an NXM field as described above.

For best performance, segregate learned flows into a table (using **table=***number*) that is not used for any other flows except possibly for a lowest-priority ''catch-all'' flow, that is, a flow with no match criteria. (This is why the default **table** is 1, to keep the learned flows separate from the primary flow table 0.)

clear_actions
Clears all the actions in the action set immediately.

write_actions([*action*][**,***action*...]**)**
Add the specific actions to the action set. The syntax of *actions* is the same as in the **actions=** field. The action set is carried between flow tables and then executed at the end of the pipeline.

The actions in the action set are applied in the following order, as required by the OpenFlow specification, regardless of the order in which they were added to the action set. Except as specified otherwise below, the action set only holds at most a single action of each type. When more than one action of a single type is written to the action set, the one written later replaces the earlier action:

1. **strip_vlan**
   pop_mpls
2. **push_mpls**
3. **push_vlan**
4. **dec_ttl**
   dec_mpls_ttl
5. **load**
   move
   mod_dl_dst
   mod_dl_src
   mod_nw_dst
   mod_nw_src
   mod_nw_tos
   mod_nw_ecn
   mod_nw_ttl
   mod_tp_dst
   mod_tp_src
   mod_vlan_pcp
   mod_vlan_vid
   set_field
   set_tunnel
   set_tunnel64
   The action set can contain any number of these actions, with cumulative effect. They will be applied in the order as added. That is, when multiple actions modify the same part of a field, the later modification takes effect, and when they modify different parts of a field (or different fields), then both modifications are applied.
6. **set_queue**
7. **group**
   output
   resubmit
   If more than one of these actions is present, then the one listed earliest above is executed and the others are ignored, regardless of the order in which they were added to the action set. (If none of these actions is present, the action set has no real effect, because the modified packet is not sent anywhere and thus the modifications are not visible.)

Only the actions listed above may be written to the action set.

write_metadata:*value*[/*mask*]
Updates the metadata field for the flow. If *mask* is omitted, the metadata field is set exactly to *value*; if *mask* is specified, then a 1-bit in *mask* indicates that the corresponding bit in the metadata field will be replaced with the corresponding bit from *value*. Both *value* and *mask* are 64-bit values that are decimal by default; use a **0x** prefix to specify them in hexadecimal.

meter:*meter_id*
Apply the *meter_id* before any other actions. If a meter band rate is exceeded, the packet may be dropped, or modified, depending on the meter band type. See the description of the **Meter Table Commands**, above, for more details.

goto_table:*table*
Indicates the next table in the process pipeline.

fin_timeout(*argument*[**,***argument*]**)**
This action changes the idle timeout or hard timeout, or both, of this OpenFlow rule when the rule matches a TCP packet with the FIN or RST flag. When such a packet is observed, the action reduces the rule's timeouts to those specified on the action. If the rule's existing timeout is already shorter than the one that the action specifies, then that timeout is unaffected.

argument takes the following forms:

idle_timeout=*seconds*
Causes the flow to expire after the given number of seconds of inactivity.

hard_timeout=*seconds*
Causes the flow to expire after the given number of seconds, regardless of activity. (*seconds* specifies time since the flow's creation, not since the receipt of the FIN or RST.)

This action was added in Open vSwitch 1.5.90.

sample(*argument*[**,***argument*]...**)**
Samples packets and sends one sample for every sampled packet.

argument takes the following forms:

probability=*packets*
The number of sampled packets out of 65535. Must be greater or equal to 1.

collector_set_id=*id*
The unsigned 32-bit integer identifier of the set of sample collectors to send sampled packets to. Defaults to 0.

obs_domain_id=*id*
When sending samples to IPFIX collectors, the unsigned 32-bit integer Observation Domain ID sent in every IPFIX flow record. Defaults to 0.

obs_point_id=*id*
When sending samples to IPFIX collectors, the unsigned 32-bit integer Observation Point ID sent in every IPFIX flow record. Defaults to 0.

sampling_port=*port*
Sample packets on *port*, which should be the ingress or egress port. This option, which was added in Open vSwitch 2.5.90, allows the IPFIX implementation to export egress tunnel information.

ingress
egress Specifies explicitly that the packet is being sampled on ingress to or egress from the switch. IPFIX reports sent by Open vSwitch before version 2.5.90 did not include a direction. From 2.5.90 until 2.6.90, IPFIX reports inferred a direction

from **sampling_port**: if it was the packet's output port, then the direction was reported as egress, otherwise as ingress. Open vSwitch 2.6.90 introduced these options, which allow the inferred direction to be overridden. This is particularly useful when the ingress (or egress) port is not a tunnel.

Refer to **ovs−vswitchd.conf.db**(5) for more details on configuring sample collector sets.

This action was added in Open vSwitch 1.10.90.

exit This action causes Open vSwitch to immediately halt execution of further actions. Those actions which have already been executed are unaffected. Any further actions, including those which may be in other tables, or different levels of the **resubmit** call stack, are ignored. Actions in the action set is still executed (specify **clear_actions** before **exit** to discard them).

conjunction(*id***,** *k*/*n***)**
This action allows for sophisticated ''conjunctive match'' flows. Refer to **CONJUNCTIVE MATCH FIELDS** in **ovs−fields**(7) for details.

The **conjunction** action and **conj_id** field were introduced in Open vSwitch 2.4.

clone([*action*][**,***action*...]**)**
Executes each nested *action*, saving much of the packet and pipeline state beforehand and then restoring it afterward. The state that is saved and restored includes all flow data and metadata (including, for example, **ct_state**), the stack accessed by **push** and **pop** actions, and the OpenFlow action set.

This action was added in Open vSwitch 2.6.90.

An opaque identifier called a cookie can be used as a handle to identify a set of flows:

cookie=*value*
A cookie can be associated with a flow using the **add−flow**, **add−flows**, and **mod−flows** commands. *value* can be any 64-bit number and need not be unique among flows. If this field is omitted, a default cookie value of 0 is used.

cookie=*value*/*mask*
When using NXM, the cookie can be used as a handle for querying, modifying, and deleting flows. *value* and *mask* may be supplied for the **del−flows**, **mod−flows**, **dump−flows**, and **dump−aggregate** commands to limit matching cookies. A 1-bit in *mask* indicates that the corresponding bit in *cookie* must match exactly, and a 0-bit wildcards that bit. A mask of −1 may be used to exactly match a cookie.

The **mod−flows** command can update the cookies of flows that match a cookie by specifying the *cookie* field twice (once with a mask for matching and once without to indicate the new value):

ovs−ofctl mod−flows br0 cookie=1,actions=normal
Change all flows' cookies to 1 and change their actions to **normal**.

ovs−ofctl mod−flows br0 cookie=1/−1,cookie=2,actions=normal
Update cookies with a value of 1 to 2 and change their actions to **normal**.

The ability to match on cookies was added in Open vSwitch 1.5.0.

The following additional field sets the priority for flows added by the **add−flow** and **add−flows** commands. For **mod−flows** and **del−flows** when **−−strict** is specified, priority must match along with the rest of the flow specification. For **mod-flows** without **−−strict**, priority is only significant if the command creates a new flow, that is, non-strict **mod−flows** does not match on priority and will not change the priority of existing flows. Other commands do not allow priority to be specified.

priority=*value*
The priority at which a wildcarded entry will match in comparison to others. *value* is a number between 0 and 65535, inclusive. A higher *value* will match before a lower one. An exact-match entry will always have priority over an entry containing wildcards, so it has an implicit priority

value of 65535. When adding a flow, if the field is not specified, the flow's priority will default to 32768.

OpenFlow leaves behavior undefined when two or more flows with the same priority can match a single packet. Some users expect ''sensible'' behavior, such as more specific flows taking precedence over less specific flows, but OpenFlow does not specify this and Open vSwitch does not implement it. Users should therefore take care to use priorities to ensure the behavior that they expect.

The **add−flow**, **add−flows**, and **mod−flows** commands support the following additional options. These options affect only new flows. Thus, for **add−flow** and **add−flows**, these options are always significant, but for **mod−flows** they are significant only if the command creates a new flow, that is, their values do not update or affect existing flows.

idle_timeout=*seconds*
Causes the flow to expire after the given number of seconds of inactivity. A value of 0 (the default) prevents a flow from expiring due to inactivity.

hard_timeout=*seconds*
Causes the flow to expire after the given number of seconds, regardless of activity. A value of 0 (the default) gives the flow no hard expiration deadline.

importance=*value*
Sets the importance of a flow. The flow entry eviction mechanism can use importance as a factor in deciding which flow to evict. A value of 0 (the default) makes the flow non-evictable on the basis of importance. Specify a value between 0 and 65535.

Only OpenFlow 1.4 and later support **importance**.

send_flow_rem
Marks the flow with a flag that causes the switch to generate a ''flow removed'' message and send it to interested controllers when the flow later expires or is removed.

check_overlap
Forces the switch to check that the flow match does not overlap that of any different flow with the same priority in the same table. (This check is expensive so it is best to avoid it.)

The **dump−flows**, **dump−aggregate**, **del−flow** and **del−flows** commands support these additional optional fields:

out_port=*port*
If set, a matching flow must include an output action to *port*, which must be an OpenFlow port number or name (e.g. **local**).

out_group=*port*
If set, a matching flow must include an **group** action naming *group*, which must be an OpenFlow group number. This field is supported in Open vSwitch 2.5 and later and requires OpenFlow 1.1 or later.

Table Entry Output

The **dump−tables** and **dump−aggregate** commands print information about the entries in a datapath's tables. Each line of output is a flow entry as described in **Flow Syntax**, above, plus some additional fields:

duration=*secs*
The time, in seconds, that the entry has been in the table. *secs* includes as much precision as the switch provides, possibly to nanosecond resolution.

n_packets
The number of packets that have matched the entry.

n_bytes
The total number of bytes from packets that have matched the entry.

The following additional fields are included only if the switch is Open vSwitch 1.6 or later and the NXM

flow format is used to dump the flow (see the description of the **−−flow-format** option below). The values of these additional fields are approximations only and in particular **idle_age** will sometimes become nonzero even for busy flows.

hard_age=*secs*
The integer number of seconds since the flow was added or modified. **hard_age** is displayed only if it differs from the integer part of **duration**. (This is separate from **duration** because **mod−flows** restarts the **hard_timeout** timer without zeroing **duration**.)

idle_age=*secs*
The integer number of seconds that have passed without any packets passing through the flow.

Packet−Out Syntax

ovs−ofctl bundle command accepts packet-outs to be specified in the bundle file. Each packet-out comprises of a series of *field=value* assignments, separated by commas or white space. (Embedding spaces into a packet-out description normally requires quoting to prevent the shell from breaking the description into multiple arguments.). Unless noted otherwise only the last instance of each field is honoured. This same syntax is also supported by the **ovs−ofctl packet-out** command.

in_port=*port*
The port number to be considered the in_port when processing actions. This can be any valid OpenFlow port number, or any of the **LOCAL**, **CONTROLLER**, or **NONE**. This field is required.

packet=*hex-string*
The actual packet to send, expressed as a string of hexadecimal bytes. This field is required.

actions=[*action*][**,***action*...]
The syntax of actions are identical to the **actions=** field described in **Flow Syntax** above. Specifying **actions=** is optional, but omitting actions is interpreted as a drop, so the packet will not be sent anywhere from the switch. **actions** must be specified at the end of each line, like for flow mods.

Group Syntax

Some **ovs−ofctl** commands accept an argument that describes a group or groups. Such flow descriptions comprise a series *field=value* assignments, separated by commas or white space. (Embedding spaces into a group description normally requires quoting to prevent the shell from breaking the description into multiple arguments.). Unless noted otherwise only the last instance of each field is honoured.

group_id=*id*
The integer group id of group. When this field is specified in **del−groups** or **dump−groups**, the keyword "all" may be used to designate all groups. This field is required.

type=*type*
The type of the group. The **add-group**, **add-groups** and **mod-groups** commands require this field. It is prohibited for other commands. The following keywords designated the allowed types:

all Execute all buckets in the group.

select Execute one bucket in the group. The switch should select the bucket in such a way that should implement equal load sharing is achieved. The switch may optionally select the bucket based on bucket weights.

indirect
Executes the one bucket in the group.

ff

fast_failover
Executes the first live bucket in the group which is associated with a live port or group.

command_bucket_id=*id*
The bucket to operate on. The **insert-buckets** and **remove-buckets** commands require this field. It is prohibited for other commands. *id* may be an integer or one of the following keywords:

all Operate on all buckets in the group. Only valid when used with the **remove-buckets** command in which case the effect is to remove all buckets from the group.

first Operate on the first bucket present in the group. In the case of the **insert-buckets** command the effect is to insert new bucets just before the first bucket already present in the group; or to replace the buckets of the group if there are no buckets already present in the group. In the case of the **remove-buckets** command the effect is to remove the first bucket of the group; or do nothing if there are no buckets present in the group.

last Operate on the last bucket present in the group. In the case of the **insert-buckets** command the effect is to insert new bucets just after the last bucket already present in the group; or to replace the buckets of the group if there are no buckets already present in the group. In the case of the **remove-buckets** command the effect is to remove the last bucket of the group; or do nothing if there are no buckets present in the group.

If *id* is an integer then it should correspond to the **bucket_id** of a bucket present in the group. In case of the **insert-buckets** command the effect is to insert buckets just before the bucket in the group whose **bucket_id** is *id*. In case of the **iremove-buckets** command the effect is to remove the in the group whose **bucket_id** is *id*. It is an error if there is no bucket persent group in whose **bucket_id** is *id*.

selection_method=*method*
The selection method used to select a bucket for a select group. This is a string of 1 to 15 bytes in length known to lower layers. This field is optional for **add−group**, **add−groups** and **mod−group** commands on groups of type **select**. Prohibited otherwise. The default value is the empty string.

hash Use a hash computed over the fields specified with the **fields** option, see below. **hash** uses the **selection_method_param** as the hash basis.

Note that the hashed fields become exact matched by the datapath flows. For example, if the TCP source port is hashed, the created datapath flows will match the specific TCP source port value present in the packet received. Since each TCP connection generally has a different source port value, a separate datapath flow will be need to be inserted for each TCP connection thus hashed to a select group bucket.

dp_hash
Use a datapath computed hash value. The hash algorithm varies accross different datapath implementations. **dp_hash** uses the upper 32 bits of the **selection_method_param** as the datapath hash algorithm selector, which currently must always be 0, corresponding to hash computation over the IP 5-tuple (selecting specific fields with the **fields** option is not allowed with **dp_hash**). The lower 32 bits are used as the hash basis.

Using **dp_hash** has the advantage that it does not require the generated datapath flows to exact match any additional packet header fields. For example, even if multiple TCP connections thus hashed to different select group buckets have different source port numbers, generally all of them would be handled with a small set of already established datapath flows, resulting in less latency for TCP SYN packets. The downside is that the shared datapath flows must match each packet twice, as the datapath hash value calculation happens only when needed, and a second match is required to match some bits of its value. This double-matching incurs a small additional latency cost for each packet, but this latency is orders of magnitude less than the latency of creating new datapath flows for

new TCP connections.

This option will use a Netronome OpenFlow extension which is only supported when using Open vSwitch 2.4 and later with OpenFlow 1.5 and later.

selection_method_param=*param*

64-bit integer parameter to the selection method selected by the **selection_method** field. The parameter's use is defined by the lower-layer that implements the **selection_method**. It is optional if the **selection_method** field is specified as a non-empty string. Prohibited otherwise. The default value is zero.

This option will use a Netronome OpenFlow extension which is only supported when using Open vSwitch 2.4 and later with OpenFlow 1.5 and later.

fields=*field*
fields(*field*[=*mask*]...**)**

The field parameters to selection method selected by the **selection_method** field. The syntax is described in **Flow Syntax** with the additional restrictions that if a value is provided it is treated as a wildcard mask and wildcard masks following a slash are prohibited. The pre-requisites of fields must be provided by any flows that output to the group. The use of the fields is defined by the lower-layer that implements the **selection_method**. They are optional if the **selection_method** field is specified as ''hash', prohibited otherwise. The default is no fields.

This option will use a Netronome OpenFlow extension which is only supported when using Open vSwitch 2.4 and later with OpenFlow 1.5 and later.

bucket=*bucket_parameters*

The **add-group**, **add-groups** and **mod-group** commands require at least one bucket field. Bucket fields must appear after all other fields. Multiple bucket fields to specify multiple buckets. The order in which buckets are specified corresponds to their order in the group. If the type of the group is "indirect" then only one group may be specified. *bucket_parameters* consists of a list of *field=value* assignments, separated by commas or white space followed by a comma-separated list of actions. The fields for *bucket_parameters* are:

bucket_id=*id*

The 32-bit integer group id of the bucket. Values greater than 0xffffff00 are reserved. This field was added in Open vSwitch 2.4 to conform with the OpenFlow 1.5 specification. It is not supported when earlier versions of OpenFlow are used. Open vSwitch will automatically allocate bucket ids when they are not specified.

actions=[*action*][**,***action*...]

The syntax of actions are identical to the **actions=** field described in **Flow Syntax** above. Specifying **actions=** is optional, any unknown bucket parameter will be interpreted as an action.

weight=*value*

The relative weight of the bucket as an integer. This may be used by the switch during bucket select for groups whose **type** is **select**.

watch_port=*port*

Port used to determine liveness of group. This or the **watch_group** field is required for groups whose **type** is **ff** or **fast_failover**.

watch_group=*group_id*

Group identifier of group used to determine liveness of group. This or the **watch_port** field is required for groups whose **type** is **ff** or **fast_failover**.

Meter Syntax

The meter table commands accept an argument that describes a meter. Such meter descriptions comprise a series *field=value* assignments, separated by commas or white space. (Embedding spaces into a group description normally requires quoting to prevent the shell from breaking the description into multiple arguments.). Unless noted otherwise only the last instance of each field is honoured.

meter=*id*

The integer meter id of the meter. When this field is specified in **del-meter**, **dump-meter**, or **meter-stats**, the keyword "all" may be used to designate all meters. This field is required, exept for **meter-stats**, which dumps all stats when this field is not specified.

kbps

pktps The unit for the meter band rate parameters, either kilobits per second, or packets per second, respectively. One of these must be specified. The burst size unit corresponds to the rate unit by dropping the "per second", i.e., burst is in units of kilobits or packets, respectively.

burst Specify burst size for all bands, or none of them, if this flag is not given.

stats Collect meter and band statistics.

bands=*band_parameters*

The **add-meter** and **mod-meter** commands require at least one band specification. Bands must appear after all other fields.

type=*type*

The type of the meter band. This keyword starts a new band specification. Each band specifies a rate above which the band is to take some action. The action depends on the band type. If multiple bands' rate is exceeded, then the band with the highest rate among the exceeded bands is selected. The following keywords designate the allowed meter band types:

drop Drop packets exceeding the band's rate limit.

The other *band_parameters* are:

rate=*value*

The relative rate limit for this band, in kilobits per second or packets per second, depending on the meter flags defined above.

burst_size=*size*

The maximum burst allowed for the band. If **pktps** is specified, then *size* is a packet count, otherwise it is in kilobits. If unspecified, the switch is free to select some reasonable value depending on its configuration.

OPTIONS

−−strict

Uses strict matching when running flow modification commands.

−−read-only

Do not execute read/write commands.

−−bundle

Execute flow mods as an OpenFlow 1.4 atomic bundle transaction.

- Within a bundle, all flow mods are processed in the order they appear and as a single atomic transaction, meaning that if one of them fails, the whole transaction fails and none of the changes are made to the *switch*'s flow table, and that each given datapath packet traversing the OpenFlow tables sees the flow tables either as before the transaction, or after all the flow mods in the bundle have been successfully applied.

- The beginning and the end of the flow table modification commands in a bundle are delimited with OpenFlow 1.4 bundle control messages, which makes it possible to stream the included commands without explicit OpenFlow barriers, which are otherwise used after each flow table modification command. This may make large modifications execute faster as a bundle.
- Bundles require OpenFlow 1.4 or higher. An explicit **-O OpenFlow14** option is not needed, but you may need to enable OpenFlow 1.4 support for OVS by setting the OVSDB *protocols* column in the *bridge* table.

−O [*version*[**,***version*]...]
−−protocols=[*version*[**,***version*]...]
Sets the OpenFlow protocol versions that are allowed when establishing an OpenFlow session.

The following versions are considered to be ready for general use. These protocol versions are enabled by default:

- **OpenFlow10**, for OpenFlow 1.0.

Support for the following protocol versions is provided for testing and development purposes. They are not enabled by default:

- **OpenFlow11**, for OpenFlow 1.1.
- **OpenFlow12**, for OpenFlow 1.2.
- **OpenFlow13**, for OpenFlow 1.3.

−F *format*[**,***format*...]
−−flow−format=*format*[**,***format*...]
ovs−ofctl supports the following individual flow formats, any number of which may be listed as *format*:

OpenFlow10−table_id
This is the standard OpenFlow 1.0 flow format. All OpenFlow switches and all versions of Open vSwitch support this flow format.

OpenFlow10+table_id
This is the standard OpenFlow 1.0 flow format plus a Nicira extension that allows **ovs−ofctl** to specify the flow table in which a particular flow should be placed. Open vSwitch 1.2 and later supports this flow format.

NXM−table_id (Nicira Extended Match)
This Nicira extension to OpenFlow is flexible and extensible. It supports all of the Nicira flow extensions, such as **tun_id** and registers. Open vSwitch 1.1 and later supports this flow format.

NXM+table_id (Nicira Extended Match)
This combines Nicira Extended match with the ability to place a flow in a specific table. Open vSwitch 1.2 and later supports this flow format.

OXM-OpenFlow12
OXM-OpenFlow13
OXM-OpenFlow14
These are the standard OXM (OpenFlow Extensible Match) flow format in OpenFlow 1.2, 1.3, and 1.4, respectively.

ovs−ofctl also supports the following abbreviations for collections of flow formats:

any Any supported flow format.

OpenFlow10
OpenFlow10−table_id or **OpenFlow10+table_id**.

NXM **NXM−table_id** or **NXM+table_id**.

OXM **OXM-OpenFlow12**, **OXM-OpenFlow13**, or **OXM-OpenFlow14**.

For commands that modify the flow table, **ovs−ofctl** by default negotiates the most widely supported flow format that supports the flows being added. For commands that query the flow table, **ovs−ofctl** by default uses the most advanced format supported by the switch.

This option, where *format* is a comma-separated list of one or more of the formats listed above, limits **ovs−ofctl**'s choice of flow format. If a command cannot work as requested using one of the specified flow formats, **ovs−ofctl** will report a fatal error.

−P *format*
−−packet−in−format=*format*

ovs−ofctl supports the following ''packet-in'' formats, in order of increasing capability:

standard
This uses the **OFPT_PACKET_IN** message, the standard ''packet-in'' message for any given OpenFlow version. Every OpenFlow switch that supports a given OpenFlow version supports this format.

nxt_packet_in
This uses the **NXT_PACKET_IN** message, which adds many of the capabilities of the OpenFlow 1.1 and later ''packet-in'' messages before those OpenFlow versions were available in Open vSwitch. Open vSwitch 1.1 and later support this format. Only Open vSwitch 2.6 and later, however, support it for OpenFlow 1.1 and later (but there is little reason to use it with those versions of OpenFlow).

nxt_packet_in2
This uses the **NXT_PACKET_IN2** message, which is extensible and should avoid the need to define new formats later. In particular, this format supports passing arbitrary user-provided data to a controller using the **userdata option on the controller** action. Open vSwitch 2.6 and later support this format.

Without this option, **ovs−ofctl** prefers **nxt_packet_in2** if the switch supports it. Otherwise, if OpenFlow 1.0 is in use, **ovs−ofctl** prefers **nxt_packet_in** if the switch supports it. Otherwise, **ovs−ofctl** falls back to the **standard** packet-in format. When this option is specified, **ovs−ofctl** insists on the selected format. If the switch does not support the requested format, **ovs−ofctl** will report a fatal error.

Before version 2.6, Open vSwitch called **standard** format **openflow10** and **nxt_packet_in** format **nxm**, and **ovs−ofctl** still accepts these names as synonyms. (The name **openflow10** was a misnomer because this format actually varies from one OpenFlow version to another; it is not consistently OpenFlow 1.0 format. Similarly, when **nxt_packet_in2** was introduced, the name **nxm** became confusing because it also uses OXM/NXM.)

This option affects only the **monitor** command.

−−timestamp
Print a timestamp before each received packet. This option only affects the **monitor**, **snoop**, and **ofp−parse−pcap** commands.

−m
−−more
Increases the verbosity of OpenFlow messages printed and logged by **ovs−ofctl** commands. Specify this option more than once to increase verbosity further.

−−sort[=*field*]
−−rsort[=*field*]
Display output sorted by flow *field* in ascending (**−−sort**) or descending (**−−rsort**) order, where *field* is any of the fields that are allowed for matching or **priority** to sort by priority. When *field* is omitted, the output is sorted by priority. Specify these options multiple times to sort by multiple

fields.

Any given flow will not necessarily specify a value for a given field. This requires special treatement:

- A flow that does not specify any part of a field that is used for sorting is sorted after all the flows that do specify the field. For example, **−−sort=tcp_src** will sort all the flows that specify a TCP source port in ascending order, followed by the flows that do not specify a TCP source port at all.
- A flow that only specifies some bits in a field is sorted as if the wildcarded bits were zero. For example, **−−sort=nw_src** would sort a flow that specifies **nw_src=192.168.0.0/24** the same as **nw_src=192.168.0.0**.

These options currently affect only **dump−flows** output. The following options are valid on POSIX based platforms.

−−pidfile[=*pidfile*]
Causes a file (by default, **ovs−ofctl.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−ofctl** 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−ofctl** 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−ofctl** detaches only when executing the **monitor** or **snoop** commands.

−−monitor
Creates an additional process to monitor the **ovs−ofctl** 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−ofctl** changes its current working directory to the root directory after it detaches. Otherwise, invoking **ovs−ofctl** 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−ofctl** 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−ofctl** 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.

−−unixctl=*socket*

Sets the name of the control socket on which **ovs−ofctl** 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−ofctl.***pid***.ctl**, where *pid* is **ovs−ofctl**'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−ofctl.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.

Public Key Infrastructure Options

−p *privkey.pem*
−−private−key=*privkey.pem*

Specifies a PEM file containing the private key used as **ovs−ofctl**'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−ofctl** 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.

−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−ofctl** 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−ofctl.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=*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.

−−color[=*when*]
Colorize the output (for some commands); *when* can be **never**, **always**, or **auto** (the default).

Only some commands support output coloring. Color names and default colors may change in future releases.

The environment variable **OVS_COLORS** can be used to specify user-defined colors and other attributes used to highlight various parts of the output. If set, its value is a colon-separated list of capabilities that defaults to **ac:01;31:dr=34:le=31:pm=36:pr=35:sp=33:vl=32**. Supported capabilities were initially designed for coloring flows from **ovs-ofctl dump-flows** *switch* command, and they are as follows.

ac=01;31
SGR substring for **actions=** keyword in a flow. The default is a bold red text foreground.

dr=34 SGR substring for **drop** keyword. The default is a dark blue text foreground.

le=31 SGR substring for **learn=** keyword in a flow. The default is a red text foreground.

pm=36 SGR substring for flow match attribute names. The default is a cyan text foreground.

pr=35 SGR substring for keywords in a flow that are followed by arguments inside parenthesis. The default is a magenta text foreground.

sp=33 SGR substring for some special keywords in a flow, notably: **table=**, **priority=**, **load:**, **output:**, **move:**, **group:**, **CONTROLLER:**, **set_field:**, **resubmit:**, **exit**. The default is a yellow text foreground.

vl=32 SGR substring for a lone flow match attribute with no field name. The default is a green text foreground.

See the Select Graphic Rendition (SGR) section in the documentation of the text terminal that is used for permitted values and their meaning as character attributes.

−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−ofctl** process. The supported commands are listed below.

exit Causes **ovs−ofctl** to gracefully terminate. This command applies only when executing the **monitor** or **snoop** commands.

ofctl/set−output−file *file*
Causes all subsequent output to go to *file* instead of stderr. This command applies only when executing the **monitor** or **snoop** commands.

ofctl/send *ofmsg...*
Sends each *ofmsg*, specified as a sequence of hex digits that express an OpenFlow message, on the OpenFlow connection. This command is useful only when executing the **monitor** command.

ofctl/packet−out *packet-out*
Sends an OpenFlow PACKET_OUT message specified in **Packet−Out Syntax**, on the OpenFlow connection. See **Packet−Out Syntax** section for more information. This command is useful only when executing the **monitor** command.

ofctl/barrier
Sends an OpenFlow barrier request on the OpenFlow connection and waits for a reply. This command is useful only for the **monitor** command.

EXAMPLES

The following examples assume that **ovs−vswitchd** has a bridge named **br0** configured.

ovs−ofctl dump−tables br0
Prints out the switch's table stats. (This is more interesting after some traffic has passed through.)

ovs−ofctl dump−flows br0
Prints the flow entries in the switch.

ovs−ofctl add−flow table=0 actions=learn(table=1,hard_timeout=10, NXM_OF_VLAN_TCI[0..11],output:NXM_OF_IN_PORT[]), resubmit(,1)
ovs−ofctl add−flow table=1 priority=0 actions=flood Implements a level 2 MAC learning switch using the learn.

ovs−ofctl add−flow br0 'table=0,priority=0 actions=load:3->NXM_NX_REG0[0..15],learn(table=0,priority=1,idle_timeout=10,NXM_OF_ETH_SRC[],NXM_OF_VLAN_TCI[0..11],output:NXM_NX_REG0[0..15]),output:2
In this use of a learn action, the first packet from each source MAC will be sent to port 2. Subsequent packets will be output to port 3, with an idle timeout of 10 seconds. NXM field names and match field names are both accepted, e.g. **NXM_NX_REG0** or **reg0** for the first register, and empty brackets may be omitted.

Additional examples may be found documented as part of related sections.

SEE ALSO

ovs−fields(7), **ovs−appctl**(8), **ovs−vswitchd**(8), **ovs−vswitchd.conf.db**(8)