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*]
: The specified SSL *port* (default: 6633) on the host at the given *ip*, which must be expressed as an IP address (not a DNS name). The **−−private−key**, **−−certificate**, and **−−ca−cert** options are mandatory when this form is used.

tcp:*ip*[**:***port*]
: The specified TCP *port* (default: 6633) on the host at the given *ip*, which must be expressed as an IP address (not a DNS name).

unix:*file*
: The Unix domain server socket named *file*.

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/***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−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*
: Prints to the console detailed information about network devices associated with *switch* (version 1.7 or later). This is a subset of the information provided by the **show** command.

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.

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, and *file* is a text file that contains zero or more flows in the same syntax, one per line.

add−flow *switch flow*
add−flow *switch* − < *file*
add−flows *switch file*
Add each flow entry to *switch*'s tables.

[−−**strict**] **mod−flows** *switch flow*
[−−**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.

del−flows *switch*
[−−**strict**] **del−flows** *switch* [*flow*]
[−−**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.

[−−**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 in_port actions packet...*
Connects to *switch* and instructs it to execute the OpenFlow *actions* on each *packet*. For the purpose of executing the actions, the packets are considered to have arrived on *in_port*, which may be an OpenFlow port number or name (e.g. **eth0**), the keyword **LOCAL** (the preferred way to refer to the OpenFlow ''local'' port), or the keyword **NONE** to indicate that the packet was generated by the switch itself.

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/***bridge***.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), 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.

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 **RUNTIME 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.

Flow Syntax

Some **ovs−ofctl** commands accept an argument that describes a flow or flows. Such flow descriptions comprise a series *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 **tp_dst** and **tp_src**, which are L4 fields. **ovs−ofctl** will warn about flows not in normal form.

The following field assignments describe how a flow matches a packet. If any of these assignments is omitted from the flow syntax, the field is treated as a wildcard; thus, if all of them are omitted, the resulting flow matches all packets. The string ***** may be specified to explicitly mark any of these fields as a wildcard. (***** should be quoted to protect it from shell expansion.)

in_port=*port*
Matches OpenFlow port *port*, which may be an OpenFlow port number or keyword (e.g. **LOCAL**). **ovs−ofctl show**.

(The **resubmit** action can search OpenFlow flow tables with arbitrary **in_port** values, so flows that match port numbers that do not exist from an OpenFlow perspective can still potentially be

matched.)

dl_vlan=*vlan*
Matches IEEE 802.1q Virtual LAN tag *vlan*. Specify **0xffff** as *vlan* to match packets that are not tagged with a Virtual LAN; otherwise, specify a number between 0 and 4095, inclusive, as the 12-bit VLAN ID to match.

dl_vlan_pcp=*priority*
Matches IEEE 802.1q Priority Code Point (PCP) *priority*, which is specified as a value between 0 and 7, inclusive. A higher value indicates a higher frame priority level.

dl_src=*xx***:***xx***:***xx***:***xx***:***xx***:***xx*
dl_dst=*xx***:***xx***:***xx***:***xx***:***xx***:***xx*
Matches an Ethernet source (or destination) address specified as 6 pairs of hexadecimal digits delimited by colons (e.g. **00:0A:E4:25:6B:B0**).

dl_src=*xx***:***xx***:***xx***:***xx***:***xx***:***xx***/***xx***:***xx***:***xx***:***xx***:***xx***:***xx*
dl_dst=*xx***:***xx***:***xx***:***xx***:***xx***:***xx***/***xx***:***xx***:***xx***:***xx***:***xx***:***xx*
Matches an Ethernet destination address specified as 6 pairs of hexadecimal digits delimited by colons (e.g. **00:0A:E4:25:6B:B0**), with a wildcard mask following the slash. Open vSwitch 1.8 and later support arbitrary masks for source and/or destination. Earlier versions only support masking the destination with the following masks:

01:00:00:00:00:00
Match only the multicast bit. Thus, **dl_dst=01:00:00:00:00:00/01:00:00:00:00:00** matches all multicast (including broadcast) Ethernet packets, and **dl_dst=00:00:00:00:00:00/01:00:00:00:00:00** matches all unicast Ethernet packets.

fe:ff:ff:ff:ff:ff
Match all bits except the multicast bit. This is probably not useful.

ff:ff:ff:ff:ff:ff
Exact match (equivalent to omitting the mask).

00:00:00:00:00:00
Wildcard all bits (equivalent to **dl_dst=*.**)

dl_type=*ethertype*
Matches Ethernet protocol type *ethertype*, which is specified as an integer between 0 and 65535, inclusive, either in decimal or as a hexadecimal number prefixed by **0x** (e.g. **0x0806** to match ARP packets).

nw_src=*ip*[/*netmask*]
nw_dst=*ip*[/*netmask*]
When **dl_type** is 0x0800 (possibly via shorthand, e.g. **ip** or **tcp**), matches IPv4 source (or destination) address *ip*, which may be specified as an IP address or host name (e.g. **192.168.1.1** or **www.example.com**). The optional *netmask* allows restricting a match to an IPv4 address prefix. The netmask may be specified as a dotted quad (e.g. **192.168.1.0/255.255.255.0**) or as a CIDR block (e.g. **192.168.1.0/24**). Open vSwitch 1.8 and later support arbitrary dotted quad masks; earlier versions support only CIDR masks, that is, the dotted quads that are equivalent to some CIDR block.

When **dl_type=0x0806** or **arp** is specified, matches the **ar_spa** or **ar_tpa** field, respectively, in ARP packets for IPv4 and Ethernet.

When **dl_type=0x8035** or **rarp** is specified, matches the **ar_spa** or **ar_tpa** field, respectively, in RARP packets for IPv4 and Ethernet.

When **dl_type** is wildcarded or set to a value other than 0x0800, 0x0806, or 0x8035, the values of **nw_src** and **nw_dst** are ignored (see **Flow Syntax** above).

nw_proto=*proto*

When **ip** or **dl_type=0x0800** is specified, matches IP protocol type *proto*, which is specified as a decimal number between 0 and 255, inclusive (e.g. 1 to match ICMP packets or 6 to match TCP packets).

When **ipv6** or **dl_type=0x86dd** is specified, matches IPv6 header type *proto*, which is specified as a decimal number between 0 and 255, inclusive (e.g. 58 to match ICMPv6 packets or 6 to match TCP). The header type is the terminal header as described in the **DESIGN** document.

When **arp** or **dl_type=0x0806** is specified, matches the lower 8 bits of the ARP opcode. ARP opcodes greater than 255 are treated as 0.

When **rarp** or **dl_type=0x8035** is specified, matches the lower 8 bits of the ARP opcode. ARP opcodes greater than 255 are treated as 0.

When **dl_type** is wildcarded or set to a value other than 0x0800, 0x0806, 0x8035 or 0x86dd, the value of **nw_proto** is ignored (see **Flow Syntax** above).

nw_tos=*tos*

Matches IP ToS/DSCP or IPv6 traffic class field *tos*, which is specified as a decimal number between 0 and 255, inclusive. Note that the two lower reserved bits are ignored for matching purposes.

When **dl_type** is wildcarded or set to a value other than 0x0800 or 0x86dd, the value of **nw_tos** is ignored (see **Flow Syntax** above).

nw_ecn=*ecn*

Matches *ecn* bits in IP ToS or IPv6 traffic class fields, which is specified as a decimal number between 0 and 3, inclusive.

When **dl_type** is wildcarded or set to a value other than 0x0800 or 0x86dd, the value of **nw_ecn** is ignored (see **Flow Syntax** above).

nw_ttl=*ttl*

Matches IP TTL or IPv6 hop limit value *ttl*, which is specified as a decimal number between 0 and 255, inclusive.

When **dl_type** is wildcarded or set to a value other than 0x0800 or 0x86dd, the value of **nw_ttl** is ignored (see **Flow Syntax** above).

tp_src=*port*
tp_dst=*port*

When **dl_type** and **nw_proto** specify TCP or UDP, **tp_src** and **tp_dst** match the UDP or TCP source or destination port *port*, respectively, which is specified as a decimal number between 0 and 65535, inclusive (e.g. 80 to match packets originating from a HTTP server).

When **dl_type** and **nw_proto** take other values, the values of these settings are ignored (see **Flow Syntax** above).

tp_src=*port*/*mask*
tp_dst=*port*/*mask*

Bitwise match on TCP (or UDP) source or destination port, respectively. The *port* and *mask* are 16-bit numbers written in decimal or in hexadecimal prefixed by **0x**. Each 1-bit in *mask* requires that the corresponding bit in *port* must match. Each 0-bit in *mask* causes the corresponding bit to be ignored.

Bitwise matches on transport ports are rarely useful in isolation, but a group of them can be used to reduce the number of flows required to match on a range of transport ports. For example, suppose that the goal is to match TCP source ports 1000 to 1999, inclusive. One way is to insert 1000 flows, each of which matches on a single source port. Another way is to look at the binary representations of 1000 and 1999, as follows:
01111101000

11111001111
and then to transform those into a series of bitwise matches that accomplish the same results:
01111101xxx
0111111xxxx
10xxxxxxxx
110xxxxxxx
1110xxxxxx
11110xxxxx
1111100xxxx
which become the following when written in the syntax required by **ovs−ofctl**:
tcp,tp_src=0x03e8/0xfff8
tcp,tp_src=0x03f0/0xfff0
tcp,tp_src=0x0400/0xfe00
tcp,tp_src=0x0600/0xff00
tcp,tp_src=0x0700/0xff80
tcp,tp_src=0x0780/0xffc0
tcp,tp_src=0x07c0/0xfff0

Only Open vSwitch 1.6 and later supports bitwise matching on transport ports.

Like the exact-match forms of **tp_src** and **tp_dst** described above, the bitwise match forms apply only when **dl_type** and **nw_proto** specify TCP or UDP.

icmp_type=*type*
icmp_code=*code*
When **dl_type** and **nw_proto** specify ICMP or ICMPv6, *type* matches the ICMP type and *code* matches the ICMP code. Each is specified as a decimal number between 0 and 255, inclusive.

When **dl_type** and **nw_proto** take other values, the values of these settings are ignored (see **Flow Syntax** above).

table=*number*
If specified, limits the flow manipulation and flow dump commands to only apply to the table with the given *number* between 0 and 254. Behavior varies if **table** is not specified (equivalent to specifying 255 as *number*). For flow table modification commands without −−**strict**, the switch will choose the table for these commands to operate on. For flow table modification commands 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 **dump-flows** and **dump-aggregate** commands will gather statistics about flows from all tables.

When this field is specified in **add-flow**, **add-flows**, **mod-flows** and **del-flows** commands, it activates a Nicira extension to OpenFlow, which as of this writing is only known to be implemented by Open vSwitch.

metadata=*value*[/*mask*]
Matches *value* either exactly or with optional *mask* in the metadata field. *value* and *mask* are 64-bit integers, by default in decimal (use a **0x** prefix to specify hexadecimal). Arbitrary *mask* values are allowed: a 1-bit in *mask* indicates that the corresponding bit in *value* must match exactly, and a 0-bit wildcards that bit. Matching on metadata was added in Open vSwitch 1.8.

The following shorthand notations are also available:

ip Same as **dl_type=0x0800**.

icmp Same as **dl_type=0x0800,nw_proto=1**.

tcp Same as **dl_type=0x0800,nw_proto=6**.

udp Same as **dl_type=0x0800,nw_proto=17**.

arp Same as **dl_type=0x0806**.

rarp Same as **dl_type=0x8035**.

The following field assignments require support for the NXM (Nicira Extended Match) extension to OpenFlow. When one of these is specified, **ovs−ofctl** will automatically attempt to negotiate use of this extension. If the switch does not support NXM, then **ovs−ofctl** will report a fatal error.

vlan_tci=*tci*[**/***mask*]
Matches modified VLAN TCI *tci*. If *mask* is omitted, *tci* is the exact VLAN TCI to match; if *mask* is specified, then a 1-bit in *mask* indicates that the corresponding bit in *tci* must match exactly, and a 0-bit wildcards that bit. Both *tci* and *mask* are 16-bit values that are decimal by default; use a **0x** prefix to specify them in hexadecimal.

The value that **vlan_tci** matches against is 0 for a packet that has no 802.1Q header. Otherwise, it is the TCI value from the 802.1Q header with the CFI bit (with value **0x1000**) forced to 1.

Examples:

vlan_tci=0
Match only packets without an 802.1Q header.

vlan_tci=0xf123
Match packets tagged with priority 7 in VLAN 0x123.

vlan_tci=0x1123/0x1fff
Match packets tagged with VLAN 0x123 (and any priority).

vlan_tci=0x5000/0xf000
Match packets tagged with priority 2 (in any VLAN).

vlan_tci=0/0xfff
Match packets with no 802.1Q header or tagged with VLAN 0 (and any priority).

vlan_tci=0x5000/0xe000
Match packets with no 802.1Q header or tagged with priority 2 (in any VLAN).

vlan_tci=0/0xefff
Match packets with no 802.1Q header or tagged with VLAN 0 and priority 0.

Some of these matching possibilities can also be achieved with **dl_vlan** and **dl_vlan_pcp**.

ip_frag=*frag_type*
When **dl_type** specifies IP or IPv6, *frag_type* specifies what kind of IP fragments or non-fragments to match. The following values of *frag_type* are supported:

no Matches only non-fragmented packets.

yes Matches all fragments.

first Matches only fragments with offset 0.

later Matches only fragments with nonzero offset.

not_later
Matches non-fragmented packets and fragments with zero offset.

The **ip_frag** match type is likely to be most useful in **nx−match** mode. See the description of the **set−frags** command, above, for more details.

arp_sha=*xx***:***xx***:***xx***:***xx***:***xx***:***xx*
arp_tha=*xx***:***xx***:***xx***:***xx***:***xx***:***xx*
When **dl_type** specifies either ARP or RARP, **arp_sha** and **arp_tha** match the source and target hardware address, respectively. An address is specified as 6 pairs of hexadecimal digits delimited by colons.

ipv6_src=*ipv6*[**/***netmask*]

ipv6_dst=*ipv6*[/*netmask*]
When **dl_type** is 0x86dd (possibly via shorthand, e.g., **ipv6** or **tcp6**), matches IPv6 source (or destination) address *ipv6*, which may be specified as defined in RFC 2373. The preferred format is *x*:*x*:*x*:*x*:*x*:*x*:*x*:*x*, where *x* are the hexadecimal values of the eight 16-bit pieces of the address. A single instance of **::** may be used to indicate multiple groups of 16-bits of zeros. The optional *netmask* allows restricting a match to an IPv6 address prefix. A netmask is specified as an IPv6 address (e.g. **2001:db8:3c4d:1::/ffff:ffff:ffff:ffff::**) or a CIDR block (e.g. **2001:db8:3c4d:1::/64**). Open vSwitch 1.8 and later support arbitrary masks; earlier versions support only CIDR masks, that is, CIDR block and IPv6 addresses that are equivalent to CIDR blocks.

ipv6_label=*label*
When **dl_type** is 0x86dd (possibly via shorthand, e.g., **ipv6** or **tcp6**), matches IPv6 flow label *label*.

nd_target=*ipv6*[/*netmask*]
When **dl_type**, **nw_proto**, and **icmp_type** specify IPv6 Neighbor Discovery (ICMPv6 type 135 or 136), matches the target address *ipv6*. *ipv6* is in the same format described earlier for the **ipv6_src** and **ipv6_dst** fields.

nd_sll=*xx*:*xx*:*xx*:*xx*:*xx*:*xx*
When **dl_type**, **nw_proto**, and **icmp_type** specify IPv6 Neighbor Solicitation (ICMPv6 type 135), matches the source link−layer address option. An address is specified as 6 pairs of hexadecimal digits delimited by colons.

nd_tll=*xx*:*xx*:*xx*:*xx*:*xx*:*xx*
When **dl_type**, **nw_proto**, and **icmp_type** specify IPv6 Neighbor Advertisement (ICMPv6 type 136), matches the target link−layer address option. An address is specified as 6 pairs of hexadecimal digits delimited by colons.

tun_id=*tunnel-id*[/*mask*]
Matches tunnel identifier *tunnel-id*. Only packets that arrive over a tunnel that carries a key (e.g. GRE with the RFC 2890 key extension and a nonzero key value) will have a nonzero tunnel ID. If *mask* is omitted, *tunnel-id* is the exact tunnel ID to match; if *mask* is specified, then a 1-bit in *mask* indicates that the corresponding bit in *tunnel-id* must match exactly, and a 0-bit wildcards that bit.

tun_src=*ip*[/*netmask*]
tun_dst=*ip*[/*netmask*]
Matches tunnel IPv4 source (or destination) address *ip*. Only packets that arrive over a tunnel will have nonzero tunnel addresses. The address may be specified as an IP address or host name (e.g. **192.168.1.1** or **www.example.com**). The optional *netmask* allows restricting a match to a masked IPv4 address. The netmask may be specified as a dotted quad (e.g. **192.168.1.0/255.255.255.0**) or as a CIDR block (e.g. **192.168.1.0/24**).

reg*idx***=***value*[/*mask*]
Matches *value* either exactly or with optional *mask* in register number *idx*. The valid range of *idx* depends on the switch. *value* and *mask* are 32-bit integers, by default in decimal (use a **0x** prefix to specify hexadecimal). Arbitrary *mask* values are allowed: a 1-bit in *mask* indicates that the corresponding bit in *value* must match exactly, and a 0-bit wildcards that bit.

When a packet enters an OpenFlow switch, all of the registers are set to 0. Only explicit Nicira extension actions change register values.

Defining IPv6 flows (those with **dl_type** equal to 0x86dd) requires support for NXM. The following shorthand notations are available for IPv6-related flows:

ipv6 Same as **dl_type=0x86dd**.

tcp6 Same as **dl_type=0x86dd,nw_proto=6**.

udp6 Same as **dl_type=0x86dd,nw_proto=17**.

icmp6 Same as **dl_type=0x86dd,nw_proto=58**.

Finally, field assignments to **duration**, **n_packets**, or **n_bytes** are ignored 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=[*target*][**,***target*...]
Specifies a comma-separated list of actions to take on a packet when the flow entry matches. If no *target* is specified, then packets matching the flow are dropped. The *target* may be an OpenFlow port number designating the physical port on which to output the packet, or one of the following keywords:

output:*port*
Outputs the packet to *port*, which must be an OpenFlow port number or keyword (e.g. **LOCAL**).

output:*src***[***start***..***end***]**
Outputs the packet to the OpenFlow port number read from *src*, which must be an NXM field as described above. For example, **output:NXM_NX_REG0[16..31]** outputs to the OpenFlow port number written in the upper half of register 0. This form of **output** uses an OpenFlow extension that is not supported by standard OpenFlow switches.

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.

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.

controller(*key***=***value*...**)**
Sends the packet 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.

Any *reason* other than **action** and any nonzero *controller-id* uses a Nicira vendor extension that, as of this writing, is only known to be implemented by Open vSwitch (version 1.6 or later).

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

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.

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 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*
If the packet does not already contain any MPLS labels, changes the packet’s Ethertype to *ethertype*, which must be either the MPLS unicast Ethertype **0x8847** or the MPLS multicast Ethertype **0x8848**, and then pushes an initial label stack entry. The label stack entry’s default 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.

There are some limitations in the implementation. **push_mpls** followed by another **push_mpls** will result in the first **push_mpls** being discarded.

pop_mpls:*ethertype*
Strips the outermost MPLS label stack entry. If the MPLS label stripped was the only one, changes the ethertype of a packet to *ethertype*, which should not ordinarily be an MPLS Ethertype.

There are some limitations in the implementation. **pop_mpls** followed by another **push_mpls** without an intermediate **push_mpls** will result in the first **push_mpls** being discarded.

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 source port to *port*.

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

mod_nw_tos:*tos*
Sets the IPv4 ToS/DSCP 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).

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*]**)**
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 executes the actions found, if any, in addition to any other actions in this flow entry.

Recursive **resubmit** actions are obeyed up to an implementation-defined maximum depth. Open vSwitch 1.0.1 and earlier did not support recursion; Open vSwitch before 1.2.90 did not support *table*.

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.

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, no decrement occurs. 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 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_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, no decrement occurs. Instead, a ‘‘packet-in’’ message with reason code **OFPR_INVALID_TTL** is sent to each connected controller with controller id zer that

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* must be NXM field names as defined in **nicira−ext.h**, e.g. **NXM_OF_UDP_SRC** or **NXM_NX_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.

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:NXM_NX_REG0[0..15]−>NXM_OF_VLAN_TCI[]** copies the least significant 16 bits of register 0 into the VLAN TCI field.

load:*value***−>***dst***[***start***..***end***]**
Writes *value* to bits *start* through *end*, inclusive, in field *dst*.

Example: **load:55−>NXM_NX_REG2[0..5]** loads value 55 (bit pattern **110111**) into bits 0 through 5, inclusive, in register 2.

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

Example: **push:NXM_NX_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]** 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.

set_field:*value***−>***dst*
Writes the literal *value* into the field *dst*, which should be specified as a name used for matching. (This is similar to **load** but more closely matches the set-field action defined in Open Flow 1.2 and above.)

Example: **set_field:fe80:0123:4567:890a:a6ba:dbff:fefe:59fa−>ipv6_src**

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.

Currently, *fields* must be either **eth_src** or **symmetric_l4** and *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** or **symmetric_l4** 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[]**.

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*
These key-value pairs 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.

*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.

The second form specifies that *field***[***start***..***end***]** in the new flow must match *src***[***start***..***end***]** taken from the flow currently being processed.

The third form is a shorthand for the second form. It specifies that *field***[***start***..***end***]** in the new flow must match *field***[***start***..***end***]** taken from the flow currently being processed.

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.)

apply_actions([*action*][**,***action*...]**)**
Applies the specific action(s) immediately. The syntax of actions are same to **actions=** field.

clear_actions
Clears all the actions in the action set immediately.

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.

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.

Refer to **ovs−vswitchd.conf.db**(8) 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.

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.

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 one additional optional field:

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

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.

OPTIONS

−−strict
Uses strict matching when running flow modification commands.

−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
These are the standard OXM (OpenFlow Extensible Match) flow format in OpenFlow 1.2 and 1.3, 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** or **OXM-OpenFlow13**.

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:

openflow10
This is the standard OpenFlow 1.0 packet in format. It should be supported by all OpenFlow switches.

nxm (Nicira Extended Match)
This packet_in format includes flow metadata encoded using the NXM format.

Usually, **ovs−ofctl** prefers the **nxm** packet_in format, but will allow the switch to choose its default if **nxm** is unsupported. When *format* is one of the formats listed in the above table, **ovs−ofctl** will insist on the selected format. If the switch does not support the requested format, **ovs−ofctl** will report a fatal error. This option only affects the **monitor** command.

−−timestamp
Print a timestamp before each received packet. This option only affects the **monitor** and **snoop** 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.

−−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**.

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
Causes **ovs−ofctl** to detach itself from the foreground session and run as a background process. **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 (e.g. **SIGSEGV**, **SIGABRT**), then the monitor process starts a new copy of it. If the daemon die 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.

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 facility 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.
- **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**.

−−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**.

−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/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.

SEE ALSO

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