EstiNet MT198T OpenFlow Command Reference

USER GUIDE

EstiNet

Table of Contents

1 General Guide

EstiNet MT198T switch supports most of the OpenFlow protocol version 1.3.4 functions . It is installed with the Open vSwitch agent, which is responsible for handling OpenFlow commands from an SDN controller or a local CLI console. This way, users can use the OpenFlow functions provided by the switch via OVS commands, which are explained in detail in following chapters.

2 ovs-appctl

```
NAME
       ovs-appctl - utility for configuring running Open vSwitch daemons

SYNOPSIS
       ovs-appctl [--target=target | -t target] command [arg...]
       ovs-appctl --help
       ovs-appctl --version

DESCRIPTION
       Open  vSwitch  daemons  accept  certain  commands at runtime to control
       their behavior and query their settings.  Every daemon accepts a common
       set of  commands documented under COMMON COMMANDS below.  Some daemons
       support additional  commands  documented  in   their   own   manpages.
       ovs-vswitchd in particular accepts a number of additional commands documented in
ovs-vswitchd(8).

       The ovs-appctl program provides a simple way to invoke these commands.
       The command to  be  sent is specified on ovs-appctl's command line as
       non-option arguments.  ovs-appctl sends the command and prints the daemon’s
response on standard output.

       In normal use only a single option is accepted:

       -t target
       --target=target
              Tells ovs-appctl which daemon to contact.

              If  target  begins  with  / it must name a Unix domain socket on
              which an Open vSwitch daemon is listening  for  control  channel
              connections.   By  default, each daemon listens on a Unix domain
              socket named //var/run/openvswitch/program.pid.ctl,  where  pro-
              gram is the program's name and pid is its process ID.  For exam-
              ple,  if  ovs-vswitchd  has  PID  123,  it   would   listen   on
              //var/run/openvswitch/ovs-vswitchd.123.ctl.

              Otherwise, ovs-appctl looks for a pidfile, that is, a file whose
              contents are the process ID of a running process  as  a  decimal
              number,  named //var/run/openvswitch/target.pid.  (The --pidfile
              option  makes  an  Open  vSwitch  daemon  create   a   pidfile.)
              ovs-appctl reads the pidfile, then looks for a Unix socket named
              //var/run/openvswitch/target.pid.ctl, where pid is  replaced  by
              the  process  ID read from the pidfile, and uses that file as if
              it had been specified directly as the target.

              On Windows, target can be an absolute path to a file  that  con-
              tains  a  localhost  TCP port on which an Open vSwitch daemon is
              listening for control channel connections. By default, each dae-
              mon  writes  the  TCP  port on which it is listening for control
```

connection into the file program.ctl located inside the configured OVS_RUNDIR directory. If target is not an absolute path, ovs-appctl looks for a file named target.ctl in the configured OVS_RUNDIR directory.

The default target is ovs-vswitchd.

2.1 COMMON COMMANDS

Every Open vSwitch daemon supports a common set of commands, which are documented in this section.

2.1.1 GENERAL COMMANDS

These commands display daemon-specific commands and the running version. Note that these commands are different from the --help and --version options that return information about the ovs-appctl utility itself.

list-commands
Lists the commands supported by the target.

version
Displays the version and compilation date of the target.

2.1.2 LOGGING COMMANDS

Open vSwitch has several log levels. The highest-severity log level is:

off
No message is ever logged at this level, so setting a logging destination's log level to off disables logging to that destination.

The following log levels, in order of descending severity, are available:

emer
A major failure forced a process to abort.

err
A high-level operation or a subsystem failed. Attention is warranted.

warn
A low-level operation failed, but higher-level subsystems may be able to recover.

info
Information that may be useful in retrospect when investigating a problem.

dbg
Information useful only to someone with intricate knowledge of the system, or that would commonly cause too-voluminous log output. Log messages at this level are not logged by default.

Every Open vSwitch daemon supports the following commands for examining and adjusting log levels.

vlog/list
Lists the known logging modules and their current levels.

vlog/list-pattern
Lists logging pattern used for each destination.

vlog/set [spec]

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

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

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

  On Windows platform, syslog is accepted as a word and is only useful if the target was started 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.

Case is not significant within spec.

Regardless of the log levels set for file, logging to a file will not take place unless the target application was invoked with the --log-file option.

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

vlog/set PATTERN:destination:pattern

Sets the log pattern for destination to pattern. Each time a message is logged to destination, pattern determines the message's formatting. Most characters in pattern are copied literally to the log, but special escapes beginning with % are expanded as follows:

%A The name of the application logging the message, e.g. ovs-vswitchd.

%B The RFC5424 syslog PRI of the message.

%c The name of the module (as shown by ovs-appctl --list) logging the message.

%d The current date and time in ISO 8601 format (YYYY-MM-DD HH:MM:SS).

%d{format}

The current date and time in the specified format, which takes the same format as the template argument to strftime(3). As an extension, any # characters in format will be replaced by fractional seconds, e.g. use %H:%M:%S.### for the time to the nearest millisecond. Sub-second times are only approximate and currently decimal places after the third will always be reported as zero.

%D The current UTC date and time in ISO 8601 format (YYYY-MM-DD HH:MM:SS).

%D{format}

The current UTC date and time in the specified format, which takes the same format as the template argument to strftime(3). Supports the same extension for sub-second resolution as %d{...}.

%E The hostname of the node running the application.

%m The message being logged.

%N A serial number for this message within this run of the program, as a decimal number. The first message a program logs has serial number 1, the second one has serial number 2, and so on.

%n A new-line.

%p The level at which the message is logged, e.g. DBG.

%P The program's process ID (pid), as a decimal number.

%r The number of milliseconds elapsed from the start of the application to the time the message was logged.

%t The subprogram name, that is, an identifying name for the process or thread that emitted the log message, such as monitor for the process used for --monitor or main for the primary process or thread in a program.

%T The subprogram name enclosed in parentheses, e.g. (monitor), or the empty string for the primary process or thread in a program.

%% A literal %.

A few options may appear between the % and the format specifier character, in this order:

- Left justify the escape's expansion within its field width. Right justification is the default.

0 Pad the field to the field width with 0s. Padding with spaces is the default.

width A number specifies the minimum field width. If the escape expands to fewer characters than width then it is padded to fill the field width. (A field wider than width is not truncated to fit.)

The default pattern for console and file output is %D{%Y-%m-%dT %H:%M:%SZ}|%05N|%c|%p|%m; for syslog output, %05N|%c|%p|%m.

Daemons written in Python (e.g. ovs-xapi-sync, ovs-monitor-ipsec) do not allow control over the log pattern.

vlog/set 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.

```
vlog/close
       Causes the daemon to close its log file, if it  is  open.   (Use
       vlog/reopen to reopen it later.)

vlog/reopen
       Causes the daemon to close its log file, if it is open, and then
       reopen it.  (This is useful after rotating log files, to cause a
       new log file to be used.)

       This  has  no  effect  if the target application was not invoked
       with the --log-file option.
```

2.2 OPTIONS

```
-h
--help Prints a brief help message to the console.
-V
--version
       Prints version information to the console.
```

3 ovs-ofctl

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.

3.1 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, a localhost TCP port written in file.

file

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

bridge

This is short for unix://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.

3.1.1 show switch

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

3.1.2 dump-tables switch

Prints to the console statistics for each of the flow tables used by switch.

3.1.3 dump-table-features switch

Prints to the console features for each of the flow tables used by switch.

3.1.4 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.

3.1.5 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.

3.1.6 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.

3.1.7 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.

3.1.8 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.

3.1.9 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.

3.2 OpenFlow 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.

3.2.1 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.

3.2.2 dump-group-features switch

Prints to the console the group features of the switch.

3.2.3 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.

3.3 OpenFlow 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.

3.3.1 add-meter switch meter

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

3.3.2 mod-meter switch meter

Modify an existing meter.

3.3.3 del-meters switch

3.3.4 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.

3.3.5 dump-meters switch

3.3.6 dump-meter switch [meter]

Print meter configuration. meter can specify a single meter with syntax meter=id, or all meters with syntax meter=all.

3.3.7 meter-stats switch [meter]

Print meter statistics. meter can specify a single meter with syntax meter=id, or all meters with syntax meter=all.

3.3.8 meter-features switch

Print meter features.

3.4 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 transaction, 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 in_port actions packet...
```

Connects to switch and instructs it to execute the OpenFlow actions on each packet. Each packet is specified as a series of hex digits. 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.

3.5 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.

```
add-group switch group
add-group switch - < file
add-groups switch file
```

Add each group entry to switch's tables.

```
[--may-create] mod-group switch group
[--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.

```
del-groups switch
del-groups switch [group]
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.

```
insert-buckets switch group
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.

```
remove-buckets switch group
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.

3.6 OpenFlow Switch Monitoring Commands

3.6.1 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 //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.

3.6.2 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 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.

3.7 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.

3.7.1 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.

3.7.2 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.

3.7.3 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.

3.8 Other Commands

3.8.1 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.

3.8.2 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.

3.9 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 the L4 fields tcp_dst and tcp_src. 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
ip_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).

ip_dscp=dscp

Matches IP ToS/DSCP or IPv6 traffic class field dscp, which is specified as a decimal number between 0 and 63, inclusive.

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

nw_ecn=ecn
ip_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).

tcp_src=port
tcp_dst=port
udp_src=port
udp_dst=port
sctp_src=port
sctp_dst=port

Matches a TCP, UDP, or SCTP source or destination port port, which is specified as a decimal number between 0 and 65535, inclusive.

When dl_type and nw_proto are wildcarded or set to values that do not indicate an appropriate protocol, the values of these settings are ignored (see Flow Syntax above).

tcp_src=port/mask
tcp_dst=port/mask
udp_src=port/mask
udp_dst=port/mask
sctp_src=port/mask
sctp_dst=port/mask

Bitwise match on TCP (or UDP or SCTP) source or destination port. 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
10xxxxxxxxx
110xxxxxxxx
1110xxxxxxx
11110xxxxxx
1111100xxxx
```

which become the following when written in the syntax required by ovs-ofctl:

```
tcp,tcp_src=0x03e8/0xfff8
tcp,tcp_src=0x03f0/0xfff0
tcp,tcp_src=0x0400/0xfe00
tcp,tcp_src=0x0600/0xff00
tcp,tcp_src=0x0700/0xff80
tcp,tcp_src=0x0780/0xffc0
tcp,tcp_src=0x07c0/0xfff0
```

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

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

tp_src=port
tp_dst=port

These are deprecated generic forms of L4 port matches. In new code, please use the TCP-, UDP-, or SCTP-specific forms described above.

tcp_flags=flags/mask
tcp_flags=[+flag...][-flag...]

Bitwise match on TCP flags. The flags 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 flags must match. Each 0-bit in mask causes the corresponding bit to be ignored.

Alternatively, the flags can be specified by their symbolic names (listed below), each preceded by either + for a flag that must be set, or - for a flag that must be unset, without any other delimiters between the flags. Flags not mentioned are wildcarded. For example, tcp,tcp_flags=+syn-ack matches TCP SYNs that are not ACKs.

TCP protocol currently defines 9 flag bits, and additional 3 bits are reserved (must be transmitted as zero), see RFCs 793, 3168, and 3540. The flag bits are, numbering from the least significant bit:

0: fin No more data from sender.

1: syn Synchronize sequence numbers.

2: rst Reset the connection.

3: psh Push function.

4: ack Acknowledgement field significant.

5: urg Urgent pointer field significant.

6: ece ECN Echo.

7: cwr Congestion Windows Reduced.

8: ns Nonce Sum.

9-11: Reserved.

12-15: Not matchable, must be zero.

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

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.

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.

ipv6 Same as dl_type=0x86dd.

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

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

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

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

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

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

sctp Same as dl_type=0x0800,nw_proto=132.

sctp6 Same as dl_type=0x86dd,nw_proto=132.

arp Same as dl_type=0x0806.

rarp Same as dl_type=0x8035.

mpls Same as dl_type=0x8847.

mplsm Same as dl_type=0x8848.

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 packets without an 802.1Q header.

```
              vlan_tci=0x1000/0x1000
                     Match packets with an 802.1Q header, regardless  of  VLAN
                     and priority values.

              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 prior-
                     ity 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_spa=ip[/netmask]
       arp_tpa=ip[/netmask]
              When dl_type specifies either ARP or RARP, arp_spa  and  arp_tpa
              match  the  source  and  target  IPv4 address, respectively.  An
              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 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).

       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 (e.g. 00:0A:E4:25:6B:B0).

arp_sha=xx:xx:xx:xx:xx:xx/xx:xx:xx:xx:xx:xx
arp_tha=xx:xx:xx:xx:xx:xx/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 (e.g. 00:0A:E4:25:6B:B0), with a wildcard mask following the slash.

arp_op=opcode

When dl_type specifies either ARP or RARP, arp_op matches the ARP opcode. Only ARP opcodes between 1 and 255 should be specified for matching.

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.

mpls_bos=bos

When dl_type is 0x8847 or 0x8848 (possibly via shorthand e.g., mpls or mplsm), matches the bottom-of-stack bit of the outermost MPLS label stack entry. Valid values are 0 and 1.

If 1 then for a packet with a well-formed MPLS label stack the bottom-of-stack bit indicates that the outer label stack entry is also the inner-most label stack entry and thus that is that there is only one label stack entry present. Conversely, if 0 then for a packet with a well-formed MPLS label stack the bottom-of-stack bit indicates that the outer label stack entry is not the inner-most label stack entry and thus there is more than one label stack entry present.

mpls_label=label

When dl_type is 0x8847 or 0x8848 (possibly via shorthand e.g., mpls or mplsm), matches the label of the outer MPLS label stack entry. The label is a 20-bit value that is decimal by default; use a 0x prefix to specify them in hexadecimal.

mpls_tc=tc

When dl_type is 0x8847 or 0x8848 (possibly via shorthand e.g., mpls or mplsm), matches the traffic-class of the outer MPLS label stack entry. Valid values are between 0 (lowest) and 7 (highest).

regidx=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 actions change register values.

xregidx=value[/mask]

Matches value either exactly or with optional mask in 64-bit ``extended register'' number idx. Each of the 64-bit extended registers overlays two of the 32-bit registers: xreg0 overlays reg0 and reg1, with reg0 supplying the most-significant bits of xreg0 and reg1 the least-significant. xreg1 similarly overlays reg2 and reg3, and so on.

These fields were added in Open vSwitch 2.3 to conform with the OpenFlow 1.5 specification. OpenFlow 1.5 calls these fields just the ``packet registers,'' but Open vSwitch already had 32-bit registers by that name, which is why Open vSwitch refers to the standard registers as ``extended registers''.

pkt_mark=value[/mask]

Matches packet metadata mark value either exactly or with optional mask. The mark is associated data that may be passed into other system components in order to facilitate interaction between subsystems. On Linux this corresponds to the skb mark but the exact implementation is platform-dependent.

actset_output=port

Matches the output port currently in the OpenFlow action set, where port may be an OpenFlow port number or keyword (e.g. LOCAL). If there is no output port in the OpenFlow action set, or if the output port will be ignored (e.g. because there is an output group in the OpenFlow action set), then the value will be UNSET.

This field was introduced in Open vSwitch 2.4 to conform with the OpenFlow 1.5 specification.

conj_id=value

Matches the given 32-bit value against the conjunction ID. This is used only with the conjunction action (see below).

This field was introduced in Open vSwitch 2.4.

ct_state=flags/mask
ct_state=[+flag...][-flag...]

Bitwise match on connection state flags. This is used with the ct action (see below).

The ct_state field provides information from a connection tracking module. It describes whether the packet has previously traversed the connection tracker (tracked, or trk) and, if it has been tracked, any additional information that the connection tracker was able to provide about the connection that the current packet belongs to.

Individual packets may be in one of two states: Untracked or tracked. When the ct action is executed on a packet, it becomes tracked for the the remainder of OpenFlow pipeline processing. Once a packet has become tracked, the state of its corresponding connection may be determined. Note that the ct_state is only significant for the current ct_zone.

Connections may be in one of two states: uncommitted or committed. Connections are uncommitted by default. To determine ongoing information about a connection, like whether the connection is established or not, the connection must be committed. When the ct action is executed on a packet with the commit parameter, the connection will become committed and will remain in this state until the end of the connection. Committed connections store state beyond the duration of packet processing.

The flags and mask are 32-bit numbers written in decimal or in hexadecimal prefixed by 0x. Each 1-bit in mask requires that the corresponding bit in flags must match. Each 0-bit in mask causes the corresponding bit to be ignored.

Alternatively, the flags can be specified by their symbolic names (listed below), each preceded by either + for a flag that must be set, or - for a flag that must be unset, without any other delimiters between the flags. Flags not mentioned are wildcarded. For example, tcp,ct_state=+trk-new matches TCP packets that have been run through the connection tracker and do not establish a new connection.

The following flags describe the state of the tracking:

0x01: new

This is the beginning of a new connection. This flag may only be present for uncommitted connections.

0x02: est

This is part of an already existing connection. This flag may only be present for committed connections.

0x04: rel

This is a connection that is related to an existing connection, for instance ICMP "destination unreachable" messages or FTP data connections. This flag may only be present for committed connections.

0x08: rpl

The flow is in the reply direction, meaning it did not initiate the connection. This flag may only be present for committed connections.

0x10: inv

The state is invalid, meaning that the connection tracker couldn't identify the connection. This flag is a catch-all for any problems that the connection tracker may have, for example:

- L3/L4 protocol handler is not loaded/unavailable. With the Linux kernel datapath, this may mean that the "nf_conntrack_ipv4" or "nf_conntrack_ipv6" modules are not loaded.

- L3/L4 protocol handler determines that the packet is malformed.

- Packets are unexpected length for protocol.

0x20: trk

This packet is tracked, meaning that it has previously traversed the connection tracker. If this flag is not set, then no other flags will be set. If this flag is set, then the packet is tracked and other flags may also be set.

0x40: snat

This packet was transformed by source address/port translation by a preceding ct action.

0x80: dnat

This packet was transformed by destination address/port translation by a preceding ct action.

This field was introduced in Open vSwitch 2.5. The snat and dnat bits were added in Open vSwitch 2.6.

The following fields are associated with the connection tracker and will only be populated for tracked packets. The ct action will populate these fields, and allows modification of some of the below fields.

ct_zone=zone

Matches the given 16-bit connection zone exactly. This represents the most recent connection tracking context that ct was executed in. Each zone is an independent connection tracking context, so if you wish to track the same packet in multiple contexts then you must use the ct action multiple times. Introduced in Open vSwitch 2.5.

ct_mark=value[/mask]

Matches the given 32-bit connection mark value either exactly or with optional mask. This represents metadata associated with the connection that the current packet is part of. Introduced in Open vSwitch 2.5.

ct_label=value[/mask]

Matches the given 128-bit connection labels value either exactly or with optional mask. This represents metadata associated with the connection that the current packet is part of. Introduced in Open vSwitch 2.5.

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.

sctp6 Same as dl_type=0x86dd,nw_proto=132.

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=[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 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. 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. Group tables are only supported in OpenFlow 1.1+. 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.

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])

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 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.

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.

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. If the connection is committed, then 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. If the connection is committed, then 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. Supported types include:

ftp Look for negotiation of FTP data connections. If a subsequent FTP data connection arrives which is related, the ct action will set the rel flag in the ct_state field for packets sent through ct.

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=...)).

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.

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.

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.

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 is the NXM or OXM name for the field. For example, load:0x001122334455->OXM_OF_ETH_DST[] 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] 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.

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[].

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.

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.)

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. 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 the port. It can be set as input port or output port. When this option is omitted, or specified as NONE, IPFIX does not differentiate between ingress packets and egress packets and does not export egress tunnel information. This option was added in Open vSwitch 2.5.90.

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)

An individual OpenFlow flow can match only a single value for each field. However, situations often arise where one wants to match one of a set of values within a field or fields. For matching a single field against a set, it is straightforward and efficient to add multiple flows to the flow table, one for each value in the set. For example, one might use the following flows to send packets with IP source address a, b, c, or d to the OpenFlow controller:

```
   ip,ip_src=a actions=controller
   ip,ip_src=b actions=controller
   ip,ip_src=c actions=controller
   ip,ip_src=d actions=controller
```

Similarly, these flows send packets with IP destination address e, f, g, or h to the OpenFlow controller:

```
   ip,ip_dst=e actions=controller
   ip,ip_dst=f actions=controller
   ip,ip_dst=g actions=controller
   ip,ip_dst=h actions=controller
```

Installing all of the above flows in a single flow table yields a disjunctive effect: a packet is sent to the controller if ip_src ∈ {a,b,c,d} or ip_dst ∈ {e,f,g,h} (or both). (Pedantically, if both of the above sets of flows are present in the flow table, they should have different priorities, because OpenFlow says that the results are undefined when two flows with same priority can both match a single packet.)

Suppose, on the other hand, one wishes to match conjunctively, that is, to send a packet to the controller only if both ip_src ∈ {a,b,c,d} and ip_dst ∈ {e,f,g,h}. This requires 4 × 4 = 16 flows, one for each possible pairing of ip_src and ip_dst. That is acceptable for our small example, but it does not gracefully extend to larger sets or greater numbers of dimensions.

The conjunction action is a solution for conjunctive matches that is built into Open vSwitch. A conjunction action ties groups of individual OpenFlow flows into higher-level ``conjunctive flows''. Each group corresponds to one dimension, and each flow within the group matches one possible value for the dimension. A packet that matches one flow from each group matches the conjunctive flow.

To implement a conjunctive flow with conjunction, assign the conjunctive flow a 32-bit id, which must be unique within an OpenFlow table. Assign each of the n ≥ 2 dimensions a unique number from 1 to n; the ordering is unimportant. Add one flow to the OpenFlow flow table for each possible value of each dimension with conjunction(id, k/n) as the flow's actions, where k is the number assigned to the flow's dimension. Together, these flows specify the conjunctive flow's match condition. When the conjunctive match condition is met, Open vSwitch looks up one more flow that specifies the conjunctive flow's actions and receives its statistics. This flow is found by setting conj_id to the specified id and then again searching the flow table.

The following flows provide an example. Whenever the IP source is one of the values in the flows that match on the IP source (dimension 1 of 2), and the IP destination is one of the values in the flows that match on IP destination (dimension 2 of 2), Open vSwitch searches for a flow that matches conj_id against the conjunction ID (1234), finding the first flow listed below.

```
   conj_id=1234 actions=controller
   ip,ip_src=10.0.0.1 actions=conjunction(1234, 1/2)
   ip,ip_src=10.0.0.4 actions=conjunction(1234, 1/2)
   ip,ip_src=10.0.0.6 actions=conjunction(1234, 1/2)
   ip,ip_src=10.0.0.7 actions=conjunction(1234, 1/2)
   ip,ip_dst=10.0.0.2 actions=conjunction(1234, 2/2)
   ip,ip_dst=10.0.0.5 actions=conjunction(1234, 2/2)
   ip,ip_dst=10.0.0.7 actions=conjunction(1234, 2/2)
   ip,ip_dst=10.0.0.8 actions=conjunction(1234, 2/2)
```

Many subtleties exist:

- In the example above, every flow in a single dimension has the same form, that is, dimension 1 matches on ip_src, dimension 2 on ip_dst, but this is not a requirement. Different flows within a dimension may match on different bits within a field (e.g. IP network prefixes of different lengths, or TCP/UDP port ranges as bitwise matches), or even on entirely different fields (e.g. to match packets for TCP source port 80 or TCP destination port 80).

- The flows within a dimension can vary their matches across more than one field, e.g. to match only specific pairs of IP source and destination addresses or L4 port numbers.

- A flow may have multiple conjunction actions, with different id values. This is useful for multiple conjunctive flows with overlapping sets. If one conjunctive flow matches packets with both ip_src ∈ {a,b} and ip_dst ∈ {d,e} and a second conjunctive flow matches ip_src ∈ {b,c} and ip_dst ∈ {f,g}, for example, then the flow that matches ip_src=b would have two conjunction actions, one for each conjunctive flow. The order of conjunction actions within a list of actions is not significant.

- A flow with conjunction actions may also include note actions for annotations, but not any other kind of actions. (They would not be useful because they would never be executed.)

- All of the flows that constitute a conjunctive flow with a given id must have the same priority. (Flows with the same id but different priorities are currently treated as different conjunctive flows, that is, currently id values need only be unique within an OpenFlow table at a given priority. This behavior isn't guaranteed to stay the same in later releases, so please use id values unique within an OpenFlow table.)

- Conjunctive flows must not overlap with each other, at a given priority, that is, any given packet must be able to match at most one conjunctive flow at a given priority. Overlapping conjunctive flows yield unpredictable results.

- Following a conjunctive flow match, the search for the flow with conj_id=id is done in the same general-purpose way as other flow table searches, so one can use flows with conj_id=id to act differently depending on circumstances. (One exception is that the search for the conj_id=id flow itself ignores conjunctive flows, to avoid recursion.) If the search with conj_id=id fails, Open vSwitch acts as if the conjunctive flow had not matched at all, and continues searching the flow table for other matching flows.

- OpenFlow prerequisite checking occurs for the flow with conj_id=id in the same way as any other flow, e.g. in an OpenFlow 1.1+ context, putting a mod_nw_src action into the example above would require adding an ip match, like this:

  ```
  conj_id=1234,ip actions=mod_nw_src:1.2.3.4,controller
  ```

- OpenFlow prerequisite checking also occurs for the individual flows that comprise a conjunctive match in the same way as any other flow.

- The flows that constitute a conjunctive flow do not have useful statistics. They are never updated with byte or packet counts, and so on. (For such a flow, therefore, the idle and hard timeouts work much the same way.)

- Conjunctive flows can be a useful building block for negation, that is, inequality matches like tcp_src ≠ 80. To implement an inequality match, convert it to a pair of range matches, e.g. 0 ≤ tcp_src < 80 and 80 < tcp_src ≤ 65535, then convert each of the range matches into a collection of bitwise matches as explained above in the description of tcp_src.

- Sometimes there is a choice of which flows include a particular match. For example, suppose that we added an extra constraint to our example, to match on ip_src ∈ {a,b,c,d} and ip_dst ∈ {e,f,g,h} and tcp_dst = i. One way to implement this is to add the new constraint to the conj_id flow, like this:

  ```
  conj_id=1234,tcp,tcp_dst=i
  actions=mod_nw_src:1.2.3.4,controller
  ```

  but this is not recommended because of the cost of the extra flow table lookup. Instead, add the constraint to the individual flows, either in one of the dimensions or (slightly better) all of them.

- A conjunctive match must have n ≥ 2 dimensions (otherwise a conjunctive match is not necessary). Open vSwitch enforces this.

- Each dimension within a conjunctive match should ordinarily have more than one flow. Open vSwitch does not enforce this.

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

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.

3.10 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 buckets 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 buckets 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.

Other than the empty string, hash is currently the only defined selection method.

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 Open-

Flow 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 a non-empty string. 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. Specyfing 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.

3.11 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 honored.

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.

3.12 OPTIONS

--strict

Uses strict matching when running flow modification 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  exten-
                sible.   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  collec-
         tions 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 ver-
                sion 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  avail-
                able 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 //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 interact with 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 //var/run/openvswitch. If --unixctl is not used at all, the default socket is //var/run/openvswitch/ovs-ofctl.pid.ctl, where pid is ovs-ofctl's process ID.

On Windows, uses a kernel chosen TCP port on the localhost to listen for runtime management commands. The kernel chosen TCP port value is written in a file whose absolute path is pointed by socket. If --unixctl is not used at all, the file is created as ovs-ofctl.ctl in the configured OVS_RUNDIR directory.

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 //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 fol-
                           lowed  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 fore-
                           ground.

              See the Select Graphic Rendition (SGR) section in the documenta-
              tion  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/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-flows br0
              Prints the flow entries in the switch.
```

4 ovs-vsctl

NAME

ovs-vsctl - utility for querying and configuring ovs-vswitchd

SYNOPSIS

ovs-vsctl [options] -- [options] command [args] [-- [options] command [args]]...

DESCRIPTION

The ovs-vsctl program configures ovs-vswitchd(8) by providing a high-level interface to its configuration database. See ovs-vswitchd.conf.db(5) for comprehensive documentation of the database schema.

ovs-vsctl connects to an ovsdb-server process that maintains an Open vSwitch configuration database. Using this connection, it queries and possibly applies changes to the database, depending on the supplied commands. Then, if it applied any changes, by default it waits until ovs-vswitchd has finished reconfiguring itself before it exits. (If you use ovs-vsctl when ovs-vswitchd is not running, use --no-wait.)

ovs-vsctl can perform any number of commands in a single run, implemented as a single atomic transaction against the database.

The ovs-vsctl command line begins with global options (see OPTIONS below for details). The global options are followed by one or more commands. Each command should begin with -- by itself as a command-line argument, to separate it from the following commands. (The -- before the first command is optional.) The command itself starts with command-specific options, if any, followed by the command name and any arguments. See EXAMPLES below for syntax examples.

Linux VLAN Bridging Compatibility

The ovs-vsctl program supports the model of a bridge implemented by Open vSwitch, in which a single bridge supports ports on multiple VLANs. In this model, each port on a bridge is either a trunk port that potentially passes packets tagged with 802.1Q headers that designate VLANs or it is assigned a single implicit VLAN that is never tagged with an 802.1Q header.

For compatibility with software designed for the Linux bridge, ovs-vsctl also supports a model in which traffic associated with a given 802.1Q VLAN is segregated into a separate bridge. A special form of the add-br command (see below) creates a ``fake bridge'' within an Open vSwitch bridge to simulate this behavior. When such a ``fake bridge'' is active, ovs-vsctl will treat it much like a bridge separate from its ``parent bridge,'' but the actual implementation in Open vSwitch uses only a single bridge, with ports on the fake bridge assigned the implicit VLAN of the fake bridge of which they are members. (A fake bridge for VLAN 0 receives packets that have no 802.1Q tag or a tag with VLAN 0.)

OPTIONS

The following options affect the behavior ovs-vsctl as a whole. Some individual commands also accept their own options, which are given just before the command name. If the first command on the command line has

```
options,  then  those options must be separated from the global options
by --.

--db=server
       Sets server as the database server that  ovs-vsctl  contacts  to
       query    or    modify    configuration.     The    default    is
       unix://var/run/openvswitch/db.sock.  server must take one of the
       following forms:

       ssl:ip:port
              The specified SSL port on the host at the given ip, which
              must be expressed as an IP address (not a  DNS  name)  in
              IPv4  or  IPv6 address format.  If ip is an IPv6 address,
              then wrap ip with square brackets, e.g.:  ssl:[::1]:6640.
              The  --private-key,  --certificate, and --ca-cert options
              are mandatory when this form is used.

       tcp:ip:port
              Connect to the given TCP port on ip, where ip can be IPv4
              or  IPv6  address. If ip is an IPv6 address, then wrap ip
              with square brackets, e.g.: tcp:[::1]:6640.

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

              On  Windows,  connect to a localhost TCP port whose value
              is written in file.

       pssl:port[:ip]
              Listen  on the  given  SSL  port  for  a  connection.   By
              default,  connections are not bound to a particular local
              IP address and it listens only on  IPv4  (but  not  IPv6)
              addresses,  but specifying ip limits connections to those
              from the given ip, either IPv4 or IPv6 address.  If ip is
              an IPv6 address, then wrap ip with square brackets, e.g.:
              pssl:6640:[::1].  The --private-key,  --certificate,  and
              --ca-cert options are mandatory when this form is used.

       ptcp:port[:ip]
              Listen  on  the  given  TCP  port  for  a connection.  By
              default, connections are not bound to a particular  local
              IP  address  and  it  listens only on IPv4 (but not IPv6)
              addresses, but ip may be specified  to  listen  only  for
              connections to the given ip, either IPv4 or IPv6 address.
              If ip is an IPv6 address, then wrap ip with square brack-
              ets, e.g.: ptcp:6640:[::1].

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

              On Windows, listen on a kernel chosen  TCP  port  on  the
              localhost. The kernel chosen TCP port value is written in
              file.

--no-wait
       Prevents ovs-vsctl from waiting for ovs-vswitchd to  reconfigure
       itself  according  to the modified database.  This option should
       be used if ovs-vswitchd is  not  running;  otherwise,  ovs-vsctl
       will not exit until ovs-vswitchd starts.
```

This option has no effect if the commands specified do not change the database.

--no-syslog

By default, ovs-vsctl logs its arguments and the details of any changes that it makes to the system log. This option disables this logging.

This option is equivalent to --verbose=vsctl:syslog:warn.

--oneline

Modifies the output format so that the output for each command is printed on a single line. New-line characters that would otherwise separate lines are printed as \n, and any instances of \ that would otherwise appear in the output are doubled. Prints a blank line for each command that has no output. This option does not affect the formatting of output from the list or find commands; see Table Formatting Options below.

--dry-run

Prevents ovs-vsctl from actually modifying the database.

-t secs
--timeout=secs

By default, or with a secs of 0, ovs-vsctl waits forever for a response from the database. This option limits runtime to approximately secs seconds. If the timeout expires, ovs-vsctl will exit with a SIGALRM signal. (A timeout would normally happen only if the database cannot be contacted, or if the system is overloaded.)

--retry

Without this option, if ovs-vsctl connects outward to the database server (the default) then ovs-vsctl will try to connect once and exit with an error if the connection fails (which usually means that ovsdb-server is not running).

With this option, or if --db specifies that ovs-vsctl should listen for an incoming connection from the database server, then ovs-vsctl will wait for a connection to the database forever.

Regardless of this setting, --timeout always limits how long ovs-vsctl will wait.

Table Formatting Options

These options control the format of output from the list and find commands.

-f format
--format=format

Sets the type of table formatting. The following types of format are available:

table 2-D text tables with aligned columns.

list (default)

A list with one column per line and rows separated by a blank line.

html HTML tables.

csv — Comma-separated values as defined in RFC 4180.

json — JSON format as defined in RFC 4627. The output is a sequence of JSON objects, each of which corresponds to one table. Each JSON object has the following members with the noted values:

caption
The table's caption. This member is omitted if the table has no caption.

headings
An array with one element per table column. Each array element is a string giving the corresponding column's heading.

data — An array with one element per table row. Each element is also an array with one element per table column. The elements of this second-level array are the cells that constitute the table. Cells that represent OVSDB data or data types are expressed in the format described in the OVSDB specification; other cells are simply expressed as text strings.

-d format
--data=format
Sets the formatting for cells within output tables. The following types of format are available:

string (default)
The simple format described in the Database Values section below.

bare — The simple format with punctuation stripped off: [] and {} are omitted around sets, maps, and empty columns, items within sets and maps are space-separated, and strings are never quoted. This format may be easier for scripts to parse.

json — JSON.

The json output format always outputs cells in JSON format, ignoring this option.

--no-heading
This option suppresses the heading row that otherwise appears in the first row of table output.

--pretty
By default, JSON in output is printed as compactly as possible. This option causes JSON in output to be printed in a more readable fashion. Members of objects and elements of arrays are printed one per line, with indentation.

This option does not affect JSON in tables, which is always printed compactly.

--bare Equivalent to --format=list --data=bare --no-headings.

Public Key Infrastructure Options

-p privkey.pem
--private-key=privkey.pem

Specifies a PEM file containing the private key used as ovs-vsctl'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-vsctl should use to verify certificates presented to it by SSL peers. (This may be the same certificate that SSL peers use to verify the certificate specified on -c or --certificate, or it may be a different one, depending on the PKI design in use.)

-C none
--ca-cert=none

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

--bootstrap-ca-cert=cacert.pem

When cacert.pem exists, this option has the same effect as -C or --ca-cert. If it does not exist, then ovs-vsctl will attempt to obtain the CA certificate from the SSL peer on its first SSL connection and save it to the named PEM file. If it is successful, it will immediately drop the connection and reconnect, and from then on all SSL connections must be authenticated by a certificate signed by the CA certificate thus obtained.

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

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

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

--peer-ca-cert=peer-cacert.pem

Specifies a PEM file that contains one or more additional certificates to send to SSL peers. peer-cacert.pem should be the CA certificate used to sign ovs-vsctl's own certificate, that is, the certificate specified on -c or --certificate. If ovs-vsctl's certificate is self-signed, then --certificate and --peer-ca-cert should specify the same file.

This option is not useful in normal operation, because the SSL peer must already have the CA certificate for the peer to have any confidence in ovs-vsctl's identity. However, this offers a way for a new installation to bootstrap the CA certificate on its first SSL connection.

-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-vsctl 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 //var/log/open-vswitch/ovs-vsctl.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.

-h
--help Prints a brief help message to the console.

-V
--version
Prints version information to the console.

COMMANDS

The commands implemented by ovs-vsctl are described in the sections below.

4.1 Open vSwitch Commands

These commands work with an Open vSwitch as a whole.

init Initializes the Open vSwitch database, if it is empty. If the database has already been initialized, this command has no effect.

Any successful ovs-vsctl command automatically initializes the Open vSwitch database if it is empty. This command is provided to initialize the database without executing any other command.

show Prints a brief overview of the database contents.

emer-reset

Reset the configuration into a clean state. It deconfigures OpenFlow controllers, OVSDB servers, and SSL, and deletes port mirroring, fail_mode, NetFlow, sFlow, and IPFIX configuration.

This command also removes all other-config keys from all database records, except that other-config:hwaddr is preserved if it is present in a Bridge record. Other networking configuration is left as-is.

4.2 Bridge Commands

These commands examine and manipulate Open vSwitch bridges.

[--may-exist] add-br bridge

Creates a new bridge named bridge. Initially the bridge will have no ports (other than bridge itself).

Without --may-exist, attempting to create a bridge that exists is an error. With --may-exist, this command does nothing if bridge already exists as a real bridge.

[--may-exist] add-br bridge parent vlan

Creates a ``fake bridge'' named bridge within the existing Open vSwitch bridge parent, which must already exist and must not itself be a fake bridge. The new fake bridge will be on 802.1Q VLAN vlan, which must be an integer between 0 and 4095. The parent bridge must not already have a fake bridge for vlan. Initially bridge will have no ports (other than bridge itself).

Without --may-exist, attempting to create a bridge that exists is an error. With --may-exist, this command does nothing if bridge already exists as a VLAN bridge under parent for vlan.

[--if-exists] del-br bridge

Deletes bridge and all of its ports. If bridge is a real bridge, this command also deletes any fake bridges that were created with bridge as parent, including all of their ports.

Without --if-exists, attempting to delete a bridge that does not exist is an error. With --if-exists, attempting to delete a bridge that does not exist has no effect.

[--real|--fake] list-br

Lists all existing real and fake bridges on standard output, one per line. With --real or --fake, only bridges of that type are returned.

br-exists bridge

Tests whether bridge exists as a real or fake bridge. If so, ovs-vsctl exits successfully with exit code 0. If not, ovs-vsctl exits unsuccessfully with exit code 2.

br-to-vlan bridge

If bridge is a fake bridge, prints the bridge's 802.1Q VLAN as a decimal integer. If bridge is a real bridge, prints 0.

br-to-parent bridge

If bridge is a fake bridge, prints the name of its parent bridge. If bridge is a real bridge, print bridge.

br-set-external-id bridge key [value]

Sets or clears an ``external ID'' value on bridge. These values are intended to identify entities external to Open vSwitch with which bridge is associated, e.g. the bridge's identifier in a virtualization management platform. The Open vSwitch database schema specifies well-known key values, but key and value are otherwise arbitrary strings.

If value is specified, then key is set to value for bridge, overwriting any previous value. If value is omitted, then key is removed from bridge's set of external IDs (if it was present).

For real bridges, the effect of this command is similar to that of a set or remove command in the external-ids column of the Bridge table. For fake bridges, it actually modifies keys with names prefixed by fake-bridge- in the Port table.

br-get-external-id bridge [key]

Queries the external IDs on bridge. If key is specified, the output is the value for that key or the empty string if key is unset. If key is omitted, the output is key=value, one per line, for each key-value pair.

For real bridges, the effect of this command is similar to that of a get command in the external-ids column of the Bridge table. For fake bridges, it queries keys with names prefixed by fake-bridge- in the Port table.

4.3 Port Commands

These commands examine and manipulate Open vSwitch ports. These commands treat a bonded port as a single entity.

list-ports bridge

Lists all of the ports within bridge on standard output, one per line. The local port bridge is not included in the list.

[--may-exist] add-port bridge port [column[:key]=value]...

Creates on bridge a new port named port from the network device of the same name.

Optional arguments set values of column in the Port record created by the command. For example, tag=9 would make the port an access port for VLAN 9. The syntax is the same as that for the set command (see Database Commands below).

Without --may-exist, attempting to create a port that exists is an error. With --may-exist, this command does nothing if port already exists on bridge and is not a bonded port.

[--fake-iface] add-bond bridge port iface... [column[:key]=value]...

Creates on bridge a new port named port that bonds together the network devices given as each iface. At least two interfaces must be named. If the interfaces are DPDK enabled then the transaction will need to include operations to explicitly set the interface type to 'dpdk'.

Optional arguments set values of column in the Port record created by the command. The syntax is the same as that for the set command (see Database Commands below).

With --fake-iface, a fake interface with the name port is created. This should only be used for compatibility with legacy software that requires it.

Without --may-exist, attempting to create a port that exists is an error. With --may-exist, this command does nothing if port already exists on bridge and bonds together exactly the specified interfaces.

[--if-exists] del-port [bridge] port

Deletes port. If bridge is omitted, port is removed from whatever bridge contains it; if bridge is specified, it must be the real or fake bridge that contains port.

Without --if-exists, attempting to delete a port that does not exist is an error. With --if-exists, attempting to delete a port that does not exist has no effect.

[--if-exists] --with-iface del-port [bridge] iface

Deletes the port named iface or that has an interface named iface. If bridge is omitted, the port is removed from whatever bridge contains it; if bridge is specified, it must be the real or fake bridge that contains the port.

Without --if-exists, attempting to delete the port for an interface that does not exist is an error. With --if-exists, attempting to delete the port for an interface that does not exist has no effect.

port-to-br port

Prints the name of the bridge that contains port on standard output.

4.4 Interface Commands

These commands examine the interfaces attached to an Open vSwitch bridge. These commands treat a bonded port as a collection of two or more interfaces, rather than as a single port.

list-ifaces bridge

Lists all of the interfaces within bridge on standard output, one per line. The local port bridge is not included in the list.

iface-to-br iface

Prints the name of the bridge that contains iface on standard output.

4.5 OpenFlow Controller Connectivity

ovs-vswitchd can perform all configured bridging and switching locally, or it can be configured to communicate with one or more external OpenFlow controllers. The switch is typically configured to connect to a primary controller that takes charge of the bridge's flow table to implement a network policy. In addition, the switch can be configured to listen to connections from service controllers. Service controllers are typically used for occasional support and maintenance, e.g. with ovs-ofctl.

get-controller bridge

Prints the configured controller target.

del-controller bridge

Deletes the configured controller target.

set-controller bridge target...

Sets the configured controller target or targets. Each target may use any of the following forms:

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, a localhost TCP port written in file.

pssl:[port][:ip]
ptcp:[port][:ip]

Listens for OpenFlow connections on port. The default port is 6653. By default, connections are allowed from any IPv4 address. Specify ip as an IPv4 address or a bracketed IPv6 address (e.g. ptcp:6653:[::1]). DNS names may not be used. For pssl, the --private-key,--certificate, and --ca-cert options are mandatory.

punix:file

Listens for OpenFlow connections on the Unix domain server socket named file.

4.6 Controller Failure Settings

When a controller is configured, it is, ordinarily, responsible for setting up all flows on the switch. Thus, if the connection to the controller fails, no new network connections can be set up. If the connection to the controller stays down long enough, no packets can pass through the switch at all.

If the value is standalone, or if neither of these settings is set, ovs-vswitchd will take over responsibility for setting up flows when no message has been received from the controller for three times the inactivity probe interval. In this mode, ovs-vswitchd causes the datapath to act like an ordinary MAC-learning switch. ovs-vswitchd will continue to retry connecting to the controller in the background and, when the connection succeeds, it discontinues its standalone behavior.

If this option is set to secure, ovs-vswitchd will not set up flows on its own when the controller connection fails.

get-fail-mode bridge

Prints the configured failure mode.

del-fail-mode bridge

Deletes the configured failure mode.

set-fail-mode bridge standalone|secure

Sets the configured failure mode.

4.7 Manager Connectivity

These commands manipulate the manager_options column in the Open_vSwitch table and rows in the Managers table. When ovsdb-server is configured to use the manager_options column for OVSDB connections (as described in INSTALL.Linux and in the startup scripts provided with Open vSwitch), this allows the administrator to use ovs-vsctl to configure database connections.

get-manager

Prints the configured manager(s).

del-manager

Deletes the configured manager(s).

set-manager target...

Sets the configured manager target or targets. Each target may use any of the following forms:

ssl:ip:port

The specified SSL port on the host at the given ip, which must be expressed as an IP address (not a DNS name) in IPv4 or IPv6 address format. If ip is an IPv6 address, then wrap ip with square brackets, e.g.: ssl:[::1]:6640. The --private-key, --certificate, and --ca-cert options are mandatory when this form is used.

tcp:ip:port

Connect to the given TCP port on ip, where ip can be IPv4 or IPv6 address. If ip is an IPv6 address, then wrap ip with square brackets, e.g.: tcp:[::1]:6640.

unix:file

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

On Windows, connect to a localhost TCP port whose value is written in file.

pssl:port[:ip]

Listen on the given SSL port for a connection. By default, connections are not bound to a particular local IP address and it listens only on IPv4 (but not IPv6) addresses, but specifying ip limits connections to those from the given ip, either IPv4 or IPv6 address. If ip is an IPv6 address, then wrap ip with square brackets, e.g.: pssl:6640:[::1]. The --private-key, --certificate, and --ca-cert options are mandatory when this form is used.

ptcp:port[:ip]

Listen on the given TCP port for a connection. By default, connections are not bound to a particular local IP address and it listens only on IPv4 (but not IPv6) addresses, but ip may be specified to listen only for connections to the given ip, either IPv4 or IPv6 address.

If ip is an IPv6 address, then wrap ip with square brackets, e.g.: ptcp:6640:[::1].

punix:file

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

On Windows, listen on a kernel chosen TCP port on the localhost. The kernel chosen TCP port value is written in file.

4.8 SSL Configuration

When ovs-vswitchd is configured to connect over SSL for management or controller connectivity, the following parameters are required:

private-key

Specifies a PEM file containing the private key used as the virtual switch's identity for SSL connections to the controller.

certificate

Specifies a PEM file containing a certificate, signed by the certificate authority (CA) used by the controller and manager, that certifies the virtual switch's private key, identifying a trustworthy switch.

ca-cert

Specifies a PEM file containing the CA certificate used to verify that the virtual switch is connected to a trustworthy controller.

These files are read only once, at ovs-vswitchd startup time. If their contents change, ovs-vswitchd must be killed and restarted.

These SSL settings apply to all SSL connections made by the virtual switch.

get-ssl

Prints the SSL configuration.

del-ssl

Deletes the current SSL configuration.

[--bootstrap] set-ssl private-key certificate ca-cert

Sets the SSL configuration. The --bootstrap option is described below.

CA Certificate Bootstrap

Ordinarily, all of the files named in the SSL configuration must exist when ovs-vswitchd starts. However, if the ca-cert file does not exist and the --bootstrap option is given, then ovs-vswitchd will attempt to obtain the CA certificate from the controller on its first SSL connection and save it to the named PEM file. If it is successful, it will immediately drop the connection and reconnect, and from then on all SSL connections must be authenticated by a certificate signed by the CA certificate thus obtained.

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

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

4.9 Auto-Attach Commands

The IETF Auto-Attach SPBM draft standard describes a compact method of using IEEE 802.1AB Link Layer Discovery Protocol (LLDP) together with a IEEE 802.1aq Shortest Path Bridging (SPB) network to automatically attach network devices to individual services in a SPB network. The intent here is to allow network applications and devices using OVS to be able to easily take advantage of features offered by industry standard SPB networks. A fundamental element of the Auto-Attach feature is to map traditional VLANs onto SPB I_SIDs. These commands manage the Auto-Attach I-SID/VLAN mappings.

add-aa-mapping bridge i-sid vlan
    Creates a new Auto-Attach mapping on bridge for i-sid and vlan.

del-aa-mapping bridge i-sid vlan
    Deletes an Auto-Attach mapping on bridge for i-sid and vlan.

get-aa-mapping bridge
    Lists all of the Auto-Attach mappings within bridge on standard output.

4.10 Database Commands

These commands query and modify the contents of ovsdb tables. They are a slight abstraction of the ovsdb interface and as such they operate at a lower level than other ovs-vsctl commands.

Identifying Tables, Records, and Columns

Each of these commands has a table parameter to identify a table within the database. Many of them also take a record parameter that identifies a particular record within a table. The record parameter may be the UUID for a record, and many tables offer additional ways to identify records. Some commands also take column parameters that identify a particular field within the records in a table.

The following tables are currently defined:

Open_vSwitch
    Global configuration for an ovs-vswitchd. This table contains exactly one record, identified by specifying . as the record name.

Bridge Configuration for a bridge within an Open vSwitch. Records may be identified by bridge name.

Port A bridge port. Records may be identified by port name.

Interface
    A network device attached to a port. Records may be identified by name.

Flow_Table
Configuration for a particular OpenFlow flow table. Records may be identified by name.

QoS
Quality-of-service configuration for a Port. Records may be identified by port name.

Queue
Configuration for one queue within a QoS configuration. Records may only be identified by UUID.

Mirror
A port mirroring configuration attached to a bridge. Records may be identified by mirror name.

Controller
Configuration for an OpenFlow controller. A controller attached to a particular bridge may be identified by the bridge's name.

Manager
Configuration for an OVSDB connection. Records may be identified by target (e.g. tcp:1.2.3.4).

NetFlow
A NetFlow configuration attached to a bridge. Records may be identified by bridge name.

SSL
The global SSL configuration for ovs-vswitchd. The record attached to the Open_vSwitch table may be identified by specifying . as the record name.

sFlow
An sFlow exporter configuration attached to a bridge. Records may be identified by bridge name.

IPFIX
An IPFIX exporter configuration attached to a bridge. Records may be identified by bridge name.

Flow_Sample_Collector_Set
An IPFIX exporter configuration attached to a bridge for sampling packets on a per-flow basis using OpenFlow sample actions.

AutoAttach
Configuration for Auto Attach within a bridge.

Record names must be specified in full and with correct capitalization. Names of tables and columns are not case-sensitive, and -- and _ are treated interchangeably. Unique abbreviations are acceptable, e.g. net or n is sufficient to identify the NetFlow table.

Database Values

Each column in the database accepts a fixed type of data. The currently defined basic types, and their representations, are:

integer
A decimal integer in the range -2**63 to 2**63-1, inclusive.

real
A floating-point number.

Boolean
True or false, written true or false, respectively.

string An arbitrary Unicode string, except that null bytes are not allowed. Quotes are optional for most strings that begin with an English letter or underscore and consist only of letters, underscores, hyphens, and periods. However, true and false and strings that match the syntax of UUIDs (see below) must be enclosed in double quotes to distinguish them from other basic types. When double quotes are used, the syntax is that of strings in JSON, e.g. backslashes may be used to escape special characters. The empty string must be represented as a pair of double quotes ("").

UUID Either a universally unique identifier in the style of RFC 4122, e.g. f81d4fae-7dec-11d0-a765-00a0c91e6bf6, or an @name defined by a get or create command within the same ovs-vsctl invocation.

Multiple values in a single column may be separated by spaces or a single comma. When multiple values are present, duplicates are not allowed, and order is not important. Conversely, some database columns can have an empty set of values, represented as [], and square brackets may optionally enclose other non-empty sets or single values as well.

A few database columns are ``maps'' of key-value pairs, where the key and the value are each some fixed database type. These are specified in the form key=value, where key and value follow the syntax for the column's key type and value type, respectively. When multiple pairs are present (separated by spaces or a comma), duplicate keys are not allowed, and again the order is not important. Duplicate values are allowed. An empty map is represented as {}. Curly braces may optionally enclose non-empty maps as well (but use quotes to prevent the shell from expanding other-config={0=x,1=y} into other-config=0=x other-config=1=y, which may not have the desired effect).

Database Command Syntax

[--if-exists] [--columns=column[,column]...] list table [record]...

Lists the data in each specified record. If no records are specified, lists all the records in table.

If --columns is specified, only the requested columns are listed, in the specified order. Otherwise, all columns are listed, in alphabetical order by column name.

Without --if-exists, it is an error if any specified record does not exist. With --if-exists, the command ignores any record that does not exist, without producing any output.

[--columns=column[,column]...] find table [column[:key]=value]...

Lists the data in each record in table whose column equals value or, if key is specified, whose column contains a key with the specified value. The following operators may be used where = is written in the syntax summary:

= != < > <= >=

Selects records in which column[:key] equals, does not equal, is less than, is greater than, is less than or equal to, or is greater than or equal to value, respectively.

Consider column[:key] and value as sets of elements. Identical sets are considered equal. Otherwise, if the sets have different numbers of elements, then the set with more elements is considered to be larger. Otherwise, consider a element from each set pairwise, in increasing order within each set. The first pair that differs determines the result. (For a column that contains key-value pairs, first all the keys are compared, and values are considered only if the two sets contain identical keys.)

{=} {!=}
Test for set equality or inequality, respectively.

{<=} Selects records in which column[:key] is a subset of value. For example, flood-vlans{<=}1,2 selects records in which the flood-vlans column is the empty set or contains 1 or 2 or both.

{<} Selects records in which column[:key] is a proper subset of value. For example, flood-vlans{<}1,2 selects records in which the flood-vlans column is the empty set or contains 1 or 2 but not both.

{>=} {>}
Same as {<=} and {<}, respectively, except that the relationship is reversed. For example, flood-vlans{>=}1,2 selects records in which the flood-vlans column contains both 1 and 2.

For arithmetic operators (= != < > <= >=), when key is specified but a particular record's column does not contain key, the record is always omitted from the results. Thus, the condition other-config:mtu!=1500 matches records that have a mtu key whose value is not 1500, but not those that lack an mtu key.

For the set operators, when key is specified but a particular record's column does not contain key, the comparison is done against an empty set. Thus, the condition other-config:mtu{!=}1500 matches records that have a mtu key whose value is not 1500 and those that lack an mtu key.

Don't forget to escape < or > from interpretation by the shell.

If --columns is specified, only the requested columns are listed, in the specified order. Otherwise all columns are listed, in alphabetical order by column name.

The UUIDs shown for rows created in the same ovs-vsctl invocation will be wrong.

[--if-exists] [--id=@name] get table record [column[:key]]...
Prints the value of each specified column in the given record in table. For map columns, a key may optionally be specified, in which case the value associated with key in the column is printed, instead of the entire map.

Without --if-exists, it is an error if record does not exist or key is specified, if key does not exist in record. With --if-exists, a missing record yields no output and a missing key prints a blank line.

If @name is specified, then the UUID for record may be referred to by that name later in the same ovs-vsctl invocation in contexts where a UUID is expected.

Both --id and the column arguments are optional, but usually at least one or the other should be specified. If both are omitted, then get has no effect except to verify that record exists in table.

--id and --if-exists cannot be used together.

[--if-exists] set table record column[:key]=value...

Sets the value of each specified column in the given record in table to value. For map columns, a key may optionally be specified, in which case the value associated with key in that column is changed (or added, if none exists), instead of the entire map.

Without --if-exists, it is an error if record does not exist. With --if-exists, this command does nothing if record does not exist.

[--if-exists] add table record column [key=]value...

Adds the specified value or key-value pair to column in record in table. If column is a map, then key is required, otherwise it is prohibited. If key already exists in a map column, then the current value is not replaced (use the set command to replace an existing value).

Without --if-exists, it is an error if record does not exist. With --if-exists, this command does nothing if record does not exist.

[--if-exists] remove table record column value...
[--if-exists] remove table record column key...
[--if-exists] remove table record column key=value...

Removes the specified values or key-value pairs from column in record in table. The first form applies to columns that are not maps: each specified value is removed from the column. The second and third forms apply to map columns: if only a key is specified, then any key-value pair with the given key is removed, regardless of its value; if a value is given then a pair is removed only if both key and value match.

It is not an error if the column does not contain the specified key or value or pair.

Without --if-exists, it is an error if record does not exist. With --if-exists, this command does nothing if record does not exist.

[--if-exists] clear table record column...

Sets each column in record in table to the empty set or empty map, as appropriate. This command applies only to columns that are allowed to be empty.

Without --if-exists, it is an error if record does not exist.

With --if-exists, this command does nothing if record does not exist.

[--id=@name] create table column[:key]=value...

Creates a new record in table and sets the initial values of each column. Columns not explicitly set will receive their default values. Outputs the UUID of the new row.

If @name is specified, then the UUID for the new row may be referred to by that name elsewhere in the same ovs-vsctl invocation in contexts where a UUID is expected. Such references may precede or follow the create command.

Caution (ovs-vsctl as exmaple)

> Records in the Open vSwitch database are significant only when they can be reached directly or indirectly from the Open_vSwitch table. Except for records in the QoS or Queue tables, records that are not reachable from the Open_vSwitch table are automatically deleted from the database. This deletion happens immediately, without waiting for additional ovs-vsctl commands or other database activity. Thus, a create command must generally be accompanied by additional commands within the same ovs-vsctl invocation to add a chain of references to the newly created record from the top-level Open_vSwitch record. The EXAMPLES section gives some examples that show how to do this.

[--if-exists] destroy table record...

Deletes each specified record from table. Unless --if-exists is specified, each records must exist.

--all destroy table

Deletes all records from the table.

Caution (ovs-vsctl as exmaple)

> The destroy command is only useful for records in the QoS or Queue tables. Records in other tables are automatically deleted from the database when they become unreachable from the Open_vSwitch table. This means that deleting the last reference to a record is sufficient for deleting the record itself. For records in these tables, destroy is silently ignored. See the EXAMPLES section below for more information.

wait-until table record [column[:key]=value]...

Waits until table contains a record named record whose column equals value or, if key is specified, whose column contains a key with the specified value. Any of the operators !=, <, >, <=, or >= may be substituted for = to test for inequality, less than, greater than, less than or equal to, or greater than or equal to, respectively. (Don't forget to escape < or > from interpretation by the shell.)

If no column[:key]=value arguments are given, this command waits only until record exists. If more than one such argument is given, the command waits until all of them are satisfied.

Caution (ovs-vsctl as exmaple)

Usually wait-until should be placed at the beginning of a set of ovs-vsctl commands. For example, wait-until bridge br0 -- get bridge br0 datapath_id waits until a bridge named br0 is created, then prints its datapath_id column, whereas get bridge br0 datapath_id -- wait-until bridge br0 will abort if no bridge named br0 exists when ovs-vsctl initially connects to the database.

Consider specifying --timeout=0 along with --wait-until, to prevent ovs-vsctl from terminating after waiting only at most 5 seconds.

comment [arg]...

This command has no effect on behavior, but any database log record created by the command will include the command and its arguments.

EXAMPLES

Create a new bridge named br0 and add port eth0 to it:

```
ovs-vsctl add-br br0
ovs-vsctl add-port br0 eth0
```

Alternatively, perform both operations in a single atomic transaction:

```
ovs-vsctl add-br br0 -- add-port br0 eth0
```

Delete bridge br0, reporting an error if it does not exist:

```
ovs-vsctl del-br br0
```

Delete bridge br0 if it exists:

```
ovs-vsctl --if-exists del-br br0
```

Set the qos column of the Port record for eth0 to point to a new QoS record, which in turn points with its queue 0 to a new Queue record:

```
ovs-vsctl -- set port eth0 qos=@newqos -- --id=@newqos create
qos type=linux-htb other-config:max-rate=1000000
queues:0=@newqueue -- --id=@newqueue create queue other-config:min-rate=1000000 other-config:max-rate=1000000
```

CONFIGURATION COOKBOOK

Port Configuration

Add an ``internal port'' vlan10 to bridge br0 as a VLAN access port for VLAN 10, and configure it with an IP address:

```
ovs-vsctl add-port br0 vlan10 tag=10 -- set Interface vlan10
type=internal

ifconfig vlan10 192.168.0.123
```

Add a GRE tunnel port gre0 to remote IP address 1.2.3.4 to bridge br0:

```
ovs-vsctl add-port br0 gre0 -- set Interface gre0 type=gre
options:remote_ip=1.2.3.4
```

Port Mirroring

Mirror all packets received or sent on eth0 or eth1 onto eth2, assuming that all of those ports exist on bridge br0 (as a side-effect this causes any packets received on eth2 to be ignored):

```
ovs-vsctl -- set Bridge br0 mirrors=@m \

-- --id=@eth0 get Port eth0 \

-- --id=@eth1 get Port eth1 \

-- --id=@eth2 get Port eth2 \

-- --id=@m create Mirror name=mymirror select-dst-port=@eth0,@eth1 select-src-port=@eth0,@eth1 output-port=@eth2
```

Remove the mirror created above from br0, which also destroys the Mirror record (since it is now unreferenced):

```
ovs-vsctl -- --id=@rec get Mirror mymirror \

-- remove Bridge br0 mirrors @rec
```

The following simpler command also works:

```
ovs-vsctl clear Bridge br0 mirrors
```

Quality of Service (QoS)

Create a linux-htb QoS record that points to a few queues and use it on eth0 and eth1:

```
ovs-vsctl -- set Port eth0 qos=@newqos \

-- set Port eth1 qos=@newqos \

-- --id=@newqos create QoS type=linux-htb other-config:max-rate=1000000000 queues=0=@q0,1=@q1 \

-- --id=@q0 create Queue other-config:min-rate=100000000 other-config:max-rate=100000000 \

-- --id=@q1 create Queue other-config:min-rate=500000000
```

Deconfigure the QoS record above from eth1 only:

```
ovs-vsctl clear Port eth1 qos
```

To deconfigure the QoS record from both eth0 and eth1 and then delete the QoS record (which must be done explicitly because unreferenced QoS records are not automatically destroyed):

```
ovs-vsctl -- destroy QoS eth0 -- clear Port eth0 qos -- clear Port eth1 qos
```

(This command will leave two unreferenced Queue records in the database. To delete them, use "ovs-vsctl list Queue" to find their UUIDs, then "ovs-vsctl destroy Queue uuid1 uuid2" to destroy each of them or use "ovs-vsctl -- --all destroy Queue" to delete all records.)

Connectivity Monitoring

Monitor connectivity to a remote maintenance point on eth0.

```
ovs-vsctl set Interface eth0 cfm_mpid=1
```

Deconfigure connectivity monitoring from above:

```
ovs-vsctl clear Interface eth0 cfm_mpid
```

NetFlow

Configure bridge br0 to send NetFlow records to UDP port 5566 on host 192.168.0.34, with an active timeout of 30 seconds:

```
ovs-vsctl -- set Bridge br0 netflow=@nf \

-- --id=@nf create NetFlow targets=\"192.168.0.34:5566\" active-timeout=30
```

Update the NetFlow configuration created by the previous command to instead use an active timeout of 60 seconds:

```
ovs-vsctl set NetFlow br0 active_timeout=60
```

Deconfigure the NetFlow settings from br0, which also destroys the NetFlow record (since it is now unreferenced):

```
ovs-vsctl clear Bridge br0 netflow
```

sFlow

Configure bridge br0 to send sFlow records to a collector on 10.0.0.1 at port 6343, using eth1´s IP address as the source, with specific sampling parameters:

```
ovs-vsctl -- --id=@s create sFlow agent=eth1 target=\"10.0.0.1:6343\" header=128 sampling=64 polling=10 \

-- set Bridge br0 sflow=@s
```

Deconfigure sFlow from br0, which also destroys the sFlow record (since it is now unreferenced):

```
ovs-vsctl -- clear Bridge br0 sflow
```

IPFIX

Configure bridge br0 to send one IPFIX flow record per packet sample to UDP port 4739 on host 192.168.0.34, with Observation Domain ID 123 and Observation Point ID 456, a flow cache active timeout of 1 minute (60 seconds), maximum flow cache size of 13 flows, and flows sampled on output port with tunnel info(sampling on input and output port is enabled by default if not disabled) :

```
ovs-vsctl -- set Bridge br0 ipfix=@i \

-- --id=@i create IPFIX targets=\"192.168.0.34:4739\" obs_domain_id=123 obs_point_id=456 cache_active_timeout=60 cache_max_flows=13 \

other_config:enable-input-sampling=false other_config:enable-tunnel-sampling=true
```

Deconfigure the IPFIX settings from br0, which also destroys the IPFIX record (since it is now unreferenced):

```
ovs-vsctl clear Bridge br0 ipfix
```

802.1D Spanning Tree Protocol (STP)

Configure bridge br0 to participate in an 802.1D spanning tree:

```
ovs-vsctl set Bridge br0 stp_enable=true
```

Set the bridge priority of br0 to 0x7800:

```
ovs-vsctl set Bridge br0 other_config:stp-priority=0x7800
```

Set the path cost of port eth0 to 10:

```
ovs-vsctl set Port eth0 other_config:stp-path-cost=10
```

Deconfigure STP from above:

```
ovs-vsctl set Bridge br0 stp_enable=false
```

Multicast Snooping

Configure bridge br0 to enable multicast snooping:

```
ovs-vsctl set Bridge br0 mcast_snooping_enable=true
```

Set the multicast snooping aging time br0 to 300 seconds:

```
ovs-vsctl set Bridge br0 other_config:mcast-snooping-aging-time=300
```

Set the multicast snooping table size br0 to 2048 entries:

```
ovs-vsctl set Bridge br0 other_config:mcast-snooping-table-size=2048
```

Disable flooding of unregistered multicast packets to all ports. When set to true, the switch will send unregistered multicast packets only to ports connected to multicast routers. When it is set to false, the switch will send them to all ports. This command disables the flood of unregistered packets on bridge br0.

```
ovs-vsctl set Bridge br0 other_config:mcast-snooping-disable-flood-unregistered=true
```

Enable flooding of multicast packets (except Reports) on a specific port.

```
ovs-vsctl set Port eth1 other_config:mcast-snooping-flood=true
```

Enable flooding of Reports on a specific port.

```
ovs-vsctl set Port eth1 other_config:mcast-snooping-flood-reports=true
```

Deconfigure multicasting snooping from above:

```
ovs-vsctl set Bridge br0 mcast_snooping_enable=false
```

802.1D-2004 Rapid Spanning Tree Protocol (RSTP)

Configure bridge br0 to participate in an 802.1D-2004 Rapid Spanning Tree:

```
ovs-vsctl set Bridge br0 rstp_enable=true
```

Set the bridge address of br0 to 00:aa:aa:aa:aa:aa :

```
ovs-vsctl set Bridge br0 other_config:rstp-address=00:aa:aa:aa:aa:aa
```

Set the bridge priority of br0 to 0x7000. The value must be specified in decimal notation and should be a multiple of 4096 (if not, it is rounded down to the nearest multiple of 4096). The default priority value is 0x800 (32768).

```
ovs-vsctl set Bridge br0 other_config:rstp-priority=28672
```

Set the bridge ageing time of br0 to 1000 s. The ageing time value should be between 10 s and 1000000 s. The default value is 300 s.

```
ovs-vsctl set Bridge br0 other_config:rstp-ageing-time=1000
```

Set the bridge force protocol version of br0 to 0. The force protocol version has two acceptable values: 0 (STP compatibility mode) and 2 (normal operation).

```
ovs-vsctl set Bridge br0 other_config:rstp-force-protocol-version=0
```

Set the bridge max age of br0 to 10 s. The max age value should be between 6 s and 40 s. The default value is 20 s.

```
ovs-vsctl set Bridge br0 other_config:rstp-max-age=10
```

Set the bridge forward delay of br0 to 15 s. This value should be between 4 s and 30 s. The default value is 15 s.

```
ovs-vsctl set Bridge br0 other_config:rstp-forward-delay=15
```

Set the bridge transmit hold count of br0 to 7 s. This value should be between 1 s and 10 s. The default value is 6 s.

```
ovs-vsctl set Bridge br0 other_config:rstp-transmit-hold-count=7
```

Enable RSTP on the Port eth0:

```
ovs-vsctl set Port eth0 other_config:rstp-enable=true
```

Disable RSTP on the Port eth0:

```
ovs-vsctl set Port eth0 other_config:rstp-enable=false
```

Set the priority of port eth0 to 32. The value must be specified in decimal notation and should be a multiple of 16 (if not, it is rounded down to the nearest multiple of 16). The default priority value is 0x80 (128).

```
ovs-vsctl set Port eth0 other_config:rstp-port-priority=32
```

Set the port number of port eth0 to 3:

```
ovs-vsctl set Port eth0 other_config:rstp-port-num=3
```

Set the path cost of port eth0 to 150:

```
ovs-vsctl set Port eth0 other_config:rstp-path-cost=150
```

Set the admin edge value of port eth0:

```
ovs-vsctl set Port eth0 other_config:rstp-port-admin-edge=true
```

Set the auto edge value of port eth0:

```
ovs-vsctl set Port eth0 other_config:rstp-port-auto-edge=true
```

Set the admin point to point MAC value of port eth0. Acceptable values are 0 (not point-to-point), 1 (point-to-point, the default value) or 2 (automatic detection). The auto-detection mode is not currently implemented, and the value 2 has the same effect of 0 (not point-to-point).

```
ovs-vsctl set Port eth0 other_config:rstp-admin-p2p-mac=1
```

Set the admin port state value of port eth0. true is the default value.

```
ovs-vsctl set Port eth0 other_config:rstp-admin-port-state=false
```

Set the mcheck value of port eth0:

```
ovs-vsctl set Port eth0 other_config:rstp-port-mcheck=true
```

Deconfigure RSTP from above:

```
ovs-vsctl set Bridge br0 rstp_enable=false
```

OpenFlow Version

Configure bridge br0 to support OpenFlow versions 1.0, 1.2, and 1.3:

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

Flow Table Configuration

Limit flow table 0 on bridge br0 to a maximum of 100 flows:

```
ovs-vsctl -- --id=@ft create Flow_Table flow_limit=100 overflow_policy=refuse -- set Bridge br0 flow_tables=0=@ft
```

EXIT STATUS

0 Successful program execution.

1 Usage, syntax, or configuration file error.

2 The bridge argument to br-exists specified the name of a bridge that does not exist.