NAME
ovs-fields – protocol header fields supported by Open vSwitch

SYNOPSIS
TUNNEL FIELDS
The fields in this group relate to tunnels, which Open vSwitch supports in several forms (GRE, VXLAN, and so on). Most of these fields do appear in the wire format of a packet, so they are data fields from that
point of view, but they are metadata from an OpenFlow flow table point of view because they do not appear
in packets that are forwarded to the controller or to ordinary (non-tunnel) output ports.

Open vSwitch supports a spectrum of usage models for mapping tunnels to OpenFlow ports:

“Port-based” tunnels
In this model, an OpenFlow port represents one tunnel: it matches a particular type of
tunnel traffic between two IP endpoints, with a particular tunnel key (if keys are in use).
In this situation, MFF_IN_PORT suffices to distinguish one tunnel from another, so the
tunnel header fields have little importance for OpenFlow processing. (They are still pop-
ulated and may be used if it is convenient.) The tunnel header fields play no role in sending
packets out such an OpenFlow port, either, because the OpenFlow port itself fully
specifies the tunnel headers.

The following Open vSwitch commands create a bridge br-int, add port tap0 to the
bridge as OpenFlow port 1, establish a port-based GRE tunnel between the local host and
remote IP 192.168.1.1 using GRE key 5001 as OpenFlow port 2, and arranges to forward
all traffic from tap0 to the tunnel and vice versa:

```bash
ovs-vsctl add-br br-int
ovs-vsctl add-port br-int tap0 -- set interface tap0 ofport_request=1
ovs-vsctl add-port br-int gre0 --
    set interface gre0 ofport_request=2 type=gre
        options:remote_ip=192.168.1.1 options:key=5001
ovs-ofctl add-flow br-int in_port=1,actions=2
ovs-ofctl add-flow br-int in_port=2,actions=1
```

“Flow-based” tunnels
In this model, one OpenFlow port represents all possible tunnels of a given type with an
endpoint on the current host, for example, all GRE tunnels. In this situation, MFF_IN_PORT only indicates that traffic was received on the particular kind of tunnel.
This is where the tunnel header fields are most important: they allow the OpenFlow tables
to discriminate among tunnels based on their IP endpoints or keys. Tunnel header fields
also determine the IP endpoints and keys of packets sent out such a tunnel port.

The following Open vSwitch commands create a bridge br-int, add port tap0 to the
bridge as OpenFlow port 1, establish a flow-based GRE tunnel port 3, and arranges to forward
all traffic from tap0 to remote IP 192.168.1.1 over a GRE tunnel with key 5001 and
vice versa:

```bash
ovs-vsctl add-br br-int
ovs-vsctl add-port br-int tap0 -- set interface tap0 ofport_request=1
ovs-vsctl add-port br-int allgre --
    set interface gre0 ofport_request=2 type=gre
        options:remote_ip=flow options:key=flow
ovs-ofctl add-flow br-int
    'in_port=1 actions=set_tunnel:5001,set_field:192.168.1.1->tun_dst,3'
   ovs-ofctl add-flow br-int 'in_port=3,tun_src=192.168.1.1,tun_id=5001 actions=1'
```
Mixed models.
One may define both flow-based and port-based tunnels at the same time. For example, it
is valid and possibly useful to create and configure both gre0 and allgre tunnel ports
described above.

Traffic is attributed on ingress to the most specific matching tunnel. For example, gre0 is
more specific than allgre. Therefore, if both exist, then gre0 will be the input port for
any GRE traffic received from 192.168.1.1 with key 5001.

On egress, traffic may be directed to any appropriate tunnel port. If both gre0 and allgre
are configured as already described, then the actions 2 and set_tunnel:5001,set_field:192.168.1.1−>tun_dst,3 send the same tunnel traffic.

Intermediate models.
Ports may be configured as partially flow-based. For example, one may define an Open-
Flow port that represents tunnels between a pair of endpoints but leaves the flow table to
discriminate on the flow key.

ovs-vswitchd.conf.db(5) describes all the details of tunnel configuration.
These fields do not have any prerequisites, which means that a flow may match on any or all of them, in any
combination.

These fields are zeros for packets that did not arrive on a tunnel.

Tunnel ID Field
Name: tun_id (aka tunnel_id)
Width: 64 bits
Masking: arbitrary bitwise masks
Prerequisites: MFP_NONE
Access: read/write
OpenFlow 1.0: no
OpenFlow 1.1: no
OpenFlow 1.2: yes (via OpenFlow 1.3 code point)
OpenFlow 1.3: yes
NXM: yes
Code Points: OXM_OF_TUNNEL_ID (0x80004c08), introduced in OpenFlow 1.3
NXM_NX_TUN_ID (0x00012008), introduced in Open vSwitch 1.1

Many kinds of tunnels support a tunnel ID:
- VXLAN has a 24-bit virtual network identifier (VNI).
- LISP has a 24-bit instance ID.
- GRE has an optional 32-bit key.
- GRE64 (a non-standard protocol) has a 64-bit ID constructed from the 32-bit GRE key and
  32-bit GRE sequence number.

When a packet is received from a tunnel, this field holds the tunnel ID in its least significant bits, zero-
extended to fit. This field is zero if the tunnel does not support an ID, or if no ID is in use for a tunnel type
that has an optional ID, or if an ID of zero received, or if the packet was not received over a tunnel.

When a packet is output to a tunnel port, the tunnel configuration determines whether the tunnel ID is taken
from this field or bound to a fixed value. See the earlier description of “port-based” and “flow-based” tun-
nels for more information.

The following diagram shows the origin of this field in a typical keyed GRE tunnel:

```
<table>
<thead>
<tr>
<th>Ethernet</th>
<th>IPv4</th>
<th>GRE</th>
<th>Ethernet</th>
</tr>
</thead>
<tbody>
<tr>
<td>dst</td>
<td>src</td>
<td>type</td>
<td>dst</td>
</tr>
<tr>
<td>48</td>
<td>48</td>
<td>16</td>
<td>48</td>
</tr>
<tr>
<td>src</td>
<td>dst</td>
<td>type</td>
<td>src</td>
</tr>
<tr>
<td>16</td>
<td>32</td>
<td>32</td>
<td>16</td>
</tr>
<tr>
<td>type</td>
<td>dst</td>
<td>src</td>
<td>type</td>
</tr>
<tr>
<td>32</td>
<td>32</td>
<td>16</td>
<td>32</td>
</tr>
<tr>
<td>dst</td>
<td>src</td>
<td>type</td>
<td>dst</td>
</tr>
<tr>
<td>16</td>
<td>47</td>
<td>0x6558</td>
<td>0x800</td>
</tr>
</tbody>
</table>
```

0x8000
Tunnel IPv4 Source Field

Name: tun_src
Width: 32 bits
Masking: arbitrary bitwise masks
Prerequisites: MFP_NONE
Access: read/write
OpenFlow 1.0: no
OpenFlow 1.1: no
OpenFlow 1.2: yes (via NXM code point)
OpenFlow 1.3: yes (via NXM code point)
NXM: yes
Code Points: NXM_NX_TUN_IPV4_SRC (0x00013e04), introduced in Open vSwitch 2.0

When a packet is received from a tunnel, this field is the source address in the outer IP header of the tunneled packet. This field is zero if the packet was not received over a tunnel.

When a packet is output to a flow-based tunnel port, this field influences the IPv4 source address used to send the packet. If it is zero, then the kernel chooses an appropriate IP address based using the routing table.

The following diagram shows the origin of this field in a typical keyed GRE tunnel:

```
<table>
<thead>
<tr>
<th>Ethernet</th>
<th>IPv4</th>
<th>GRE</th>
<th>Ethernet</th>
</tr>
</thead>
<tbody>
<tr>
<td>dst</td>
<td>src</td>
<td>type</td>
<td></td>
</tr>
<tr>
<td>0x800</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Tunnel IPv4 Destination Field

Name: tun_dst
Width: 32 bits
Masking: arbitrary bitwise masks
Prerequisites: MFP_NONE
Access: read/write
OpenFlow 1.0: no
OpenFlow 1.1: no
OpenFlow 1.2: yes (via NXM code point)
OpenFlow 1.3: yes (via NXM code point)
NXM: yes
Code Points: NXM_NX_TUN_IPV4_DST (0x00014004), introduced in Open vSwitch 2.0

When a packet is received from a tunnel, this field is the destination address in the outer IP header of the tunneled packet. This field is zero if the packet was not received over a tunnel.

When a packet is output to a flow-based tunnel port, this field specifies the destination to which the tunnel packet is sent.

The following diagram shows the origin of this field in a typical keyed GRE tunnel:

```
<table>
<thead>
<tr>
<th>Ethernet</th>
<th>IPv4</th>
<th>GRE</th>
<th>Ethernet</th>
</tr>
</thead>
<tbody>
<tr>
<td>dst</td>
<td>src</td>
<td>type</td>
<td></td>
</tr>
<tr>
<td>0x800</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
METADATA FIELDS
These fields relate to the origin or treatment of a packet, but they are not extracted from the packet data itself.

Input Port Field
Name: in_port
Width: 16 bits
Masking: not maskable
Prerequisites: MFP_NONE
Access: read/write
OpenFlow 1.0: yes
OpenFlow 1.1: yes
OpenFlow 1.2: yes (via NXM code point)
OpenFlow 1.3: yes (via NXM code point)
NXM: yes
Code Points: NXM_OF_IN_PORT (0x00000002), introduced in Open vSwitch 1.1

The OpenFlow port on which the packet being processed arrived. This is a 16-bit field that holds an OpenFlow 1.0 port number. For receiving a packet, the only values that appear in this field are:

- 1 through 0xeff (65,279), inclusive. Conventional OpenFlow port numbers.
- OFPP_LOCAL (0xfffe or 65,534).
  The “local” port, which in Open vSwitch is always named the same as the bridge itself. This represents a connection between the switch and the local TCP/IP stack. This port is where an IP address is most commonly configured on an Open vSwitch switch.
  OpenFlow does not require a switch to have a local port, but all existing versions of Open vSwitch have always included a local port. (Some future version of Open vSwitch might be able to optionally omit the local port, if someone submits code to implement such a feature.)
- OFPP_NONE (0xffff or 65,535).
- OFPP_CONTROLLER (0xffffd or 65,533).
  When a controller injects a packet into an OpenFlow switch with a “packet-out” request, it can specify one of these input ports to indicate that the packet was generated internally rather than having been received on some port.
  OpenFlow 1.0 specified OFPP_NONE for this purpose. Despite that, some controllers used OFPP_CONTROLLER, and some switches only accepted OFPP_CONTROLLER, so OpenFlow 1.0.2 required support for both ports. OpenFlow 1.1 and later were more clearly drafted to allow only OFPP_CONTROLLER. For maximum compatibility, Open vSwitch allows both ports with all OpenFlow versions.

Values not mentioned above will never appear when receiving a packet, including the following notable values:

- 0  Zero is not a valid OpenFlow port number.
- OFPP_MAX (0xfff0 or 65,280).
  This value has only been clearly specified as a valid port number as of OpenFlow 1.3.3. Before that, its status was unclear, and so Open vSwitch has never allowed OFPP_MAX to be used as a port number, so packets will never be received on this port. (Other OpenFlow switches, of course, might use it.)
- OFPP_IN_PORT (0xff8 or 65,528)
- OFPP_TABLE (0xfff9 or 65,529)
- OFPP_NORMAL (0xfff or 65,530)
OFPP_FLOOD (0xfffb or 65,531)
OFPP_ALL (0xffffffff or 65,532)

These port numbers are used only in output actions and never appear as input ports. Values that will never appear when receiving a packet may still be matched against in the flow table. There are still circumstances in which those flows can be matched:

- The `resubmit` Nicira extension action allows a flow table lookup with an arbitrary input port.
- An action that modifies the input port field (see below), such as e.g. `load` or `set_field`, followed by an action or instruction that performs another flow table lookup, such as `resubmit` or `goto_table`.

This field is heavily used for matching in OpenFlow tables, but for packet egress, it has only very limited roles:

- OpenFlow requires suppressing output actions to MFF_IN_PORT. That is, the following two flows both drop all packets that arrive on port 1:

  ```
  in_port=1,actions=1
  in_port=1,actions=drop
  ```

  (This behavior is occasionally useful for flooding to a subset of ports. Specifying `actions=1,2,3,4`, for example, outputs to ports 1, 2, 3, and 4, omitting the input port.)

- OpenFlow has a special port OFPP_IN_PORT (with value 0xff8) that outputs to the input port. For example, in a switch that has four ports numbered 1 through 4, `actions=1,2,3,4,in_port` outputs to ports 1, 2, 3, and 4, including the input port.

Because the input port field has so little influence on packet processing, it does not ordinarily make sense to modify the input port field. The field is writable only to support the occasional use case where the input port’s roles in packet egress, described above, become troublesome. For example, `actions=load:0->NXM_OF_IN_PORT[],output:123` will output to port 123 regardless of whether it is in the input port. If the input port is important, then one may save and restore it on the stack:

```
actions=push:NXM_OF_IN_PORT[],load:0->NXM_OF_IN_PORT[],output:123,pop:NXM_OF_IN_PORT[]
```

The ability to modify the input port is an Open vSwitch extension to OpenFlow.

Modifying the input port does not prevent or frustrate specifying an input port in the `resubmit` action, because `resubmit` only (optionally) changes the `in_port` used for `resubmit`’s flow table lookup. It does not otherwise affect the input port.

**OXM Input Port Field**

Name: `in_port_oxm`
Width: 32 bits
Masking: not maskable
Prerequisites: MFP_NONE
Access: read/write
OpenFlow 1.0: no
OpenFlow 1.1: yes
OpenFlow 1.2: yes
OpenFlow 1.3: yes
NXM: no
Code Points: OXM_OF_IN_PORT (0x80000004), introduced in OpenFlow 1.2

OpenFlow 1.1 and later use a 32-bit port number, so this field supplies a 32-bit view of the input port. Current versions of Open vSwitch support only a 16-bit range of ports:

- OpenFlow 1.0 ports 0x0000 to 0xfffe, inclusive, map to OpenFlow 1.1 port numbers with the same values.
OpenFlow 1.0 ports 0xff00 to 0xffff, inclusive, map to OpenFlow 1.1 port numbers 0xffffffff to 0xffffffff.

OpenFlow 1.1 ports 0x0000ff00 to 0xffffffff are not mapped and not supported.

MFF_IN_PORT and MFF_IN_PORT_OXM are two views of the same information, so all of the comments on MFF_IN_PORT apply to MFF_IN_PORT_OXM too. Modifying MFF_IN_PORT changes MFF_IN_PORT_OXM, and vice versa.

Setting MFF_IN_PORT_OXM to an unsupported value yields unspecified behavior.

Output Queue Field

- **Name**: skb_priority
- **Width**: 32 bits
- **Masking**: not maskable
- **Prerequisites**: MFP_NONE
- **Access**: read-only
- **OpenFlow 1.0**: no
- **OpenFlow 1.1**: no
- **OpenFlow 1.2**: no
- **OpenFlow 1.3**: no
- **NXM**: no
- **Code Points**: none

This field influences how packets in the flow will be queued, for quality of service (QoS) purposes, when they egress the switch. Its range of meaningful values, and their meanings, varies greatly from one OpenFlow implementation to another. Even within a single implementation, there is no guarantee that all OpenFlow ports have the same queues configured or that all OpenFlow ports in an implementation can be configured the same way queue-wise.

Configuring queues on OpenFlow is not well standardized. On Linux, Open vSwitch supports queue configuration via OVSDB, specifically the QoS and Queue tables (see ovsvswitchd.conf.db(5) for details). Ports of Open vSwitch to other platforms might require queue configuration through some separate protocol (such as a CLI). Even on Linux, Open vSwitch exposes only a fraction of the kernel’s queuing features through OVSDB, so advanced or unusual uses might require use of separate utilities (e.g. tc). OpenFlow switches other than Open vSwitch might use OF-CONFIG or any of the configuration methods mentioned above. Finally, some OpenFlow switches have a fixed number of fixed-function queues (e.g. eight queues with strictly defined priorities) and others do not support any control over queuing.

The only output queue that all OpenFlow implementations must support is zero, to identify a default queue, whose properties are implementation-defined. Outputting a packet to a queue that does not exist on the output port yields unpredictable behavior: among the possibilities are that the packet might be dropped or transmitted with a very high or very low priority.

OpenFlow 1.0 only allowed output queues to be specified as part of an “enqueue” action that specified both a queue and an output port. That is, OpenFlow 1.0 treats the queue as an argument to an action, not as a field.

OpenFlow switch and controller implementers soon realized that separating the decisions for output queue and output port increased flexibility, so OpenFlow 1.1 added an action to set the output queue. This model was carried forward, without change, through OpenFlow 1.4.

Open vSwitch implements the native queuing model of each OpenFlow version it supports. Open vSwitch also includes an extension for setting the output queue as an action in OpenFlow 1.0.

When a packet ingresses into an OpenFlow switch, the output queue is ordinarily set to 0, indicating the default queue. However, Open vSwitch supports various ways to forward a packet from one OpenFlow switch to another within a single host. In these cases, Open vSwitch maintains the output queue across the forwarding step. For example:

- A hop across a Open vSwitch “patch port” (which does not actually involve queuing) preserves the output queue.
• When a flow sets the output queue then outputs to an OpenFlow tunnel port, the encapsulation preserves the output queue. If the kernel TCP/IP stack routes the encapsulated packet directly to a physical interface, then that output honors the output queue. Alternatively, if the kernel routes the encapsulated packet to another Open vSwitch bridge, then the output queue set previously becomes the initial output queue on ingress to the second bridge and will thus be used for further output actions (unless overridden by a new “set queue” action).

(This description reflects the current behavior of Open vSwitch on Linux. This behavior relies on details of the Linux TCP/IP stack. It could be difficult to make ports to other operating systems behave the same way.)

Open vSwitch implements the output queue as a field, but does not currently expose it through OXM or NXM for matching purposes. If this turns out to be a useful feature, it could be implemented in future versions.

**Packet Mark Field**

<table>
<thead>
<tr>
<th>Name</th>
<th>pkt_mark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Width</td>
<td>32 bits</td>
</tr>
<tr>
<td>Masking</td>
<td>arbitrary bitwise masks</td>
</tr>
<tr>
<td>Prerequisites</td>
<td>MFP_NONE</td>
</tr>
<tr>
<td>Access</td>
<td>read/write</td>
</tr>
<tr>
<td>OpenFlow 1.0:</td>
<td>no</td>
</tr>
<tr>
<td>OpenFlow 1.1:</td>
<td>no</td>
</tr>
<tr>
<td>OpenFlow 1.2:</td>
<td>yes (via NXM code point)</td>
</tr>
<tr>
<td>OpenFlow 1.3:</td>
<td>yes (via NXM code point)</td>
</tr>
<tr>
<td>NXM:</td>
<td>yes</td>
</tr>
</tbody>
</table>

Packet mark comes to Open vSwitch from the Linux kernel, in which the sk_buff data structure that represents a packet contains a 32-bit member named skb_mark. The value of skb_mark propagates along with the packet it accompanies wherever the packet goes in the kernel. It has no predefined semantics but various kernel-user interfaces can set and match on it, which makes it suitable for “marking” packets at one point in their handling and then acting on the mark later. With iptables, for example, one can mark some traffic specially at ingress and then handle that traffic differently at egress based on the marked value.

Packet mark is an attempt at a generalization of the skb_mark concept beyond Linux, at least through more generic naming. Like MFF_SKB_PRIORITY, packet mark is preserved across forwarding steps within a machine. Unlike MFF_SKB_PRIORITY, packet mark has no direct effect on packet forwarding: the value set in packet mark does not matter unless some later OpenFlow table or switch matches on packet mark, or unless the packet passes through some other kernel subsystem that has been configured to interpret packet mark in specific ways, e.g. through iptables configuration mentioned above.

Preserving packet mark across kernel forwarding steps relies heavily on kernel support, which ports to non-Linux operating systems may not have. Regardless of operating system support, Open vSwitch supports packet mark within a single bridge and across patch ports.

The value of packet mark when a packet ingresses into the first Open vSwitch bridge is typically zero, but it could be nonzero if its value was previously set by some kernel subsystem.
REGISTER FIELDS

These fields give an OpenFlow switch space for temporary storage while the pipeline is running. Whereas metadata fields can have a meaningful initial value and can persist across some hops across OpenFlow switches, registers are always initially 0 and their values never persist across inter-switch hops.

OpenFlow Metadata Field

Name: metadata
Width: 64 bits
Masking: arbitrary bitwise masks
Prerequisites: MFP_NONE
Access: read/write
OpenFlow 1.0: no
OpenFlow 1.1: yes
OpenFlow 1.2: yes
OpenFlow 1.3: yes
NXM: yes (via OXM code point)
Code Points: OXM_OF_METADATA (0x80000408), introduced in OpenFlow 1.2

This field is the only standardized OpenFlow register field. Because ASIC-based switches can carry a limited number of user-defined bits through their pipelines, OpenFlow allows switches to support writing and masking only an implementation-defined subset of bits, even no bits at all. The Open vSwitch software switch always supports all 64 bits, but of course an Open vSwitch port to an ASIC would have the same restriction as the ASIC itself.

This field has an OXM code point, but OpenFlow 1.1 through 1.4 allow it to be modified only with a specialized instruction, not with a "set-field" action. As of this writing, OpenFlow 1.5 seems likely to remove this restriction. Open vSwitch does not enforce this restriction, regardless of OpenFlow version.

Register 0 Field

Name: reg0
Width: 32 bits
Masking: arbitrary bitwise masks
Prerequisites: MFP_NONE
Access: read/write
OpenFlow 1.0: no
OpenFlow 1.1: no
OpenFlow 1.2: yes (via NXM code point)
OpenFlow 1.3: yes (via NXM code point)
NXM: yes
Code Points: NXM_NX_REG0 (0x00010004), introduced in Open vSwitch 1.1

This is the first of several Open vSwitch registers, all of which have the same properties. Open vSwitch 1.1 introduced registers 0, 1, 2, and 3, version 1.3 added register 4, and version 1.7 added registers 5, 6, and 7.

Register 1 Field

Name: reg1
Width: 32 bits
Masking: arbitrary bitwise masks
Prerequisites: MFP_NONE
Access: read/write
OpenFlow 1.0: no
OpenFlow 1.1: no
OpenFlow 1.2: yes (via NXM code point)
OpenFlow 1.3: yes (via NXM code point)
NXM: yes
Code Points: NXM_NX_REG1 (0x00010204), introduced in Open vSwitch 1.1
Register 2 Field
Name: reg2
Width: 32 bits
Masking: arbitrary bitwise masks
Prerequisites: MFP_NONE
Access: read/write
OpenFlow 1.0: no
OpenFlow 1.1: no
OpenFlow 1.2: yes (via NXM code point)
OpenFlow 1.3: yes (via NXM code point)
NXM: yes
Code Points: NXM_NX_REG2 (0x00010404), introduced in Open vSwitch 1.1

Register 3 Field
Name: reg3
Width: 32 bits
Masking: arbitrary bitwise masks
Prerequisites: MFP_NONE
Access: read/write
OpenFlow 1.0: no
OpenFlow 1.1: no
OpenFlow 1.2: yes (via NXM code point)
OpenFlow 1.3: yes (via NXM code point)
NXM: yes
Code Points: NXM_NX_REG3 (0x00010604), introduced in Open vSwitch 1.1

Register 4 Field
Name: reg4
Width: 32 bits
Masking: arbitrary bitwise masks
Prerequisites: MFP_NONE
Access: read/write
OpenFlow 1.0: no
OpenFlow 1.1: no
OpenFlow 1.2: yes (via NXM code point)
OpenFlow 1.3: yes (via NXM code point)
NXM: yes
Code Points: NXM_NX_REG4 (0x00010804), introduced in Open vSwitch 1.3

Register 5 Field
Name: reg5
Width: 32 bits
Masking: arbitrary bitwise masks
Prerequisites: MFP_NONE
Access: read/write
OpenFlow 1.0: no
OpenFlow 1.1: no
OpenFlow 1.2: yes (via NXM code point)
OpenFlow 1.3: yes (via NXM code point)
NXM: yes
Code Points: NXM_NX_REG5 (0x00010a04), introduced in Open vSwitch 1.7

Register 6 Field
Name: reg6
Width: 32 bits
Masking: arbitrary bitwise masks
Register 7 Field

Name: reg7
Width: 32 bits
Masking: arbitrary bitwise masks
Prerequisites: MFP_NONE
Access: read/write
OpenFlow 1.0: no
OpenFlow 1.1: no
OpenFlow 1.2: yes (via NXM code point)
OpenFlow 1.3: yes (via NXM code point)
NXM: yes
Code Points: NXM_NX_REG6 (0x00010c04), introduced in Open vSwitch 1.7

Layer 2 (Ethernet) Fields

Ethernet is the only layer-2 protocol that Open vSwitch supports. As with most software, Open vSwitch and OpenFlow regard an Ethernet frame to begin with the 14-byte header and end with the final byte of the payload; that is, the frame check sequence is not considered part of the frame.

Ethernet Source Field

Name: eth_src (aka dl_src)
Width: 48 bits
Masking: arbitrary bitwise masks
Prerequisites: MFP_NONE
Access: read/write
OpenFlow 1.0: yes (exact match only)
OpenFlow 1.1: yes
OpenFlow 1.2: yes
OpenFlow 1.3: yes
NXM: yes (maskable since Open vSwitch 1.8)
Code Points: OXM_OF_ETH_SRC (0x80000806), introduced in OpenFlow 1.2
NXM_OF_ETH_SRC (0x00000406), introduced in Open vSwitch 1.1

The Ethernet source address:

```
48 48 16
```

<table>
<thead>
<tr>
<th>dst</th>
<th>src</th>
<th>type</th>
</tr>
</thead>
</table>

Ethernet Destination Field

Name: eth_dst (aka dl_dst)
Width: 48 bits
Masking: arbitrary bitwise masks
Prerequisites: MFP_NONE
Access: read/write
OpenFlow 1.0: yes (exact match only)
OpenFlow 1.1: yes
OpenFlow 1.2: yes
OpenFlow 1.3: yes
NXM: yes (only partially maskable before Open vSwitch 1.8, see notes)

Code Points:
- OXM_OF_ETH_DST (0x80000606), introduced in OpenFlow 1.2
- NXM_OF_ETH_DST (0x00000206), introduced in Open vSwitch 1.1

The Ethernet destination address:

```
<table>
<thead>
<tr>
<th>dst</th>
<th>src</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>48</td>
<td>48</td>
<td>16</td>
</tr>
</tbody>
</table>
```

Ethertype Field

Name: eth_type (aka dl_type)
Width: 16 bits
Masking: not maskable
Prerequisites: MFP_NONE
Access: read-only
OpenFlow 1.0: yes
OpenFlow 1.1: yes
OpenFlow 1.2: yes
OpenFlow 1.3: yes
NXM: yes

Code Points:
- OXM_OF_ETH_TYPE (0x80000a02), introduced in OpenFlow 1.2
- NXM_OF_ETH_TYPE (0x00000602), introduced in Open vSwitch 1.1

The most commonly seen Ethernet frames today use a format called “Ethernet II,” in which the last two bytes of the Ethernet header specify the Ethertype. For such a frame, this field is copied from those bytes of the header, like so:

```
<table>
<thead>
<tr>
<th>dst</th>
<th>src</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>48</td>
<td>48</td>
<td>16</td>
</tr>
</tbody>
</table>
```

Every Ethernet type has a value 0x600 (1,536) or greater. When the last two bytes of the Ethernet header have a value too small to be an Ethernet type, then the value found there is the total length of the frame in bytes, excluding the Ethernet header. An 802.2 LLC header typically follows the Ethernet header. OpenFlow and Open vSwitch only support LLC headers with DSAP and SSAP 0xaa and control byte 0x03, which indicate that a SNAP header follows the LLC header. In turn, OpenFlow and Open vSwitch only support a SNAP header with organization 0x000000. In such a case, this field is copied from the type field in the SNAP header, like this:

```
<table>
<thead>
<tr>
<th>dst</th>
<th>src</th>
<th>length</th>
<th>DSAP</th>
<th>SSAP</th>
<th>cntl</th>
<th>org</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>48</td>
<td>48</td>
<td>&lt;0x600</td>
<td>0xa0</td>
<td>0xa0</td>
<td>0x03</td>
<td>0x000000</td>
<td>&lt;0x600</td>
</tr>
</tbody>
</table>
```

When an 802.1Q header is inserted after the Ethernet source and destination, this field is populated with the encapsulated Ethertype, not the 802.1Q Ethertype. With an Ethernet II inner frame, the result looks like this:

```
<table>
<thead>
<tr>
<th>dst</th>
<th>src</th>
<th>TPID</th>
<th>TCI</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>48</td>
<td>48</td>
<td>0x8100</td>
<td>0x03</td>
<td>&lt;0x600</td>
</tr>
</tbody>
</table>
```

LLC and SNAP encapsulation look like this with an 802.1Q header:

```
<table>
<thead>
<tr>
<th>dst</th>
<th>src</th>
<th>TPID</th>
<th>TCI</th>
<th>length</th>
<th>DSAP</th>
<th>SSAP</th>
<th>cntl</th>
<th>org</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>48</td>
<td>48</td>
<td>0x8100</td>
<td>0x03</td>
<td>&lt;0x600</td>
<td>0xa0</td>
<td>0xa0</td>
<td>0x03</td>
<td>0x000000</td>
<td>&lt;0x600</td>
</tr>
</tbody>
</table>
```

When a packet does’t match any of the header formats described above, Open vSwitch and OpenFlow set
this field to 0x5ff (OFP_DL_TYPE_NOT_ETH_TYPE).
VLAN FIELDS

VLAN TCI Field
Name: vlan_tci
Width: 16 bits
Masking: arbitrary bitwise masks
Prerequisites: MFP_NONE
Access: read/write
OpenFlow 1.0: yes (exact match only)
OpenFlow 1.1: yes (exact match only)
OpenFlow 1.2: yes (via NXM code point)
OpenFlow 1.3: yes (via NXM code point)
NXM: yes
Code Points: NXM_OF_VLAN_TCI (0x00000802), introduced in Open vSwitch 1.1

OpenFlow 1.0 VLAN ID Field
Name: dl_vlan
Width: 16 bits (only the least-significant 12 bits may be nonzero)
Masking: not maskable
Prerequisites: MFP_NONE
Access: read/write
OpenFlow 1.0: yes
OpenFlow 1.1: yes
OpenFlow 1.2: yes
OpenFlow 1.3: yes
NXM: yes
Code Points: none

OpenFlow 1.0 VLAN Priority Field
Name: dl_vlan_pcp
Width: 8 bits (only the least-significant 3 bits may be nonzero)
Masking: not maskable
Prerequisites: MFP_NONE
Access: read/write
OpenFlow 1.0: yes
OpenFlow 1.1: yes
OpenFlow 1.2: yes
OpenFlow 1.3: yes
NXM: yes
Code Points: none

OpenFlow 1.2+ VLAN ID Field
Name: vlan_vid
Width: 16 bits (only the least-significant 12 bits may be nonzero)
Masking: arbitrary bitwise masks
Prerequisites: MFP_NONE
Access: read/write
OpenFlow 1.0: yes (exact match only)
OpenFlow 1.1: yes (exact match only)
OpenFlow 1.2: yes
OpenFlow 1.3: yes
NXM: yes (via OXM code point)
Code Points: OXM_OF_VLAN_VID (0x80000c02), introduced in OpenFlow 1.2

OpenFlow 1.2+ VLAN Priority Field
Name: vlan_pcp
Width: 8 bits (only the least-significant 3 bits may be nonzero)
Masking: not maskable
Prerequisites: MFP_VLAN_VID
Access: read/write
OpenFlow 1.0: yes
OpenFlow 1.1: yes
OpenFlow 1.2: yes
OpenFlow 1.3: yes
NXM: yes (via OXM code point)
Code Points: OXM_OF_VLAN_PCP (0x80000e01), introduced in OpenFlow 1.2

**LAYER 2.5 (MPLS) FIELDS**

**MPLS Label Field**

Name: mpls_label
Width: 32 bits (only the least-significant 20 bits may be nonzero)
Masking: not maskable
Prerequisites: MFP_MPLS
Access: read/write
OpenFlow 1.0: no
OpenFlow 1.1: yes
OpenFlow 1.2: yes
OpenFlow 1.3: yes
NXM: yes (via OXM code point)
Code Points: OXM_OF_MPLS_LABEL (0x80004404), introduced in OpenFlow 1.2

**MPLS Traffic Class Field**

Name: mpls_tc
Width: 8 bits (only the least-significant 3 bits may be nonzero)
Masking: not maskable
Prerequisites: MFP_MPLS
Access: read/write
OpenFlow 1.0: no
OpenFlow 1.1: yes
OpenFlow 1.2: yes
OpenFlow 1.3: yes
NXM: yes (via OXM code point)
Code Points: OXM_OF_MPLS_TC (0x80004601), introduced in OpenFlow 1.2

**MPLS Bottom of Stack Field**

Name: mpls_bos
Width: 8 bits (only the least-significant 1 bits may be nonzero)
Masking: not maskable
Prerequisites: MFP_MPLS
Access: read-only
OpenFlow 1.0: no
OpenFlow 1.1: no
OpenFlow 1.2: yes (via OpenFlow 1.3 code point)
OpenFlow 1.3: yes
NXM: yes (via OXM code point)
Code Points: OXM_OF_MPLS_BOS (0x80004801), introduced in OpenFlow 1.3

**LAYER 3 FIELDS**

**IPv4 Source Address Field**

Name: ip_src (aka nw_src)
Width: 32 bits
Masking: arbitrary bitwise masks
Prerequisites: MFP_IPV4
Access: read/write
OpenFlow 1.0: yes (CIDR masks only)
OpenFlow 1.1: yes
OpenFlow 1.2: yes
OpenFlow 1.3: yes
NXM: yes (CIDR masks only before Open vSwitch 1.8)
Code Points: OXM_OF_IPV4_SRC (0x80001604), introduced in OpenFlow 1.2
            NXM_OF_IP_SRC (0x00000e04), introduced in Open vSwitch 1.1

IPv4 Destination Address Field
Name: ip_dst (aka nw_dst)
Width: 32 bits
Masking: arbitrary bitwise masks
Prerequisites: MFP_IPV4
Access: read/write
OpenFlow 1.0: yes (CIDR masks only)
OpenFlow 1.1: yes
OpenFlow 1.2: yes
OpenFlow 1.3: yes
NXM: yes (CIDR masks only before Open vSwitch 1.8)
Code Points: OXM_OF_IPV4_DST (0x80001804), introduced in OpenFlow 1.2
            NXM_OF_IP_DST (0x00001004), introduced in Open vSwitch 1.1

IPv6 Source Address Field
Name: ipv6_src
Width: 128 bits
Masking: arbitrary bitwise masks
Prerequisites: MFP_IPV6
Access: read/write
OpenFlow 1.0: no
OpenFlow 1.1: no
OpenFlow 1.2: yes
OpenFlow 1.3: yes
NXM: yes (CIDR masks only before Open vSwitch 1.8)
Code Points: OXM_OF_IPV6_SRC (0x80003410), introduced in OpenFlow 1.2
            NXM_NX_IPV6_SRC (0x00012610), introduced in Open vSwitch 1.1

IPv6 Destination Address Field
Name: ipv6_dst
Width: 128 bits
Masking: arbitrary bitwise masks
Prerequisites: MFP_IPV6
Access: read/write
OpenFlow 1.0: no
OpenFlow 1.1: no
OpenFlow 1.2: yes
OpenFlow 1.3: yes
NXM: yes (CIDR masks only before Open vSwitch 1.8)
Code Points: OXM_OF_IPV6_DST (0x80003410), introduced in OpenFlow 1.2
            NXM_NX_IPV6_DST (0x00012610), introduced in Open vSwitch 1.1

IPv6 Flow Label Field
Name: ipv6_label
Width: 32 bits (only the least-significant 20 bits may be nonzero)
Masking: arbitrary bitwise masks
Prerequisites: MFP_IPV6
IPv4/v6 Protocol Field
Name: nw_proto (aka ip_proto)
Width: 8 bits
Masking: not maskable
Prerequisites: MFP_IP_ANY
Access: read-only
OpenFlow 1.0: yes
OpenFlow 1.1: yes
OpenFlow 1.2: yes
OpenFlow 1.3: yes
NXM: yes
Code Points: OXM_OF_IP_PROTO (0x80001401), introduced in OpenFlow 1.2
NXM_OF_IP_PROTO (0x00000c01), introduced in Open vSwitch 1.1

IPv4/v6 DSCP (Bits 2-7) Field
Name: nw_tos
Width: 8 bits
Masking: not maskable
Prerequisites: MFP_IP_ANY
Access: read/write
OpenFlow 1.0: yes
OpenFlow 1.1: yes
OpenFlow 1.2: yes (via NXM code point)
OpenFlow 1.3: yes (via NXM code point)
NXM: yes
Code Points: NXM_OF_IP_TOS (0x00000a01), introduced in Open vSwitch 1.1

IPv4/v6 DSCP (Bits 0-5) Field
Name: ip_dscp
Width: 8 bits (only the least-significant 6 bits may be nonzero)
Masking: not maskable
Prerequisites: MFP_IP_ANY
Access: read/write
OpenFlow 1.0: yes
OpenFlow 1.1: yes
OpenFlow 1.2: yes
OpenFlow 1.3: yes
NXM: yes (via OXM code point)
Code Points: OXM_OF_IP_DSCP (0x80001001), introduced in OpenFlow 1.2

IPv4/v6 ECN Field
Name: nw_ecn (aka ip_ecn)
Width: 8 bits (only the least-significant 2 bits may be nonzero)
Masking: not maskable
Prerequisites: MFP_IP_ANY
Access: read/write
OpenFlow 1.0: no
OpenFlow 1.1: no
OpenFlow 1.2: yes
OpenFlow 1.3: yes
NXM: yes
Code Points: OXM_OF_IP_ECN (0x80001201), introduced in OpenFlow 1.2
            NXM_NX_IP_ECN (0x00013801), introduced in Open vSwitch 1.4

IPv4/v6 TTL/Hop Limit Field
Name: nw_ttl
Width: 8 bits
Masking: not maskable
Prerequisites: MFP_IP_ANY
Access: read/write
OpenFlow 1.0: no
OpenFlow 1.1: no
OpenFlow 1.2: yes (via NXM code point)
OpenFlow 1.3: yes (via NXM code point)
NXM: yes
Code Points: NXM_NX_IP_TTL (0x00013a01), introduced in Open vSwitch 1.4

IPv4/v6 Fragment Bitmask Field
Name: ip_frag
Width: 8 bits (only the least-significant 2 bits may be nonzero)
Masking: arbitrary bitwise masks
Prerequisites: MFP_IP_ANY
Access: read-only
OpenFlow 1.0: no
OpenFlow 1.1: no
OpenFlow 1.2: yes (via NXM code point)
OpenFlow 1.3: yes (via NXM code point)
NXM: yes
Code Points: NXM_NX_IP_FRAG (0x00013401), introduced in Open vSwitch 1.3

ARP Opcode Field
Name: arp_op
Width: 16 bits
Masking: not maskable
Prerequisites: MFP_ARP
Access: read/write
OpenFlow 1.0: yes
OpenFlow 1.1: yes
OpenFlow 1.2: yes
OpenFlow 1.3: yes
NXM: yes
Code Points: OXM_OF_ARP_OP (0x80002a02), introduced in OpenFlow 1.2
             NXM_OF_ARP_OP (0x00001e02), introduced in Open vSwitch 1.1

ARP Source IPv4 Address Field
Name: arp_spa
Width: 32 bits
Masking: arbitrary bitwise masks
Prerequisites: MFP_ARP
Access: read/write
OpenFlow 1.0: yes (exact match only)
OpenFlow 1.1: yes
OpenFlow 1.2: yes
ARP Target IPv4 Address Field
Name: arp_tpa
Width: 32 bits
Masking: arbitrary bitwise masks
Prerequisites: MFP_ARP
Access: read/write
OpenFlow 1.0: yes (exact match only)
OpenFlow 1.1: yes
OpenFlow 1.2: yes
OpenFlow 1.3: yes
NXM: yes
Code Points: OXM_OF_ARP_TPA (0x80002e04), introduced in OpenFlow 1.2
NXM_OF_ARP_TPA (0x00002204), introduced in Open vSwitch 1.1

ARP Source Ethernet Address Field
Name: arp_sha
Width: 48 bits
Masking: arbitrary bitwise masks
Prerequisites: MFP_ARP
Access: read/write
OpenFlow 1.0: no
OpenFlow 1.1: no
OpenFlow 1.2: yes
OpenFlow 1.3: yes
NXM: yes (maskable since Open vSwitch 1.9)
Code Points: OXM_OF_ARP_SHA (0x80003006), introduced in OpenFlow 1.2
NXM_NX_ARP_SHA (0x00012206), introduced in Open vSwitch 1.1

ARP Target Ethernet Address Field
Name: arp_tha
Width: 48 bits
Masking: arbitrary bitwise masks
Prerequisites: MFP_ARP
Access: read/write
OpenFlow 1.0: no
OpenFlow 1.1: no
OpenFlow 1.2: yes
OpenFlow 1.3: yes
NXM: yes (maskable since Open vSwitch 1.9)
Code Points: OXM_OF_ARP_THA (0x80003206), introduced in OpenFlow 1.2
NXM_NX_ARP_THA (0x00012406), introduced in Open vSwitch 1.1

Layer 4 Fields
TCP Source Port Field
Name: tcp_src (aka tp_src)
Width: 16 bits
Masking: arbitrary bitwise masks
Prerequisites: MFP_TCP
Access: read/write
OpenFlow 1.0: yes (exact match only)
OpenFlow 1.1: yes (exact match only)
TCP Destination Port Field
Name: tcp_dst (aka tp_dst)
Width: 16 bits
Masking: arbitrary bitwise masks
Prerequisites: MFP_TCP
Access: read/write
OpenFlow 1.0: yes (exact match only)
OpenFlow 1.1: yes (exact match only)
OpenFlow 1.2: yes
OpenFlow 1.3: yes
NXM: yes (maskable since Open vSwitch 1.6)
Code Points: OXM_OF_TCP_DST (0x80001c02), introduced in OpenFlow 1.2
NXM_OF_TCP_DST (0x00001402), introduced in Open vSwitch 1.1

TCP Flags Field
Name: tcp_flags
Width: 16 bits (only the least-significant 12 bits may be nonzero)
Masking: arbitrary bitwise masks
Prerequisites: MFP_TCP
Access: read-only
OpenFlow 1.0: no
OpenFlow 1.1: no
OpenFlow 1.2: yes (via NXM code point)
OpenFlow 1.3: yes (via NXM code point)
NXM: yes
Code Points: NXM_NX_TCP_FLAGS (0x00014402), introduced in Open vSwitch 2.1

UDP Source Port Field
Name: udp_src
Width: 16 bits
Masking: arbitrary bitwise masks
Prerequisites: MFP_UDP
Access: read/write
OpenFlow 1.0: yes (exact match only)
OpenFlow 1.1: yes (exact match only)
OpenFlow 1.2: yes
OpenFlow 1.3: yes
NXM: yes (maskable since Open vSwitch 1.6)
Code Points: OXM_OF_UDP_SRC (0x80001e02), introduced in OpenFlow 1.2
NXM_OF_UDP_SRC (0x00001602), introduced in Open vSwitch 1.1

UDP Destination Port Field
Name: udp_dst
Width: 16 bits
Masking: arbitrary bitwise masks
Prerequisites: MFP_UDP
Access: read/write
OpenFlow 1.0: yes (exact match only)
OpenFlow 1.1: yes (exact match only)
OpenFlow 1.2: yes
SCTP Source Port Field
Name: sctp_src
Width: 16 bits
Masking: arbitrary bitwise masks
Prerequisites: MFP_SCTP
Access: read/write
OpenFlow 1.0: no
OpenFlow 1.1: yes (exact match only)
OpenFlow 1.2: yes
OpenFlow 1.3: yes
NXM: yes (via OXM code point)
Code Points: OXM_OF_SCTP_SRC (0x80002202), introduced in OpenFlow 1.2

SCTP Destination Port Field
Name: sctp_dst
Width: 16 bits
Masking: arbitrary bitwise masks
Prerequisites: MFP_SCTP
Access: read/write
OpenFlow 1.0: no
OpenFlow 1.1: yes (exact match only)
OpenFlow 1.2: yes
OpenFlow 1.3: yes
NXM: yes (via OXM code point)
Code Points: OXM_OF_SCTP_DST (0x80002402), introduced in OpenFlow 1.2

ICMPv4 Type Field
Name: icmp_type
Width: 8 bits
Masking: not maskable
Prerequisites: MFP_ICMPV4
Access: read-only
OpenFlow 1.0: yes
OpenFlow 1.1: yes
OpenFlow 1.2: yes
OpenFlow 1.3: yes
NXM: yes
Code Points: OXM_OF_ICMPV4_TYPE (0x80002601), introduced in OpenFlow 1.2
NXM_OF_ICMP_TYPE (0x00001a01), introduced in Open vSwitch 1.1

ICMPv4 Code Field
Name: icmp_code
Width: 8 bits
Masking: not maskable
Prerequisites: MFP_ICMPV4
Access: read-only
OpenFlow 1.0: yes
OpenFlow 1.1: yes
OpenFlow 1.2: yes
OpenFlow 1.3: yes
NXM: yes
ICMPv6 Type Field

Name: icmpv6_type
Width: 8 bits
Masking: not maskable
Prerequisites: MFP_ICMPV6
Access: read-only
OpenFlow 1.0: no
OpenFlow 1.1: no
OpenFlow 1.2: yes
OpenFlow 1.3: yes
NXM: yes
Code Points: OXM_OF_ICMPV6_CODE (0x80003a01), introduced in OpenFlow 1.2
NXM_NX_ICMPV6_CODE (0x00012a01), introduced in Open vSwitch 1.1

ICMPv6 Code Field

Name: icmpv6_code
Width: 8 bits
Masking: not maskable
Prerequisites: MFP_ICMPV6
Access: read-only
OpenFlow 1.0: no
OpenFlow 1.1: no
OpenFlow 1.2: yes
OpenFlow 1.3: yes
NXM: yes
Code Points: OXM_OF_ICMPV6_CODE (0x80003c01), introduced in OpenFlow 1.2
NXM_NX_ICMPV6_CODE (0x00012c01), introduced in Open vSwitch 1.1

ICMPv6 Neighbor Discovery Target IPv6 Field

Name: nd_target
Width: 128 bits
Masking: arbitrary bitwise masks
Prerequisites: MFP_ND
Access: read-only
OpenFlow 1.0: no
OpenFlow 1.1: no
OpenFlow 1.2: yes
OpenFlow 1.3: yes
NXM: yes (CIDR masks only before Open vSwitch 1.8)
Code Points: OXM_OF_IPV6_ND_TARGET (0x80003e01), introduced in OpenFlow 1.2
NXM_NX_IPV6_ND_TARGET (0x00012e01), introduced in Open vSwitch 1.1

ICMPv6 Neighbor Discovery Source Ethernet Address Field

Name: nd_sll
Width: 48 bits
Masking: arbitrary bitwise masks
Prerequisites: MFP_ND_SOLICIT
Access: read-only
OpenFlow 1.0: no
OpenFlow 1.1: no
OpenFlow 1.2: yes
OpenFlow 1.3: yes
NXM: yes (maskable since Open vSwitch 1.9)
Code Points:  
**OXM_OF_IPV6_ND_SLL** (0x00004006), introduced in OpenFlow 1.2  
**NXM_NX_ND_SLL** (0x00013006), introduced in Open vSwitch 1.1

**ICMPv6 Neighbor Discovery Target Ethernet Address Field**

- **Name:** nd_tll
- **Width:** 48 bits
- **Masking:** arbitrary bitwise masks
- **Prerequisites:** MFP_ND_ADVERT
- **Access:** read-only
- **OpenFlow 1.0:** no
- **OpenFlow 1.1:** no
- **OpenFlow 1.2:** yes
- **OpenFlow 1.3:** yes
- **NXM:** yes (maskable since Open vSwitch 1.9)

Code Points:  
**OXM_OF_IPV6_ND_TLL** (0x00004206), introduced in OpenFlow 1.2  
**NXM_NX_ND_TLL** (0x00013206), introduced in Open vSwitch 1.1