

---

# os-ken Documentation

*Release 2.11.3.dev6*

**OpenStack Developers**

**Feb 14, 2025**

# CONTENTS

<b>1</b>	<b>Overview</b>	<b>1</b>
<b>2</b>	<b>Usage</b>	<b>2</b>
<b>3</b>	<b>Contributor Documentation</b>	<b>3</b>
<b>4</b>	<b>Configuration</b>	<b>4</b>
<b>5</b>	<b>Users guide</b>	<b>5</b>
<b>6</b>	<b>Administrators guide</b>	<b>6</b>
<b>7</b>	<b>References</b>	<b>7</b>
<b>8</b>	<b>Archived Contents</b>	<b>8</b>
8.1	Writing Your OS-Ken Application . . . . .	8
8.1.1	The First Application . . . . .	8
8.1.2	Components of OS-Ken . . . . .	10
8.1.3	OS-Ken application API . . . . .	13
8.1.4	Library . . . . .	19
8.1.5	OpenFlow protocol API Reference . . . . .	160
8.1.6	Nicira Extension Structures . . . . .	543
8.1.7	OS-Ken API Reference . . . . .	569
8.2	Configuration . . . . .	571
8.2.1	Setup TLS Connection . . . . .	571
8.3	Tests . . . . .	572
8.3.1	Testing VRRP Module . . . . .	572
8.3.2	Testing OF-config support with LINC . . . . .	576
8.4	Snort Intergration . . . . .	580
8.4.1	Overview . . . . .	580
8.4.2	Installation Snort . . . . .	581
8.4.3	Configure Snort . . . . .	581
8.4.4	Usage . . . . .	581
8.5	Built-in OS-Ken applications . . . . .	583
8.5.1	os_ken.app.ofctl . . . . .	583
	<b>Python Module Index</b>	<b>585</b>
	<b>Index</b>	<b>586</b>

## OVERVIEW

A component-based software defined networking framework in OpenStack.

os-ken is a fork of the Ryu library tailored for OpenStack Neutron.

- License: Apache License, Version 2.0
- Documentation: <https://docs.openstack.org/os-ken/latest/>
- Source: <https://opendev.org/openstack/os-ken/>
- Bugs: <https://storyboard.openstack.org/#!/project/openstack/os-ken>
- Release Notes: <https://docs.openstack.org/releasenotes/os-ken/>

---

## CHAPTER TWO

---

### USAGE

To use os-ken in a project:

```
import os\_ken
```

## CONTRIBUTOR DOCUMENTATION

If you would like to contribute to the development of OpenStack, you must follow the steps in this page:

<https://docs.openstack.org/infra/manual/developers.html>

If you already have a good understanding of how the system works and your OpenStack accounts are set up, you can skip to the development workflow section of this documentation to learn how changes to OpenStack should be submitted for review via the Gerrit tool:

<https://docs.openstack.org/infra/manual/developers.html#development-workflow>

Pull requests submitted through GitHub will be ignored.

Bugs should be filed on Launchpad: <https://bugs.launchpad.net/neutron>

---

**CHAPTER  
FOUR**

---

**CONFIGURATION**

T.B.D.

---

**CHAPTER  
FIVE**

---

**USERS GUIDE**

T.B.D.

**ADMINISTRATORS GUIDE**

T.B.D.



---

**CHAPTER  
SEVEN**

---

**REFERENCES**

T.B.D.

## ARCHIVED CONTENTS

### Note

Contents here are imported from the upstream Ryu documentation. They will be merged into the OS-Ken documentation gradually.

## 8.1 Writing Your OS-Ken Application

### 8.1.1 The First Application

#### Whetting Your Appetite

If you want to manage the network gears (switches, routers, etc) at your way, you need to write your OS-Ken application. Your application tells OS-Ken how you want to manage the gears. Then OS-Ken configures the gears by using OpenFlow protocol, etc.

Writing OS-Ken application is easy. It's just Python scripts.

#### Start Writing

We show a OS-Ken application that make OpenFlow switches work as a dumb layer 2 switch.

Open a text editor creating a new file with the following content:

```
from os_ken.base import app_manager

class L2Switch(app_manager.OSKenApp):
    def __init__(self, *args, **kwargs):
        super(L2Switch, self).__init__(*args, **kwargs)
```

OS-Ken application is just a Python script so you can save the file with any name, extensions, and any place you want. Let's name the file 'l2.py' at your home directory.

This application does nothing useful yet, however it's a complete OS-Ken application. In fact, you can run this OS-Ken application:

```
% osken-manager ~/l2.py
loading app /Users/fujita/l2.py
instantiating app /Users/fujita/l2.py
```

All you have to do is defining needs a new subclass of OSKenApp to run your Python script as a OS-Ken application.

Next let's add the functionality of sending a received packet to all the ports.

```

from os_ken.base import app_manager
from os_ken.controller import ofp_event
from os_ken.controller.handler import MAIN_DISPATCHER
from os_ken.controller.handler import set_ev_cls
from os_ken.ofproto import ofproto_v1_0

class L2Switch(app_manager.OSKenApp):
    OFP_VERSIONS = [ofproto_v1_0.OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super(L2Switch, self).__init__(*args, **kwargs)

    @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
    def packet_in_handler(self, ev):
        msg = ev.msg
        dp = msg.datapath
        ofp = dp.ofproto
        ofp_parser = dp.ofproto_parser

        actions = [ofp_parser.OFPActionOutput(ofp.OFPP_FLOOD)]
        out = ofp_parser.OFPPacketOut(
            datapath=dp, buffer_id=msg.buffer_id, in_port=msg.in_port,
            actions=actions)
        dp.send_msg(out)

```

A new method 'packet\_in\_handler' is added to L2Switch class. This is called when OS-Ken receives an OpenFlow packet\_in message. The trick is 'set\_ev\_cls' decorator. This decorator tells OS-Ken when the decorated function should be called.

The first argument of the decorator indicates an event that makes function called. As you expect easily, every time OS-Ken gets a packet\_in message, this function is called.

The second argument indicates the state of the switch. Probably, you want to ignore packet\_in messages before the negotiation between OS-Ken and the switch finishes. Using 'MAIN\_DISPATCHER' as the second argument means this function is called only after the negotiation completes.

Next let's look at the first half of the 'packet\_in\_handler' function.

- ev.msg is an object that represents a packet\_in data structure.
- msg.dp is an object that represents a datapath (switch).
- dp.ofproto and dp.ofproto\_parser are objects that represent the OpenFlow protocol that OS-Ken and the switch negotiated.

Ready for the second half.

- OFPActionOutput class is used with a packet\_out message to specify a switch port that you want to send the packet out of. This application need a switch to send out of all the ports so OFPP\_FLOOD constant is used.
- OFPPacketOut class is used to build a packet\_out message.
- If you call Datapath class's send\_msg method with a OpenFlow message class object, OS-Ken

builds and send the on-wire data format to the switch.

Here, you finished implementing your first OS-Ken application. You are ready to run this OS-Ken application that does something useful.

A dumb I2 switch is too dumb? You want to implement a learning I2 switch? Move to [the next step](#). You can learn from the existing OS-Ken applications at [os\\_ken/app](#) directory and [integrated tests](#) directory.

## 8.1.2 Components of OS-Ken

### Executables

#### **osken-manager**

The main executable.

### Base components

#### **os\_ken.base.app\_manager**

The central management of OSKen applications.

- Load OSKen applications
- Provide *contexts* to OSKen applications
- Route messages among OSKen applications

### OpenFlow controller

#### **os\_ken.controller.controller**

The main component of OpenFlow controller.

- Handle connections from switches
- Generate and route events to appropriate entities like OSKen applications

#### **os\_ken.controller.dpset**

Manage switches.

Planned to be replaced by [os\\_ken/topology](#).

#### **os\_ken.controller.ofp\_event**

OpenFlow event definitions.

#### **os\_ken.controller.ofp\_handler**

Basic OpenFlow handling including negotiation.

### OpenFlow wire protocol encoder and decoder

#### **os\_ken.ofproto.ofproto\_v1\_0**

OpenFlow 1.0 definitions.

### **os\_ken.ofproto.ofproto\_v1\_0\_parser**

Decoder/Encoder implementations of OpenFlow 1.0.

### **os\_ken.ofproto.ofproto\_v1\_2**

OpenFlow 1.2 definitions.

### **os\_ken.ofproto.ofproto\_v1\_2\_parser**

Decoder/Encoder implementations of OpenFlow 1.2.

### **os\_ken.ofproto.ofproto\_v1\_3**

OpenFlow 1.3 definitions.

### **os\_ken.ofproto.ofproto\_v1\_3\_parser**

This module implements OpenFlow 1.3.x.

This module also implements some of extensions shown in "OpenFlow Extensions for 1.3.X Pack 1". Namely, the following extensions are implemented.

- EXT-230 Bundle Extension (without bundle properties)
- EXT-236 Bad flow entry priority error Extension
- EXT-237 Set async config error Extension
- EXT-256 PBB UCA header field Extension
- EXT-260 Duplicate instruction error Extension
- EXT-264 Multipart timeout errors Extension

The following extensions are partially implemented.

- EXT-187 Flow entry notifications Extension (ONFMP\_FLOW\_MONITOR only)
- EXT-232 Table synchronisation Extension (Error codes only)

The following extensions are not implemented yet.

- EXT-191 Role Status Extension
- EXT-192-e Flow entry eviction Extension
- EXT-192-v Vacancy events Extension

### **os\_ken.ofproto.ofproto\_v1\_4**

OpenFlow 1.4 definitions.

### **os\_ken.ofproto.ofproto\_v1\_4\_parser**

Decoder/Encoder implementations of OpenFlow 1.4.

### **os\_ken.ofproto.ofproto\_v1\_5**

OpenFlow 1.5 definitions.

### **os\_ken.ofproto.ofproto\_v1\_5\_parser**

Decoder/Encoder implementations of OpenFlow 1.5.

## **OS-Ken applications**

### **os\_ken.topology**

Switch and link discovery module. Planned to replace os\_ken/controller/dpset.

## **Libraries**

### **os\_ken.lib.packet**

OSKen packet library. Decoder/Encoder implementations of popular protocols like TCP/IP.

### **os\_ken.lib.ovs**

ovsdb interaction library.

### **os\_ken.lib.of\_config**

OF-Config implementation.

### **os\_ken.lib.netconf**

NETCONF definitions used by os\_ken/lib/of\_config.

### **os\_ken.lib.xflow**

An implementation of sFlow and NetFlow.

## **Third party libraries**

### **os\_ken.contrib.ovs**

Open vSwitch python binding. Used by os\_ken.lib.ovs.

### **os\_ken.contrib.oslo.config**

Oslo configuration library. Used for osken-manager's command-line options and configuration files.

### **os\_ken.contrib.ncclient**

Python library for NETCONF client. Used by os\_ken.lib.of\_config.

### 8.1.3 OS-Ken application API

#### OS-Ken application programming model

##### Threads, events, and event queues

OS-Ken applications are single-threaded entities which implement various functionalities in OS-Ken. Events are messages between them.

OS-Ken applications send asynchronous events to each other. Besides that, there are some OS-Ken-internal event sources which are not OS-Ken applications. One of the examples of such event sources is the OpenFlow controller. While an event can currently contain arbitrary python objects, it's discouraged to pass complex objects (eg. unpickleable objects) between OS-Ken applications.

Each OS-Ken application has a receive queue for events. The queue is FIFO and preserves the order of events. Each OS-Ken application has a thread for event processing. The thread keeps draining the receive queue by dequeuing an event and calling the appropriate event handler for the event type. Because the event handler is called in the context of the event processing thread, it should be careful when blocking. While an event handler is blocked, no further events for the OS-Ken application will be processed.

There are kinds of events which are used to implement synchronous inter-application calls between OS-Ken applications. While such requests use the same machinery as ordinary events, their replies are put on a queue dedicated to the transaction to avoid deadlock.

While threads and queues are currently implemented with eventlet/greenlet, a direct use of them in a OS-Ken application is strongly discouraged.

##### Contexts

Contexts are ordinary python objects shared among OS-Ken applications. The use of contexts is discouraged for new code.

##### Create a OS-Ken application

A OS-Ken application is a python module which defines a subclass of `os_ken.base.app_manager.OSKenApp`. If two or more such classes are defined in a module, the first one (by name order) will be picked by `app_manager`. An OS-Ken application is singleton: only a single instance of a given OS-Ken application is supported.

##### Observe events

A OS-Ken application can register itself to listen for specific events using `os_ken.controller.handler.set_ev_cls` decorator.

##### Generate events

A OS-Ken application can raise events by calling appropriate `os_ken.base.app_manager.OSKenApp`'s methods like `send_event` or `send_event_to_observers`.

##### Event classes

An event class describes a OS-Ken event generated in the system. By convention, event class names are prefixed by "Event". Events are generated either by the core part of OS-Ken or OS-Ken applications. A OS-Ken application can register its interest for a specific type of event by providing a handler method using the `os_ken.controller.handler.set_ev_cls` decorator.

## OpenFlow event classes

os\_ken.controller.ofp\_event module exports event classes which describe receptions of OpenFlow messages from connected switches. By convention, they are named as os\_ken.controller.ofp\_event.EventOFPxxxx where xxxx is the name of the corresponding OpenFlow message. For example, EventOFPPacketIn for the packet-in message. The OpenFlow controller part of OS-Ken automatically decodes OpenFlow messages received from switches and send these events to OS-Ken applications which expressed an interest using os\_ken.controller.handler.set\_ev\_cls. OpenFlow event classes are subclasses of the following class.

**class** os\_ken.controller.ofp\_event.EventOFPMsgBase(msg)

The base class of OpenFlow event class.

OpenFlow event classes have at least the following attributes.

Attribute	Description
msg	An object which describes the corresponding OpenFlow message.
msg.datapath	A os_ken.controller.controller.Datapath instance which describes an OpenFlow switch from which we received this OpenFlow message.
timestamp	Timestamp when Datapath instance generated this event.

The msg object has some more additional members whose values are extracted from the original OpenFlow message.

See *OpenFlow protocol API Reference* for more info about OpenFlow messages.

## os\_ken.base.app\_manager.OSKenApp

See *OS-Ken API Reference*.

## os\_ken.controller.handler.set\_ev\_cls

os\_ken.controller.handler.set\_ev\_cls(ev\_cls, dispatchers=None)

A decorator for OSKen application to declare an event handler.

Decorated method will become an event handler. ev\_cls is an event class whose instances this OS-KenApp wants to receive. dispatchers argument specifies one of the following negotiation phases (or a list of them) for which events should be generated for this handler. Note that, in case an event changes the phase, the phase before the change is used to check the interest.

Negotiation phase	Description
os_ken.controller.handler.HANDSHAKE_DISPATCHER	Sending and waiting for hello message
os_ken.controller.handler.CONFIG_DISPATCHER	Version negotiated and sent features-request message
os_ken.controller.handler.MAIN_DISPATCHER	Switch-features message received and sent set-config message
os_ken.controller.handler.DEAD_DISPATCHER	Disconnect from the peer. Or disconnecting due to some unrecoverable errors.



### os\_ken.controller.controller.Datapath

**class** os\_ken.controller.controller.Datapath(*socket, address*)

A class to describe an OpenFlow switch connected to this controller.

An instance has the following attributes.

Attribute	Description
id	64-bit OpenFlow Datapath ID. Only available for os_ken.controller.handler.MAIN_DISPATCHER phase.
ofproto	A module which exports OpenFlow definitions, mainly constants appeared in the specification, for the negotiated OpenFlow version. For example, os_ken.ofproto.ofproto_v1_0 for OpenFlow 1.0.
ofproto_parser	A module which exports OpenFlow wire message encoder and decoder for the negotiated OpenFlow version. For example, os_ken.ofproto.ofproto_v1_0_parser for OpenFlow 1.0.
ofproto_parser.OFPxxxx(datapath,...)	A callable to prepare an OpenFlow message for the given switch. It can be sent with Datapath.send_msg later. xxxx is a name of the message. For example OFPFlowMod for flow-mod message. Arguments depend on the message.
set_xid(self, msg)	Generate an OpenFlow XID and put it in msg.xid.
send_msg(self, msg)	Queue an OpenFlow message to send to the corresponding switch. If msg.xid is None, set_xid is automatically called on the message before queueing.
send_packet_out	deprecated
send_flow_mod	deprecated
send_flow_del	deprecated
send_delete_all_flows	deprecated
send_barrier	Queue an OpenFlow barrier message to send to the switch.
send_nxt_set_flow_format	deprecated
is_reserved_port	deprecated

### os\_ken.controller.event.EventBase

**class** os\_ken.controller.event.EventBase

The base of all event classes.

A OSKen application can define its own event type by creating a subclass.

### os\_ken.controller.event.EventRequestBase

**class** os\_ken.controller.event.EventRequestBase

The base class for synchronous request for OSKenApp.send\_request.

**os\_ken.controller.event.EventReplyBase**

**class** os\_ken.controller.event.EventReplyBase(*dst*)

The base class for synchronous request reply for OSKenApp.send\_reply.

**os\_ken.controller.ofp\_event.EventOFPPStateChange**

**class** os\_ken.controller.ofp\_event.EventOFPPStateChange(*dp*)

An event class for negotiation phase change notification.

An instance of this class is sent to observer after changing the negotiation phase. An instance has at least the following attributes.

Attribute	Description
datapath	os_ken.controller.controller.Datapath instance of the switch

**os\_ken.controller.ofp\_event.EventOFPPortStateChange**

**class** os\_ken.controller.ofp\_event.EventOFPPortStateChange(*dp, reason, port\_no*)

An event class to notify the port state changes of Datapath instance.

This event performs like EventOFPPortStatus, but OSKen will send this event after updating ports dict of Datapath instances. An instance has at least the following attributes.

Attribute	Description
datapath	os_ken.controller.controller.Datapath instance of the switch
reason	one of OFPPR_*
port_no	Port number which state was changed

**os\_ken.controller.dpset.EventDP**

**class** os\_ken.controller.dpset.EventDP(*dp, enter\_leave*)

An event class to notify connect/disconnect of a switch.

For OpenFlow switches, one can get the same notification by observing os\_ken.controller.ofp\_event.EventOFPPStateChange. An instance has at least the following attributes.

Attribute	Description
dp	A os_ken.controller.controller.Datapath instance of the switch
enter	True when the switch connected to our controller. False for disconnect.
ports	A list of port instances.

**os\_ken.controller.dpset.EventPortAdd**

**class** os\_ken.controller.dpset.EventPortAdd(*dp, port*)

An event class for switch port status "ADD" notification.

This event is generated when a new port is added to a switch. For OpenFlow switches, one can get the same notification by observing `os_ken.controller.ofp_event.EventOFPPortStatus`. An instance has at least the following attributes.

Attribute	Description
<code>dp</code>	A <code>os_ken.controller.controller.Datapath</code> instance of the switch
<code>port</code>	port number

### `os_ken.controller.dpset.EventPortDelete`

**class** `os_ken.controller.dpset.EventPortDelete`(*dp*, *port*)

An event class for switch port status "DELETE" notification.

This event is generated when a port is removed from a switch. For OpenFlow switches, one can get the same notification by observing `os_ken.controller.ofp_event.EventOFPPortStatus`. An instance has at least the following attributes.

Attribute	Description
<code>dp</code>	A <code>os_ken.controller.controller.Datapath</code> instance of the switch
<code>port</code>	port number

### `os_ken.controller.dpset.EventPortModify`

**class** `os_ken.controller.dpset.EventPortModify`(*dp*, *new\_port*)

An event class for switch port status "MODIFY" notification.

This event is generated when some attribute of a port is changed. For OpenFlow switches, one can get the same notification by observing `os_ken.controller.ofp_event.EventOFPPortStatus`. An instance has at least the following attributes.

Attribute	Description
<code>dp</code>	A <code>os_ken.controller.controller.Datapath</code> instance of the switch
<code>port</code>	port number

### `os_ken.controller.network.EventNetworkPort`

**class** `os_ken.controller.network.EventNetworkPort`(*network\_id*, *dpid*, *port\_no*, *add\_del*)

An event class for notification of port arrival and departure.

This event is generated when a port is introduced to or removed from a network by the REST API. An instance has at least the following attributes.

Attribute	Description
<code>network_id</code>	Network ID
<code>dpid</code>	OpenFlow Datapath ID of the switch to which the port belongs.
<code>port_no</code>	OpenFlow port number of the port
<code>add_del</code>	True for adding a port. False for removing a port.

### os\_ken.controller.network.EventNetworkDel

**class** os\_ken.controller.network.**EventNetworkDel**(*network\_id*)

An event class for network deletion.

This event is generated when a network is deleted by the REST API. An instance has at least the following attributes.

Attribute	Description
network_id	Network ID

### os\_ken.controller.network.EventMacAddress

**class** os\_ken.controller.network.**EventMacAddress**(*dpid*, *port\_no*, *network\_id*,  
*mac\_address*, *add\_del*)

An event class for end-point MAC address registration.

This event is generated when a end-point MAC address is updated by the REST API. An instance has at least the following attributes.

Attribute	Description
network_id	Network ID
dpid	OpenFlow Datapath ID of the switch to which the port belongs.
port_no	OpenFlow port number of the port
mac_address	The old MAC address of the port if add_del is False. Otherwise the new MAC address.
add_del	False if this event is a result of a port removal. Otherwise True.

### os\_ken.controller.tunnels.EventTunnelKeyAdd

**class** os\_ken.controller.tunnels.**EventTunnelKeyAdd**(*network\_id*, *tunnel\_key*)

An event class for tunnel key registration.

This event is generated when a tunnel key is registered or updated by the REST API. An instance has at least the following attributes.

Attribute	Description
network_id	Network ID
tunnel_key	Tunnel Key

### os\_ken.controller.tunnels.EventTunnelKeyDel

**class** os\_ken.controller.tunnels.**EventTunnelKeyDel**(*network\_id*, *tunnel\_key*)

An event class for tunnel key registration.

This event is generated when a tunnel key is removed by the REST API. An instance has at least the following attributes.

Attribute	Description
network_id	Network ID
tunnel_key	Tunnel Key

### os\_ken.controller.tunnels.EventTunnelPort

**class** os\_ken.controller.tunnels.**EventTunnelPort**(*dpid*, *port\_no*, *remote\_dpid*, *add\_del*)

An event class for tunnel port registration.

This event is generated when a tunnel port is added or removed by the REST API. An instance has at least the following attributes.

Attribute	Description
dpid	OpenFlow Datapath ID
port_no	OpenFlow port number
remote_dpid	OpenFlow port number of the tunnel peer
add_del	True for adding a tunnel. False for removal.

## 8.1.4 Library

OS-Ken provides some useful library for your network applications.

### Packet library

#### Introduction

OS-Ken packet library helps you to parse and build various protocol packets. dpkt is the popular library for the same purpose, however it is not designed to handle protocols that are interleaved; vlan, mpls, gre, etc. So we implemented our own packet library.

#### Network Addresses

Unless otherwise specified, MAC/IPv4/IPv6 addresses are specified using human readable strings for this library. For example, '08:60:6e:7f:74:e7', '192.0.2.1', 'fe80::a60:6eff:fe7f:74e7'.

#### Parsing Packet

First, let's look at how we can use the library to parse the received packets in a handler for OFPPacketIn messages.

```
from os_ken.lib.packet import packet

@handler.set_ev_cls(ofp_event.EventOFPPacketIn, handler.MAIN_DISPATCHER)
def packet_in_handler(self, ev):
    pkt = packet.Packet(array.array('B', ev.msg.data))
    for p in pkt.protocols:
        print p
```

You can create a Packet class instance with the received raw data. Then the packet library parses the data and creates protocol class instances included the data. The packet class 'protocols' has the protocol class instances.

If a TCP packet is received, something like the following is printed:

```
<os_ken.lib.packet.ethernet.ethernet object at 0x107a5d790>
<os_ken.lib.packet.vlan.vlan object at 0x107a5d7d0>
<os_ken.lib.packet.ipv4.ipv4 object at 0x107a5d810>
<os_ken.lib.packet.tcp.tcp object at 0x107a5d850>
```

If vlan is not used, you see something like:

```
<os_ken.lib.packet.ethernet.ethernet object at 0x107a5d790>
<os_ken.lib.packet.ipv4.ipv4 object at 0x107a5d810>
<os_ken.lib.packet.tcp.tcp object at 0x107a5d850>
```

You can access to a specific protocol class instance by using the packet class iterator. Let's try to check VLAN id if VLAN is used:

```
from os_ken.lib.packet import packet

@handler.set_ev_cls(ofp_event.EventOFPPacketIn, handler.MAIN_DISPATCHER)
def packet_in_handler(self, ev):
    pkt = packet.Packet(array.array('B', ev.msg.data))
    for p in pkt:
        print p.protocol_name, p
        if p.protocol_name == 'vlan':
            print 'vid = ', p.vid
```

You see something like:

```
ethernet <os_ken.lib.packet.ethernet.ethernet object at 0x107a5d790>
vlan <os_ken.lib.packet.vlan.vlan object at 0x107a5d7d0>
vid = 10
ipv4 <os_ken.lib.packet.ipv4.ipv4 object at 0x107a5d810>
tcp <os_ken.lib.packet.tcp.tcp object at 0x107a5d850>
```

## Building Packet

You need to create protocol class instances that you want to send, add them to a packet class instance via `add_protocol` method, and then call `serialize` method. You have the raw data to send. The following example is building an arp packet.

```
from os_ken.ofproto import ether
from os_ken.lib.packet import ethernet, arp, packet

e = ethernet.ethernet(dst='ff:ff:ff:ff:ff:ff',
                      src='08:60:6e:7f:74:e7',
                      ethertype=ether.ETH_TYPE_ARP)
a = arp.arp(hwtype=1, proto=0x0800, hlen=6, plen=4, opcode=2,
           src_mac='08:60:6e:7f:74:e7', src_ip='192.0.2.1',
```

(continues on next page)

(continued from previous page)

```

        dst_mac='00:00:00:00:00:00', dst_ip='192.0.2.2')
p = packet.Packet()
p.add_protocol(e)
p.add_protocol(a)
p.serialize()
print repr(p.data) # the on-wire packet

```

## Packet library API Reference

### Packet class

**class** `os_ken.lib.packet.packet.Packet`(*data=None, protocols=None, parse\_cls=<class 'os\_ken.lib.packet.ethernet.ethernet'>*)

A packet decoder/encoder class.

An instance is used to either decode or encode a single packet.

*data* is a bytearray to describe a raw datagram to decode. When decoding, a Packet object is iterable. Iterated values are protocol (ethernet, ipv4, ...) headers and the payload. Protocol headers are instances of subclass of `packet_base.PacketBase`. The payload is a bytearray. They are iterated in on-wire order.

*data* should be omitted when encoding a packet.

#### **add\_protocol**(*proto*)

Register a protocol *proto* for this packet.

This method is legal only when encoding a packet.

When encoding a packet, register a protocol (ethernet, ipv4, ...) header to add to this packet. Protocol headers should be registered in on-wire order before calling `self.serialize`.

**classmethod** `from_jsondict`(*dict\_, decode\_string=<function b64decode>*,  
\*\**additional\_args*)

Create an instance from a JSON style dict.

Instantiate this class with parameters specified by the dict.

This method takes the following arguments.

Argument	Description
<code>dict_</code>	A dictionary which describes the parameters. For example, {"Param1": 100, "Param2": 200}
<code>decode_string</code>	(Optional) specify how to decode strings. The default is base64. This argument is used only for attributes which don't have explicit type annotations in <code>_TYPE</code> class attribute.
<code>additional_args</code>	(Optional) Additional kwargs for constructor.

#### **get\_protocol**(*protocol*)

Returns the firstly found protocol that matches to the specified protocol.

**get\_protocols**(*protocol*)

Returns a list of protocols that matches to the specified protocol.

**serialize**()

Encode a packet and store the resulted bytearray in `self.data`.

This method is legal only when encoding a packet.

## Stream Parser class

**class** `os_ken.lib.packet.stream_parser.StreamParser`

Streaming parser base class.

An instance of a subclass of this class is used to extract messages from a raw byte stream.

It's designed to be used for data read from a transport which doesn't preserve message boundaries. A typical example of such a transport is TCP.

**exception** `TooSmallException`

**parse**(*data*)

Tries to extract messages from a raw byte stream.

The data argument would be python bytes newly read from the input stream.

Returns an ordered list of extracted messages. It can be an empty list.

The rest of data which doesn't produce a complete message is kept internally and will be used when more data is come. I.e. next time this method is called again.

**abstract try\_parse**(*q*)

Try to extract a message from the given bytes.

This is an override point for subclasses.

This method tries to extract a message from bytes given by the argument.

Raises `TooSmallException` if the given data is not enough to extract a complete message but there's still a chance to extract a message if more data is come later.

List of the sub-classes:

- `os_ken.lib.packet.bgp.StreamParser`

## Protocol Header classes

### Packet Base Class

**class** `os_ken.lib.packet.packet_base.PacketBase`

A base class for a protocol (ethernet, ipv4, ...) header.

**classmethod** `get_packet_type`(*type\_*)

Per-protocol dict-like get method.

Provided for convenience of protocol implementers. Internal use only.



**abstract classmethod parser**(*buf*)

Decode a protocol header.

This method is used only when decoding a packet.

Decode a protocol header at offset 0 in bytearray *buf*. Returns the following three objects.

- An object to describe the decoded header.
- A `packet_base.PacketBase` subclass appropriate for the rest of the packet. None when the rest of the packet should be considered as raw payload.
- The rest of packet.

**classmethod register\_packet\_type**(*cls\_*, *type\_*)

Per-protocol dict-like set method.

Provided for convenience of protocol implementers. Internal use only.

**serialize**(*payload*, *prev*)

Encode a protocol header.

This method is used only when encoding a packet.

Encode a protocol header. Returns a bytearray which contains the header.

*payload* is the rest of the packet which will immediately follow this header.

*prev* is a `packet_base.PacketBase` subclass for the outer protocol header. *prev* is None if the current header is the outer-most. For example, *prev* is `ipv4` or `ipv6` for `tcp.serialize`.

**ARP**

```
class os_ken.lib.packet.arp.arp(hwtype=1, proto=2048, hlen=6, plen=4, opcode=1,
                               src_mac='ff:ff:ff:ff:ff:ff', src_ip='0.0.0.0',
                               dst_mac='ff:ff:ff:ff:ff:ff', dst_ip='0.0.0.0')
```

ARP (RFC 826) header encoder/decoder class.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. IPv4 addresses are represented as a string like '192.0.2.1'. MAC addresses are represented as a string like '08:60:6e:7f:74:e7'. `__init__` takes the corresponding args in this order.

Attribute	Description	Example
<code>hwtype</code>	Hardware address.	
<code>proto</code>	Protocol address.	
<code>hlen</code>	byte length of each hardware address.	
<code>plen</code>	byte length of each protocol address.	
<code>opcode</code>	operation codes.	
<code>src_mac</code>	Hardware address of sender.	'08:60:6e:7f:74:e7'
<code>src_ip</code>	Protocol address of sender.	'192.0.2.1'
<code>dst_mac</code>	Hardware address of target.	'00:00:00:00:00:00'
<code>dst_ip</code>	Protocol address of target.	'192.0.2.2'

**classmethod parser(buf)**

Decode a protocol header.

This method is used only when decoding a packet.

Decode a protocol header at offset 0 in bytearray *buf*. Returns the following three objects.

- An object to describe the decoded header.
- A packet\_base.PacketBase subclass appropriate for the rest of the packet. None when the rest of the packet should be considered as raw payload.
- The rest of packet.

**serialize(payload, prev)**

Encode a protocol header.

This method is used only when encoding a packet.

Encode a protocol header. Returns a bytearray which contains the header.

*payload* is the rest of the packet which will immediately follow this header.

*prev* is a packet\_base.PacketBase subclass for the outer protocol header. *prev* is None if the current header is the outer-most. For example, *prev* is ipv4 or ipv6 for tcp.serialize.

**os\_ken.lib.packet.arp.arp\_ip(opcode, src\_mac, src\_ip, dst\_mac, dst\_ip)**

A convenient wrapper for IPv4 ARP for Ethernet.

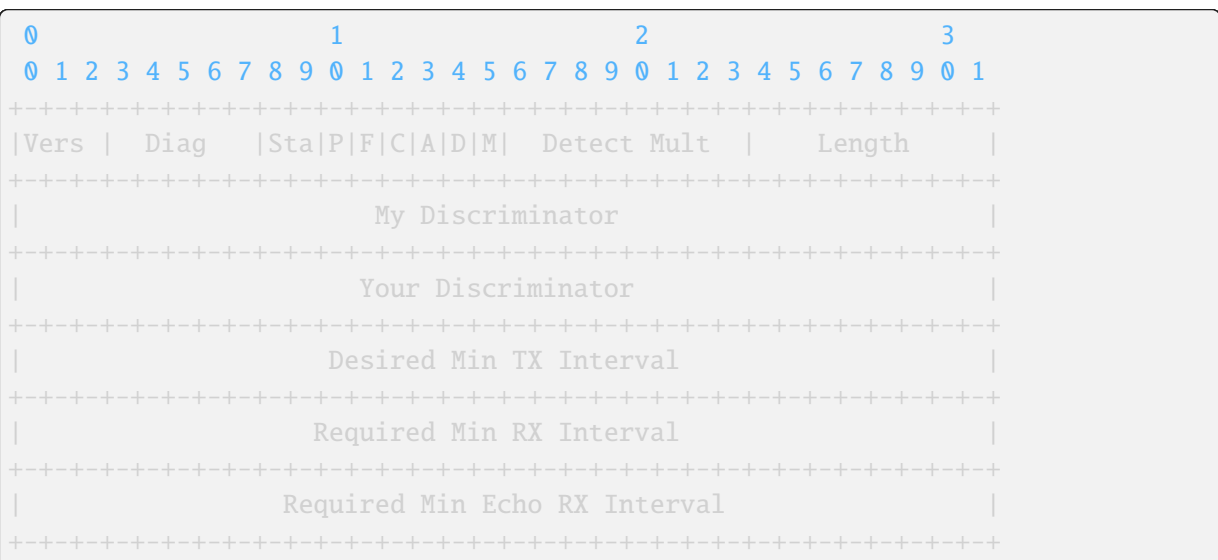
This is an equivalent of the following code.

```
arp(ARP_HW_TYPE_ETHERNET, ether.ETH_TYPE_IP, 6, 4, opcode, src_mac,
src_ip, dst_mac, dst_ip)
```

**BFD**

BFD Control packet parser/serializer

[RFC 5880] BFD Control packet format:

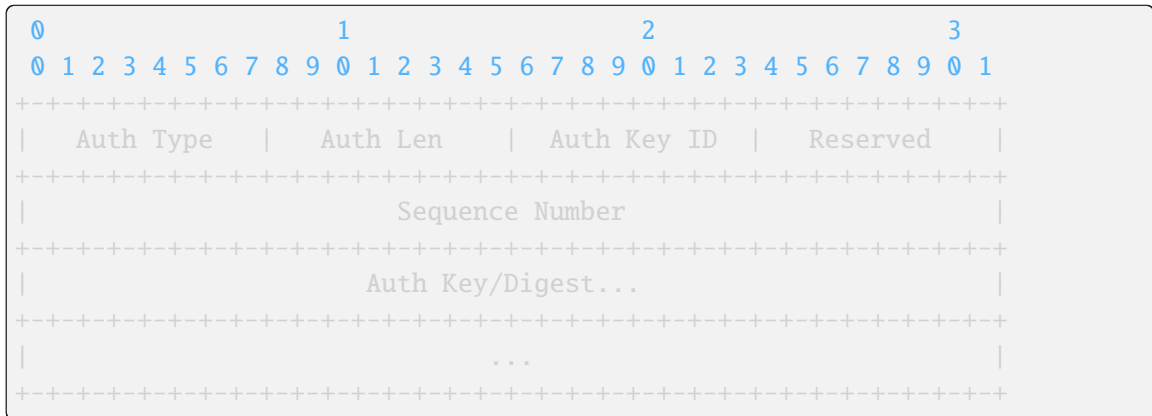


An optional Authentication Section MAY be present in the following format of types:

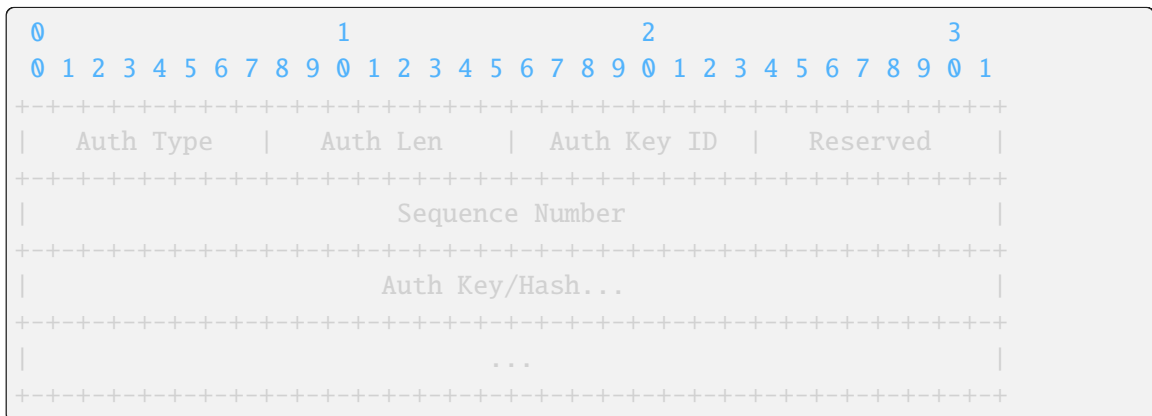
1. Format of Simple Password Authentication Section:



2. Format of Keyed MD5 and Meticulous Keyed MD5 Authentication Section:



3. Format of Keyed SHA1 and Meticulous Keyed SHA1 Authentication Section:



**class** os\_ken.lib.packet.bfd.BFDAuth(auth\_len=None)

Base class of BFD (RFC 5880) Authentication Section

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order.

Attribute	Description
auth_type	The authentication type in use.
auth_len	The length, in bytes, of the authentication section, including the auth_type and auth_len fields.

**classmethod** parser\_hdr(buf)

Parser for common part of authentication section.

**serialize\_hdr()**

Serialization function for common part of authentication section.

```
class os_ken.lib.packet.bfd.KeyedMD5(auth_key_id, seq, auth_key=None, digest=None,  
auth_len=None)
```

BFD (RFC 5880) Keyed MD5 Authentication Section class

An instance has the following attributes. Most of them are same to the on-wire counterparts but in host byte order.

Attribute	Description
auth_type	(Fixed) The authentication type in use.
auth_key_id	The authentication Key ID in use.
seq	The sequence number for this packet. This value is incremented occasionally.
auth_key	The shared MD5 key for this packet.
digest	(Optional) The 16-byte MD5 digest for the packet.
auth_len	(Fixed) The length of the authentication section is 24 bytes.

**authenticate**(*prev, auth\_keys=None*)

Authenticate the MD5 digest for this packet.

This method can be invoked only when `self.digest` is defined.

Returns a boolean indicates whether the digest can be authenticated by the correspondent Auth Key or not.

`prev` is a `bfd` instance for the BFD Control header which this authentication section belongs to. It's necessary to be assigned because an MD5 digest must be calculated over the entire BFD Control packet.

`auth_keys` is a dictionary of authentication key chain which key is an integer of *Auth Key ID* and value is a string of *Auth Key*.

**serialize**(*payload, prev*)

Encode a Keyed MD5 Authentication Section.

This method is used only when encoding an BFD Control packet.

`payload` is the rest of the packet which will immediately follow this section.

`prev` is a `bfd` instance for the BFD Control header which this authentication section belongs to. It's necessary to be assigned because an MD5 digest must be calculated over the entire BFD Control packet.

```
class os_ken.lib.packet.bfd.KeyedSHA1(auth_key_id, seq, auth_key=None, auth_hash=None,  
auth_len=None)
```

BFD (RFC 5880) Keyed SHA1 Authentication Section class

An instance has the following attributes. Most of them are same to the on-wire counterparts but in host byte order.

Attribute	Description
auth_type	(Fixed) The authentication type in use.
auth_key_id	The authentication Key ID in use.
seq	The sequence number for this packet. This value is incremented occasionally.
auth_key	The shared SHA1 key for this packet.
auth_hash	(Optional) The 20-byte SHA1 hash for the packet.
auth_len	(Fixed) The length of the authentication section is 28 bytes.

**authenticate**(*prev*, *auth\_keys=None*)

Authenticate the SHA1 hash for this packet.

This method can be invoked only when `self.auth_hash` is defined.

Returns a boolean indicates whether the hash can be authenticated by the correspondent Auth Key or not.

*prev* is a `bfd` instance for the BFD Control header which this authentication section belongs to. It's necessary to be assigned because an SHA1 hash must be calculated over the entire BFD Control packet.

*auth\_keys* is a dictionary of authentication key chain which key is an integer of *Auth Key ID* and value is a string of *Auth Key*.

**serialize**(*payload*, *prev*)

Encode a Keyed SHA1 Authentication Section.

This method is used only when encoding an BFD Control packet.

*payload* is the rest of the packet which will immediately follow this section.

*prev* is a `bfd` instance for the BFD Control header which this authentication section belongs to. It's necessary to be assigned because an SHA1 hash must be calculated over the entire BFD Control packet.

**class** `os_ken.lib.packet.bfd.MeticulousKeyedMD5`(*auth\_key\_id*, *seq*, *auth\_key=None*,  
*digest=None*, *auth\_len=None*)

BFD (RFC 5880) Meticulous Keyed MD5 Authentication Section class

All methods of this class are inherited from `KeyedMD5`.

An instance has the following attributes. Most of them are same to the on-wire counterparts but in host byte order.

Attribute	Description
auth_type	(Fixed) The authentication type in use.
auth_key_id	The authentication Key ID in use.
seq	The sequence number for this packet. This value is incremented for each successive packet transmitted for a session.
auth_key	The shared MD5 key for this packet.
digest	(Optional) The 16-byte MD5 digest for the packet.
auth_len	(Fixed) The length of the authentication section is 24 bytes.

```
class os_ken.lib.packet.bfd.MeticulousKeyedSHA1(auth_key_id, seq, auth_key=None,  
auth_hash=None, auth_len=None)
```

BFD (RFC 5880) Meticulous Keyed SHA1 Authentication Section class

All methods of this class are inherited from KeyedSHA1.

An instance has the following attributes. Most of them are same to the on-wire counterparts but in host byte order.

Attribute	Description
<code>auth_type</code>	(Fixed) The authentication type in use.
<code>auth_key_id</code>	The authentication Key ID in use.
<code>seq</code>	The sequence number for this packet. This value is incremented for each successive packet transmitted for a session.
<code>auth_key</code>	The shared SHA1 key for this packet.
<code>auth_hash</code>	(Optional) The 20-byte SHA1 hash for the packet.
<code>auth_len</code>	(Fixed) The length of the authentication section is 28 bytes.

```
class os_ken.lib.packet.bfd.SimplePassword(auth_key_id, password, auth_len=None)
```

BFD (RFC 5880) Simple Password Authentication Section class

An instance has the following attributes. Most of them are same to the on-wire counterparts but in host byte order.

Attribute	Description
<code>auth_type</code>	(Fixed) The authentication type in use.
<code>auth_key_id</code>	The authentication Key ID in use.
<code>password</code>	The simple password in use on this session. The password is a binary string, and MUST be from 1 to 16 bytes in length.
<code>auth_len</code>	The length, in bytes, of the authentication section, including the <code>auth_type</code> and <code>auth_len</code> fields.

```
authenticate(prev=None, auth_keys=None)
```

Authenticate the password for this packet.

This method can be invoked only when `self.password` is defined.

Returns a boolean indicates whether the password can be authenticated or not.

`prev` is a `bfd` instance for the BFD Control header. It's not necessary for authenticating the Simple Password.

`auth_keys` is a dictionary of authentication key chain which key is an integer of *Auth Key ID* and value is a string of *Password*.

```
serialize(payload, prev)
```

Encode a Simple Password Authentication Section.

`payload` is the rest of the packet which will immediately follow this section.

`prev` is a `bfd` instance for the BFD Control header. It's not necessary for encoding only the Simple Password section.

```
class os_ken.lib.packet.bfd.bfd(ver=1, diag=0, state=0, flags=0, detect_mult=0,
                               my_discr=0, your_discr=0, desired_min_tx_interval=0,
                               required_min_rx_interval=0,
                               required_min_echo_rx_interval=0, auth_cls=None,
                               length=None)
```

BFD (RFC 5880) Control packet encoder/decoder class.

The serialized packet would look like the ones described in the following sections.

- RFC 5880 Generic BFD Control Packet Format

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order.

`__init__` takes the corresponding args in this order.

Attribute	Description
<code>ver</code>	The version number of the protocol. This class implements protocol version 1.
<code>diag</code>	A diagnostic code specifying the local system's reason for the last change in session state.
<code>state</code>	The current BFD session state as seen by the transmitting system.
<code>flags</code>	Bitmap of the following flags: <code>BFD_FLAG_POLL</code> , <code>BFD_FLAG_FINAL</code> , <code>BFD_FLAG_CTRL_PLANE_INDEP</code> , <code>BFD_FLAG_AUTH_PRESENT</code> , <code>BFD_FLAG_DEMAND</code> , <code>BFD_FLAG_MULTIPPOINT</code>
<code>detect_mult</code>	Detection time multiplier.
<code>my_discr</code>	My Discriminator.
<code>your_discr</code>	Your Discriminator.
<code>desired_min_tx_interval</code>	Desired Min TX Interval. (in microseconds)
<code>required_min_rx_interval</code>	Required Min RX Interval. (in microseconds)
<code>required_min_echo_rx_interval</code>	Required Min Echo RX Interval. (in microseconds)
<code>auth_cls</code>	(Optional) Authentication Section instance. It's defined only when the Authentication Present (A) bit is set in flags. Assign an instance of the following classes: <code>SimplePassword</code> , <code>KeyedMD5</code> , <code>MeticulousKeyedMD5</code> , <code>KeyedSHA1</code> , and <code>MeticulousKeyedSHA1</code> .
<code>length</code>	(Optional) Length of the BFD Control packet, in bytes.

**authenticate**(\*args, \*\*kwargs)

Authenticate this packet.

Returns a boolean indicates whether the packet can be authenticated or not.

Returns `False` if the Authentication Present (A) is not set in the flag of this packet.

Returns `False` if the Authentication Section for this packet is not present.

For the description of the arguments of this method, refer to the authentication method of the Authentication Section classes.

**pack**()

Encode a BFD Control packet without authentication section.

**classmethod** `parser(buf)`

Decode a protocol header.

This method is used only when decoding a packet.

Decode a protocol header at offset 0 in bytearray *buf*. Returns the following three objects.

- An object to describe the decoded header.
- A `packet_base.PacketBase` subclass appropriate for the rest of the packet. None when the rest of the packet should be considered as raw payload.
- The rest of packet.

**serialize(payload, prev)**

Encode a protocol header.

This method is used only when encoding a packet.

Encode a protocol header. Returns a bytearray which contains the header.

*payload* is the rest of the packet which will immediately follow this header.

*prev* is a `packet_base.PacketBase` subclass for the outer protocol header. *prev* is None if the current header is the outer-most. For example, *prev* is `ipv4` or `ipv6` for `tcp.serialize`.

**BGP**

RFC 4271 BGP-4

**exception** `os_ken.lib.packet.bgp.AdminReset(data=)`

**CODE = 6**

BGP error code.

**SUB\_CODE = 4**

BGP error sub-code.

**exception** `os_ken.lib.packet.bgp.AdminShutdown(data=)`

Error to indicate Administrative shutdown.

RFC says: If a BGP speaker decides to administratively shut down its peering with a neighbor, then the speaker SHOULD send a NOTIFICATION message with the Error Code Cease and the Error Subcode 'Administrative Shutdown'.

**CODE = 6**

BGP error code.

**SUB\_CODE = 2**

BGP error sub-code.

**exception** `os_ken.lib.packet.bgp.AttrFlagError(data=)`

Error to indicate recognized path attributes have incorrect flags.

RFC says: If any recognized attribute has Attribute Flags that conflict with the Attribute Type Code, then the Error Subcode MUST be set to Attribute Flags Error. The Data field MUST contain the erroneous attribute (type, length, and value).



**CODE = 3**  
BGP error code.

**SUB\_CODE = 4**  
BGP error sub-code.

**exception** `os_ken.lib.packet.bgp.AttrLenError(data=)`

**CODE = 3**  
BGP error code.

**SUB\_CODE = 5**  
BGP error sub-code.

**exception** `os_ken.lib.packet.bgp.AuthFailure(data=)`

**CODE = 2**  
BGP error code.

**SUB\_CODE = 5**  
BGP error sub-code.

**class** `os_ken.lib.packet.bgp.BGPEvpnEsImportRTEExtendedCommunity(**kwargs)`  
ES-Import Route Target Extended Community

**class** `os_ken.lib.packet.bgp.BGPEvpnEsiLabelExtendedCommunity(label=None, mpls_label=None, vni=None, **kwargs)`  
ESI Label Extended Community

**class** `os_ken.lib.packet.bgp.BGPEvpnMacMobilityExtendedCommunity(**kwargs)`  
MAC Mobility Extended Community

**class** `os_ken.lib.packet.bgp.BGPFlowSpecRedirectCommunity(**kwargs)`  
Flow Specification Traffic Filtering Actions for Redirect.

Attribute	Description
<code>as_number</code>	Autonomous System number.
<code>local_administrator</code>	Local Administrator.

**class** `os_ken.lib.packet.bgp.BGPFlowSpecTPIDActionCommunity(**kwargs)`  
Flow Specification TPID Actions.

At-tribute	Description
<code>actions</code>	Bit representation of actions. Supported actions are TI(inner TPID action) and T0(outer TPID action).
<code>tpid_1</code>	TPID used by TI.
<code>tpid_2</code>	TPID used by T0.

**class** `os_ken.lib.packet.bgp.BGPFlowSpecTrafficActionCommunity(**kwargs)`

Flow Specification Traffic Filtering Actions for Traffic Action.

Attribute	Description
action	Apply action. The supported action are <code>SAMPLE</code> and <code>TERMINAL</code> .

**class** `os_ken.lib.packet.bgp.BGPFlowSpecTrafficMarkingCommunity(**kwargs)`

Flow Specification Traffic Filtering Actions for Traffic Marking.

Attribute	Description
dscp	Differentiated Services Code Point.

**class** `os_ken.lib.packet.bgp.BGPFlowSpecTrafficRateCommunity(**kwargs)`

Flow Specification Traffic Filtering Actions for Traffic Rate.

Attribute	Description
as_number	Autonomous System number.
rate_info	rate information.

**class** `os_ken.lib.packet.bgp.BGPFlowSpecVlanActionCommunity(**kwargs)`

Flow Specification Vlan Actions.

Attribute	Description
actions_1	Bit representation of actions. Supported actions are <code>POP</code> , <code>PUSH</code> , <code>SWAP</code> , <code>REWRITE_INNER</code> , <code>REWRITE_OUTER</code> .
actions_2	Same as <code>actions_1</code> .
vlan_1	VLAN ID used by <code>actions_1</code> .
cos_1	Class of Service used by <code>actions_1</code> .
vlan_2	VLAN ID used by <code>actions_2</code> .
cos_2	Class of Service used by <code>actions_2</code> .

**class** `os_ken.lib.packet.bgp.BGPKeepAlive(type_=4, len_=None, marker=None)`

BGP-4 KEEPALIVE Message encoder/decoder class.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
marker	Marker field. Ignored when encoding.
len	Length field. Ignored when encoding.
type	Type field.

**classmethod parser**(*buf*)

Decode a protocol header.

This method is used only when decoding a packet.

Decode a protocol header at offset 0 in bytearray *buf*. Returns the following three objects.

- An object to describe the decoded header.
- A `packet_base.PacketBase` subclass appropriate for the rest of the packet. None when the rest of the packet should be considered as raw payload.
- The rest of packet.

**class** `os_ken.lib.packet.bgp.BGPMessage`(*marker=None, len\_=None, type\_=None*)

Base class for BGP-4 messages.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
<code>marker</code>	Marker field. Ignored when encoding.
<code>len</code>	Length field. Ignored when encoding.
<code>type</code>	Type field. one of <code>BGP_MSG_*</code> constants.

**classmethod parser**(*buf*)

Decode a protocol header.

This method is used only when decoding a packet.

Decode a protocol header at offset 0 in bytearray *buf*. Returns the following three objects.

- An object to describe the decoded header.
- A `packet_base.PacketBase` subclass appropriate for the rest of the packet. None when the rest of the packet should be considered as raw payload.
- The rest of packet.

**serialize**(*payload=None, prev=None*)

Encode a protocol header.

This method is used only when encoding a packet.

Encode a protocol header. Returns a bytearray which contains the header.

*payload* is the rest of the packet which will immediately follow this header.

*prev* is a `packet_base.PacketBase` subclass for the outer protocol header. *prev* is None if the current header is the outer-most. For example, *prev* is `ipv4` or `ipv6` for `tcp.serialize`.

**class** `os_ken.lib.packet.bgp.BGPNotification`(*error\_code, error\_subcode, data=b'', type\_=3, len\_=None, marker=None*)

BGP-4 NOTIFICATION Message encoder/decoder class.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
marker	Marker field. Ignored when encoding.
len	Length field. Ignored when encoding.
type	Type field.
error_code	Error code field.
error_subcode	Error subcode field.
data	Data field.

**classmethod parser**(*buf*)

Decode a protocol header.

This method is used only when decoding a packet.

Decode a protocol header at offset 0 in bytearray *buf*. Returns the following three objects.

- An object to describe the decoded header.
- A `packet_base.PacketBase` subclass appropriate for the rest of the packet. None when the rest of the packet should be considered as raw payload.
- The rest of packet.

```
class os_ken.lib.packet.bgp.BGPOpen(my_as, bgp_identifier, type_=1, opt_param_len=0,
                                     opt_param=None, version=4, hold_time=0,
                                     len_=None, marker=None)
```

BGP-4 OPEN Message encoder/decoder class.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
marker	Marker field. Ignored when encoding.
len	Length field. Ignored when encoding.
type	Type field.
version	Version field.
my_as	My Autonomous System field. 2 octet unsigned integer.
hold_time	Hold Time field. 2 octet unsigned integer.
bgp_identifier	BGP Identifier field. An IPv4 address. For example, '192.0.2.1'
opt_param_len	Optional Parameters Length field. Ignored when encoding.
opt_param	Optional Parameters field. A list of BGPOptParam instances. The default is [].

**classmethod parser**(*buf*)

Decode a protocol header.

This method is used only when decoding a packet.

Decode a protocol header at offset 0 in bytearray *buf*. Returns the following three objects.

- An object to describe the decoded header.
- A `packet_base.PacketBase` subclass appropriate for the rest of the packet. None when the rest of the packet should be considered as raw payload.

- The rest of packet.

```
class os_ken.lib.packet.bgp.BGPPathAttributePmsiTunnel(pmsi_flags, tunnel_type,
                                                    mpls_label=None,
                                                    label=None, vni=None,
                                                    tunnel_id=None, flags=0,
                                                    type_=None, length=None)
```

P-Multicast Service Interface Tunnel (PMSI Tunnel) attribute

```
classmethod from_jsondict(dict_, decode_string=<function b64decode>,
                          **additional_args)
```

Create an instance from a JSON style dict.

Instantiate this class with parameters specified by the dict.

This method takes the following arguments.

Argument	Description
<code>dict_</code>	A dictionary which describes the parameters. For example, {"Param1": 100, "Param2": 200}
<code>decode_string</code>	(Optional) specify how to decode strings. The default is base64. This argument is used only for attributes which don't have explicit type annotations in <code>_TYPE</code> class attribute.
<code>additional_args</code>	(Optional) Additional kwargs for constructor.

```
class os_ken.lib.packet.bgp.BGPRouteRefresh(afi, safi, demarcation=0, type_=5,
                                           len_=None, marker=None)
```

BGP-4 ROUTE REFRESH Message (RFC 2918) encoder/decoder class.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
<code>marker</code>	Marker field. Ignored when encoding.
<code>len</code>	Length field. Ignored when encoding.
<code>type</code>	Type field.
<code>afi</code>	Address Family Identifier
<code>safi</code>	Subsequent Address Family Identifier

```
classmethod parser(buf)
```

Decode a protocol header.

This method is used only when decoding a packet.

Decode a protocol header at offset 0 in bytearray *buf*. Returns the following three objects.

- An object to describe the decoded header.
- A `packet_base.PacketBase` subclass appropriate for the rest of the packet. None when the rest of the packet should be considered as raw payload.
- The rest of packet.

```
class os_ken.lib.packet.bgp.BGPUpdate(type_=2, withdrawn_routes_len=None,
                                     withdrawn_routes=None,
                                     total_path_attribute_len=None,
                                     path_attributes=None, nlri=None, len_=None,
                                     marker=None)
```

BGP-4 UPDATE Message encoder/decoder class.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
marker	Marker field. Ignored when encoding.
len	Length field. Ignored when encoding.
type	Type field.
withdrawn_routes_len	Withdrawn Routes Length field. Ignored when encoding.
withdrawn_routes	Withdrawn Routes field. A list of BGPWithdrawnRoute instances. The default is [].
total_path_attribute_len	Total Path Attribute Length field. Ignored when encoding.
path_attributes	Path Attributes field. A list of BGPPathAttribute instances. The default is [].
nlri	Network Layer Reachability Information field. A list of BGPNLRI instances. The default is [].

**classmethod** `parser(buf)`

Decode a protocol header.

This method is used only when decoding a packet.

Decode a protocol header at offset 0 in bytearray *buf*. Returns the following three objects.

- An object to describe the decoded header.
- A `packet_base.PacketBase` subclass appropriate for the rest of the packet. None when the rest of the packet should be considered as raw payload.
- The rest of packet.

**exception** `os_ken.lib.packet.bgp.BadBgpId(data=)`

Error to indicate incorrect BGP Identifier.

RFC says: If the BGP Identifier field of the OPEN message is syntactically incorrect, then the Error Subcode MUST be set to Bad BGP Identifier. Syntactic correctness means that the BGP Identifier field represents a valid unicast IP host address.

**CODE = 2**

BGP error code.

**SUB\_CODE = 3**

BGP error sub-code.

**exception** `os_ken.lib.packet.bgp.BadLen(msg_type_code, message_length)`

**CODE = 1**

BGP error code.

**SUB\_CODE = 2**

BGP error sub-code.

**exception** `os_ken.lib.packet.bgp.BadMsg(msg_type)`

Error to indicate un-recognized message type.

RFC says: If the Type field of the message header is not recognized, then the Error Subcode MUST be set to Bad Message Type. The Data field MUST contain the erroneous Type field.

**CODE = 1**

BGP error code.

**SUB\_CODE = 3**

BGP error sub-code.

**exception** `os_ken.lib.packet.bgp.BadNotification(data=)`

**SEND\_ERROR = False**

Flag if set indicates Notification message should be sent to peer.

**exception** `os_ken.lib.packet.bgp.BadPeerAs(data=)`

Error to indicate open message has incorrect AS number.

RFC says: If the Autonomous System field of the OPEN message is unacceptable, then the Error Subcode MUST be set to Bad Peer AS. The determination of acceptable Autonomous System numbers is configure peer AS.

**CODE = 2**

BGP error code.

**SUB\_CODE = 2**

BGP error sub-code.

**exception** `os_ken.lib.packet.bgp.BgpExc(data=)`

Base bgp exception.

**CODE = 0**

BGP error code.

**SEND\_ERROR = True**

Flag if set indicates Notification message should be sent to peer.

**SUB\_CODE = 0**

BGP error sub-code.

**exception** `os_ken.lib.packet.bgp.CollisionResolution(data=)`

Error to indicate Connection Collision Resolution.

RFC says: If a BGP speaker decides to send a NOTIFICATION message with the Error Code Cease as a result of the collision resolution procedure (as described in [BGP-4]), then the subcode SHOULD be set to "Connection Collision Resolution".

**CODE = 6**

BGP error code.

**SUB\_CODE = 7**

BGP error sub-code.

**exception** `os_ken.lib.packet.bgp.ConnRejected(data=’')`

Error to indicate Connection Rejected.

RFC says: If a BGP speaker decides to disallow a BGP connection (e.g., the peer is not configured locally) after the speaker accepts a transport protocol connection, then the BGP speaker SHOULD send a NOTIFICATION message with the Error Code Cease and the Error Subcode "Connection Rejected".

**CODE = 6**

BGP error code.

**SUB\_CODE = 5**

BGP error sub-code.

**class** `os_ken.lib.packet.bgp.EvpnASBasedEsi(as_number, local_disc, type_=None)`

AS based ESI value

This type indicates an Autonomous System(AS)-based ESI Value that can be auto-generated or configured by the operator.

**class** `os_ken.lib.packet.bgp.EvpnArbitraryEsi(value, type_=None)`

Arbitrary 9-octet ESI value

This type indicates an arbitrary 9-octet ESI value, which is managed and configured by the operator.

**class** `os_ken.lib.packet.bgp.EvpnEsi(type_=None)`

Ethernet Segment Identifier

The supported ESI Types:

- `EvpnEsi.ARBITRARY` indicates `EvpnArbitraryEsi`.
- `EvpnEsi.LACP` indicates `EvpnLACPEsi`.
- `EvpnEsi.L2_BRIDGE` indicates `EvpnL2BridgeEsi`.
- `EvpnEsi.MAC_BASED` indicates `EvpnMacBasedEsi`.
- `EvpnEsi.ROUTER_ID` indicates `EvpnRouterIDEsi`.
- `EvpnEsi.AS_BASED` indicates `EvpnASBasedEsi`.

**class** `os_ken.lib.packet.bgp.EvpnEthernetAutoDiscoveryNLRI(route_dist, esi, ethernet_tag_id, mpl_label=None, vni=None, label=None, type_=None, length=None)`

Ethernet A-D route type specific EVPN NLRI

**class** `os_ken.lib.packet.bgp.EvpnEthernetSegmentNLRI(route_dist, esi, ip_addr, ip_addr_len=None, type_=None, length=None)`

Ethernet Segment route type specific EVPN NLRI



```
class os_ken.lib.packet.bgp.EvpnInclusiveMulticastEthernetTagNLRI(route_dist,
                                                                ethernet_tag_id,
                                                                ip_addr,
                                                                ip_addr_len=None,
                                                                type_=None,
                                                                length=None)
```

Inclusive Multicast Ethernet Tag route type specific EVPN NLRI

```
class os_ken.lib.packet.bgp.EvpnIpPrefixNLRI(route_dist, ethernet_tag_id, ip_prefix,
                                             esi=None, gw_ip_addr=None,
                                             mpls_label=None, vni=None, label=None,
                                             type_=None, length=None)
```

IP Prefix advertisement route NLRI

```
class os_ken.lib.packet.bgp.EvpnL2BridgeEsi(mac_addr, priority, type_=None)
```

ESI value for Layer 2 Bridge

This type is used in the case of indirectly connected hosts via a bridged LAN between the CEs and the PEs. The ESI Value is auto-generated and determined based on the Layer 2 bridge protocol.

```
class os_ken.lib.packet.bgp.EvpnLACPesi(mac_addr, port_key, type_=None)
```

ESI value for LACP

When IEEE 802.1AX LACP is used between the PEs and CEs, this ESI type indicates an auto-generated ESI value determined from LACP.

```
class os_ken.lib.packet.bgp.EvpnMacBasedEsi(mac_addr, local_disc, type_=None)
```

MAC-based ESI Value

This type indicates a MAC-based ESI Value that can be auto-generated or configured by the operator.

```
class os_ken.lib.packet.bgp.EvpnMacIPAdvertisementNLRI(route_dist, ethernet_tag_id,
                                                       mac_addr, ip_addr, esi=None,
                                                       mpls_labels=None, vni=None,
                                                       labels=None,
                                                       mac_addr_len=None,
                                                       ip_addr_len=None,
                                                       type_=None, length=None)
```

MAC/IP Advertisement route type specific EVPN NLRI

```
class os_ken.lib.packet.bgp.EvpnNLRI(type_=None, length=None)
```

BGP Network Layer Reachability Information (NLRI) for EVPN

```
class os_ken.lib.packet.bgp.EvpnRouterIDesi(router_id, local_disc, type_=None)
```

Router-ID ESI Value

This type indicates a router-ID ESI Value that can be auto-generated or configured by the operator.

```
class os_ken.lib.packet.bgp.EvpnUnknownEsi(value, type_=None)
```

ESI value for unknown type

```
class os_ken.lib.packet.bgp.EvpnUnknownNLRI(value, type_, length=None)
```

Unknown route type specific EVPN NLRI

**exception** `os_ken.lib.packet.bgp.FiniteStateMachineError`(*data=""*)

Error to indicate any Finite State Machine Error.

RFC says: Any error detected by the BGP Finite State Machine (e.g., receipt of an unexpected event) is indicated by sending the NOTIFICATION message with the Error Code Finite State Machine Error.

**CODE = 5**

BGP error code.

**SUB\_CODE = 1**

BGP error sub-code.

**class** `os_ken.lib.packet.bgp.FlowSpecComponentUnknown`(*buf, type\_=None*)

Unknown component type for Flow Specification NLRI component

**class** `os_ken.lib.packet.bgp.FlowSpecDSCP`(*operator, value, type\_=None*)

Diffserv Code Point for Flow Specification NLRI component

Set the 6-bit DSCP field at value. [RFC2474]

**class** `os_ken.lib.packet.bgp.FlowSpecDestPort`(*operator, value, type\_=None*)

Destination port number for Flow Specification NLRI component

Set the destination port of a TCP or UDP packet at value.

**class** `os_ken.lib.packet.bgp.FlowSpecDestPrefix`(*length, addr, type\_=None*)

Destination Prefix for Flow Specification NLRI component

**class** `os_ken.lib.packet.bgp.FlowSpecDestinationMac`(*length, addr, type\_=None*)

Destination Mac Address.

Set the Mac Address at value.

**class** `os_ken.lib.packet.bgp.FlowSpecEtherType`(*operator, value, type\_=None*)

Ethernet Type field in an Ethernet frame.

Set the 2 byte value of an Ethernet Type field at value.

**class** `os_ken.lib.packet.bgp.FlowSpecFragment`(*operator, value, type\_=None*)

Fragment for Flow Specification NLRI component

Set the bitmask for operand format at value. The following values are supported.

Attribute	Description
LF	Last fragment
FF	First fragment
ISF	Is a fragment
DF	Don't fragment

**class** `os_ken.lib.packet.bgp.FlowSpecIPProtocol`(*operator, value, type\_=None*)

IP Protocol for Flow Specification NLRI component

Set the IP protocol number at value.

**class** `os_ken.lib.packet.bgp.FlowSpecIPv4NLRI`(*length=0, rules=None*)

Flow Specification NLRI class for IPv4 [RFC 5575]

**classmethod** `from_user`(\*\**kwargs*)

Utility method for creating a NLRI instance.

This function returns a NLRI instance from human readable format value.

#### Parameters

**kwargs** -- The following arguments are available.

Argument	Value	Operator	Description
<code>dst_pre</code>	IPv4 Prefix	Nothing	Destination Prefix.
<code>src_pre</code>	IPv4 Prefix	Nothing	Source Prefix.
<code>ip_prot</code>	Integer	Numeric	IP Protocol.
<code>port</code>	Integer	Numeric	Port number.
<code>dst_por</code>	Integer	Numeric	Destination port number.
<code>src_por</code>	Integer	Numeric	Source port number.
<code>icmp_t</code>	Integer	Numeric	ICMP type.
<code>icmp_c</code>	Integer	Numeric	ICMP code.
<code>tcp_flags</code>	Fixed string	Bit-mask	TCP flags. Supported values are CWR, ECN, URGENT, ACK, PUSH, RST, SYN and FIN.
<code>packet_</code>	Integer	Numeric	Packet length.
<code>dscp</code>	Integer	Numeric	Differentiated Services Code Point.
<code>frag- ment</code>	Fixed string	Bit-mask	Fragment. Supported values are DF (Don't fragment), ISF (Is a fragment), FF (First fragment) and LF (Last fragment)

Example:

```
>>> msg = bgp.FlowSpecIPv4NLRI.from_user(
...     dst_prefix='10.0.0.0/24',
...     src_prefix='20.0.0.1/24',
...     ip_proto=6,
...     port='80 | 8000',
...     dst_port='>9000 & <9050',
...     src_port='>=8500 & <=9000',
...     icmp_type=0,
...     icmp_code=6,
...     tcp_flags='SYN+ACK & !=URGENT',
```

(continues on next page)

(continued from previous page)

```

...     packet_len=1000,
...     dscp='22 | 24',
...     fragment='LF | ==FF')
>>>

```

You can specify conditions with the following keywords.

The following keywords can be used when the operator type is Numeric.

Keyword	Description
<	Less than comparison between data and value.
<=	Less than or equal to comparison between data and value.
>	Greater than comparison between data and value.
>=	Greater than or equal to comparison between data and value.
==	Equality between data and value. This operator can be omitted.

The following keywords can be used when the operator type is Bitmask.

Keyword	Description
!=	Not equal operation.
==	Exact match operation if specified. Otherwise partial match operation.
+	Used for the summation of bitmask values. (e.g., SYN+ACK)

You can combine the multiple conditions with the following operators.

Keyword	Description
	Logical OR operation
&	Logical AND operation

### Returns

A instance of FlowSpecVPNv4NLRI.

```

class os_ken.lib.packet.bgp.FlowSpecIPv6DestPrefix(length, addr, offset=0,
                                                    type_=None)

```

IPv6 destination Prefix for Flow Specification NLRI component

```

class os_ken.lib.packet.bgp.FlowSpecIPv6FlowLabel(operator, value, type_=None)

```

```

class os_ken.lib.packet.bgp.FlowSpecIPv6Fragment(operator, value, type_=None)

```

Fragment for Flow Specification for IPv6 NLRI component

Attribute	Description
LF	Last fragment
FF	First fragment
ISF	Is a fragment

**class** `os_ken.lib.packet.bgp.FlowSpecIPv6NLRI`(*length=0, rules=None*)

Flow Specification NLRI class for IPv6 [RFC draft-ietf-idr-flow-spec-v6-08]

**classmethod** `from_user`(\*\**kwargs*)

Utility method for creating a NLRI instance.

This function returns a NLRI instance from human readable format value.

#### Parameters

**kwargs** -- The following arguments are available.

Argument	Value	Operator	Description
<code>dst_prefix</code>	IPv6 Prefix	Nothing	Destination Prefix.
<code>src_prefix</code>	IPv6 Prefix	Nothing	Source Prefix.
<code>next_header</code>	Integer	Numeric	Next Header.
<code>port</code>	Integer	Numeric	Port number.
<code>dst_port</code>	Integer	Numeric	Destination port number.
<code>src_port</code>	Integer	Numeric	Source port number.
<code>icmp_type</code>	Integer	Numeric	ICMP type.
<code>icmp_code</code>	Integer	Numeric	ICMP code.
<code>tcp_flags</code>	Fixed string	Bit-mask	TCP flags. Supported values are CWR, ECN, URGENT, ACK, PUSH, RST, SYN and FIN.
<code>packet_length</code>	Integer	Numeric	Packet length.
<code>dscp</code>	Integer	Numeric	Differentiated Services Code Point.
<code>fragment</code>	Fixed string	Bit-mask	Fragment. Supported values are ISF (Is a fragment), FF (First fragment) and LF (Last fragment)
<code>flow_label</code>	Integer	Numeric	Flow Label.

#### Note

For `dst_prefix` and `src_prefix`, you can give "offset" value like this: `2001::2/128/32`. At this case, offset is 32. offset can be omitted, then offset is treated as 0.

**class** `os_ken.lib.packet.bgp.FlowSpecIPv6SrcPrefix`(*length, addr, offset=0, type\_=None*)

IPv6 source Prefix for Flow Specification NLRI component

**class** `os_ken.lib.packet.bgp.FlowSpecIcmpCode`(*operator, value, type\_=None*)

ICMP code Flow Specification NLRI component

Set the code field of an ICMP packet at value.

**class** `os_ken.lib.packet.bgp.FlowSpecIcmpType`(*operator, value, type\_=None*)

ICMP type for Flow Specification NLRI component

Set the type field of an ICMP packet at value.

**class** `os_ken.lib.packet.bgp.FlowSpecInnerVLANCoS`(*operator, value, type\_=None*)

VLAN CoS Fields in an Inner Ethernet frame.

Set the 3 bit CoS field at value..

**class** `os_ken.lib.packet.bgp.FlowSpecInnerVLANID`(*operator, value, type\_=None*)

Inner VLAN ID.

Set VLAN ID at value.

**class** `os_ken.lib.packet.bgp.FlowSpecL2VPNLR`(*length=0, route\_dist=None, rules=None*)

Flow Specification NLRI class for L2VPN [draft-ietf-idr-flowspec-l2vpn-05]

**classmethod** `from_user`(*route\_dist, \*\*kwargs*)

Utility method for creating a L2VPN NLRI instance.

This function returns a L2VPN NLRI instance from human readable format value.

#### Parameters

**kwargs** -- The following arguments are available.

Argument	Value	Operator	Description
<code>ether_type</code>	Integer	Numeric	Ethernet Type.
<code>src_mac</code>	Mac Address	Nothing	Source Mac address.
<code>dst_mac</code>	Mac Address	Nothing	Destination Mac address.
<code>llc_ssap</code>	Integer	Numeric	Source Service Access Point in LLC.
<code>llc_dsap</code>	Integer	Numeric	Destination Service Access Point in LLC.
<code>llc_control</code>	Integer	Numeric	Control field in LLC.
<code>snap</code>	Integer	Numeric	Sub-Network Access Protocol field.
<code>vlan_id</code>	Integer	Numeric	VLAN ID.
<code>vlan_cos</code>	Integer	Numeric	VLAN COS field.
<code>inner_vlan_id</code>	Integer	Numeric	Inner VLAN ID.
<code>inner_vlan_cos</code>	Integer	Numeric	Inner VLAN COS field.

**class** `os_ken.lib.packet.bgp.FlowSpecLLCControl`(*operator, value, type\_=None*)

Control field in LLC header in an Ethernet frame.

Set the Control field at value.

**class** `os_ken.lib.packet.bgp.FlowSpecLLCDSAP`(*operator, value, type\_=None*)

Destination SAP field in LLC header in an Ethernet frame.

Set the 2 byte value of an Destination SAP at value.

**class** `os_ken.lib.packet.bgp.FlowSpecLLCSSAP`(*operator, value, type\_=None*)  
Source SAP field in LLC header in an Ethernet frame.  
Set the 2 byte value of an Source SAP at value.

**class** `os_ken.lib.packet.bgp.FlowSpecNextHeader`(*operator, value, type\_=None*)  
Next Header value in IPv6 packets  
Set the IP protocol number at value

**class** `os_ken.lib.packet.bgp.FlowSpecPacketLen`(*operator, value, type\_=None*)  
Packet length for Flow Specification NLRI component  
Set the total IP packet length at value.

**class** `os_ken.lib.packet.bgp.FlowSpecPort`(*operator, value, type\_=None*)  
Port number for Flow Specification NLRI component  
Set the source or destination TCP/UDP ports at value.

**class** `os_ken.lib.packet.bgp.FlowSpecSNAP`(*operator, value, type\_=None*)  
Sub-Network Access Protocol field in an Ethernet frame.  
Set the 5 byte SNAP field at value.

**class** `os_ken.lib.packet.bgp.FlowSpecSourceMac`(*length, addr, type\_=None*)  
Source Mac Address.  
Set the Mac Address at value.

**class** `os_ken.lib.packet.bgp.FlowSpecSrcPort`(*operator, value, type\_=None*)  
Source port number for Flow Specification NLRI component  
Set the source port of a TCP or UDP packet at value.

**class** `os_ken.lib.packet.bgp.FlowSpecSrcPrefix`(*length, addr, type\_=None*)  
Source Prefix for Flow Specification NLRI component

**class** `os_ken.lib.packet.bgp.FlowSpecTCPFlags`(*operator, value, type\_=None*)  
TCP flags for Flow Specification NLRI component  
Supported TCP flags are CWR, ECN, URGENT, ACK, PUSH, RST, SYN and FIN.

**class** `os_ken.lib.packet.bgp.FlowSpecVLANCoS`(*operator, value, type\_=None*)  
VLAN CoS Fields in an Ethernet frame.  
Set the 3 bit CoS field at value.

**class** `os_ken.lib.packet.bgp.FlowSpecVLANID`(*operator, value, type\_=None*)  
VLAN ID.  
Set VLAN ID at value.

**class** `os_ken.lib.packet.bgp.FlowSpecVPNv4NLRI`(*length=0, route\_dist=None, rules=None*)  
Flow Specification NLRI class for VPNv4 [RFC 5575]

**classmethod** `from_user`(*route\_dist, \*\*kwargs*)  
Utility method for creating a NLRI instance.  
This function returns a NLRI instance from human readable format value.

**Parameters**

- **route\_dist** -- Route Distinguisher.
- **kwargs** -- See *os\_ken.lib.packet.bgp.FlowSpecIPv4NLRI*

Example:

```
>>> msg = bgp.FlowSpecIPv4NLRI.from_user(
...     route_dist='65000:1000',
...     dst_prefix='10.0.0.0/24',
...     src_prefix='20.0.0.1/24',
...     ip_proto=6,
...     port='80 | 8000',
...     dst_port='>9000 & <9050',
...     src_port='>=8500 & <=9000',
...     icmp_type=0,
...     icmp_code=6,
...     tcp_flags='SYN+ACK & !=URGENT',
...     packet_len=1000,
...     dscp='22 | 24',
...     fragment='LF | ==FF')
>>>
```

**class** *os\_ken.lib.packet.bgp.FlowSpecVPNv6NLRI*(length=0, route\_dist=None, rules=None)

Flow Specification NLRI class for VPNv6 [draft-ietf-idr-flow-spec-v6-08]

**classmethod** *from\_user*(route\_dist, \*\*kwargs)

Utility method for creating a NLRI instance.

This function returns a NLRI instance from human readable format value.

**Parameters**

- **route\_dist** -- Route Distinguisher.
- **kwargs** -- See *os\_ken.lib.packet.bgp.FlowSpecIPv6NLRI*

**exception** *os\_ken.lib.packet.bgp.HoldTimerExpired*(data="")

Error to indicate Hold Timer expired.

RFC says: If a system does not receive successive KEEPALIVE, UPDATE, and/or NOTIFICATION messages within the period specified in the Hold Time field of the OPEN message, then the NOTIFICATION message with the Hold Timer Expired Error Code is sent and the BGP connection is closed.

**CODE** = 4

BGP error code.

**SUB\_CODE** = 1

BGP error sub-code.

**exception** *os\_ken.lib.packet.bgp.InvalidNetworkField*(data="")

**CODE** = 3

BGP error code.



**SUB\_CODE = 10**

BGP error sub-code.

**exception** `os_ken.lib.packet.bgp.InvalidNextHop(data=)`

**CODE = 3**

BGP error code.

**SUB\_CODE = 8**

BGP error sub-code.

**exception** `os_ken.lib.packet.bgp.InvalidOriginError(data=)`

Error indicates undefined Origin attribute value.

RFC says: If the ORIGIN attribute has an undefined value, then the Error Sub-code MUST be set to Invalid Origin Attribute. The Data field MUST contain the unrecognized attribute (type, length, and value).

**CODE = 3**

BGP error code.

**SUB\_CODE = 6**

BGP error sub-code.

**exception** `os_ken.lib.packet.bgp.MalformedAsPath(data=)`

Error to indicate if AP\_PATH attribute is syntactically incorrect.

RFC says: The AS\_PATH attribute is checked for syntactic correctness. If the path is syntactically incorrect, then the Error Subcode MUST be set to Malformed AS\_PATH.

**CODE = 3**

BGP error code.

**SUB\_CODE = 11**

BGP error sub-code.

**exception** `os_ken.lib.packet.bgp.MalformedAttrList(data=)`

Error to indicate UPDATE message is malformed.

RFC says: Error checking of an UPDATE message begins by examining the path attributes. If the Withdrawn Routes Length or Total Attribute Length is too large (i.e., if Withdrawn Routes Length + Total Attribute Length + 23 exceeds the message Length), then the Error Subcode MUST be set to Malformed Attribute List.

**CODE = 3**

BGP error code.

**SUB\_CODE = 1**

BGP error sub-code.

**exception** `os_ken.lib.packet.bgp.MalformedOptionalParam(data=)`

If recognized optional parameters are malformed.

RFC says: If one of the Optional Parameters in the OPEN message is recognized, but is malformed, then the Error Subcode MUST be set to 0 (Unspecific).

**CODE = 2**

BGP error code.

**SUB\_CODE = 0**

BGP error sub-code.

**exception** `os_ken.lib.packet.bgp.MaxPrefixReached(data=)`

**CODE = 6**

BGP error code.

**SUB\_CODE = 1**

BGP error sub-code.

**exception** `os_ken.lib.packet.bgp.MissingWellKnown(attr_type_code)`

Error to indicate missing well-known attribute.

RFC says: If any of the well-known mandatory attributes are not present, then the Error Subcode MUST be set to Missing Well-known Attribute. The Data field MUST contain the Attribute Type Code of the missing, well-known attribute.

**CODE = 3**

BGP error code.

**SUB\_CODE = 3**

BGP error sub-code.

**exception** `os_ken.lib.packet.bgp.NotSync(data=)`

**CODE = 1**

BGP error code.

**SUB\_CODE = 1**

BGP error sub-code.

**exception** `os_ken.lib.packet.bgp.OptAttrError(data=)`

Error indicates Optional Attribute is malformed.

RFC says: If an optional attribute is recognized, then the value of this attribute MUST be checked. If an error is detected, the attribute MUST be discarded, and the Error Subcode MUST be set to Optional Attribute Error. The Data field MUST contain the attribute (type, length, and value).

**CODE = 3**

BGP error code.

**SUB\_CODE = 9**

BGP error sub-code.

**exception** `os_ken.lib.packet.bgp.OtherConfChange(data=)`

**CODE = 6**

BGP error code.

**SUB\_CODE = 6**

BGP error sub-code.

**exception** `os_ken.lib.packet.bgp.OutOfResource(data=)`

**CODE** = 6

BGP error code.

**SUB\_CODE** = 8

BGP error sub-code.

**exception** `os_ken.lib.packet.bgp.PeerDeConfig(data=)`

**CODE** = 6

BGP error code.

**SUB\_CODE** = 3

BGP error sub-code.

**class** `os_ken.lib.packet.bgp.PmsiTunnelIdUnknown(value)`

Unknown route type specific `_PmsiTunnelId`

**class** `os_ken.lib.packet.bgp.RouteTargetMembershipNLRI(origin_as, route_target)`

Route Target Membership NLRI.

Route Target membership NLRI is advertised in BGP UPDATE messages using the `MP_REACH_NLRI` and `MP_UNREACH_NLRI` attributes.

**exception** `os_ken.lib.packet.bgp.RoutingLoop(data=)`

**CODE** = 3

BGP error code.

**SUB\_CODE** = 7

BGP error sub-code.

**class** `os_ken.lib.packet.bgp.StreamParser`

Streaming parser for BGP-4 messages.

This is a subclass of `os_ken.lib.packet.stream_parser.StreamParser`. Its parse method returns a list of `BGPMessage` subclass instances.

**try\_parse(data)**

Try to extract a message from the given bytes.

This is an override point for subclasses.

This method tries to extract a message from bytes given by the argument.

Raises `TooSmallException` if the given data is not enough to extract a complete message but there's still a chance to extract a message if more data is come later.

**exception** `os_ken.lib.packet.bgp.UnRegWellKnowAttr(data=)`

**CODE** = 3

BGP error code.

**SUB\_CODE** = 2

BGP error sub-code.

**exception** `os_ken.lib.packet.bgp.UnacceptableHoldTime(data=)`

Error to indicate Unacceptable Hold Time in open message.

RFC says: If the Hold Time field of the OPEN message is unacceptable, then the Error Subcode MUST be set to Unacceptable Hold Time.

**CODE = 2**

BGP error code.

**SUB\_CODE = 6**

BGP error sub-code.

**exception** `os_ken.lib.packet.bgp.UnsupportedOptParam(data=)`

Error to indicate unsupported optional parameters.

RFC says: If one of the Optional Parameters in the OPEN message is not recognized, then the Error Subcode MUST be set to Unsupported Optional Parameters.

**CODE = 2**

BGP error code.

**SUB\_CODE = 4**

BGP error sub-code.

**exception** `os_ken.lib.packet.bgp.UnsupportedVersion(locally_support_version)`

Error to indicate unsupported bgp version number.

RFC says: If the version number in the Version field of the received OPEN message is not supported, then the Error Subcode MUST be set to Unsupported Version Number. The Data field is a 2-octet unsigned integer, which indicates the largest, locally-supported version number less than the version the remote BGP peer bid (as indicated in the received OPEN message), or if the smallest, locally-supported version number is greater than the version the remote BGP peer bid, then the smallest, locally-supported version number.

**CODE = 2**

BGP error code.

**SUB\_CODE = 1**

BGP error sub-code.

## BMP

BGP Monitoring Protocol draft-ietf-grow-bmp-07

**class** `os_ken.lib.packet.bmp.BMPInitiation(info, type_=4, len_=None, version=3)`

BMP Initiation Message

Attribute	Description
version	Version. this packet lib defines BMP ver. 3
len	Length field. Ignored when encoding.
type	Type field. one of BMP_MSG_ constants.
info	One or more piece of information encoded as a TLV

**classmethod parser**(*buf*)

Decode a protocol header.

This method is used only when decoding a packet.

Decode a protocol header at offset 0 in bytearray *buf*. Returns the following three objects.

- An object to describe the decoded header.
- A `packet_base.PacketBase` subclass appropriate for the rest of the packet. None when the rest of the packet should be considered as raw payload.
- The rest of packet.

**class** `os_ken.lib.packet.bmp.BMPMessage`(*type\_*, *len\_=None*, *version=3*)

Base class for BGP Monitoring Protocol messages.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
<code>version</code>	Version. this packet lib defines BMP ver. 3
<code>len</code>	Length field. Ignored when encoding.
<code>type</code>	Type field. one of <code>BMP_MSG_constants</code> .

**classmethod parser**(*buf*)

Decode a protocol header.

This method is used only when decoding a packet.

Decode a protocol header at offset 0 in bytearray *buf*. Returns the following three objects.

- An object to describe the decoded header.
- A `packet_base.PacketBase` subclass appropriate for the rest of the packet. None when the rest of the packet should be considered as raw payload.
- The rest of packet.

**serialize**()

Encode a protocol header.

This method is used only when encoding a packet.

Encode a protocol header. Returns a bytearray which contains the header.

*payload* is the rest of the packet which will immediately follow this header.

*prev* is a `packet_base.PacketBase` subclass for the outer protocol header. *prev* is None if the current header is the outer-most. For example, *prev* is `ipv4` or `ipv6` for `tcp.serialize`.

**class** `os_ken.lib.packet.bmp.BMPPeerDownNotification`(*reason*, *data*, *peer\_type*,  
*is\_post\_policy*, *peer\_distinguisher*,  
*peer\_address*, *peer\_as*,  
*peer\_bgp\_id*, *timestamp*,  
*version=3*, *type\_=2*, *len\_=None*,  
*is\_adj\_rib\_out=False*)

## BMP Peer Down Notification Message

Attribute	Description
version	Version. this packet lib defines BMP ver. 3
len	Length field. Ignored when encoding.
type	Type field. one of BMP_MSG_constants.
reason	Reason indicates why the session was closed.
data	vary by the reason.

**classmethod parser**(*buf*)

Decode a protocol header.

This method is used only when decoding a packet.

Decode a protocol header at offset 0 in bytearray *buf*. Returns the following three objects.

- An object to describe the decoded header.
- A packet\_base.PacketBase subclass appropriate for the rest of the packet. None when the rest of the packet should be considered as raw payload.
- The rest of packet.

```
class os_ken.lib.packet.bmp.BMPPeerMessage(peer_type, is_post_policy, peer_distinguisher,
peer_address, peer_as, peer_bgp_id,
timestamp, version=3, type_=None,
len_=None, is_adj_rib_out=False)
```

## BMP Message with Per Peer Header

Following BMP Messages contain Per Peer Header after Common BMP Header.

- BMP\_MSG\_TYPE\_ROUTE\_MONITRING
- BMP\_MSG\_TYPE\_STATISTICS\_REPORT
- BMP\_MSG\_PEER\_UP\_NOTIFICATION

Attribute	Description
version	Version. this packet lib defines BMP ver. 3
len	Length field. Ignored when encoding.
type	Type field. one of BMP_MSG_constants.
peer_type	The type of the peer.
is_post_policy	Indicate the message reflects the post-policy
is_adj_rib_out	Indicate the message reflects Adj-RIB-Out (defaults to Adj-RIB-In)
peer_distinguisher	Use for L3VPN router which can have multiple instance.
peer_address	The remote IP address associated with the TCP session.
peer_as	The Autonomous System number of the peer.
peer_bgp_id	The BGP Identifier of the peer
timestamp	The time when the encapsulated routes were received.

**classmethod parser**(*buf*)

Decode a protocol header.

This method is used only when decoding a packet.

Decode a protocol header at offset 0 in bytearray *buf*. Returns the following three objects.

- An object to describe the decoded header.
- A `packet_base.PacketBase` subclass appropriate for the rest of the packet. None when the rest of the packet should be considered as raw payload.
- The rest of packet.

```
class os_ken.lib.packet.bmp.BMPPeerUpNotification(local_address, local_port,
                                                remote_port, sent_open_message,
                                                received_open_message, peer_type,
                                                is_post_policy, peer_distinguisher,
                                                peer_address, peer_as, peer_bgp_id,
                                                timestamp, version=3, type_=3,
                                                len_=None, is_adj_rib_out=False)
```

#### BMP Peer Up Notification Message

Attribute	Description
version	Version. this packet lib defines BMP ver. 3
len	Length field. Ignored when encoding.
type	Type field. one of BMP_MSG_ constants.
peer_type	The type of the peer.
peer_flags	Provide more information about the peer.
peer_distinguisher	Use for L3VPN router which can have multiple instance.
peer_address	The remote IP address associated with the TCP session.
peer_as	The Autonomous System number of the peer.
peer_bgp_id	The BGP Identifier of the peer
timestamp	The time when the encapsulated routes were received.
local_address	The local IP address associated with the peering TCP session.
local_port	The local port number associated with the peering TCP session.
remote_port	The remote port number associated with the peering TCP session.
sent_open_message	The full OPEN message transmitted by the monitored router to its peer.
re- ceived_open_message	The full OPEN message received by the monitored router from its peer.

#### **classmethod** parser(*buf*)

Decode a protocol header.

This method is used only when decoding a packet.

Decode a protocol header at offset 0 in bytearray *buf*. Returns the following three objects.

- An object to describe the decoded header.
- A `packet_base.PacketBase` subclass appropriate for the rest of the packet. None when the rest of the packet should be considered as raw payload.
- The rest of packet.

```
class os_ken.lib.packet.bmp.BMPRouteMonitoring(bgp_update, peer_type, is_post_policy,
peer_distinguisher, peer_address,
peer_as, peer_bgp_id, timestamp,
version=3, type_=0, len_=None,
is_adj_rib_out=False)
```

BMP Route Monitoring Message

Attribute	Description
version	Version. this packet lib defines BMP ver. 3
len	Length field. Ignored when encoding.
type	Type field. one of BMP_MSG_constants.
peer_type	The type of the peer.
peer_flags	Provide more information about the peer.
peer_distinguisher	Use for L3VPN router which can have multiple instance.
peer_address	The remote IP address associated with the TCP session.
peer_as	The Autonomous System number of the peer.
peer_bgp_id	The BGP Identifier of the peer
timestamp	The time when the encapsulated routes were received.
bgp_update	BGP Update PDU

**classmethod** **parser**(*buf*)

Decode a protocol header.

This method is used only when decoding a packet.

Decode a protocol header at offset 0 in bytearray *buf*. Returns the following three objects.

- An object to describe the decoded header.
- A packet\_base.PacketBase subclass appropriate for the rest of the packet. None when the rest of the packet should be considered as raw payload.
- The rest of packet.

```
class os_ken.lib.packet.bmp.BMPStatisticsReport(stats, peer_type, is_post_policy,
peer_distinguisher, peer_address,
peer_as, peer_bgp_id, timestamp,
version=3, type_=1, len_=None,
is_adj_rib_out=False)
```

BMP Statistics Report Message



Attribute	Description
version	Version. this packet lib defines BMP ver. 3
len	Length field. Ignored when encoding.
type	Type field. one of BMP_MSG_ constants.
peer_type	The type of the peer.
peer_flags	Provide more information about the peer.
peer_distinguisher	Use for L3VPN router which can have multiple instance.
peer_address	The remote IP address associated with the TCP session.
peer_as	The Autonomous System number of the peer.
peer_bgp_id	The BGP Identifier of the peer
timestamp	The time when the encapsulated routes were received.
stats	Statistics (one or more stats encoded as a TLV)

### classmethod `parser(buf)`

Decode a protocol header.

This method is used only when decoding a packet.

Decode a protocol header at offset 0 in bytearray *buf*. Returns the following three objects.

- An object to describe the decoded header.
- A `packet_base.PacketBase` subclass appropriate for the rest of the packet. None when the rest of the packet should be considered as raw payload.
- The rest of packet.

**class** `os_ken.lib.packet.bmp.BMPTermination`(*info*, *type\_=5*, *len\_=None*, *version=3*)

BMP Termination Message

Attribute	Description
version	Version. this packet lib defines BMP ver. 3
len	Length field. Ignored when encoding.
type	Type field. one of BMP_MSG_ constants.
info	One or more piece of information encoded as a TLV

### classmethod `parser(buf)`

Decode a protocol header.

This method is used only when decoding a packet.

Decode a protocol header at offset 0 in bytearray *buf*. Returns the following three objects.

- An object to describe the decoded header.
- A `packet_base.PacketBase` subclass appropriate for the rest of the packet. None when the rest of the packet should be considered as raw payload.
- The rest of packet.

## BPDU

Bridge Protocol Data Unit(BPDU, IEEE 802.1D) parser/serializer <http://standards.ieee.org/getieee802/download/802.1D-2004.pdf>

Configuration BPDUs format

Structure	Octet
Protocol Identifier = 0000 0000 0000 0000	1 - 2
Protocol Version Identifier = 0000 0000	3
BPDU Type = 0000 0000	4
Flags	5
<b>Root Identifier</b> <b>include - priority</b> system ID extension MAC address	6 - 13
Root Path Cost	14 - 17
<b>Bridge Identifier</b> <b>include - priority</b> system ID extension MAC address	18 - 25
<b>Port Identifier</b> <b>include - priority</b> port number	26 - 27
Message Age	28 - 29
Max Age	30 - 31
Hello Time	32 - 33
Forward Delay	34 - 35

Topology Change Notification BPDUs format

Structure	Octet
Protocol Identifier = 0000 0000 0000 0000	1 - 2
Protocol Version Identifier = 0000 0000	3
BPDU Type = 1000 0000	4

Rapid Spanning Tree BPDUs(RST BPDUs) format

Structure	Octet
Protocol Identifier = 0000 0000 0000 0000	1 - 2
Protocol Version Identifier = 0000 0010	3
BPDU Type = 0000 0010	4
Flags	5
<b>Root Identifier</b> <b>include - priority</b> system ID extension MAC address	6 - 13
Root Path Cost	14 - 17
<b>Bridge Identifier</b> <b>include - priority</b> system ID extension MAC address	18 - 25
<b>Port Identifier</b> <b>include - priority</b> port number	26 - 27
Message Age	28 - 29
Max Age	30 - 31
Hello Time	32 - 33
Forward Delay	34 - 35
Version 1 Length = 0000 0000	36

```
class os_ken.lib.packet.bpdu.ConfigurationBPDUs(flags=0, root_priority=32768,
root_system_id_extension=0,
root_mac_address='00:00:00:00:00:00',
root_path_cost=0,
bridge_priority=32768,
bridge_system_id_extension=0,
bridge_mac_address='00:00:00:00:00:00',
port_priority=128, port_number=0,
message_age=0, max_age=20,
hello_time=2, forward_delay=15)
```

Configuration BPDUs(IEEE 802.1D) header encoder/decoder class.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
flags	Bit 1: Topology Change flag Bits 2 through 7: unused and take the value 0 Bit 8: Topology Change Acknowledgment flag
root_priority	Root Identifier priority set 0-61440 in steps of 4096
root_system_id_extension	Root Identifier system ID extension
root_mac_address	Root Identifier MAC address
root_path_cost	Root Path Cost
bridge_priority	Bridge Identifier priority set 0-61440 in steps of 4096
bridge_system_id_extension	Bridge Identifier system ID extension
bridge_mac_address	Bridge Identifier MAC address
port_priority	Port Identifier priority set 0-240 in steps of 16
port_number	Port Identifier number
message_age	Message Age timer value
max_age	Max Age timer value
hello_time	Hello Time timer value
forward_delay	Forward Delay timer value

**classmethod parser(*buf*)**

Decode a protocol header.

This method is used only when decoding a packet.

Decode a protocol header at offset 0 in bytearray *buf*. Returns the following three objects.

- An object to describe the decoded header.
- A `packet_base.PacketBase` subclass appropriate for the rest of the packet. None when the rest of the packet should be considered as raw payload.
- The rest of packet.

**serialize(*payload*, *prev*)**

Encode a protocol header.

This method is used only when encoding a packet.

Encode a protocol header. Returns a bytearray which contains the header.

*payload* is the rest of the packet which will immediately follow this header.

*prev* is a `packet_base.PacketBase` subclass for the outer protocol header. *prev* is None if the current header is the outer-most. For example, *prev* is `ipv4` or `ipv6` for `tcp.serialize`.

```
class os_ken.lib.packet.bpdu.RstBPDUs(flags=0, root_priority=32768,
                                       root_system_id_extension=0,
                                       root_mac_address='00:00:00:00:00:00',
                                       root_path_cost=0, bridge_priority=32768,
                                       bridge_system_id_extension=0,
                                       bridge_mac_address='00:00:00:00:00:00',
                                       port_priority=128, port_number=0,
                                       message_age=0, max_age=20, hello_time=2,
                                       forward_delay=15)
```

Rapid Spanning Tree BPDUs(RST BPDUs, IEEE 802.1D) header encoder/decoder class.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
flags	Bit 1: Topology Change flag Bit 2: Proposal flag Bits 3 and 4: Port Role Bit 5: Learning flag Bit 6: Forwarding flag Bit 7: Agreement flag Bit 8: Topology Change Acknowledgment flag
root_priority	Root Identifier priority set 0-61440 in steps of 4096
root_system_id_extension	Root Identifier system ID extension
root_mac_address	Root Identifier MAC address
root_path_cost	Root Path Cost
bridge_priority	Bridge Identifier priority set 0-61440 in steps of 4096
bridge_system_id_extension	Bridge Identifier system ID extension
bridge_mac_address	Bridge Identifier MAC address
port_priority	Port Identifier priority set 0-240 in steps of 16
port_number	Port Identifier number
message_age	Message Age timer value
max_age	Max Age timer value
hello_time	Hello Time timer value
forward_delay	Forward Delay timer value

**classmethod** `parser(buf)`

Decode a protocol header.

This method is used only when decoding a packet.

Decode a protocol header at offset 0 in bytearray *buf*. Returns the following three objects.

- An object to describe the decoded header.

- A `packet_base.PacketBase` subclass appropriate for the rest of the packet. None when the rest of the packet should be considered as raw payload.
- The rest of packet.

**serialize**(*payload*, *prev*)

Encode a protocol header.

This method is used only when encoding a packet.

Encode a protocol header. Returns a bytearray which contains the header.

*payload* is the rest of the packet which will immediately follow this header.

*prev* is a `packet_base.PacketBase` subclass for the outer protocol header. *prev* is None if the current header is the outer-most. For example, *prev* is `ipv4` or `ipv6` for `tcp.serialize`.

**class** `os_ken.lib.packet.bpdu.TopologyChangeNotificationBPDUs`

Topology Change Notification BPDUs(IEEE 802.1D) header encoder/decoder class.

**classmethod** `parser`(*buf*)

Decode a protocol header.

This method is used only when decoding a packet.

Decode a protocol header at offset 0 in bytearray *buf*. Returns the following three objects.

- An object to describe the decoded header.
- A `packet_base.PacketBase` subclass appropriate for the rest of the packet. None when the rest of the packet should be considered as raw payload.
- The rest of packet.

**class** `os_ken.lib.packet.bpdu.bpdu`

Bridge Protocol Data Unit(BPDU) header encoder/decoder base class.

**classmethod** `parser`(*buf*)

Decode a protocol header.

This method is used only when decoding a packet.

Decode a protocol header at offset 0 in bytearray *buf*. Returns the following three objects.

- An object to describe the decoded header.
- A `packet_base.PacketBase` subclass appropriate for the rest of the packet. None when the rest of the packet should be considered as raw payload.
- The rest of packet.

**serialize**(*payload*, *prev*)

Encode a protocol header.

This method is used only when encoding a packet.

Encode a protocol header. Returns a bytearray which contains the header.

*payload* is the rest of the packet which will immediately follow this header.

*prev* is a `packet_base.PacketBase` subclass for the outer protocol header. *prev* is None if the current header is the outer-most. For example, *prev* is `ipv4` or `ipv6` for `tcp.serialize`.

**CFM**

```
class os_ken.lib.packet.cfm.cc_message(md_lv=0, version=0, rdi=0, interval=4,
                                       seq_num=0, mep_id=1, md_name_format=4,
                                       md_name_length=0, md_name=b'0',
                                       short_ma_name_format=2,
                                       short_ma_name_length=0, short_ma_name=b'1',
                                       tlvs=None)
```

CFM (IEEE Std 802.1ag-2007) Continuity Check Message (CCM) encoder/decoder class.

This is used with `os_ken.lib.packet.cfm.cfm`.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
<code>md_lv</code>	Maintenance Domain Level.
<code>version</code>	The protocol version number.
<code>rdi</code>	RDI bit.
<code>interval</code>	CCM Interval. The default is 4 (1 frame/s)
<code>seq_num</code>	Sequence Number.
<code>mep_id</code>	Maintenance association End Point Identifier.
<code>md_name_format</code>	Maintenance Domain Name Format. The default is 4 (Character string)
<code>md_name_length</code>	Maintenance Domain Name Length. (0 means automatically-calculate when encoding.)
<code>md_name</code>	Maintenance Domain Name.
<code>short_ma_name_format</code>	Short MA Name Format. The default is 2 (Character string)
<code>short_ma_name_length</code>	Short MA Name Format Length. (0 means automatically-calculate when encoding.)
<code>short_ma_name</code>	Short MA Name.
<code>tlvs</code>	TLVs.

```
class os_ken.lib.packet.cfm.cfm(op=None)
```

CFM (Connectivity Fault Management) Protocol header class.

<http://standards.ieee.org/getieee802/download/802.1ag-2007.pdf>

OpCode Field range assignments

OpCode range	CFM PDU or organization
0	Reserved for IEEE 802.1
1	Continuity Check Message (CCM)
2	Loopback Reply (LBR)
3	Loopback Message (LBM)
4	Linktrace Reply (LTR)
5	Linktrace Message (LTM)
06 - 31	Reserved for IEEE 802.1
32 - 63	Defined by ITU-T Y.1731
64 - 255	Reserved for IEEE 802.1.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
op	CFM PDU

**classmethod parser**(*buf*)

Decode a protocol header.

This method is used only when decoding a packet.

Decode a protocol header at offset 0 in bytearray *buf*. Returns the following three objects.

- An object to describe the decoded header.
- A `packet_base.PacketBase` subclass appropriate for the rest of the packet. None when the rest of the packet should be considered as raw payload.
- The rest of packet.

**serialize**(*payload, prev*)

Encode a protocol header.

This method is used only when encoding a packet.

Encode a protocol header. Returns a bytearray which contains the header.

*payload* is the rest of the packet which will immediately follow this header.

*prev* is a `packet_base.PacketBase` subclass for the outer protocol header. *prev* is None if the current header is the outer-most. For example, *prev* is `ipv4` or `ipv6` for `tcp.serialize`.

**class os\_ken.lib.packet.cfm.data\_tlv**(*length=0, data\_value=b''*)

CFM (IEEE Std 802.1ag-2007) Data TLV encoder/decoder class.

This is used with `os_ken.lib.packet.cfm.cfm`.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
length	Length of Value field. (0 means automatically-calculate when encoding)
data_value	Bit pattern of any of n octets.(n = length)

**class os\_ken.lib.packet.cfm.interface\_status\_tlv**(*length=0, interface\_status=1*)

CFM (IEEE Std 802.1ag-2007) Interface Status TLV encoder/decoder class.

This is used with `os_ken.lib.packet.cfm.cfm`.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
length	Length of Value field. (0 means automatically-calculate when encoding.)
interface_status	Interface Status.The default is 1 (isUp)



```
class os_ken.lib.packet.cfm.link_trace_message(md_lv=0, version=0, use_fdb_only=1,
                                              transaction_id=0, ttl=64,
                                              ltm_orig_addr='00:00:00:00:00:00',
                                              ltm_targ_addr='00:00:00:00:00:00',
                                              tlvs=None)
```

CFM (IEEE Std 802.1ag-2007) Linktrace Message (LTM) encoder/decoder class.

This is used with `os_ken.lib.packet.cfm.cfm`.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
<code>md_lv</code>	Maintenance Domain Level.
<code>version</code>	The protocol version number.
<code>use_fdb_only</code>	UseFDBOnly bit.
<code>transaction_id</code>	LTM Transaction Identifier.
<code>ttl</code>	LTM TTL.
<code>ltm_orig_addr</code>	Original MAC Address.
<code>ltm_targ_addr</code>	Target MAC Address.
<code>tlvs</code>	TLVs.

```
class os_ken.lib.packet.cfm.link_trace_reply(md_lv=0, version=0, use_fdb_only=1,
                                             fwd_yes=0, terminal_mep=1,
                                             transaction_id=0, ttl=64, relay_action=1,
                                             tlvs=None)
```

CFM (IEEE Std 802.1ag-2007) Linktrace Reply (LTR) encoder/decoder class.

This is used with `os_ken.lib.packet.cfm.cfm`.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
<code>version</code>	The protocol version number.
<code>use_fdb_only</code>	UseFDBOnly bit.
<code>fwd_yes</code>	FwdYes bit.
<code>terminal_mep</code>	TerminalMep bit.
<code>transaction_id</code>	LTR Transaction Identifier.
<code>ttl</code>	Reply TTL.
<code>relay_action</code>	Relay Action. The default is 1 (RlyHit)
<code>tlvs</code>	TLVs.

```
class os_ken.lib.packet.cfm.loopback_message(md_lv=0, version=0, transaction_id=0,
                                              tlvs=None)
```

CFM (IEEE Std 802.1ag-2007) Loopback Message (LBM) encoder/decoder class.

This is used with `os_ken.lib.packet.cfm.cfm`.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
md_lv	Maintenance Domain Level.
version	The protocol version number.
transaction_id	Loopback Transaction Identifier.
tlvs	TLVs.

```
class os_ken.lib.packet.cfm.loopback_reply(md_lv=0, version=0, transaction_id=0,
                                           tlvs=None)
```

CFM (IEEE Std 802.1ag-2007) Loopback Reply (LBR) encoder/decoder class.

This is used with `os_ken.lib.packet.cfm.cfm`.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
md_lv	Maintenance Domain Level.
version	The protocol version number.
transaction_id	Loopback Transaction Identifier.
tlvs	TLVs.

```
class os_ken.lib.packet.cfm.ltm_egress_identifier_tlv(length=0, egress_id_ui=0,
                                                    egress_id_mac='00:00:00:00:00:00')
```

CFM (IEEE Std 802.1ag-2007) LTM EGRESS TLV encoder/decoder class.

This is used with `os_ken.lib.packet.cfm.cfm`.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
length	Length of Value field. (0 means automatically-calculate when encoding.)
egress_id_ui	Egress Identifier of Unique ID.
egress_id_mac	Egress Identifier of MAC address.

```
class os_ken.lib.packet.cfm.ltr_egress_identifier_tlv(length=0, last_egress_id_ui=0,
                                                    last_egress_id_mac='00:00:00:00:00:00',
                                                    next_egress_id_ui=0,
                                                    next_egress_id_mac='00:00:00:00:00:00')
```

CFM (IEEE Std 802.1ag-2007) LTR EGRESS TLV encoder/decoder class.

This is used with `os_ken.lib.packet.cfm.cfm`.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
length	Length of Value field. (0 means automatically-calculate when encoding.)
last_egress_id_ui	Last Egress Identifier of Unique ID.
last_egress_id_mac	Last Egress Identifier of MAC address.
next_egress_id_ui	Next Egress Identifier of Unique ID.
next_egress_id_mac	Next Egress Identifier of MAC address.

```
class os_ken.lib.packet.cfm.organization_specific_tlv(length=0, oui=b'\x00\x00\x00',
                                                    subtype=0, value=b'')
```

### CFM (IEEE Std 802.1ag-2007) Organization Specific TLV

encoder/decoder class.

This is used with `os_ken.lib.packet.cfm.cfm`.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
length	Length of Value field. (0 means automatically-calculate when encoding.)
oui	Organizationally Unique Identifier.
subtype	Subtype.
value	Value.(optional)

```
class os_ken.lib.packet.cfm.port_status_tlv(length=0, port_status=2)
```

CFM (IEEE Std 802.1ag-2007) Port Status TLV encoder/decoder class.

This is used with `os_ken.lib.packet.cfm.cfm`.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
length	Length of Value field. (0 means automatically-calculate when encoding.)
port_status	Port Status.The default is 1 (psUp)

```
class os_ken.lib.packet.cfm.reply_egress_tlv(length=0, action=1,
                                              mac_address='00:00:00:00:00:00',
                                              port_id_length=0, port_id_subtype=0,
                                              port_id=b'')
```

CFM (IEEE Std 802.1ag-2007) Reply Egress TLV encoder/decoder class.

This is used with `os_ken.lib.packet.cfm.cfm`.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
length	Length of Value field. (0 means automatically-calculate when encoding.)
action	Egress Action.The default is 1 (EgrOK)
mac_address	Egress MAC Address.
port_id_length	Egress PortID Length. (0 means automatically-calculate when encoding.)
port_id_subtype	Egress PortID Subtype.
port_id	Egress PortID.

```
class os_ken.lib.packet.cfm.reply_ingress_tlv(length=0, action=1,
                                             mac_address='00:00:00:00:00:00',
                                             port_id_length=0, port_id_subtype=0,
                                             port_id=b'')
```

CFM (IEEE Std 802.1ag-2007) Reply Ingress TLV encoder/decoder class.

This is used with `os_ken.lib.packet.cfm.cfm`.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
length	Length of Value field. (0 means automatically-calculate when encoding.)
action	Ingress Action.The default is 1 (IngOK)
mac_address	Ingress MAC Address.
port_id_length	Ingress PortID Length. (0 means automatically-calculate when encoding.)
port_id_subtype	Ingress PortID Subtype.
port_id	Ingress PortID.

```
class os_ken.lib.packet.cfm.sender_id_tlv(length=0, chassis_id_length=0,
                                           chassis_id_subtype=4, chassis_id=b'',
                                           ma_domain_length=0, ma_domain=b'',
                                           ma_length=0, ma=b'')
```

CFM (IEEE Std 802.1ag-2007) Sender ID TLV encoder/decoder class.

This is used with `os_ken.lib.packet.cfm.cfm`.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
length	Length of Value field. (0 means automatically-calculate when encoding.)
chassis_id_length	Chassis ID Length. (0 means automatically-calculate when encoding.)
chassis_id_subtype	Chassis ID Subtype. The default is 4 (Mac Address)
chassis_id	Chassis ID.
ma_domain_length	Management Address Domain Length. (0 means automatically-calculate when encoding.)
ma_domain	Management Address Domain.
ma_length	Management Address Length. (0 means automatically-calculate when encoding.)
ma	Management Address.

## DHCP

DHCP packet parser/serializer

```
class os_ken.lib.packet.dhcp.dhcp(op, chaddr, options=None, htype=1, hlen=0, hops=0,
                                xid=None, secs=0, flags=0, ciaddr='0.0.0.0',
                                yiaddr='0.0.0.0', siaddr='0.0.0.0', giaddr='0.0.0.0',
                                sname="", boot_file="")
```

DHCP (RFC 2131) header encoder/decoder class.

The serialized packet would look like the ones described in the following sections.

- RFC 2131 DHCP packet format

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
op	Message op code / message type. 1 = BOOTREQUEST, 2 = BOOTREPLY
htype	Hardware address type (e.g. '1' = 10mb ethernet).
hlen	Hardware address length (e.g. '6' = 10mb ethernet).
hops	Client sets to zero, optionally used by relay agent when booting via a relay agent.
xid	Transaction ID, a random number chosen by the client, used by the client and server to associate messages and responses between a client and a server.
secs	Filled in by client, seconds elapsed since client began address acquisition or renewal process.
flags	Flags.
ciaddr	Client IP address; only filled in if client is in BOUND, RENEW or REBINDING state and can respond to ARP requests.
yiaddr	'your' (client) IP address.
siaddr	IP address of next server to use in bootstrap; returned in DHCPOFFER, DHCPACK by server.
giaddr	Relay agent IP address, used in booting via a relay agent.
chaddr	Client hardware address.
sname	Optional server host name, null terminated string.
boot_file	Boot file name, null terminated string; "generic" name or null in DHCPDISCOVER, fully qualified directory-path name in DHCPOFFER.
options	Optional parameters field ('DHCP message type' option must be included in every DHCP message).

**classmethod parser**(*buf*)

Decode a protocol header.

This method is used only when decoding a packet.

Decode a protocol header at offset 0 in bytearray *buf*. Returns the following three objects.

- An object to describe the decoded header.
- A `packet_base.PacketBase` subclass appropriate for the rest of the packet. None when the rest of the packet should be considered as raw payload.
- The rest of packet.

**serialize**(*\_payload=None, \_prev=None*)

Encode a protocol header.

This method is used only when encoding a packet.

Encode a protocol header. Returns a bytearray which contains the header.

*payload* is the rest of the packet which will immediately follow this header.

*prev* is a `packet_base.PacketBase` subclass for the outer protocol header. *prev* is None if the current header is the outer-most. For example, *prev* is `ipv4` or `ipv6` for `tcp.serialize`.

**class** `os_ken.lib.packet.dhcp.option`(*tag, value, length=0*)

DHCP (RFC 2132) options encoder/decoder class.

This is used with `os_ken.lib.packet.dhcp.dhcp.options`.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
tag	Option type. (except for the 'magic cookie', 'pad option' and 'end option'.)
value	Option's value. (set the value that has been converted to hexadecimal.)
length	Option's value length. (calculated automatically from the length of value.)

**class** `os_ken.lib.packet.dhcp.options`(*option\_list=None, options\_len=0, magic\_cookie='99.130.83.99'*)

DHCP (RFC 2132) options encoder/decoder class.

This is used with `os_ken.lib.packet.dhcp.dhcp`.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

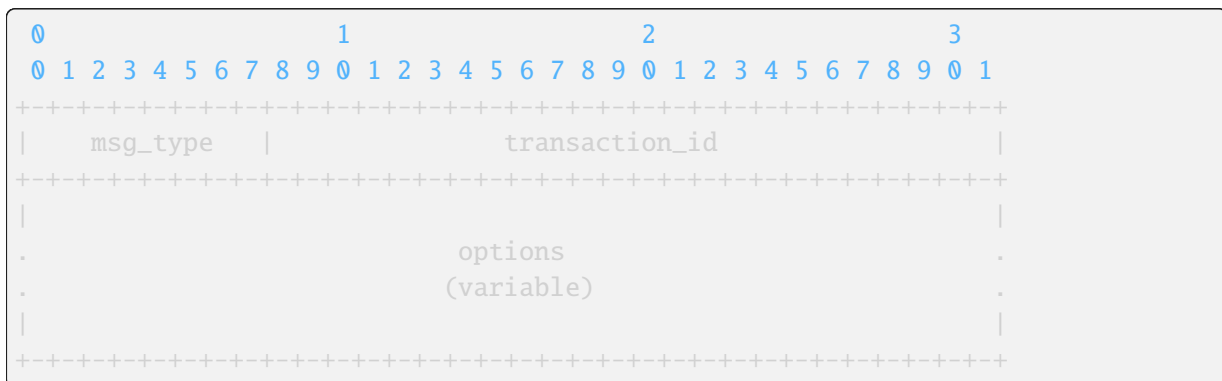
Attribute	Description
option_list	'end option' and 'pad option' are added automatically after the option class is stored in array.
options_len	Option's byte length. ('magic cookie', 'end option' and 'pad option' length including.)
magic_cookie	The first four octets contain the decimal values 99, 130, 83 and 99.

## DHCP6

DHCPv6 packet parser/serializer

[RFC 3315] DHCPv6 packet format:

The following diagram illustrates the format of DHCP messages sent between clients and servers:

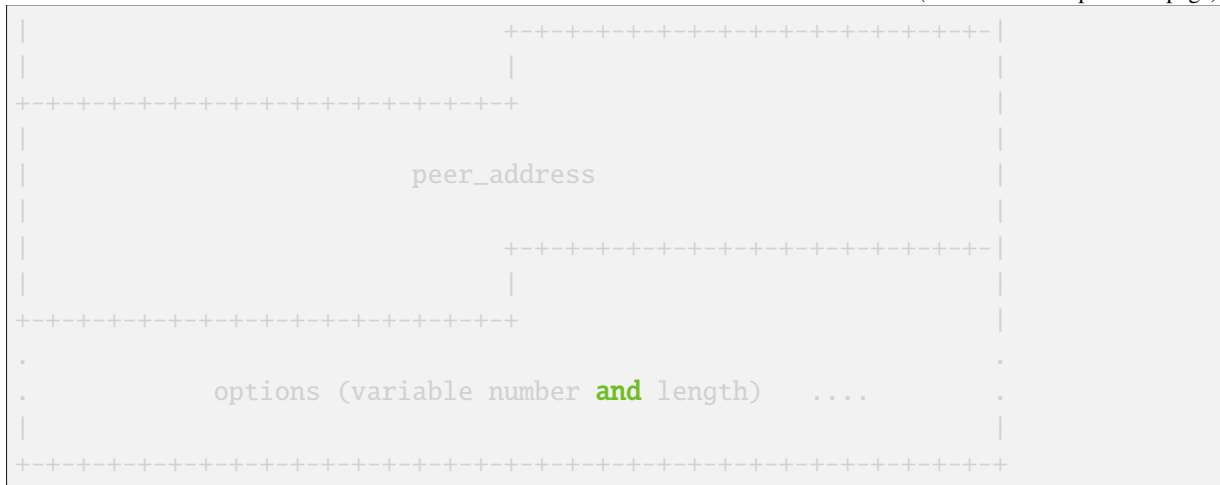


There are two relay agent messages, which share the following format:



(continues on next page)

(continued from previous page)



**class** `os_ken.lib.packet.dhcp6.dhcp6`(*msg\_type, options, transaction\_id=None, hop\_count=0, link\_address='::', peer\_address='::'*)

DHCPv6 (RFC 3315) header encoder/decoder class.

The serialized packet would look like the ones described in the following sections.

- RFC 3315 DHCP packet format

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
<code>msg_type</code>	Identifies the DHCP message type
<code>transaction_id</code>	For unrelayed messages only: the transaction ID for this message exchange.
<code>hop_count</code>	For relayed messages only: number of relay agents that have relayed this message.
<code>link_addr</code>	For relayed messages only: a global or site-local address that will be used by the server to identify the link on which the client is located.
<code>peer_addr</code>	For relayed messages only: the address of the client or relay agent from which the message to be relayed was received.
<code>options</code>	Options carried in this message

**classmethod** `parser`(*buf*)

Decode a protocol header.

This method is used only when decoding a packet.

Decode a protocol header at offset 0 in bytearray *buf*. Returns the following three objects.

- An object to describe the decoded header.
- A `packet_base.PacketBase` subclass appropriate for the rest of the packet. None when the rest of the packet should be considered as raw payload.
- The rest of packet.



**serialize**(*payload=None, prev=None*)

Encode a protocol header.

This method is used only when encoding a packet.

Encode a protocol header. Returns a bytearray which contains the header.

*payload* is the rest of the packet which will immediately follow this header.

*prev* is a `packet_base.PacketBase` subclass for the outer protocol header. *prev* is `None` if the current header is the outer-most. For example, *prev* is `ipv4` or `ipv6` for `tcp.serialize`.

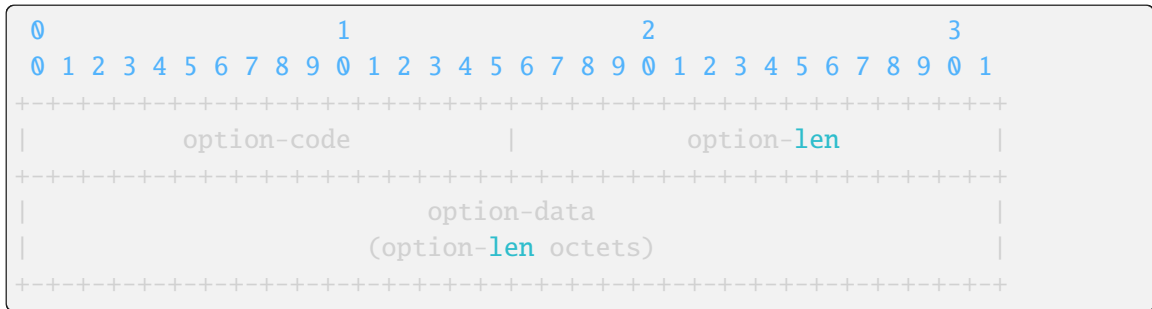
**class** `os_ken.lib.packet.dhcp6.option`(*code, data, length=0*)

DHCP (RFC 3315) options encoder/decoder class.

This is used with `os_ken.lib.packet.dhcp6.dhcp6.options`.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

The format of DHCP options is:



Attribute	Description
<code>option-code</code>	An unsigned integer identifying the specific option type carried in this option.
<code>option-len</code>	An unsigned integer giving the length of the option-data field in this option in octets.
<code>option-data</code>	The data for the option; the format of this data depends on the definition of the option.

**class** `os_ken.lib.packet.dhcp6.options`(*option\_list=None, options\_len=0*)

DHCP (RFC 3315) options encoder/decoder class.

This is used with `os_ken.lib.packet.dhcp6.dhcp6`.

## Ethernet

**class** `os_ken.lib.packet.ethernet.ethernet`(*dst='ff:ff:ff:ff:ff:ff', src='00:00:00:00:00:00', ethertype=2048*)

Ethernet header encoder/decoder class.

An instance has the following attributes at least. MAC addresses are represented as a string like `'08:60:6e:7f:74:e7'`. `__init__` takes the corresponding args in this order.

Attribute	Description	Example
dst	destination address	'ff:ff:ff:ff:ff:ff'
src	source address	'08:60:6e:7f:74:e7'
ethertype	ether type	0x0800

### **classmethod** `get_packet_type`(*type\_*)

Override method for the ethernet IEEE802.3 Length/Type field (self.ethertype).

If the value of Length/Type field is less than or equal to 1500 decimal(05DC hexadecimal), it means Length interpretation and be passed to the LLC sublayer.

### **classmethod** `parser`(*buf*)

Decode a protocol header.

This method is used only when decoding a packet.

Decode a protocol header at offset 0 in bytearray *buf*. Returns the following three objects.

- An object to describe the decoded header.
- A `packet_base.PacketBase` subclass appropriate for the rest of the packet. None when the rest of the packet should be considered as raw payload.
- The rest of packet.

### **serialize**(*payload*, *prev*)

Encode a protocol header.

This method is used only when encoding a packet.

Encode a protocol header. Returns a bytearray which contains the header.

*payload* is the rest of the packet which will immediately follow this header.

*prev* is a `packet_base.PacketBase` subclass for the outer protocol header. *prev* is None if the current header is the outer-most. For example, *prev* is `ipv4` or `ipv6` for `tcp.serialize`.

## Geneve

Geneve packet parser/serializer

```
class os_ken.lib.packet.geneve.Option(option_class=None, type_=None, length=0)
```

Tunnel Options

```
class os_ken.lib.packet.geneve.OptionDataUnknown(buf, option_class=None, type_=None,
length=0)
```

Unknown Option Class and Type specific Option

```
class os_ken.lib.packet.geneve.geneve(version=0, opt_len=0, flags=0, protocol=25944,
vni=None, options=None)
```

Geneve (RFC draft-ietf-nvo3-geneve-03) header encoder/decoder class.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
version	Version.
opt_len	The length of the options fields.
flags	Flag field for OAM packet and Critical options present.
protocol	Protocol Type field. The Protocol Type is defined as "ETHER TYPES".
vni	Identifier for unique element of virtual network.
options	List of <code>Option*</code> instance.

### **classmethod** `parser(buf)`

Decode a protocol header.

This method is used only when decoding a packet.

Decode a protocol header at offset 0 in bytearray *buf*. Returns the following three objects.

- An object to describe the decoded header.
- A `packet_base.PacketBase` subclass appropriate for the rest of the packet. None when the rest of the packet should be considered as raw payload.
- The rest of packet.

### **serialize(payload=None, prev=None)**

Encode a protocol header.

This method is used only when encoding a packet.

Encode a protocol header. Returns a bytearray which contains the header.

*payload* is the rest of the packet which will immediately follow this header.

*prev* is a `packet_base.PacketBase` subclass for the outer protocol header. *prev* is None if the current header is the outer-most. For example, *prev* is `ipv4` or `ipv6` for `tcp.serialize`.

## GRE

```
class os_ken.lib.packet.gre.gre(version=0, protocol=2048, checksum=None, key=None,
                               vsid=None, flow_id=None, seq_number=None)
```

GRE (RFC2784,RFC2890) header encoder/decoder class.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
version	Version.
protocol	Protocol Type field. The Protocol Type is defined as "ETHER TYPES".
checksum	Checksum field(optional). When you set a value other than None, this field will be automatically calculated.
key	Key field(optional) This field is intended to be used for identifying an individual traffic flow within a tunnel.
vsid	Virtual Subnet ID field(optional) This field is a 24-bit value that is used to identify the NVGRE-based Virtual Layer 2 Network.
flow_id	FlowID field(optional) This field is an 8-bit value that is used to provide per-flow entropy for flows in the same VSID.
seq_num	Sequence Number field(optional)

### classmethod parser(buf)

Decode a protocol header.

This method is used only when decoding a packet.

Decode a protocol header at offset 0 in bytearray *buf*. Returns the following three objects.

- An object to describe the decoded header.
- A packet\_base.PacketBase subclass appropriate for the rest of the packet. None when the rest of the packet should be considered as raw payload.
- The rest of packet.

### serialize(payload=None, prev=None)

Encode a protocol header.

This method is used only when encoding a packet.

Encode a protocol header. Returns a bytearray which contains the header.

*payload* is the rest of the packet which will immediately follow this header.

*prev* is a packet\_base.PacketBase subclass for the outer protocol header. *prev* is None if the current header is the outer-most. For example, *prev* is ipv4 or ipv6 for tcp.serialize.

### os\_ken.lib.packet.gre.nvgre(version=0, vsid=0, flow\_id=0)

Generate instance of GRE class with information for NVGRE (RFC7637).

#### Parameters

- **version** -- Version.
- **vsid** -- Virtual Subnet ID.
- **flow\_id** -- FlowID.

#### Returns

Instance of GRE class with information for NVGRE.

## ICMP

**class** `os_ken.lib.packet.icmp.TimeExceeded`(*data\_len=0, data=None*)

ICMP sub encoder/decoder class for Time Exceeded Message.

This is used with `os_ken.lib.packet.icmp.icmp` for ICMP Time Exceeded Message.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

[RFC4884] introduced 8-bit data length attribute.

Attribute	Description
<code>data_len</code>	data length
<code>data</code>	Internet Header + leading octets of original datagram

**class** `os_ken.lib.packet.icmp.dest_unreach`(*data\_len=0, mtu=0, data=None*)

ICMP sub encoder/decoder class for Destination Unreachable Message.

This is used with `os_ken.lib.packet.icmp.icmp` for ICMP Destination Unreachable Message.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

[RFC1191] reserves bits for the "Next-Hop MTU" field. [RFC4884] introduced 8-bit data length attribute.

Attribute	Description
<code>data_len</code>	data length
<code>mtu</code>	Next-Hop MTU NOTE: This field is required when icmp code is 4 code 4 = fragmentation needed and DF set
<code>data</code>	Internet Header + leading octets of original datagram

**class** `os_ken.lib.packet.icmp.echo`(*id\_=0, seq=0, data=None*)

ICMP sub encoder/decoder class for Echo and Echo Reply messages.

This is used with `os_ken.lib.packet.icmp.icmp` for ICMP Echo and Echo Reply messages.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
<code>id</code>	Identifier
<code>seq</code>	Sequence Number
<code>data</code>	Internet Header + 64 bits of Original Data Datagram

**class** `os_ken.lib.packet.icmp.icmp`(*type\_=8, code=0, csum=0, data=b''*)

ICMP (RFC 792) header encoder/decoder class.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
type	Type
code	Code
csum	Checksum (0 means automatically-calculate when encoding)
data	Payload. Either a bytearray, or <code>os_ken.lib.packet.icmp.echo</code> or <code>os_ken.lib.packet.icmp.dest_unreach</code> or <code>os_ken.lib.packet.icmp.TimeExceeded</code> object NOTE for <code>icmp.echo</code> : This includes "unused" 16 bits and the following "Internet Header + 64 bits of Original Data Datagram" of the ICMP header. NOTE for <code>icmp.dest_unreach</code> and <code>icmp.TimeExceeded</code> : This includes "unused" 8 or 24 bits and the following "Internet Header + leading octets of original datagram" of the original packet.

### **classmethod** `parser(buf)`

Decode a protocol header.

This method is used only when decoding a packet.

Decode a protocol header at offset 0 in bytearray *buf*. Returns the following three objects.

- An object to describe the decoded header.
- A `packet_base.PacketBase` subclass appropriate for the rest of the packet. None when the rest of the packet should be considered as raw payload.
- The rest of packet.

### **serialize(payload, prev)**

Encode a protocol header.

This method is used only when encoding a packet.

Encode a protocol header. Returns a bytearray which contains the header.

*payload* is the rest of the packet which will immediately follow this header.

*prev* is a `packet_base.PacketBase` subclass for the outer protocol header. *prev* is None if the current header is the outer-most. For example, *prev* is `ipv4` or `ipv6` for `tcp.serialize`.

## ICMPv6

### **class** `os_ken.lib.packet.icmpv6.echo(id_=0, seq=0, data=None)`

ICMPv6 sub encoder/decoder class for Echo Request and Echo Reply messages.

This is used with `os_ken.lib.packet.icmpv6.icmpv6` for ICMPv6 Echo Request and Echo Reply messages.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
id	Identifier
seq	Sequence Number
data	Data

**class** `os_ken.lib.packet.icmpv6.icmpv6`(*type\_=0, code=0, csum=0, data=b''*)

ICMPv6 (RFC 2463) header encoder/decoder class.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
<code>type_</code>	Type
<code>code</code>	Code
<code>csum</code>	Checksum (0 means automatically-calculate when encoding)
<code>data</code>	Payload. <code>os_ken.lib.packet.icmpv6.echo</code> object, <code>os_ken.lib.packet.icmpv6.nd_neighbor</code> object, <code>os_ken.lib.packet.icmpv6.nd_router_solicit</code> object, <code>os_ken.lib.packet.icmpv6.nd_router_advert</code> object, <code>os_ken.lib.packet.icmpv6.mld</code> object, or a bytearray.

**classmethod** `parser`(*buf*)

Decode a protocol header.

This method is used only when decoding a packet.

Decode a protocol header at offset 0 in bytearray *buf*. Returns the following three objects.

- An object to describe the decoded header.
- A `packet_base.PacketBase` subclass appropriate for the rest of the packet. None when the rest of the packet should be considered as raw payload.
- The rest of packet.

**serialize**(*payload, prev*)

Encode a protocol header.

This method is used only when encoding a packet.

Encode a protocol header. Returns a bytearray which contains the header.

*payload* is the rest of the packet which will immediately follow this header.

*prev* is a `packet_base.PacketBase` subclass for the outer protocol header. *prev* is None if the current header is the outer-most. For example, *prev* is `ipv4` or `ipv6` for `tcp.serialize`.

**class** `os_ken.lib.packet.icmpv6.mld`(*maxresp=0, address='::'*)

ICMPv6 sub encoder/decoder class for MLD Lister Query, MLD Listener Report, and MLD Listener Done messages. (RFC 2710)

<http://www.ietf.org/rfc/rfc2710.txt>

This is used with `os_ken.lib.packet.icmpv6.icmpv6`.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
<code>maxresp</code>	max response time in millisecond. it is meaningful only in Query Message.
<code>address</code>	a group address value.

```
class os_ken.lib.packet.icmpv6.mldv2_query(maxresp=0, address='::', s_flg=0, qrv=2,
                                           qqic=0, num=0, srcs=None)
```

ICMPv6 sub encoder/decoder class for MLD v2 Lister Query messages. (RFC 3810)

<http://www.ietf.org/rfc/rfc3810.txt>

This is used with `os_ken.lib.packet.icmpv6.icmpv6`.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
<code>maxresp</code>	max response time in millisecond. it is meaningful only in Query Message.
<code>address</code>	a group address value.
<code>s_flg</code>	when set to 1, routers suppress the timer process.
<code>qrv</code>	robustness variable for a querier.
<code>qqic</code>	an interval time for a querier in unit of seconds.
<code>num</code>	a number of the multicast servers.
<code>srcs</code>	a list of IPv6 addresses of the multicast servers.

```
class os_ken.lib.packet.icmpv6.mldv2_report(record_num=0, records=None)
```

ICMPv6 sub encoder/decoder class for MLD v2 Lister Report messages. (RFC 3810)

<http://www.ietf.org/rfc/rfc3810.txt>

This is used with `os_ken.lib.packet.icmpv6.icmpv6`.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
<code>record_num</code>	a number of the group records.
<code>records</code>	a list of <code>os_ken.lib.packet.icmpv6.mldv2_report_group</code> . None if no records.

```
class os_ken.lib.packet.icmpv6.mldv2_report_group(type_=0, aux_len=0, num=0,
                                                  address='::', srcs=None,
                                                  aux=None)
```

ICMPv6 sub encoder/decoder class for MLD v2 Lister Report Group Record messages. (RFC 3810)

This is used with `os_ken.lib.packet.icmpv6.mldv2_report`.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.



Attribute	Description
type_	a group record type for v3.
aux_len	the length of the auxiliary data in 32-bit words.
num	a number of the multicast servers.
address	a group address value.
srcs	a list of IPv6 addresses of the multicast servers.
aux	the auxiliary data.

**class** `os_ken.lib.packet.icmpv6.nd_neighbor`(*res=0, dst='::', option=None*)

ICMPv6 sub encoder/decoder class for Neighbor Solicitation and Neighbor Advertisement messages. (RFC 4861)

This is used with `os_ken.lib.packet.icmpv6.icmpv6`.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
res	R,S,O Flags for Neighbor Advertisement. The 3 MSBs of "Reserved" field for Neighbor Solicitation.
dst	Target Address
option	a derived object of <code>os_ken.lib.packet.icmpv6.nd_option</code> or a bytearray. None if no options.

**class** `os_ken.lib.packet.icmpv6.nd_option_mtu`(*mtu=1500*)

ICMPv6 sub encoder/decoder class for Neighbor discovery MTU Option. (RFC 4861)

This is used with `os_ken.lib.packet.icmpv6.nd_router_advert`.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
mtu	MTU.

**class** `os_ken.lib.packet.icmpv6.nd_option_pi`(*length=0, pl=0, res1=0, val\_l=0, pre\_l=0, res2=0, prefix='::')*

ICMPv6 sub encoder/decoder class for Neighbor discovery Prefix Information Option. (RFC 4861)

This is used with `os_ken.lib.packet.icmpv6.nd_router_advert`.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
length	length of the option. (0 means automatically-calculate when encoding)
pl	Prefix Length.
res1	L,A,R* Flags for Prefix Information.
val_l	Valid Lifetime.
pre_l	Preferred Lifetime.
res2	This field is unused. It MUST be initialized to zero.
prefix	An IP address or a prefix of an IP address.

\*R flag is defined in (RFC 3775)

```
class os_ken.lib.packet.icmpv6.nd_option_sla(length=0, hw_src='00:00:00:00:00:00',
                                             data=None)
```

ICMPv6 sub encoder/decoder class for Neighbor discovery Source Link-Layer Address Option. (RFC 4861)

This is used with `os_ken.lib.packet.icmpv6.nd_neighbor`, `os_ken.lib.packet.icmpv6.nd_router_solicit` or `os_ken.lib.packet.icmpv6.nd_router_advert`.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
length	length of the option. (0 means automatically-calculate when encoding)
hw_src	Link-Layer Address. NOTE: If the address is longer than 6 octets this contains the first 6 octets in the address. This implementation assumes the address has at least 6 octets.
data	A bytearray which contains the rest of Link-Layer Address and padding. When encoding a packet, it's user's responsibility to provide necessary padding for 8-octets alignment required by the protocol.

```
class os_ken.lib.packet.icmpv6.nd_option_tla(length=0, hw_src='00:00:00:00:00:00',
                                             data=None)
```

ICMPv6 sub encoder/decoder class for Neighbor discovery Target Link-Layer Address Option. (RFC 4861)

This is used with `os_ken.lib.packet.icmpv6.nd_neighbor`.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
length	length of the option. (0 means automatically-calculate when encoding)
hw_src	Link-Layer Address. NOTE: If the address is longer than 6 octets this contains the first 6 octets in the address. This implementation assumes the address has at least 6 octets.
data	A bytearray which contains the rest of Link-Layer Address and padding. When encoding a packet, it's user's responsibility to provide necessary padding for 8-octets alignment required by the protocol.

**class** `os_ken.lib.packet.icmpv6.nd_router_advert`(*ch\_l=0, res=0, rou\_l=0, rea\_t=0, ret\_t=0, options=None*)

ICMPv6 sub encoder/decoder class for Router Advertisement messages. (RFC 4861)

This is used with `os_ken.lib.packet.icmpv6.icmpv6`.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
<code>ch_l</code>	Cur Hop Limit.
<code>res</code>	M,O Flags for Router Advertisement.
<code>rou_l</code>	Router Lifetime.
<code>rea_t</code>	Reachable Time.
<code>ret_t</code>	Retrans Timer.
<code>options</code>	List of a derived object of <code>os_ken.lib.packet.icmpv6.nd_option</code> or a bytearray. None if no options.

**class** `os_ken.lib.packet.icmpv6.nd_router_solicit`(*res=0, option=None*)

ICMPv6 sub encoder/decoder class for Router Solicitation messages. (RFC 4861)

This is used with `os_ken.lib.packet.icmpv6.icmpv6`.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
<code>res</code>	This field is unused. It MUST be initialized to zero.
<code>option</code>	a derived object of <code>os_ken.lib.packet.icmpv6.nd_option</code> or a bytearray. None if no options.

## IGMP

Internet Group Management Protocol(IGMP) packet parser/serializer

[RFC 1112] IGMP v1 format:



[RFC 2236] IGMP v2 format:



(continues on next page)

(continued from previous page)

Type	Max Resp Time	Checksum
Group Address		

[RFC 3376] IGMP v3 Membership Query format:

0	1	2	3
0 1 2 3 4 5 6 7 8 9	0 1 2 3 4 5 6 7 8 9	0 1 2 3 4 5 6 7 8 9	0 1
Type = 0x11	Max Resp Code	Checksum	
Group Address			
Resv	S  QRV	QQIC	Number of Sources (N)
Source Address [1]			
Source Address [2]			
.			
.			
.			
Source Address [N]			

IGMP v3 Membership Report format:

0	1	2	3
0 1 2 3 4 5 6 7 8 9	0 1 2 3 4 5 6 7 8 9	0 1 2 3 4 5 6 7 8 9	0 1
Type = 0x22	Reserved	Checksum	
Reserved		Number of Group Records (M)	
Group Record [1]			
Group Record [2]			
.			
.			
.			

(continues on next page)

(continued from previous page)



Where each Group Record has the following internal format:



**class** `os_ken.lib.packet.igmp.igmp(msgtype=17, maxresp=0, csum=0, address='0.0.0.0')`  
Internet Group Management Protocol (IGMP, RFC 1112, RFC 2236) header encoder/decoder class.

<http://www.ietf.org/rfc/rfc1112.txt>

<http://www.ietf.org/rfc/rfc2236.txt>

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

At-tribute	Description
msgtype	a message type for v2, or a combination of version and a message type for v1.
maxresp	max response time in unit of 1/10 second. it is meaningful only in Query Message.
csum	a check sum value. 0 means automatically-calculate when encoding.
address	a group address value.

**classmethod** `parser(buf)`

Decode a protocol header.

This method is used only when decoding a packet.

Decode a protocol header at offset 0 in bytearray *buf*. Returns the following three objects.

- An object to describe the decoded header.
- A `packet_base.PacketBase` subclass appropriate for the rest of the packet. None when the rest of the packet should be considered as raw payload.
- The rest of packet.

**serialize**(*payload*, *prev*)

Encode a protocol header.

This method is used only when encoding a packet.

Encode a protocol header. Returns a bytearray which contains the header.

*payload* is the rest of the packet which will immediately follow this header.

*prev* is a `packet_base.PacketBase` subclass for the outer protocol header. *prev* is None if the current header is the outer-most. For example, *prev* is `ipv4` or `ipv6` for `tcp.serialize`.

```
class os_ken.lib.packet.igmp.igmpv3_query(msgtype=17, maxresp=100, csum=0,
                                         address='0.0.0.0', s_flg=0, qrv=2, qqic=0,
                                         num=0, srcs=None)
```

Internet Group Management Protocol(IGMP, RFC 3376) Membership Query message encoder/decoder class.

<http://www.ietf.org/rfc/rfc3376.txt>

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
<code>msgtype</code>	a message type for v3.
<code>maxresp</code>	max response time in unit of 1/10 second.
<code>csum</code>	a check sum value. 0 means automatically-calculate when encoding.
<code>address</code>	a group address value.
<code>s_flg</code>	when set to 1, routers suppress the timer process.
<code>qrv</code>	robustness variable for a querier.
<code>qqic</code>	an interval time for a querier in unit of seconds.
<code>num</code>	a number of the multicast servers.
<code>srcs</code>	a list of IPv4 addresses of the multicast servers.

**classmethod parser**(*buf*)

Decode a protocol header.

This method is used only when decoding a packet.

Decode a protocol header at offset 0 in bytearray *buf*. Returns the following three objects.

- An object to describe the decoded header.

- A `packet_base.PacketBase` subclass appropriate for the rest of the packet. None when the rest of the packet should be considered as raw payload.
- The rest of packet.

**serialize**(*payload*, *prev*)

Encode a protocol header.

This method is used only when encoding a packet.

Encode a protocol header. Returns a bytearray which contains the header.

*payload* is the rest of the packet which will immediately follow this header.

*prev* is a `packet_base.PacketBase` subclass for the outer protocol header. *prev* is None if the current header is the outer-most. For example, *prev* is `ipv4` or `ipv6` for `tcp.serialize`.

**class** `os_ken.lib.packet.igmp.igmpv3_report`(*msgtype=34*, *csum=0*, *record\_num=0*,  
*records=None*)

Internet Group Management Protocol(IGMP, RFC 3376) Membership Report message encoder/decoder class.

<http://www.ietf.org/rfc/rfc3376.txt>

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
<code>msgtype</code>	a message type for v3.
<code>csum</code>	a check sum value. 0 means automatically-calculate when encoding.
<code>record_num</code>	a number of the group records.
<code>records</code>	a list of <code>os_ken.lib.packet.igmp.igmpv3_report_group</code> . None if no records.

**classmethod** `parser`(*buf*)

Decode a protocol header.

This method is used only when decoding a packet.

Decode a protocol header at offset 0 in bytearray *buf*. Returns the following three objects.

- An object to describe the decoded header.
- A `packet_base.PacketBase` subclass appropriate for the rest of the packet. None when the rest of the packet should be considered as raw payload.
- The rest of packet.

**serialize**(*payload*, *prev*)

Encode a protocol header.

This method is used only when encoding a packet.

Encode a protocol header. Returns a bytearray which contains the header.

*payload* is the rest of the packet which will immediately follow this header.

*prev* is a `packet_base.PacketBase` subclass for the outer protocol header. *prev* is None if the current header is the outer-most. For example, *prev* is `ipv4` or `ipv6` for `tcp.serialize`.

```
class os_ken.lib.packet.igmp.igmpv3_report_group(type_=0, aux_len=0, num=0,  
                                                address='0.0.0.0', srcs=None,  
                                                aux=None)
```

Internet Group Management Protocol(IGMP, RFC 3376) Membership Report Group Record message encoder/decoder class.

<http://www.ietf.org/rfc/rfc3376.txt>

This is used with `os_ken.lib.packet.igmp.igmpv3_report`.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
<code>type_</code>	a group record type for v3.
<code>aux_len</code>	the length of the auxiliary data.
<code>num</code>	a number of the multicast servers.
<code>address</code>	a group address value.
<code>srcs</code>	a list of IPv4 addresses of the multicast servers.
<code>aux</code>	the auxiliary data.

## IPv4

```
class os_ken.lib.packet.ipv4.ipv4(version=4, header_length=5, tos=0, total_length=0,  
                                   identification=0, flags=0, offset=0, ttl=255, proto=0,  
                                   csum=0, src='10.0.0.1', dst='10.0.0.2', option=None)
```

IPv4 (RFC 791) header encoder/decoder class.

NOTE: When decoding, this implementation tries to decode the upper layer protocol even for a fragmented datagram. It isn't likely what a user would want.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. IPv4 addresses are represented as a string like '192.0.2.1'. `__init__` takes the corresponding args in this order.



Attribute	Description	Example
version	Version	
header_length	IHL	
tos	Type of Service	
total_length	Total Length (0 means automatically-calculate when encoding)	
identifica- tion	Identification	
flags	Flags	
offset	Fragment Offset	
ttl	Time to Live	
proto	Protocol	
csum	Header Checksum (Ignored and automatically-calculated when encoding)	
src	Source Address	'192.0.2.1'
dst	Destination Address	'192.0.2.2'
option	A bytearray which contains the entire Options, or None for no Options	

**classmethod** `parser(buf)`

Decode a protocol header.

This method is used only when decoding a packet.

Decode a protocol header at offset 0 in bytearray *buf*. Returns the following three objects.

- An object to describe the decoded header.
- A `packet_base.PacketBase` subclass appropriate for the rest of the packet. None when the rest of the packet should be considered as raw payload.
- The rest of packet.

**serialize(payload, prev)**

Encode a protocol header.

This method is used only when encoding a packet.

Encode a protocol header. Returns a bytearray which contains the header.

*payload* is the rest of the packet which will immediately follow this header.

*prev* is a `packet_base.PacketBase` subclass for the outer protocol header. *prev* is None if the current header is the outer-most. For example, *prev* is `ipv4` or `ipv6` for `tcp.serialize`.

**IPv6**

**class** `os_ken.lib.packet.ipv6.auth(nxt=6, size=2, spi=0, seq=0, data=b'\x00\x00\x00\x00')`

IP Authentication header (RFC 2402) encoder/decoder class.

This is used with `os_ken.lib.packet.ipv6.ipv6`.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
nxt	Next Header
size	the length of the Authentication Header in 64-bit words, subtracting 1.
spi	security parameters index.
seq	sequence number.
data	authentication data.

**class** `os_ken.lib.packet.ipv6.dst_opts`(*nxt=6, size=0, data=None*)

IPv6 (RFC 2460) destination header encoder/decoder class.

This is used with `os_ken.lib.packet.ipv6.ipv6`.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
nxt	Next Header
size	the length of the destination header, not include the first 8 octet.
data	IPv6 options.

**class** `os_ken.lib.packet.ipv6.fragment`(*nxt=6, offset=0, more=0, id\_=0*)

IPv6 (RFC 2460) fragment header encoder/decoder class.

This is used with `os_ken.lib.packet.ipv6.ipv6`.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
nxt	Next Header
offset	offset, in 8-octet units, relative to the start of the fragmentable part of the original packet.
more	1 means more fragments follow; 0 means last fragment.
id_	packet identification value.

**class** `os_ken.lib.packet.ipv6.header`(*nxt*)

extension header abstract class.

**class** `os_ken.lib.packet.ipv6.hop_opts`(*nxt=6, size=0, data=None*)

IPv6 (RFC 2460) Hop-by-Hop Options header encoder/decoder class.

This is used with `os_ken.lib.packet.ipv6.ipv6`.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
<code>nxt</code>	Next Header
<code>size</code>	the length of the Hop-by-Hop Options header, not include the first 8 octet.
<code>data</code>	IPv6 options.

```
class os_ken.lib.packet.ipv6.ipv6(version=6, traffic_class=0, flow_label=0,
                                payload_length=0, nxt=6, hop_limit=255, src='10::10',
                                dst='20::20', ext_hdrs=None)
```

IPv6 (RFC 2460) header encoder/decoder class.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. IPv6 addresses are represented as a string like `'ff02::1'`. `__init__` takes the corresponding args in this order.

Attribute	Description	Example
<code>version</code>	Version	
<code>traffic_class</code>	Traffic Class	
<code>flow_label</code>	When decoding, Flow Label. When encoding, the most significant 8 bits of Flow Label.	
<code>payload_length</code>	Payload Length	
<code>nxt</code>	Next Header	
<code>hop_limit</code>	Hop Limit	
<code>src</code>	Source Address	<code>'ff02::1'</code>
<code>dst</code>	Destination Address	<code>'::'</code>
<code>ext_hdrs</code>	Extension Headers	

**classmethod** `parser(buf)`

Decode a protocol header.

This method is used only when decoding a packet.

Decode a protocol header at offset 0 in bytearray `buf`. Returns the following three objects.

- An object to describe the decoded header.
- A `packet_base.PacketBase` subclass appropriate for the rest of the packet. None when the rest of the packet should be considered as raw payload.
- The rest of packet.

**serialize(payload, prev)**

Encode a protocol header.

This method is used only when encoding a packet.

Encode a protocol header. Returns a bytearray which contains the header.

`payload` is the rest of the packet which will immediately follow this header.

`prev` is a `packet_base.PacketBase` subclass for the outer protocol header. `prev` is None if the current header is the outer-most. For example, `prev` is `ipv4` or `ipv6` for `tcp.serialize`.

**class** `os_ken.lib.packet.ipv6.opt_header`(*nxt*, *size*, *data*)

an abstract class for Hop-by-Hop Options header and destination header.

**class** `os_ken.lib.packet.ipv6.option`(*type\_*=0, *len\_*=-1, *data*=None)

IPv6 (RFC 2460) Options header encoder/decoder class.

**This is used with** `os_ken.lib.packet.ipv6.hop_opts` or  
`os_ken.lib.packet.ipv6.dst_opts`.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
<code>type_</code>	option type.
<code>len_</code>	the length of data. -1 if <code>type_</code> is 0.
<code>data</code>	an option value. None if <code>len_</code> is 0 or -1.

**class** `os_ken.lib.packet.ipv6.routing`(*nxt*)

An IPv6 Routing Header decoder class. This class has only the parser method.

IPv6 Routing Header types.

<http://www.iana.org/assignments/ipv6-parameters/ipv6-parameters.xhtml>

Value	Description	Reference
0	Source Route (DEPRECATED)	[[IPV6]][RFC5095]
1	Nimrod (DEPRECATED 2009-05-06)	
2	Type 2 Routing Header	[RFC6275]
3	RPL Source Route Header	[RFC6554]
4 - 252	Unassigned	
253	RFC3692-style Experiment 1 [2]	[RFC4727]
254	RFC3692-style Experiment 2 [2]	[RFC4727]
255	Reserved	

**class** `os_ken.lib.packet.ipv6.routing_type3`(*nxt*=6, *size*=0, *type\_*=3, *seg*=0, *cmpi*=0, *cmpe*=0, *adrs*=None)

An IPv6 Routing Header for Source Routes with the RPL (RFC 6554) encoder/decoder class.

This is used with `os_ken.lib.packet.ipv6.ipv6`.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
nxt	Next Header
size	The length of the Routing header, not include the first 8 octet. (0 means automatically-calculate when encoding)
type	Identifies the particular Routing header variant.
seg	Number of route segments remaining.
cmpi	Number of prefix octets from segments 1 through n-1.
cmpe	Number of prefix octets from segment n.
pad	Number of octets that are used for padding after Address[n] at the end of the SRH.
adrs	Vector of addresses, numbered 1 to n.

## LLC

Logical Link Control(LLC, IEEE 802.2) parser/serializer <http://standards.ieee.org/getieee802/download/802.2-1998.pdf>

LLC format:

```

+-----+-----+
| DSAP address | 8 bits |
+-----+-----+
| SSAP address | 8 bits |
+-----+-----+
| Control      | 8 or 16 bits |
+-----+-----+

```

DSAP address field:

```

LSB
+-----+-----+
| I/G | D | D | D | D | D | D | D |
+-----+-----+
I/G bit = 0 : Individual DSAP
I/G bit = 1 : Group DSA
D : DSAP address

```

SSAP address field:

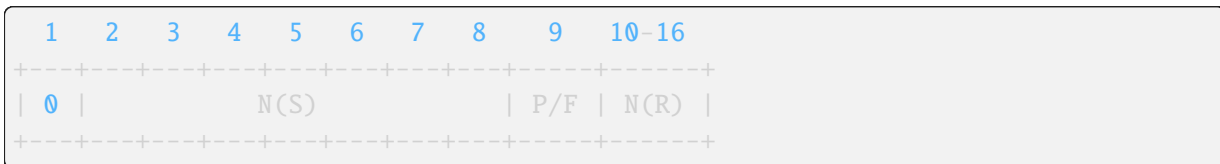
```

LSB
+-----+-----+
| C/R | S | S | S | S | S | S | S |
+-----+-----+
C/R bit = 0 : Command
C/R bit = 1 : Response
S : SSAP address

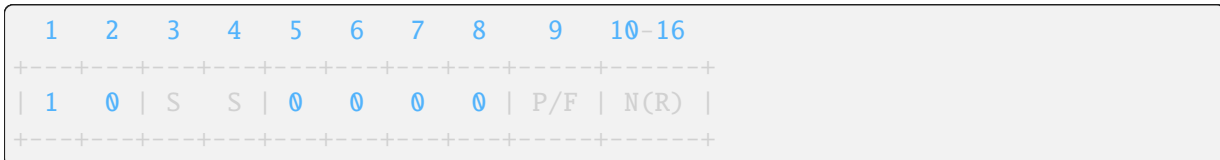
```

Control field:

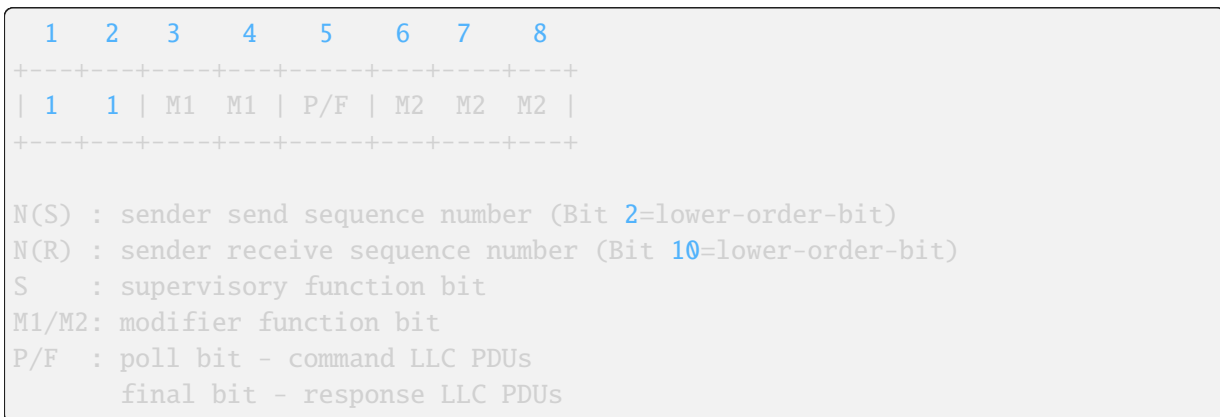
Information transfer command/response (I-format PDU):



Supervisory commands/responses (S-format PDUs):



Unnumbered commands/responses (U-format PDUs):



```
class os_ken.lib.packet.llc.ControlFormatI(send_sequence_number=0, pf_bit=0,
                                             receive_sequence_number=0)
```

LLC sub encoder/decoder class for control I-format field.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
<code>send_sequence_number</code>	sender send sequence number
<code>pf_bit</code>	poll/final bit
<code>receive_sequence_number</code>	sender receive sequence number

```
class os_ken.lib.packet.llc.ControlFormatS(supervisory_function=0, pf_bit=0,
                                             receive_sequence_number=0)
```

LLC sub encoder/decoder class for control S-format field.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
<code>supervisory_function</code>	supervisory function bit
<code>pf_bit</code>	poll/final bit
<code>receive_sequence_number</code>	sender receive sequence number

**class** `os_ken.lib.packet.llc.ControlFormatU`(*modifier\_function1=0, pf\_bit=0, modifier\_function2=0*)

LLC sub encoder/decoder class for control U-format field.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
<code>modifier_function1</code>	modifier function bit
<code>pf_bit</code>	poll/final bit
<code>modifier_function2</code>	modifier function bit

**class** `os_ken.lib.packet.llc.llc`(*dsap\_addr, ssap\_addr, control*)

LLC(IEEE 802.2) header encoder/decoder class.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
<code>dsap_addr</code>	Destination service access point address field includes I/G bit at least significant bit.
<code>ssap_addr</code>	Source service access point address field includes C/R bit at least significant bit.
<code>control</code>	Control field [16 bits for formats that include sequence numbering, and 8 bits for formats that do not]. Either <code>os_ken.lib.packet.llc.ControlFormatI</code> or <code>os_ken.lib.packet.llc.ControlFormatS</code> or <code>os_ken.lib.packet.llc.ControlFormatU</code> object.

**classmethod** `parser`(*buf*)

Decode a protocol header.

This method is used only when decoding a packet.

Decode a protocol header at offset 0 in bytearray *buf*. Returns the following three objects.

- An object to describe the decoded header.
- A `packet_base.PacketBase` subclass appropriate for the rest of the packet. None when the rest of the packet should be considered as raw payload.
- The rest of packet.

**serialize**(*payload, prev*)

Encode a protocol header.

This method is used only when encoding a packet.

Encode a protocol header. Returns a bytearray which contains the header.

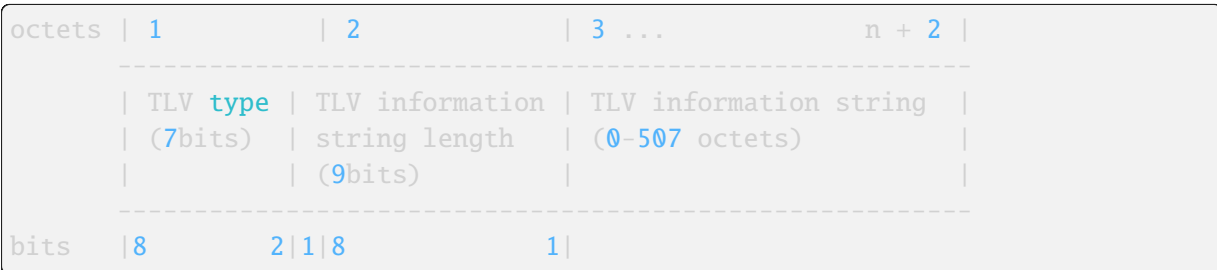
*payload* is the rest of the packet which will immediately follow this header.

*prev* is a `packet_base.PacketBase` subclass for the outer protocol header. *prev* is None if the current header is the outer-most. For example, *prev* is `ipv4` or `ipv6` for `tcp.serialize`.

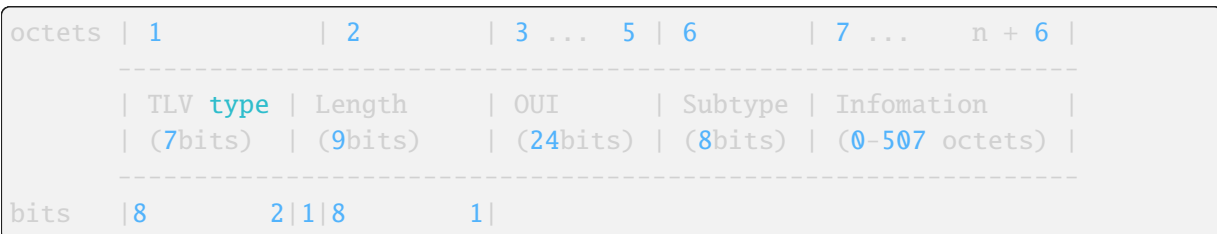
## LLDP

Link Layer Discovery Protocol(LLDP, IEEE 802.1AB) <http://standards.ieee.org/getieee802/download/802.1AB-2009.pdf>

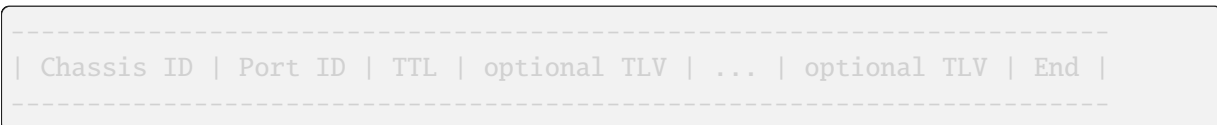
basic TLV format:



Organizationally specific TLV format:



LLDPDU format:



Chasis ID, Port ID, TTL, End are mandatory optional TLV may be inserted in any order

**class** `os_ken.lib.packet.lldp.ChassisID`(*buf=None*, \*args, \*\*kwargs)

Chassis ID TLV encoder/decoder class

Attribute	Description
buf	Binary data to parse.
subtype	Subtype.
chassis_id	Chassis id corresponding to subtype.

**class** `os_ken.lib.packet.lldp.End`(*buf=None*, \*args, \*\*kwargs)

End TLV encoder/decoder class

Attribute	Description
buf	Binary data to parse.

**class** `os_ken.lib.packet.lldp.ManagementAddress`(*buf=None*, \*args, \*\*kwargs)

Management Address TLV encoder/decoder class



Attribute	Description
buf	Binary data to parse.
addr_subtype	Address type.
addr	Device address.
intf_subtype	Interface type.
intf_num	Interface number.
oid	Object ID.

**class** os\_ken.lib.packet.lldp.**OrganizationallySpecific**(buf=None, \*args, \*\*kwargs)  
 Organizationally Specific TLV encoder/decoder class

Attribute	Description
buf	Binary data to parse.
oui	Organizationally unique ID.
subtype	Organizationally defined subtype.
info	Organizationally defined information string.

**class** os\_ken.lib.packet.lldp.**PortDescription**(buf=None, \*args, \*\*kwargs)  
 Port description TLV encoder/decoder class

Attribute	Description
buf	Binary data to parse.
port_description	Port description.

**class** os\_ken.lib.packet.lldp.**PortID**(buf=None, \*args, \*\*kwargs)  
 Port ID TLV encoder/decoder class

Attribute	Description
buf	Binary data to parse.
subtype	Subtype.
port_id	Port ID corresponding to subtype.

**class** os\_ken.lib.packet.lldp.**SystemCapabilities**(buf=None, \*args, \*\*kwargs)  
 System Capabilities TLV encoder/decoder class

Attribute	Description
buf	Binary data to parse.
system_cap	System Capabilities.
enabled_cap	Enabled Capabilities.

**class** os\_ken.lib.packet.lldp.**SystemDescription**(buf=None, \*args, \*\*kwargs)  
 System description TLV encoder/decoder class

Attribute	Description
buf	Binary data to parse.
system_description	System description.

**class** `os_ken.lib.packet.lldp.SystemName`(*buf=None, \*args, \*\*kwargs*)  
System name TLV encoder/decoder class

Attribute	Description
buf	Binary data to parse.
system_name	System name.

**class** `os_ken.lib.packet.lldp.TTL`(*buf=None, \*args, \*\*kwargs*)  
Time To Live TLV encoder/decoder class

Attribute	Description
buf	Binary data to parse.
ttl	Time To Live.

**class** `os_ken.lib.packet.lldp.lldp`(*tlvs*)  
LLDPDU encoder/decoder class.

An instance has the following attributes at least.

Attribute	Description
tlvs	List of TLV instance.

**classmethod** `parser`(*buf*)

Decode a protocol header.

This method is used only when decoding a packet.

Decode a protocol header at offset 0 in bytearray *buf*. Returns the following three objects.

- An object to describe the decoded header.
- A `packet_base.PacketBase` subclass appropriate for the rest of the packet. None when the rest of the packet should be considered as raw payload.
- The rest of packet.

**serialize**(*payload, prev*)

Encode a protocol header.

This method is used only when encoding a packet.

Encode a protocol header. Returns a bytearray which contains the header.

*payload* is the rest of the packet which will immediately follow this header.

*prev* is a `packet_base.PacketBase` subclass for the outer protocol header. *prev* is None if the current header is the outer-most. For example, *prev* is `ipv4` or `ipv6` for `tcp.serialize`.

## MPLS

`os_ken.lib.packet.mpls.label_from_bin(buf)`

Converts binary representation label to integer.

### Parameters

**buf** -- Binary representation of label.

### Returns

MPLS Label and BoS bit.

`os_ken.lib.packet.mpls.label_to_bin(mpls_label, is_bos=True)`

Converts integer label to binary representation.

### Parameters

- **mpls\_label** -- MPLS Label.
- **is\_bos** -- BoS bit.

### Returns

Binary representation of label.

**class** `os_ken.lib.packet.mpls.mpls(label=0, exp=0, bsb=1, ttl=255)`

MPLS (RFC 3032) header encoder/decoder class.

NOTE: When decoding, this implementation assumes that the inner protocol is IPv4.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
label	Label Value
exp	Experimental Use
bsb	Bottom of Stack
ttl	Time To Live

**classmethod** `parser(buf)`

Decode a protocol header.

This method is used only when decoding a packet.

Decode a protocol header at offset 0 in bytearray *buf*. Returns the following three objects.

- An object to describe the decoded header.
- A `packet_base.PacketBase` subclass appropriate for the rest of the packet. None when the rest of the packet should be considered as raw payload.
- The rest of packet.

**serialize(payload, prev)**

Encode a protocol header.

This method is used only when encoding a packet.

Encode a protocol header. Returns a bytearray which contains the header.

*payload* is the rest of the packet which will immediately follow this header.

*prev* is a `packet_base.PacketBase` subclass for the outer protocol header. *prev* is `None` if the current header is the outer-most. For example, *prev* is `ipv4` or `ipv6` for `tcp.serialize`.

## OpenFlow

**class** `os_ken.lib.packet.openflow.OFPUnparseableMsg`(*datapath*, *version*, *msg\_type*, *msg\_len*, *xid*, *body*)

Unparseable OpenFlow message encoder class.

An instance has the following attributes at least.

Attribute	Description
<code>datapath</code>	A <code>os_ken.ofproto.ofproto_protocol.ProtocolDesc</code> instance for this message or <code>None</code> if OpenFlow protocol version is unsupported version.
<code>version</code>	OpenFlow protocol version
<code>msg_ty</code>	Type of OpenFlow message
<code>msg_le</code>	Length of the message
<code>xid</code>	Transaction id
<code>body</code>	OpenFlow body data

### Note

"datapath" attribute is different from `os_ken.controller.controller.Datapath`. So you can not use "datapath" attribute to send OpenFlow messages. For example, "datapath" attribute does not have `send_msg` method.

**class** `os_ken.lib.packet.openflow.openflow`(*msg*)

OpenFlow message encoder/decoder class.

An instance has the following attributes at least.

Attribute	Description
<code>msg</code>	An instance of OpenFlow message (see <a href="#">OpenFlow protocol API Reference</a> ) or an instance of <code>OFPUnparseableMsg</code> if failed to parse packet as OpenFlow message.

**classmethod** `parser`(*buf*)

Decode a protocol header.

This method is used only when decoding a packet.

Decode a protocol header at offset 0 in bytearray *buf*. Returns the following three objects.

- An object to describe the decoded header.
- A `packet_base.PacketBase` subclass appropriate for the rest of the packet. `None` when the rest of the packet should be considered as raw payload.

- The rest of packet.

**serialize**(*\_payload*, *\_prev*)

Encode a protocol header.

This method is used only when encoding a packet.

Encode a protocol header. Returns a bytearray which contains the header.

*payload* is the rest of the packet which will immediately follow this header.

*prev* is a `packet_base.PacketBase` subclass for the outer protocol header. *prev* is `None` if the current header is the outer-most. For example, *prev* is `ipv4` or `ipv6` for `tcp.serialize`.

## OSPF

RFC 2328 OSPF version 2

**exception** `os_ken.lib.packet.ospf.InvalidChecksum`

**class** `os_ken.lib.packet.ospf.OSPFDBDesc`(*length=None*, *router\_id='0.0.0.0'*,  
*area\_id='0.0.0.0'*, *au\_type=1*, *authentication=0*,  
*checksum=None*, *version=2*, *mtu=1500*,  
*options=0*, *i\_flag=0*, *m\_flag=0*, *ms\_flag=0*,  
*sequence\_number=0*, *lsa\_headers=None*)

**classmethod** `parser`(*buf*)

Decode a protocol header.

This method is used only when decoding a packet.

Decode a protocol header at offset 0 in bytearray *buf*. Returns the following three objects.

- An object to describe the decoded header.
- A `packet_base.PacketBase` subclass appropriate for the rest of the packet. `None` when the rest of the packet should be considered as raw payload.
- The rest of packet.

**class** `os_ken.lib.packet.ospf.OSPFHello`(*length=None*, *router\_id='0.0.0.0'*,  
*area\_id='0.0.0.0'*, *au\_type=1*, *authentication=0*,  
*checksum=None*, *version=2*, *mask='0.0.0.0'*,  
*hello\_interval=10*, *options=0*, *priority=1*,  
*dead\_interval=40*, *designated\_router='0.0.0.0'*,  
*backup\_router='0.0.0.0'*, *neighbors=None*)

**classmethod** `parser`(*buf*)

Decode a protocol header.

This method is used only when decoding a packet.

Decode a protocol header at offset 0 in bytearray *buf*. Returns the following three objects.

- An object to describe the decoded header.
- A `packet_base.PacketBase` subclass appropriate for the rest of the packet. `None` when the rest of the packet should be considered as raw payload.
- The rest of packet.

```
class os_ken.lib.packet.ospf.OSPFLSack(length=None, router_id='0.0.0.0',  
                                       area_id='0.0.0.0', au_type=1, authentication=0,  
                                       checksum=None, version=2, lsa_headers=None)
```

**classmethod parser**(*buf*)

Decode a protocol header.

This method is used only when decoding a packet.

Decode a protocol header at offset 0 in bytearray *buf*. Returns the following three objects.

- An object to describe the decoded header.
- A packet\_base.PacketBase subclass appropriate for the rest of the packet. None when the rest of the packet should be considered as raw payload.
- The rest of packet.

```
class os_ken.lib.packet.ospf.OSPFLSReq(length=None, router_id='0.0.0.0',  
                                       area_id='0.0.0.0', au_type=1, authentication=0,  
                                       checksum=None, version=2, lsa_requests=None)
```

**classmethod parser**(*buf*)

Decode a protocol header.

This method is used only when decoding a packet.

Decode a protocol header at offset 0 in bytearray *buf*. Returns the following three objects.

- An object to describe the decoded header.
- A packet\_base.PacketBase subclass appropriate for the rest of the packet. None when the rest of the packet should be considered as raw payload.
- The rest of packet.

```
class os_ken.lib.packet.ospf.OSPFLSupd(length=None, router_id='0.0.0.0',  
                                       area_id='0.0.0.0', au_type=1, authentication=0,  
                                       checksum=None, version=2, lsas=None)
```

**classmethod parser**(*buf*)

Decode a protocol header.

This method is used only when decoding a packet.

Decode a protocol header at offset 0 in bytearray *buf*. Returns the following three objects.

- An object to describe the decoded header.
- A packet\_base.PacketBase subclass appropriate for the rest of the packet. None when the rest of the packet should be considered as raw payload.
- The rest of packet.

```
class os_ken.lib.packet.ospf.OSPFMessage(type_, length=None, router_id='0.0.0.0',  
                                       area_id='0.0.0.0', au_type=1, authentication=0,  
                                       checksum=None, version=2)
```

Base class for OSPF version 2 messages.

**classmethod parser**(*buf*)

Decode a protocol header.

This method is used only when decoding a packet.

Decode a protocol header at offset 0 in bytearray *buf*. Returns the following three objects.

- An object to describe the decoded header.
- A `packet_base.PacketBase` subclass appropriate for the rest of the packet. None when the rest of the packet should be considered as raw payload.
- The rest of packet.

**serialize**(*payload=None, prev=None*)

Encode a protocol header.

This method is used only when encoding a packet.

Encode a protocol header. Returns a bytearray which contains the header.

*payload* is the rest of the packet which will immediately follow this header.

*prev* is a `packet_base.PacketBase` subclass for the outer protocol header. *prev* is None if the current header is the outer-most. For example, *prev* is `ipv4` or `ipv6` for `tcp.serialize`.

**os\_ken.lib.packet.ospf.ospf**

alias of `OSPFMessage`

**PBB****class** `os_ken.lib.packet.pbb.itag`(*pcp=0, dei=0, uca=0, sid=0*)

I-TAG (IEEE 802.1ah-2008) header encoder/decoder class.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
<code>pcp</code>	Priority Code Point
<code>dei</code>	Drop Eligible Indication
<code>uca</code>	Use Customer Address
<code>sid</code>	Service Instance ID

**classmethod parser**(*buf*)

Decode a protocol header.

This method is used only when decoding a packet.

Decode a protocol header at offset 0 in bytearray *buf*. Returns the following three objects.

- An object to describe the decoded header.
- A `packet_base.PacketBase` subclass appropriate for the rest of the packet. None when the rest of the packet should be considered as raw payload.
- The rest of packet.

**serialize**(*payload*, *prev*)

Encode a protocol header.

This method is used only when encoding a packet.

Encode a protocol header. Returns a bytearray which contains the header.

*payload* is the rest of the packet which will immediately follow this header.

*prev* is a `packet_base.PacketBase` subclass for the outer protocol header. *prev* is `None` if the current header is the outer-most. For example, *prev* is `ipv4` or `ipv6` for `tcp.serialize`.

## SCTP

**class** `os_ken.lib.packet.sctp.cause_cookie_while_shutdown`(*length=0*)

Stream Control Transmission Protocol (SCTP) sub encoder/decoder class for Cookie Received While Shutting Down (RFC 4960).

This class is used with the following.

- `os_ken.lib.packet.sctp.chunk_abort`
- `os_ken.lib.packet.sctp.chunk_error`

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
<code>length</code>	length of this cause containing this header. (0 means automatically-calculate when encoding)

**class** `os_ken.lib.packet.sctp.cause_invalid_param`(*length=0*)

Stream Control Transmission Protocol (SCTP) sub encoder/decoder class for Invalid Mandatory Parameter (RFC 4960).

This class is used with the following.

- `os_ken.lib.packet.sctp.chunk_abort`
- `os_ken.lib.packet.sctp.chunk_error`

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
<code>length</code>	length of this cause containing this header. (0 means automatically-calculate when encoding)

**class** `os_ken.lib.packet.sctp.cause_invalid_stream_id`(*value=0*, *length=0*)

Stream Control Transmission Protocol (SCTP) sub encoder/decoder class for Invalid Stream Identifier (RFC 4960).

This class is used with the following.

- `os_ken.lib.packet.sctp.chunk_abort`



- `os_ken.lib.packet.sctp.chunk_error`

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
<code>value</code>	stream id.
<code>length</code>	length of this cause containing this header. (0 means automatically-calculate when encoding)

**class** `os_ken.lib.packet.sctp.cause_missing_param`(*types=None, num=0, length=0*)

Stream Control Transmission Protocol (SCTP) sub encoder/decoder class for Missing Mandatory Parameter (RFC 4960).

This class is used with the following.

- `os_ken.lib.packet.sctp.chunk_abort`
- `os_ken.lib.packet.sctp.chunk_error`

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
<code>types</code>	a list of missing params.
<code>num</code>	Number of missing params. (0 means automatically-calculate when encoding)
<code>length</code>	length of this cause containing this header. (0 means automatically-calculate when encoding)

**class** `os_ken.lib.packet.sctp.cause_no_userdata`(*value=None, length=0*)

Stream Control Transmission Protocol (SCTP) sub encoder/decoder class for No User Data (RFC 4960).

This class is used with the following.

- `os_ken.lib.packet.sctp.chunk_abort`
- `os_ken.lib.packet.sctp.chunk_error`

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
<code>value</code>	the TSN of the DATA chunk received with no user data field.
<code>length</code>	length of this cause containing this header. (0 means automatically-calculate when encoding)

**class** `os_ken.lib.packet.sctp.cause_out_of_resource`(*length=0*)

Stream Control Transmission Protocol (SCTP) sub encoder/decoder class for Out of Resource (RFC 4960).

This class is used with the following.

- `os_ken.lib.packet.sctp.chunk_abort`
- `os_ken.lib.packet.sctp.chunk_error`

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
<code>length</code>	length of this cause containing this header. (0 means automatically-calculate when encoding)

**class** `os_ken.lib.packet.sctp.cause_protocol_violation`(*value=None, length=0*)

Stream Control Transmission Protocol (SCTP) sub encoder/decoder class for Protocol Violation (RFC 4960).

This class is used with the following.

- `os_ken.lib.packet.sctp.chunk_abort`
- `os_ken.lib.packet.sctp.chunk_error`

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
<code>value</code>	Additional Information.
<code>length</code>	length of this cause containing this header. (0 means automatically-calculate when encoding)

**class** `os_ken.lib.packet.sctp.cause_restart_with_new_addr`(*value=None, length=0*)

Stream Control Transmission Protocol (SCTP) sub encoder/decoder class for Restart of an Association with New Addresses (RFC 4960).

This class is used with the following.

- `os_ken.lib.packet.sctp.chunk_abort`
- `os_ken.lib.packet.sctp.chunk_error`

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
<code>value</code>	New Address TLVs.
<code>length</code>	length of this cause containing this header. (0 means automatically-calculate when encoding)

**class** `os_ken.lib.packet.sctp.cause_stale_cookie`(*value=None, length=0*)

Stream Control Transmission Protocol (SCTP) sub encoder/decoder class for Stale Cookie Error (RFC 4960).

This class is used with the following.

- `os_ken.lib.packet.sctp.chunk_abort`
- `os_ken.lib.packet.sctp.chunk_error`

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
value	Measure of Staleness.
length	length of this cause containing this header. (0 means automatically-calculate when encoding)

**class** `os_ken.lib.packet.sctp.cause_unrecognized_chunk`(*value=None, length=0*)

Stream Control Transmission Protocol (SCTP) sub encoder/decoder class for Unrecognized Chunk Type (RFC 4960).

This class is used with the following.

- `os_ken.lib.packet.sctp.chunk_abort`
- `os_ken.lib.packet.sctp.chunk_error`

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
value	Unrecognized Chunk.
length	length of this cause containing this header. (0 means automatically-calculate when encoding)

**class** `os_ken.lib.packet.sctp.cause_unrecognized_param`(*value=None, length=0*)

Stream Control Transmission Protocol (SCTP) sub encoder/decoder class for Unrecognized Parameters (RFC 4960).

This class is used with the following.

- `os_ken.lib.packet.sctp.chunk_abort`
- `os_ken.lib.packet.sctp.chunk_error`

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
value	Unrecognized Parameter.
length	length of this cause containing this header. (0 means automatically-calculate when encoding)

**class** `os_ken.lib.packet.sctp.cause_unresolvable_addr`(*value=None, length=0*)

Stream Control Transmission Protocol (SCTP) sub encoder/decoder class for Unresolvable Address (RFC 4960).

This class is used with the following.

- `os_ken.lib.packet.sctp.chunk_abort`
- `os_ken.lib.packet.sctp.chunk_error`

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
value	Unresolvable Address. one of follows: <code>os_ken.lib.packet.sctp.param_host_addr</code> , <code>os_ken.lib.packet.sctp.param_ipv4</code> , or <code>os_ken.lib.packet.sctp.param_ipv6</code> .
length	length of this cause containing this header. (0 means automatically-calculate when encoding)

**class** `os_ken.lib.packet.sctp.cause_user_initiated_abort` (*value=None, length=0*)

Stream Control Transmission Protocol (SCTP) sub encoder/decoder class for User-Initiated Abort (RFC 4960).

This class is used with the following.

- `os_ken.lib.packet.sctp.chunk_abort`
- `os_ken.lib.packet.sctp.chunk_error`

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
value	Upper Layer Abort Reason.
length	length of this cause containing this header. (0 means automatically-calculate when encoding)

**class** `os_ken.lib.packet.sctp.chunk_abort` (*tflag=0, length=0, causes=None*)

Stream Control Transmission Protocol (SCTP) sub encoder/decoder class for Abort Association (ABORT) chunk (RFC 4960).

This class is used with the following.

- `os_ken.lib.packet.sctp.sctp`

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
tflag	'0' means the Verification tag is normal. '1' means the Verification tag is copy of the sender.
length	length of this chunk containing this header. (0 means automatically-calculate when encoding)
causes	a list of derived classes of <code>os_ken.lib.packet.sctp.causes</code> .

**class** `os_ken.lib.packet.sctp.chunk_cookie_ack`(*flags=0, length=0*)

Stream Control Transmission Protocol (SCTP) sub encoder/decoder class for Cookie Acknowledgement (COOKIE ACK) chunk (RFC 4960).

This class is used with the following.

- `os_ken.lib.packet.sctp.sctp`

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
<code>flags</code>	set to '0'. this field will be ignored.
<code>length</code>	length of this chunk containing this header. (0 means automatically-calculate when encoding)

**class** `os_ken.lib.packet.sctp.chunk_cookie_echo`(*flags=0, length=0, cookie=None*)

Stream Control Transmission Protocol (SCTP) sub encoder/decoder class for Cookie Echo (COOKIE ECHO) chunk (RFC 4960).

This class is used with the following.

- `os_ken.lib.packet.sctp.sctp`

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
<code>flags</code>	set to '0'. this field will be ignored.
<code>length</code>	length of this chunk containing this header. (0 means automatically-calculate when encoding)
<code>cookie</code>	cookie data.

**class** `os_ken.lib.packet.sctp.chunk_cwr`(*flags=0, length=0, low\_tsn=0*)

Stream Control Transmission Protocol (SCTP) sub encoder/decoder class for CWR chunk (RFC 4960 Appendix A.).

This class is used with the following.

- `os_ken.lib.packet.sctp.sctp`

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
<code>flags</code>	set to '0'. this field will be ignored.
<code>length</code>	length of this chunk containing this header. (0 means automatically-calculate when encoding)
<code>low_tsn</code>	the lowest TSN.

```
class os_ken.lib.packet.sctp.chunk_data(unordered=0, begin=0, end=0, length=0, tsn=0,
sid=0, seq=0, payload_id=0,
payload_data=None)
```

Stream Control Transmission Protocol (SCTP) sub encoder/decoder class for Payload Data (DATA) chunk (RFC 4960).

This class is used with the following.

- os\_ken.lib.packet.sctp.sctp

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
unordered	if set to '1', the receiver ignores the sequence number.
begin	if set to '1', this chunk is the first fragment.
end	if set to '1', this chunk is the last fragment.
length	length of this chunk containing this header. (0 means automatically-calculate when encoding)
tsn	Transmission Sequence Number.
sid	stream id.
seq	the sequence number.
payload_id	application specified protocol id. '0' means that no application id is identified.
payload_data	user data.

```
class os_ken.lib.packet.sctp.chunk_ecn_echo(flags=0, length=0, low_tsn=0)
```

Stream Control Transmission Protocol (SCTP) sub encoder/decoder class for ECN-Echo chunk (RFC 4960 Appendix A.).

This class is used with the following.

- os\_ken.lib.packet.sctp.sctp

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
flags	set to '0'. this field will be ignored.
length	length of this chunk containing this header. (0 means automatically-calculate when encoding)
low_tsn	the lowest TSN.

```
class os_ken.lib.packet.sctp.chunk_error(flags=0, length=0, causes=None)
```

Stream Control Transmission Protocol (SCTP) sub encoder/decoder class for Operation Error (ERROR) chunk (RFC 4960).

This class is used with the following.

- os\_ken.lib.packet.sctp.sctp

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
flags	set to '0'. this field will be ignored.
length	length of this chunk containing this header. (0 means automatically-calculate when encoding)
causes	a list of derived classes of <code>os_ken.lib.packet.sctp.causes</code> .

**class** `os_ken.lib.packet.sctp.chunk_heartbeat`(*flags=0, length=0, info=None*)

Stream Control Transmission Protocol (SCTP) sub encoder/decoder class for Heartbeat Request (HEARTBEAT) chunk (RFC 4960).

This class is used with the following.

- `os_ken.lib.packet.sctp.sctp`

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
flags	set to '0'. this field will be ignored.
length	length of this chunk containing this header. (0 means automatically-calculate when encoding)
info	<code>os_ken.lib.packet.sctp.param_heartbeat</code> .

**class** `os_ken.lib.packet.sctp.chunk_heartbeat_ack`(*flags=0, length=0, info=None*)

Stream Control Transmission Protocol (SCTP) sub encoder/decoder class for Heartbeat Acknowledgement (HEARTBEAT ACK) chunk (RFC 4960).

This class is used with the following.

- `os_ken.lib.packet.sctp.sctp`

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
flags	set to '0'. this field will be ignored.
length	length of this chunk containing this header. (0 means automatically-calculate when encoding)
info	<code>os_ken.lib.packet.sctp.param_heartbeat</code> .

**class** `os_ken.lib.packet.sctp.chunk_init`(*flags=0, length=0, init\_tag=0, a\_rwnd=0, os=0, mis=0, i\_tsn=0, params=None*)

Stream Control Transmission Protocol (SCTP) sub encoder/decoder class for Initiation (INIT) chunk (RFC 4960).

This class is used with the following.

- `os_ken.lib.packet.sctp.sctp`

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
flags	set to '0'. this field will be ignored.
length	length of this chunk containing this header. (0 means automatically-calculate when encoding)
init_tag	the tag that be used as Verification Tag.
a_rwnd	Advertised Receiver Window Credit.
os	number of outbound streams.
mis	number of inbound streams.
i_tsn	Transmission Sequence Number that the sender will use.
params	Optional/Variable-Length Parameters. a list of derived classes of os_ken.lib.packet.sctp.param.

```
class os_ken.lib.packet.sctp.chunk_init_ack(flags=0, length=0, init_tag=0, a_rwnd=0,
os=0, mis=0, i_tsn=0, params=None)
```

Stream Control Transmission Protocol (SCTP) sub encoder/decoder class for Initiation Acknowledgement (INIT ACK) chunk (RFC 4960).

This class is used with the following.

- os\_ken.lib.packet.sctp.sctp

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
flags	set to '0'. this field will be ignored.
length	length of this chunk containing this header. (0 means automatically-calculate when encoding)
init_tag	the tag that be used as Verification Tag.
a_rwnd	Advertised Receiver Window Credit.
os	number of outbound streams.
mis	number of inbound streams.
i_tsn	Transmission Sequence Number that the sender will use.
params	Optional/Variable-Length Parameters. a list of derived classes of os_ken.lib.packet.sctp.param.

```
class os_ken.lib.packet.sctp.chunk_sack(flags=0, length=0, tsn_ack=0, a_rwnd=0,
gapack_num=0, duptsn_num=0, gapacks=None,
duptsns=None)
```

Stream Control Transmission Protocol (SCTP) sub encoder/decoder class for Selective Acknowledgement (SACK) chunk (RFC 4960).

This class is used with the following.

- os\_ken.lib.packet.sctp.sctp

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.



Attribute	Description
flags	set to '0'. this field will be ignored.
length	length of this chunk containing this header. (0 means automatically-calculate when encoding)
tsn_ack	TSN of the last DATA chunk received in sequence before a gap.
a_rwnd	Advertised Receiver Window Credit.
gapack_num	number of Gap Ack blocks.
dup_tsn_num	number of duplicate TSNs.
gapacks	a list of Gap Ack blocks. one block is made of a list with the start offset and the end offset from tsn_ack. e.g.) gapacks = [[2, 3], [10, 12], [19, 21]]
dup_tsns	a list of duplicate TSN.

**class** `os_ken.lib.packet.sctp.chunk_shutdown`(*flags=0, length=0, tsn\_ack=0*)

Stream Control Transmission Protocol (SCTP) sub encoder/decoder class for Shutdown Association (SHUTDOWN) chunk (RFC 4960).

This class is used with the following.

- `os_ken.lib.packet.sctp.sctp`

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
flags	set to '0'. this field will be ignored.
length	length of this chunk containing this header. (0 means automatically-calculate when encoding)
tsn_ack	TSN of the last DATA chunk received in sequence before a gap.

**class** `os_ken.lib.packet.sctp.chunk_shutdown_ack`(*flags=0, length=0*)

Stream Control Transmission Protocol (SCTP) sub encoder/decoder class for Shutdown Acknowledgement (SHUTDOWN ACK) chunk (RFC 4960).

This class is used with the following.

- `os_ken.lib.packet.sctp.sctp`

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
flags	set to '0'. this field will be ignored.
length	length of this chunk containing this header. (0 means automatically-calculate when encoding)

**class** `os_ken.lib.packet.sctp.chunk_shutdown_complete`(*tflag=0, length=0*)

Stream Control Transmission Protocol (SCTP) sub encoder/decoder class for Shutdown Complete (SHUTDOWN COMPLETE) chunk (RFC 4960).

This class is used with the following.

- `os_ken.lib.packet.sctp.sctp`

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
<code>tflag</code>	'0' means the Verification tag is normal. '1' means the Verification tag is copy of the sender.
<code>length</code>	length of this chunk containing this header. (0 means automatically-calculate when encoding)

**class** `os_ken.lib.packet.sctp.param_cookie_preserve`(*value=0, length=0*)

Stream Control Transmission Protocol (SCTP) sub encoder/decoder class for Cookie Preservative Parameter (RFC 4960).

This class is used with the following.

- `os_ken.lib.packet.sctp.chunk_init`

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
<code>value</code>	Suggested Cookie Life-Span Increment (msec).
<code>length</code>	length of this param containing this header. (0 means automatically-calculate when encoding)

**class** `os_ken.lib.packet.sctp.param_ecn`(*value=None, length=0*)

Stream Control Transmission Protocol (SCTP) sub encoder/decoder class for ECN Parameter (RFC 4960 Appendix A.).

This class is used with the following.

- `os_ken.lib.packet.sctp.chunk_init`
- `os_ken.lib.packet.sctp.chunk_init_ack`

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
<code>value</code>	set to None.
<code>length</code>	length of this param containing this header. (0 means automatically-calculate when encoding)

**class** `os_ken.lib.packet.sctp.param_heartbeat`(*value=None, length=0*)

Stream Control Transmission Protocol (SCTP) sub encoder/decoder class for Heartbeat Info Parameter (RFC 4960).

This class is used with the following.

- `os_ken.lib.packet.sctp.chunk_heartbeat`

- `os_ken.lib.packet.sctp.chunk_heartbeat_ack`

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
<code>value</code>	the sender-specific heartbeat information.
<code>length</code>	length of this param containing this header. (0 means automatically-calculate when encoding)

**class** `os_ken.lib.packet.sctp.param_host_addr`(*value=None, length=0*)

Stream Control Transmission Protocol (SCTP) sub encoder/decoder class for Host Name Address Parameter (RFC 4960).

This class is used with the following.

- `os_ken.lib.packet.sctp.chunk_init`
- `os_ken.lib.packet.sctp.chunk_init_ack`

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
<code>value</code>	a host name that ends with null terminator.
<code>length</code>	length of this param containing this header. (0 means automatically-calculate when encoding)

**class** `os_ken.lib.packet.sctp.param_ipv4`(*value='127.0.0.1', length=0*)

Stream Control Transmission Protocol (SCTP) sub encoder/decoder class for IPv4 Address Parameter (RFC 4960).

This class is used with the following.

- `os_ken.lib.packet.sctp.chunk_init`
- `os_ken.lib.packet.sctp.chunk_init_ack`

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
<code>value</code>	IPv4 address of the sending endpoint.
<code>length</code>	length of this param containing this header. (0 means automatically-calculate when encoding)

**class** `os_ken.lib.packet.sctp.param_ipv6`(*value='::1', length=0*)

Stream Control Transmission Protocol (SCTP) sub encoder/decoder class for IPv6 Address Parameter (RFC 4960).

This class is used with the following.

- `os_ken.lib.packet.sctp.chunk_init`
- `os_ken.lib.packet.sctp.chunk_init_ack`

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
<code>value</code>	IPv6 address of the sending endpoint.
<code>length</code>	length of this param containing this header. (0 means automatically-calculate when encoding)

**class** `os_ken.lib.packet.sctp.param_state_cookie`(*value=None, length=0*)

Stream Control Transmission Protocol (SCTP) sub encoder/decoder class for State Cookie Parameter (RFC 4960).

This class is used with the following.

- `os_ken.lib.packet.sctp.chunk_init_ack`

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
<code>value</code>	the state cookie. see Section 5.1.3 in RFC 4960.
<code>length</code>	length of this param containing this header. (0 means automatically-calculate when encoding)

**class** `os_ken.lib.packet.sctp.param_supported_addr`(*value=None, length=0*)

Stream Control Transmission Protocol (SCTP) sub encoder/decoder class for Supported Address Types Parameter (RFC 4960).

This class is used with the following.

- `os_ken.lib.packet.sctp.chunk_init`

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
<code>value</code>	a list of parameter types. odd cases pad with 0x0000.
<code>length</code>	length of this param containing this header. (0 means automatically-calculate when encoding)

**class** `os_ken.lib.packet.sctp.param_unrecognized_param`(*value=None, length=0*)

Stream Control Transmission Protocol (SCTP) sub encoder/decoder class for Unrecognized Parameter (RFC 4960).

This class is used with the following.

- `os_ken.lib.packet.sctp.chunk_init_ack`

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
value	the unrecognized parameter in the INIT chunk.
length	length of this param containing this header. (0 means automatically-calculate when encoding)

**class** `os_ken.lib.packet.sctp.sctp`(*src\_port=1, dst\_port=1, vtag=0, csum=0, chunks=None*)  
Stream Control Transmission Protocol (SCTP) header encoder/decoder class (RFC 4960).

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
src_port	Source Port
dst_port	Destination Port
vtag	Verification Tag
csum	Checksum (0 means automatically-calculate when encoding)
chunks	a list of derived classes of <code>os_ken.lib.packet.sctp.chunk</code> .

**classmethod** `parser`(*buf*)

Decode a protocol header.

This method is used only when decoding a packet.

Decode a protocol header at offset 0 in bytearray *buf*. Returns the following three objects.

- An object to describe the decoded header.
- A `packet_base.PacketBase` subclass appropriate for the rest of the packet. None when the rest of the packet should be considered as raw payload.
- The rest of packet.

**serialize**(*payload, prev*)

Encode a protocol header.

This method is used only when encoding a packet.

Encode a protocol header. Returns a bytearray which contains the header.

*payload* is the rest of the packet which will immediately follow this header.

*prev* is a `packet_base.PacketBase` subclass for the outer protocol header. *prev* is None if the current header is the outer-most. For example, *prev* is `ipv4` or `ipv6` for `tcp.serialize`.

**Slow**

```

class os_ken.lib.packet.slow.lacp(version=1, actor_system_priority=0,
    actor_system='00:00:00:00:00:00', actor_key=0,
    actor_port_priority=0, actor_port=0,
    actor_state_activity=0, actor_state_timeout=0,
    actor_state_aggregation=0,
    actor_state_synchronization=0, actor_state_collecting=0,
    actor_state_distributing=0, actor_state_defaulted=0,
    actor_state_expired=0, partner_system_priority=0,
    partner_system='00:00:00:00:00:00', partner_key=0,
    partner_port_priority=0, partner_port=0,
    partner_state_activity=0, partner_state_timeout=0,
    partner_state_aggregation=0,
    partner_state_synchronization=0,
    partner_state_collecting=0, partner_state_distributing=0,
    partner_state_defaulted=0, partner_state_expired=0,
    collector_max_delay=0)

```

Link Aggregation Control Protocol(LACP, IEEE 802.1AX) header encoder/decoder class.

<http://standards.ieee.org/getieee802/download/802.1AX-2008.pdf>

LACPDU format

LACPDU structure		Octets
Subtype = LACP		1
Version Number		1
TLV Actor	TLV_type = Actor Information	1
	Actor_Information_Length = 20	1
	Actor_System_Priority	2
	Actor_System	6
	Actor_Key	2
	Actor_Port_Priority	2
	Actor_Port	2
	Actor_State	1
	Reserved	3
	TLV Partner	TLV_type = Partner Information
Partner_Information_Length = 20		1
Partner_System_Priority		2
Partner_System		6
Partner_Key		2
Partner_Port_Priority		2
Partner_Port		2
Partner_State		1
Reserved		3
TLV Collector	TLV_type = Collector Information	1
	Collector_Information_Length = 16	1
	Collector_Max_Delay	2
	Reserved	12
TLV Terminator	TLV_type = Terminator	1
	Terminator_Length = 0	1
	Reserved	50

Terminator information uses a length value of 0 (0x00).

**NOTE--The use of a Terminator\_Length of 0 is intentional.**

In TLV encoding schemes it is common practice for the terminator encoding to be 0 both for the type and the length.

Actor\_State and Partner\_State encoded as individual bits within a single octet as follows:

7	6	5	4	3	2	1	0
EXPR	DFLT	DIST	CLCT	SYNC	AGGR	TMO	ACT

**ACT**

bit 0. about the activity control value with regard to this link.

**TMO**

bit 1. about the timeout control value with regard to this link.

**AGGR**

bit 2. about how the system regards this link from the point of view of the aggregation.

**SYNC**

bit 3. about how the system regards this link from the point of view of the synchronization.

**CLCT**

bit 4. about collecting of incoming frames.

**DIST**

bit 5. about distributing of outgoing frames.

**DFLT**

bit 6. about the opposite system information which the system use.

**EXPR**

bit 7. about the expire state of the system.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
version	LACP version. This parameter must be set to LACP_VERSION_NUMBER(i.e. 1).
actor_system_priority	The priority assigned to this System.
actor_system	The Actor's System ID, encoded as a MAC address.
actor_key	The operational Key value assigned to the port by the Actor.
actor_port_priority	The priority assigned to this port.
actor_port	The port number assigned to the port by the Actor.
actor_state_activity	about the activity control value with regard to this link. LACP_STATE_ACTIVE(1) LACP_STATE_PASSIVE(0)
actor_state_timeout	about the timeout control value with regard to this link. LACP_STATE_SHORT_TIMEOUT(1) LACP_STATE_LONG_TIMEOUT(0)
actor_state_aggregation	about how the system regards this link from the point of view of the aggregation. LACP_STATE_AGGREGATEABLE(1) LACP_STATE_INDIVIDUAL(0)
actor_state_synchronization	about how the system regards this link from the point of view of the synchronization. LACP_STATE_IN_SYNC(1) LACP_STATE_OUT_OF_SYNC(0)
actor_state_collecting	about collecting of incoming frames. LACP_STATE_COLLECTING_ENABLED(1) LACP_STATE_COLLECTING_DISABLED(0)
actor_state_distributing	about distributing of outgoing frames. LACP_STATE_DISTRIBUTING_ENABLED(1) LACP_STATE_DISTRIBUTING_DISABLED(0)
actor_state_defaulted	about the Partner information which the the Actor use. LACP_STATE_DEFAULTED_PARTNER(1) LACP_STATE_OPERATIONAL_PARTNER(0)
actor_state_expired	about the state of the Actor. LACP_STATE_EXPIRED(1) LACP_STATE_NOT_EXPIRED(0)
partner_system_priority	The priority assigned to the Partner System.
partner_system	The Partner's System ID, encoded as a MAC address.
partner_key	The operational Key value assigned to the port by the Partner.
partner_port_priority	The priority assigned to this port by the Partner.
partner_port	The port number assigned to the port by the Partner.
partner_state_activity	See <i>actor_state_activity</i> .
partner_state_timeout	See <i>actor_state_timeout</i> .
partner_state_aggregation	See <i>actor_state_aggregation</i> .
partner_state_synchronization	See <i>actor_state_synchronization</i> .
partner_state_collecting	See <i>actor_state_collecting</i> .
partner_state_distributing	See <i>actor_state_distributing</i> .
partner_state_defaulted	See <i>actor_state_defaulted</i> .
partner_state_expired	See <i>actor_state_expired</i> .



**classmethod parser**(*buf*)

Decode a protocol header.

This method is used only when decoding a packet.

Decode a protocol header at offset 0 in bytearray *buf*. Returns the following three objects.

- An object to describe the decoded header.
- A `packet_base.PacketBase` subclass appropriate for the rest of the packet. None when the rest of the packet should be considered as raw payload.
- The rest of packet.

**serialize**(*payload*, *prev*)

Encode a protocol header.

This method is used only when encoding a packet.

Encode a protocol header. Returns a bytearray which contains the header.

*payload* is the rest of the packet which will immediately follow this header.

*prev* is a `packet_base.PacketBase` subclass for the outer protocol header. *prev* is None if the current header is the outer-most. For example, *prev* is `ipv4` or `ipv6` for `tcp.serialize`.

**class** `os_ken.lib.packet.slow.slow`

Slow Protocol header decoder class. This class has only the `parser` method.

[http://standards.ieee.org/getieee802/download/802.3-2012\\_section5.pdf](http://standards.ieee.org/getieee802/download/802.3-2012_section5.pdf)

Slow Protocols Subtypes

Subtype Value	Protocol Name
0	Unused - Illegal Value
1	Link Aggregation Control Protocol(LACP)
2	Link Aggregation - Marker Protocol
3	Operations, Administration, and Maintenance(OAM)
4 - 9	Reserved for future use
10	Organization Specific Slow Protocol(OSSP)
11 - 255	Unused - Illegal values

**classmethod parser**(*buf*)

Decode a protocol header.

This method is used only when decoding a packet.

Decode a protocol header at offset 0 in bytearray *buf*. Returns the following three objects.

- An object to describe the decoded header.
- A `packet_base.PacketBase` subclass appropriate for the rest of the packet. None when the rest of the packet should be considered as raw payload.
- The rest of packet.

## TCP

**class** `os_ken.lib.packet.tcp.tcp`(*src\_port=1, dst\_port=1, seq=0, ack=0, offset=0, bits=0, window\_size=0, csum=0, urgent=0, option=None*)

TCP (RFC 793) header encoder/decoder class.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
<code>src_port</code>	Source Port
<code>dst_port</code>	Destination Port
<code>seq</code>	Sequence Number
<code>ack</code>	Acknowledgement Number
<code>offset</code>	Data Offset (0 means automatically-calculate when encoding)
<code>bits</code>	Control Bits
<code>window_size</code>	Window
<code>csum</code>	Checksum (0 means automatically-calculate when encoding)
<code>urgent</code>	Urgent Pointer
<code>option</code>	List of <code>TCPOption</code> sub-classes or an bytearray containing options. None if no options.

**has\_flags**(*\*flags*)

Check if flags are set on this packet.

returns boolean if all passed flags is set

Example:

```
>>> pkt = tcp.tcp(bits=(tcp.TCP_SYN | tcp.TCP_ACK))
>>> pkt.has_flags(tcp.TCP_SYN, tcp.TCP_ACK)
True
```

**classmethod parser**(*buf*)

Decode a protocol header.

This method is used only when decoding a packet.

Decode a protocol header at offset 0 in bytearray *buf*. Returns the following three objects.

- An object to describe the decoded header.
- A `packet_base.PacketBase` subclass appropriate for the rest of the packet. None when the rest of the packet should be considered as raw payload.
- The rest of packet.

**serialize**(*payload, prev*)

Encode a protocol header.

This method is used only when encoding a packet.

Encode a protocol header. Returns a bytearray which contains the header.

*payload* is the rest of the packet which will immediately follow this header.

*prev* is a `packet_base.PacketBase` subclass for the outer protocol header. *prev* is `None` if the current header is the outer-most. For example, *prev* is `ipv4` or `ipv6` for `tcp.serialize`.

## UDP

**class** `os_ken.lib.packet.udp.udp`(*src\_port=1, dst\_port=1, total\_length=0, csum=0*)

UDP (RFC 768) header encoder/decoder class.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
<code>src_port</code>	Source Port
<code>dst_port</code>	Destination Port
<code>total_length</code>	Length (0 means automatically-calculate when encoding)
<code>csum</code>	Checksum (0 means automatically-calculate when encoding)

**static** `get_packet_type`(*src\_port, dst\_port*)

Per-protocol dict-like get method.

Provided for convenience of protocol implementers. Internal use only.

**classmethod** `parser`(*buf*)

Decode a protocol header.

This method is used only when decoding a packet.

Decode a protocol header at offset 0 in bytearray *buf*. Returns the following three objects.

- An object to describe the decoded header.
- A `packet_base.PacketBase` subclass appropriate for the rest of the packet. `None` when the rest of the packet should be considered as raw payload.
- The rest of packet.

**serialize**(*payload, prev*)

Encode a protocol header.

This method is used only when encoding a packet.

Encode a protocol header. Returns a bytearray which contains the header.

*payload* is the rest of the packet which will immediately follow this header.

*prev* is a `packet_base.PacketBase` subclass for the outer protocol header. *prev* is `None` if the current header is the outer-most. For example, *prev* is `ipv4` or `ipv6` for `tcp.serialize`.

## VLAN

**class** `os_ken.lib.packet.vlan.svlan`(*pcp=0, cfi=0, vid=0, ethertype=33024*)

S-VLAN (IEEE 802.1ad) header encoder/decoder class.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
pcp	Priority Code Point
cfi	Canonical Format Indicator. In a case to be used as B-TAG, this field means DEI(Drop Eligible Indication).
vid	VLAN Identifier
ethertype	EtherType

**classmethod** `get_packet_type(type_)`

Per-protocol dict-like get method.

Provided for convenience of protocol implementers. Internal use only.

**class** `os_ken.lib.packet.vlan.vlan(pcp=0, cfi=0, vid=0, ethertype=2048)`

VLAN (IEEE 802.1Q) header encoder/decoder class.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
pcp	Priority Code Point
cfi	Canonical Format Indicator
vid	VLAN Identifier
ethertype	EtherType

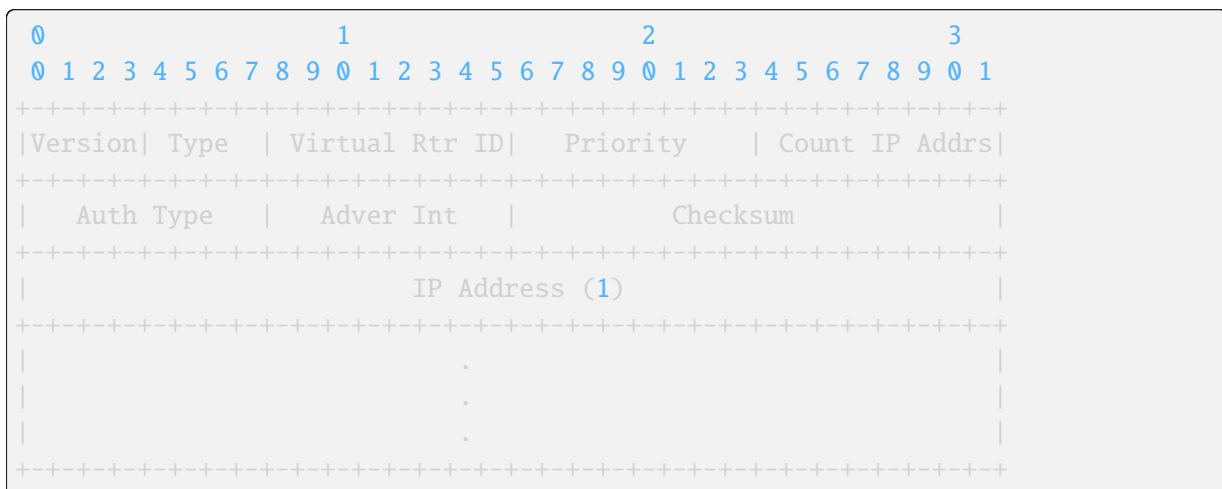
**classmethod** `get_packet_type(type_)`

Override method for the Length/Type field (`self.ethertype`). The Length/Type field means Length or Type interpretation, same as ethernet IEEE802.3. If the value of Length/Type field is less than or equal to 1500 decimal(05DC hexadecimal), it means Length interpretation and be passed to the LLC sublayer.

## VRRP

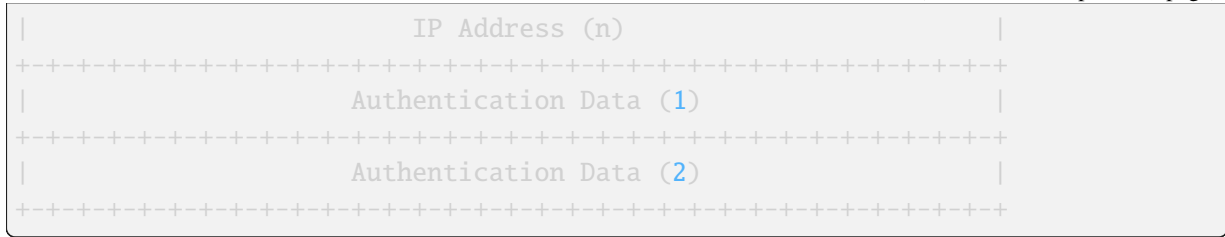
VRRP packet parser/serializer

[RFC 3768] VRRP v2 packet format:



(continues on next page)

(continued from previous page)



[RFC 5798] VRRP v3 packet format:



```

class os_ken.lib.packet.vrrp.vrrp(version, type_, vrid, priority, count_ip, max_adver_int,
    checksum, ip_addresses, auth_type=None,
    auth_data=None)
  
```

The base class for VRRPv2 (RFC 3768) and VRRPv3 (RFC 5798) header encoder/decoder classes. Unlike other `os_ken.lib.packet.packet_base.PacketBase` derived classes, This class should not be directly instantiated by user. An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order.

Attribute	Description
version	Version
type	Type
vrid	Virtual Rtr ID (VRID)
priority	Priority
count_ip	Count IPvX Addr. Calculated automatically when encoding.
max_adver_int	Maximum Advertisement Interval (Max Adver Int)
checksum	Checksum. Calculated automatically when encoding.
ip_addresses	IPvX Address(es). A python list of IP addresses.
auth_type	Authentication Type (only for VRRPv2)
auth_data	Authentication Data (only for VRRPv2)

**create\_packet**(*primary\_ip\_address*, *vlan\_id=None*)

Prepare a VRRP packet.

Returns a newly created `os_ken.lib.packet.packet.Packet` object with appropriate protocol header objects added by `add_protocol()`. It's caller's responsibility to `serialize()`. The serialized packet would look like the ones described in the following sections.

- RFC 3768 5.1. VRRP Packet Format
- RFC 5798 5.1. VRRP Packet Format

Argument	Description
<code>primary_ip_address</code>	Source IP address
<code>vlan_id</code>	VLAN ID. None for no VLAN.

**classmethod parser**(*buf*)

Decode a protocol header.

This method is used only when decoding a packet.

Decode a protocol header at offset 0 in bytearray *buf*. Returns the following three objects.

- An object to describe the decoded header.
- A `packet_base.PacketBase` subclass appropriate for the rest of the packet. None when the rest of the packet should be considered as raw payload.
- The rest of packet.

**serialize**(*payload*, *prev*)

Encode a protocol header.

This method is used only when encoding a packet.

Encode a protocol header. Returns a bytearray which contains the header.

*payload* is the rest of the packet which will immediately follow this header.

*prev* is a `packet_base.PacketBase` subclass for the outer protocol header. *prev* is None if the current header is the outer-most. For example, *prev* is `ipv4` or `ipv6` for `tcp.serialize()`.

```
class os_ken.lib.packet.vrrp.vrrpv2(version, type_, vrid, priority, count_ip, max_adver_int,  
                                     checksum, ip_addresses, auth_type=None,  
                                     auth_data=None)
```

VRRPv2 (RFC 3768) header encoder/decoder class.

Unlike other `os_ken.lib.packet.packet_base.PacketBase` derived classes, `create` method should be used to instantiate an object of this class.

```
static create(type_, vrid, priority, max_adver_int, ip_addresses)
```

Unlike other `os_ken.lib.packet.packet_base.PacketBase` derived classes, this method should be used to instantiate an object of this class.

This method's arguments are same as `os_ken.lib.packet.vrrp.vrrp` object's attributes of the same name. (except that `type_` corresponds to `type` attribute.)

```
classmethod parser(buf)
```

Decode a protocol header.

This method is used only when decoding a packet.

Decode a protocol header at offset 0 in bytearray `buf`. Returns the following three objects.

- An object to describe the decoded header.
- A `packet_base.PacketBase` subclass appropriate for the rest of the packet. None when the rest of the packet should be considered as raw payload.
- The rest of packet.

```
class os_ken.lib.packet.vrrp.vrrpv3(version, type_, vrid, priority, count_ip, max_adver_int,  
                                     checksum, ip_addresses, auth_type=None,  
                                     auth_data=None)
```

VRRPv3 (RFC 5798) header encoder/decoder class.

Unlike other `os_ken.lib.packet.packet_base.PacketBase` derived classes, `create` method should be used to instantiate an object of this class.

```
static create(type_, vrid, priority, max_adver_int, ip_addresses)
```

Unlike other `os_ken.lib.packet.packet_base.PacketBase` derived classes, this method should be used to instantiate an object of this class.

This method's arguments are same as `os_ken.lib.packet.vrrp.vrrp` object's attributes of the same name. (except that `type_` corresponds to `type` attribute.)

```
classmethod parser(buf)
```

Decode a protocol header.

This method is used only when decoding a packet.

Decode a protocol header at offset 0 in bytearray `buf`. Returns the following three objects.

- An object to describe the decoded header.
- A `packet_base.PacketBase` subclass appropriate for the rest of the packet. None when the rest of the packet should be considered as raw payload.
- The rest of packet.

## VXLAN

`os_ken.lib.packet.vxlan.vni_from_bin(buf)`

Converts binary representation VNI to integer.

**Parameters**

**buf** -- binary representation of VNI.

**Returns**

VNI integer.

`os_ken.lib.packet.vxlan.vni_to_bin(vni)`

Converts integer VNI to binary representation.

**Parameters**

**vni** -- integer of VNI

**Returns**

binary representation of VNI.

**class** `os_ken.lib.packet.vxlan.vxlan(vni)`

VXLAN (RFC 7348) header encoder/decoder class.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
<code>vni</code>	VXLAN Network Identifier

**classmethod** `parser(buf)`

Decode a protocol header.

This method is used only when decoding a packet.

Decode a protocol header at offset 0 in bytearray *buf*. Returns the following three objects.

- An object to describe the decoded header.
- A `packet_base.PacketBase` subclass appropriate for the rest of the packet. None when the rest of the packet should be considered as raw payload.
- The rest of packet.

**serialize(payload, prev)**

Encode a protocol header.

This method is used only when encoding a packet.

Encode a protocol header. Returns a bytearray which contains the header.

*payload* is the rest of the packet which will immediately follow this header.

*prev* is a `packet_base.PacketBase` subclass for the outer protocol header. *prev* is None if the current header is the outer-most. For example, *prev* is `ipv4` or `ipv6` for `tcp.serialize`.



## Zebra

Zebra protocol parser/serializer

Zebra Protocol is used to communicate with the zebra daemon.

```
class os_ken.lib.packet.zebra.InterfaceLinkParams(lp_status, te_metric, max_bw,
                                                    max_reserved_bw, unreserved_bw,
                                                    admin_group, remote_as, remote_ip,
                                                    average_delay, min_delay,
                                                    max_delay, delay_var, pkt_loss,
                                                    residual_bw, average_bw,
                                                    utilized_bw)
```

Interface Link Parameters class for if\_link\_params structure.

```
class os_ken.lib.packet.zebra.NextHopBlackhole(ifindex=None, ifname=None,
                                                addr=None, type_=None)
```

Nexthop class for ZEBRA\_NEXTHOP\_BLACKHOLE type.

```
class os_ken.lib.packet.zebra.NextHopIFIndex(ifindex=None, ifname=None, addr=None,
                                              type_=None)
```

Nexthop class for ZEBRA\_NEXTHOP\_IFINDEX type.

```
class os_ken.lib.packet.zebra.NextHopIFName(ifindex=None, ifname=None, addr=None,
                                              type_=None)
```

Nexthop class for ZEBRA\_NEXTHOP\_IFNAME type.

```
class os_ken.lib.packet.zebra.NextHopIPv4(ifindex=None, ifname=None, addr=None,
                                           type_=None)
```

Nexthop class for ZEBRA\_NEXTHOP\_IPV4 type.

```
class os_ken.lib.packet.zebra.NextHopIPv4IFIndex(ifindex=None, ifname=None,
                                                  addr=None, type_=None)
```

Nexthop class for ZEBRA\_NEXTHOP\_IPV4\_IFINDEX type.

```
class os_ken.lib.packet.zebra.NextHopIPv4IFName(ifindex=None, ifname=None,
                                                  addr=None, type_=None)
```

Nexthop class for ZEBRA\_NEXTHOP\_IPV4\_IFNAME type.

```
class os_ken.lib.packet.zebra.NextHopIPv6(ifindex=None, ifname=None, addr=None,
                                           type_=None)
```

Nexthop class for ZEBRA\_NEXTHOP\_IPV6 type.

```
class os_ken.lib.packet.zebra.NextHopIPv6IFIndex(ifindex=None, ifname=None,
                                                  addr=None, type_=None)
```

Nexthop class for ZEBRA\_NEXTHOP\_IPV6\_IFINDEX type.

```
class os_ken.lib.packet.zebra.NextHopIPv6IFName(ifindex=None, ifname=None,
                                                  addr=None, type_=None)
```

Nexthop class for ZEBRA\_NEXTHOP\_IPV6\_IFNAME type.

```
class os_ken.lib.packet.zebra.RegisteredNexthop(connected, family, prefix)
```

Unit of ZEBRA\_NEXTHOP\_REGISTER message body.

**class** `os_ken.lib.packet.zebra.ZebraBfdClientRegister`(*pid*)

Message body class for FRR\_ZEBRA\_BFD\_CLIENT\_REGISTER.

**class** `os_ken.lib.packet.zebra.ZebraBfdDestinationDeregister`(*pid*, *dst\_family*,  
*dst\_prefix*, *multi\_hop*,  
*src\_family*, *src\_prefix*,  
*multi\_hop\_count=None*,  
*ifname=None*)

Message body class for FRR\_ZEBRA\_BFD\_DEST\_DEREGISTER.

**class** `os_ken.lib.packet.zebra.ZebraBfdDestinationRegister`(*pid*, *dst\_family*, *dst\_prefix*,  
*min\_rx\_timer*,  
*min\_tx\_timer*, *detect\_mult*,  
*multi\_hop*, *src\_family*,  
*src\_prefix*,  
*multi\_hop\_count=None*,  
*ifname=None*)

Message body class for FRR\_ZEBRA\_BFD\_DEST\_REGISTER.

**class** `os_ken.lib.packet.zebra.ZebraBfdDestinationReply`

Message body class for FRR\_ZEBRA\_BFD\_DEST\_REPLAY.

**class** `os_ken.lib.packet.zebra.ZebraBfdDestinationUpdate`(*pid*, *dst\_family*, *dst\_prefix*,  
*min\_rx\_timer*, *min\_tx\_timer*,  
*detect\_mult*, *multi\_hop*,  
*src\_family*, *src\_prefix*,  
*multi\_hop\_count=None*,  
*ifname=None*)

Message body class for FRR\_ZEBRA\_BFD\_DEST\_UPDATE.

**class** `os_ken.lib.packet.zebra.ZebraHello`(*route\_type*, *instance=None*)

Message body class for ZEBRA\_HELLO.

**class** `os_ken.lib.packet.zebra.ZebraIPv4ImportLookup`(*prefix*, *metric=None*,  
*nexthops=None*,  
*from\_zebra=False*)

Message body class for ZEBRA\_IPV4\_IMPORT\_LOOKUP.

**class** `os_ken.lib.packet.zebra.ZebraIPv4NexthopAdd`(*route\_type*, *flags*, *message*,  
*safi=None*, *prefix=None*,  
*src\_prefix=None*, *nexthops=None*,  
*ifindexes=None*, *distance=None*,  
*metric=None*, *mtu=None*, *tag=None*,  
*instance=None*, *from\_zebra=False*)

Message body class for FRR\_ZEBRA\_IPV4\_NEXTHOP\_ADD.

**class** `os_ken.lib.packet.zebra.ZebraIPv4NexthopDelete`(*route\_type*, *flags*, *message*,  
*safi=None*, *prefix=None*,  
*src\_prefix=None*,  
*nexthops=None*, *ifindexes=None*,  
*distance=None*, *metric=None*,  
*mtu=None*, *tag=None*,  
*instance=None*,  
*from\_zebra=False*)

Message body class for FRR\_ZEBRA\_IPV4\_NEXTHOP\_DELETE.

```
class os_ken.lib.packet.zebra.ZebraIPv4NexthopLookup(addr, metric=None,  
                                                    nexthops=None)
```

Message body class for ZEBRA\_IPV4\_NEXTHOP\_LOOKUP.

```
class os_ken.lib.packet.zebra.ZebraIPv4NexthopLookupMRib(addr, distance=None,  
                                                         metric=None,  
                                                         nexthops=None)
```

Message body class for ZEBRA\_IPV4\_NEXTHOP\_LOOKUP\_MRIB.

```
class os_ken.lib.packet.zebra.ZebraIPv4RouteAdd(route_type, flags, message, safi=None,  
                                                prefix=None, src_prefix=None,  
                                                nexthops=None, ifindexes=None,  
                                                distance=None, metric=None,  
                                                mtu=None, tag=None, instance=None,  
                                                from_zebra=False)
```

Message body class for ZEBRA\_IPV4\_ROUTE\_ADD.

```
class os_ken.lib.packet.zebra.ZebraIPv4RouteDelete(route_type, flags, message,  
                                                  safi=None, prefix=None,  
                                                  src_prefix=None, nexthops=None,  
                                                  ifindexes=None, distance=None,  
                                                  metric=None, mtu=None,  
                                                  tag=None, instance=None,  
                                                  from_zebra=False)
```

Message body class for ZEBRA\_IPV4\_ROUTE\_DELETE.

```
class os_ken.lib.packet.zebra.ZebraIPv4RouteIPv6NexthopAdd(route_type, flags,  
                                                           message, safi=None,  
                                                           prefix=None,  
                                                           src_prefix=None,  
                                                           nexthops=None,  
                                                           ifindexes=None,  
                                                           distance=None,  
                                                           metric=None, mtu=None,  
                                                           tag=None,  
                                                           instance=None,  
                                                           from_zebra=False)
```

Message body class for FRR\_ZEBRA\_IPV4\_ROUTE\_IPV6\_NEXTHOP\_ADD.

```
class os_ken.lib.packet.zebra.ZebraIPv6ImportLookup(prefix, metric=None,  
                                                    nexthops=None,  
                                                    from_zebra=False)
```

Message body class for ZEBRA\_IPV6\_IMPORT\_LOOKUP.

```
class os_ken.lib.packet.zebra.ZebraIPv6NexthopAdd(route_type, flags, message,  
                                                  safi=None, prefix=None,  
                                                  src_prefix=None, nexthops=None,  
                                                  ifindexes=None, distance=None,  
                                                  metric=None, mtu=None, tag=None,  
                                                  instance=None, from_zebra=False)
```

Message body class for FRR\_ZEBRA\_IPV6\_NEXTHOP\_ADD.

```
class os_ken.lib.packet.zebra.ZebraIPv6NexthopDelete(route_type, flags, message,  
                                                    safi=None, prefix=None,  
                                                    src_prefix=None,  
                                                    nexthops=None, ifindexes=None,  
                                                    distance=None, metric=None,  
                                                    mtu=None, tag=None,  
                                                    instance=None,  
                                                    from_zebra=False)
```

Message body class for FRR\_ZEBRA\_IPV6\_NEXTHOP\_DELETE.

```
class os_ken.lib.packet.zebra.ZebraIPv6NexthopLookup(addr, metric=None,  
                                                    nexthops=None)
```

Message body class for ZEBRA\_IPV6\_NEXTHOP\_LOOKUP.

```
class os_ken.lib.packet.zebra.ZebraIPv6RouteAdd(route_type, flags, message, safi=None,  
                                                prefix=None, src_prefix=None,  
                                                nexthops=None, ifindexes=None,  
                                                distance=None, metric=None,  
                                                mtu=None, tag=None, instance=None,  
                                                from_zebra=False)
```

Message body class for ZEBRA\_IPV6\_ROUTE\_ADD.

```
class os_ken.lib.packet.zebra.ZebraIPv6RouteDelete(route_type, flags, message,  
                                                    safi=None, prefix=None,  
                                                    src_prefix=None, nexthops=None,  
                                                    ifindexes=None, distance=None,  
                                                    metric=None, mtu=None,  
                                                    tag=None, instance=None,  
                                                    from_zebra=False)
```

Message body class for ZEBRA\_IPV6\_ROUTE\_DELETE.

```
class os_ken.lib.packet.zebra.ZebraImportCheckUpdate(family, prefix, distance=None,  
                                                    metric=None, nexthops=None)
```

Message body class for FRR\_ZEBRA\_IMPORT\_CHECK\_UPDATE.

```
class os_ken.lib.packet.zebra.ZebraImportRouteRegister(nexthops)
```

Message body class for FRR\_ZEBRA\_IMPORT\_ROUTE\_REGISTER.

```
class os_ken.lib.packet.zebra.ZebraImportRouteUnregister(nexthops)
```

Message body class for FRR\_ZEBRA\_IMPORT\_ROUTE\_UNREGISTER.

```
class os_ken.lib.packet.zebra.ZebraInterfaceAdd(ifname=None, ifindex=None,  
                                                status=None, if_flags=None,  
                                                ptm_enable=None, ptm_status=None,  
                                                metric=None, speed=None,  
                                                ifmtu=None, ifmtu6=None,  
                                                bandwidth=None, ll_type=None,  
                                                hw_addr_len=0, hw_addr=None,  
                                                link_params=None)
```

Message body class for ZEBRA\_INTERFACE\_ADD.

```
class os_ken.lib.packet.zebra.ZebraInterfaceAddressAdd(ifindex, ifc_flags, family,  
prefix, dest)
```

Message body class for ZEBRA\_INTERFACE\_ADDRESS\_ADD.

```
class os_ken.lib.packet.zebra.ZebraInterfaceAddressDelete(ifindex, ifc_flags, family,  
prefix, dest)
```

Message body class for ZEBRA\_INTERFACE\_ADDRESS\_DELETE.

```
class os_ken.lib.packet.zebra.ZebraInterfaceBfdDestinationUpdate(ifindex,  
dst_family,  
dst_prefix, status,  
src_family,  
src_prefix)
```

Message body class for FRR\_ZEBRA\_INTERFACE\_BFD\_DEST\_UPDATE.

```
class os_ken.lib.packet.zebra.ZebraInterfaceDelete(ifname=None, ifindex=None,  
status=None, if_flags=None,  
ptm_enable=None,  
ptm_status=None, metric=None,  
speed=None, ifmtu=None,  
ifmtu6=None, bandwidth=None,  
ll_type=None, hw_addr_len=0,  
hw_addr=None,  
link_params=None)
```

Message body class for ZEBRA\_INTERFACE\_DELETE.

```
class os_ken.lib.packet.zebra.ZebraInterfaceDisableRadv(ifindex, interval)
```

Message body class for FRR\_ZEBRA\_INTERFACE\_DISABLE\_RADV.

```
class os_ken.lib.packet.zebra.ZebraInterfaceDown(ifname=None, ifindex=None,  
status=None, if_flags=None,  
ptm_enable=None, ptm_status=None,  
metric=None, speed=None,  
ifmtu=None, ifmtu6=None,  
bandwidth=None, ll_type=None,  
hw_addr_len=0, hw_addr=None,  
link_params=None)
```

Message body class for ZEBRA\_INTERFACE\_DOWN.

```
class os_ken.lib.packet.zebra.ZebraInterfaceEnableRadv(ifindex, interval)
```

Message body class for FRR\_ZEBRA\_INTERFACE\_ENABLE\_RADV.

```
class os_ken.lib.packet.zebra.ZebraInterfaceLinkParams(ifindex, link_params)
```

Message body class for ZEBRA\_INTERFACE\_LINK\_PARAMS.

```
class os_ken.lib.packet.zebra.ZebraInterfaceNbrAddressAdd(ifindex, family, prefix)
```

Message body class for FRR\_ZEBRA\_INTERFACE\_NBR\_ADDRESS\_ADD.

```
class os_ken.lib.packet.zebra.ZebraInterfaceNbrAddressDelete(ifindex, family, prefix)
```

Message body class for FRR\_ZEBRA\_INTERFACE\_NBR\_ADDRESS\_DELETE.

```
class os_ken.lib.packet.zebra.ZebraInterfaceUp(ifname=None, ifindex=None,
                                             status=None, if_flags=None,
                                             ptm_enable=None, ptm_status=None,
                                             metric=None, speed=None, ifmtu=None,
                                             ifmtu6=None, bandwidth=None,
                                             ll_type=None, hw_addr_len=0,
                                             hw_addr=None, link_params=None)
```

Message body class for ZEBRA\_INTERFACE\_UP.

```
class os_ken.lib.packet.zebra.ZebraInterfaceVrfUpdate(ifindex, vrf_id)
```

Message body class for FRR\_ZEBRA\_INTERFACE\_VRF\_UPDATE.

```
class os_ken.lib.packet.zebra.ZebraMessage(length=None, version=3, vrf_id=0,
                                           command=None, body=None)
```

Zebra protocol parser/serializer class.

An instance has the following attributes at least. Most of them are same to the on-wire counterparts but in host byte order. `__init__` takes the corresponding args in this order.

Attribute	Description
<code>length</code>	Total packet length including this header. The minimum length is 3 bytes for version 0 messages, 6 bytes for version 1/2 messages and 8 bytes for version 3 messages.
<code>version</code>	Version number of the Zebra protocol message. To instantiate messages with other than the default version, <code>version</code> must be specified.
<code>vrf_id</code>	VRF ID for the route contained in message. Not present in version 0/1/2 messages in the on-wire structure, and always 0 for these version.
<code>command</code>	Zebra Protocol command, which denotes message type.
<code>body</code>	Messages body. An instance of subclass of <code>_ZebraMessageBody</code> named like "Zebra + <message name>" (e.g., <code>ZebraHello</code> ). Or <code>None</code> if message does not contain any body.

#### Note

To instantiate Zebra messages, `command` can be omitted when the valid body is specified.

```
>>> from os_ken.lib.packet import zebra
>>> zebra.ZebraMessage(body=zebra.ZebraHello())
ZebraMessage(body=ZebraHello(route_type=14), command=23,
length=None, version=3, vrf_id=0)
```

On the other hand, if `body` is omitted, `command` must be specified.

```
>>> zebra.ZebraMessage(command=zebra.ZEBRA_INTERFACE_ADD)
ZebraMessage(body=None, command=1, length=None, version=3, vrf_id=0)
```

```
classmethod parser(buf)
```

Decode a protocol header.

This method is used only when decoding a packet.

Decode a protocol header at offset 0 in bytearray `buf`. Returns the following three objects.

- An object to describe the decoded header.
- A `packet_base.PacketBase` subclass appropriate for the rest of the packet. None when the rest of the packet should be considered as raw payload.
- The rest of packet.

**serialize**(*\_payload=None, \_prev=None*)

Encode a protocol header.

This method is used only when encoding a packet.

Encode a protocol header. Returns a bytearray which contains the header.

*payload* is the rest of the packet which will immediately follow this header.

*prev* is a `packet_base.PacketBase` subclass for the outer protocol header. *prev* is None if the current header is the outer-most. For example, *prev* is `ipv4` or `ipv6` for `tcp.serialize`.

```
class os_ken.lib.packet.zebra.ZebraMplsLabelsAdd(route_type, family, prefix, gate_addr,
                                                ifindex=None, distance=None,
                                                in_label=None, out_label=None)
```

Message body class for FRR\_ZEBRA\_MPLS\_LABELS\_ADD.

```
class os_ken.lib.packet.zebra.ZebraMplsLabelsDelete(route_type, family, prefix,
                                                    gate_addr, ifindex=None,
                                                    distance=None, in_label=None,
                                                    out_label=None)
```

Message body class for FRR\_ZEBRA\_MPLS\_LABELS\_DELETE.

```
class os_ken.lib.packet.zebra.ZebraNextHopRegister(nexthops)
```

Message body class for ZEBRA\_NEXTHOP\_REGISTER.

```
class os_ken.lib.packet.zebra.ZebraNextHopUnregister(nexthops)
```

Message body class for ZEBRA\_NEXTHOP\_UNREGISTER.

```
class os_ken.lib.packet.zebra.ZebraNextHopUpdate(family, prefix, distance=None,
                                                  metric=None, nexthops=None)
```

Message body class for ZEBRA\_NEXTHOP\_UPDATE.

```
class os_ken.lib.packet.zebra.ZebraRedistributeAdd(route_type, afi=None,
                                                  instance=None)
```

Message body class for ZEBRA\_REDISTRIBUTE\_ADD.

```
class os_ken.lib.packet.zebra.ZebraRedistributeDefaultAdd(route_type, afi=None,
                                                         instance=None)
```

Message body class for ZEBRA\_REDISTRIBUTE\_DEFAULT\_ADD.

```
class os_ken.lib.packet.zebra.ZebraRedistributeDefaultDelete(route_type, afi=None,
                                                            instance=None)
```

Message body class for ZEBRA\_REDISTRIBUTE\_DEFAULT\_DELETE.

```
class os_ken.lib.packet.zebra.ZebraRedistributeDelete(route_type, afi=None,
                                                      instance=None)
```

Message body class for ZEBRA\_REDISTRIBUTE\_DELETE.

```
class os_ken.lib.packet.zebra.ZebraRedistributeIPv4Add(route_type, flags, message,  
                                                    safi=None, prefix=None,  
                                                    src_prefix=None,  
                                                    nexthops=None,  
                                                    ifindexes=None,  
                                                    distance=None, metric=None,  
                                                    mtu=None, tag=None,  
                                                    instance=None,  
                                                    from_zebra=False)
```

Message body class for FRR\_ZEBRA\_IPV4\_ROUTE\_ADD.

```
class os_ken.lib.packet.zebra.ZebraRedistributeIPv4Delete(route_type, flags, message,  
                                                         safi=None, prefix=None,  
                                                         src_prefix=None,  
                                                         nexthops=None,  
                                                         ifindexes=None,  
                                                         distance=None,  
                                                         metric=None, mtu=None,  
                                                         tag=None, instance=None,  
                                                         from_zebra=False)
```

Message body class for FRR\_ZEBRA\_IPV4\_ROUTE\_DELETE.

```
class os_ken.lib.packet.zebra.ZebraRedistributeIPv6Add(route_type, flags, message,  
                                                       safi=None, prefix=None,  
                                                       src_prefix=None,  
                                                       nexthops=None,  
                                                       ifindexes=None,  
                                                       distance=None, metric=None,  
                                                       mtu=None, tag=None,  
                                                       instance=None,  
                                                       from_zebra=False)
```

Message body class for FRR\_ZEBRA\_REDISTRIBUTE\_IPV6\_ADD.

```
class os_ken.lib.packet.zebra.ZebraRedistributeIPv6Delete(route_type, flags, message,  
                                                          safi=None, prefix=None,  
                                                          src_prefix=None,  
                                                          nexthops=None,  
                                                          ifindexes=None,  
                                                          distance=None,  
                                                          metric=None, mtu=None,  
                                                          tag=None, instance=None,  
                                                          from_zebra=False)
```

Message body class for FRR\_ZEBRA\_REDISTRIBUTE\_IPV6\_DEL.

```
class os_ken.lib.packet.zebra.ZebraRouterIDAdd
```

Message body class for ZEBRA\_ROUTER\_ID\_ADD.

```
class os_ken.lib.packet.zebra.ZebraRouterIDDelete
```

Message body class for ZEBRA\_ROUTER\_ID\_DELETE.

```
class os_ken.lib.packet.zebra.ZebraRouterIDUpdate(family, prefix)
```

Message body class for ZEBRA\_ROUTER\_ID\_UPDATE.



**class** `os_ken.lib.packet.zebra.ZebraUnknownMessage(buf)`

Message body class for Unknown command.

**class** `os_ken.lib.packet.zebra.ZebraVrfAdd(vrf_name)`

Message body class for FRR\_ZEBRA\_VRF\_ADD.

**class** `os_ken.lib.packet.zebra.ZebraVrfDelete(vrf_name)`

Message body class for FRR\_ZEBRA\_VRF\_DELETE.

**class** `os_ken.lib.packet.zebra.ZebraVrfUnregister`

Message body class for ZEBRA\_VRF\_UNREGISTER.

`os_ken.lib.packet.zebra.zebra`

alias of *ZebraMessage*

## PCAP file library

### Introduction

OS-Ken PCAP file library helps you to read/write PCAP file which file format are described in [The Wireshark Wiki](#).

### Reading PCAP file

For loading the packet data containing in PCAP files, you can use `pcaplib.Reader`.

**class** `os_ken.lib.pcaplib.Reader(file_obj)`

PCAP file reader

Argument	Description
<code>file_obj</code>	File object which reading PCAP file in binary mode

Example of usage:

```
from os_ken.lib import pcaplib
from os_ken.lib.packet import packet

frame_count = 0
# iterate pcaplib.Reader that yields (timestamp, packet_data)
# in the PCAP file
for ts, buf in pcaplib.Reader(open('test.pcap', 'rb')):
    frame_count += 1
    pkt = packet.Packet(buf)
    print("%d, %f, %s" % (frame_count, ts, pkt))
```

### Writing PCAP file

For dumping the packet data which your OSKenApp received, you can use `pcaplib.Writer`.

**class** `os_ken.lib.pcaplib.Writer(file_obj, snaplen=65535, network=1)`

PCAP file writer

Argument	Description
file_obj	File object which writing PCAP file in binary mode
snaplen	Max length of captured packets (in octets)
network	Data link type. (e.g. 1 for Ethernet, see <a href="http://tcpdump.org">tcpdump.org</a> for details)

Example of usage:

```

...
from os_ken.lib import pcaplib

class SimpleSwitch13(app_manager.OSKenApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super(SimpleSwitch13, self).__init__(*args, **kwargs)
        self.mac_to_port = {}

        # Create pcaplib.Writer instance with a file object
        # for the PCAP file
        self.pcap_writer = pcaplib.Writer(open('mypcap.pcap', 'wb'))

    ...

    @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
    def _packet_in_handler(self, ev):
        # Dump the packet data into PCAP file
        self.pcap_writer.write_pkt(ev.msg.data)

    ...

```

## OF-Config support

OS-Ken has a library for OF-Config support.

### XML schema files for NETCONFIG and OFConfig

XML schema files for NETCONF and OFConfig are stolen from LINC whose licence is Apache 2.0. It supports only part of OFConfig so that its schema files are (intentionally) limited such that operation attributes are allowed only in several limited places. Once our library is tested with other OFConfig switches, the schema files should be updated to allow operation attribute in more places.

## References

- NETCONF ietf,
- NETCONF ietf wiki,
- OF-Config spec,
- ncclient,

- ncclient repo,
- LINC git repo

## BGP speaker library

### Introduction

OS-Ken BGP speaker library helps you to enable your code to speak BGP protocol. The library supports IPv4, IPv4 MPLS-labeled VPN, IPv6 MPLS-labeled VPN and L2VPN EVPN address families.

### Example

The following simple code creates a BGP instance with AS number 64512 and Router ID 10.0.0.1. It tries to establish a bgp session with a peer (its IP is 192.168.177.32 and the AS number is 64513). The instance advertizes some prefixes.

```
import eventlet

# BGPSpeaker needs sockets patched
eventlet.monkey_patch()

# initialize a log handler
# this is not strictly necessary but useful if you get messages like:
#   No handlers could be found for logger "os_ken.lib.hub"
import logging
import sys
log = logging.getLogger()
log.addHandler(logging.StreamHandler(sys.stderr))

from os_ken.services.protocols.bgp.bgpspeaker import BGPSpeaker

def dump_remote_best_path_change(event):
    print 'the best path changed:', event.remote_as, event.prefix, \
          event.nexthop, event.is_withdraw

def detect_peer_down(remote_ip, remote_as):
    print 'Peer down:', remote_ip, remote_as

if __name__ == "__main__":
    speaker = BGPSpeaker(as_number=64512, router_id='10.0.0.1',
                          best_path_change_handler=dump_remote_best_path_
→change,
                          peer_down_handler=detect_peer_down)

    speaker.neighbor_add('192.168.177.32', 64513)
# uncomment the below line if the speaker needs to talk with a bmp server.
# speaker.bmp_server_add('192.168.177.2', 11019)
    count = 1
    while True:
        eventlet.sleep(30)
        prefix = '10.20.' + str(count) + '.0/24'
```

(continues on next page)

(continued from previous page)

```

print "add a new prefix", prefix
speaker.prefix_add(prefix)
count += 1
if count == 4:
    speaker.shutdown()
    break

```

## BGP speaker library API Reference

### BGP speaker class

```

class os_ken.services.protocols.bgp.bgpspeaker.BGP speaker(as_number, router_id,
    bgp_server_hosts=('0.0.0.0',
    ':::'),
    bgp_server_port=179,
    refresh_stalepath_time=0,
    refresh_max_eor_time=0,
    best_path_change_handler=None,
    adj_rib_in_change_handler=None,
    peer_down_handler=None,
    peer_up_handler=None,
    ssh_console=False,
    ssh_port=None,
    ssh_host=None,
    ssh_host_key=None,
    label_range=(100,
    100000), al-
    low_local_as_in_count=0,
    cluster_id=None,
    local_pref=100)

```

Class to provide the APIs of OSKen BGP Speaker.

*as\_number* specifies an Autonomous Number. It must be an integer between 1 and 65535.

*router\_id* specifies BGP router identifier. It must be the string representation of an IPv4 address (e.g. 10.0.0.1).

*bgp\_server\_host* specifies a list of TCP listen host addresses.

*bgp\_server\_port* specifies TCP listen port number. 179 is used if not specified.

*refresh\_stalepath\_time* causes the BGP speaker to remove stale routes from the BGP table after the timer expires, even if the speaker does not receive a Router-Refresh End-of-RIB message. This feature is disabled (not implemented yet).

*refresh\_max\_eor\_time* causes the BGP speaker to generate a Route-Refresh End-of-RIB message if it was not able to generate one due to route flapping. This feature is disabled (not implemented yet).

*best\_path\_change\_handler*, if specified, is called when any best remote path is changed due to an update message or remote peer down. The handler is supposed to take one argument, the instance of an EventPrefix class instance.

*adj\_rib\_in\_change\_handler*, if specified, is called when any adj-RIB-in path is changed due

to an update message or remote peer down. The given handler should take three argument, the instance of an EventPrefix class instance, str type peer's IP address and int type peer's AS number.

`peer_down_handler`, if specified, is called when BGP peering session goes down.

`peer_up_handler`, if specified, is called when BGP peering session goes up.

`ssh_console` specifies whether or not SSH CLI need to be started.

`ssh_port` specifies the port number for SSH CLI server. The default is `bgp.operator.ssh.DEFAULT_SSH_PORT`.

`ssh_host` specifies the IP address for SSH CLI server. The default is `bgp.operator.ssh.DEFAULT_SSH_HOST`.

`ssh_host_key` specifies the path to the host key added to the keys list used by SSH CLI server. The default is `bgp.operator.ssh.DEFAULT_SSH_HOST_KEY`.

`label_range` specifies the range of MPLS labels generated automatically.

`allow_local_as_in_count` maximum number of local AS number occurrences in `AS_PATH`. This option is useful for e.g. auto RD/RT configurations in leaf/spine architecture with shared AS numbers. The default is 0 and means "local AS number is not allowed in `AS_PATH`". To allow local AS, 3 is recommended (Cisco's default).

`cluster_id` specifies the cluster identifier for Route Reflector. It must be the string representation of an IPv4 address. If omitted, "router\_id" is used for this field.

`local_pref` specifies the default local preference. It must be an integer.

**`attribute_map_get`**(*address*, *route\_dist=None*, *route\_family='ipv4'*)

This method gets in-bound filters of the specified neighbor.

`address` specifies the IP address of the neighbor.

`route_dist` specifies route distinguisher that has `attribute_maps`.

`route_family` specifies route family of the VRF. This parameter must be one of the following.

- `RF_VPN_V4` (default) = 'ipv4'
- `RF_VPN_V6` = 'ipv6'

Returns a list object containing an instance of `AttributeMap`

**`attribute_map_set`**(*address*, *attribute\_maps*, *route\_dist=None*, *route\_family='ipv4'*)

This method sets attribute mapping to a neighbor. attribute mapping can be used when you want to apply attribute to BGPUpdate under specific conditions.

`address` specifies the IP address of the neighbor

`attribute_maps` specifies `attribute_map` list that are used before paths are advertised. All the items in the list must be an instance of `AttributeMap` class

`route_dist` specifies route dist in which `attribute_maps` are added.

`route_family` specifies route family of the VRF. This parameter must be one of the following.

- `RF_VPN_V4` (default) = 'ipv4'
- `RF_VPN_V6` = 'ipv6'

We can set AttributeMap to a neighbor as follows:

```
pref_filter = PrefixFilter('192.168.103.0/30',
                          PrefixFilter.POLICY_PERMIT)

attribute_map = AttributeMap([pref_filter],
                             AttributeMap.ATTR_LOCAL_PREF, 250)

speaker.attribute_map_set('192.168.50.102', [attribute_map])
```

#### **bmp\_server\_add**(*address, port*)

This method registers a new BMP (BGP monitoring Protocol) server. The BGP speaker starts to send BMP messages to the server. Currently, only one BMP server can be registered.

*address* specifies the IP address of a BMP server.

*port* specifies the listen port number of a BMP server.

#### **bmp\_server\_del**(*address, port*)

This method unregister the registered BMP server.

*address* specifies the IP address of a BMP server.

*port* specifies the listen port number of a BMP server.

#### **evpn\_prefix\_add**(*route\_type, route\_dist, esi=0, ethernet\_tag\_id=None, mac\_addr=None, ip\_addr=None, ip\_prefix=None, gw\_ip\_addr=None, vni=None, next\_hop=None, tunnel\_type=None, pmsi\_tunnel\_type=None, redundancy\_mode=None, tunnel\_endpoint\_ip=None, mac\_mobility=None*)

This method adds a new EVPN route to be advertised.

*route\_type* specifies one of the EVPN route type name. This parameter must be one of the following.

- EVPN\_ETH\_AUTO\_DISCOVERY = 'eth\_ad'
- EVPN\_MAC\_IP\_ADV\_ROUTE = 'mac\_ip\_adv'
- EVPN\_MULTICAST\_ETAG\_ROUTE = 'multicast\_etag'
- EVPN\_ETH\_SEGMENT = 'eth\_seg'
- EVPN\_IP\_PREFIX\_ROUTE = 'ip\_prefix'

*route\_dist* specifies a route distinguisher value.

*esi* is an value to specify the Ethernet Segment Identifier. 0 is the default and denotes a single-homed site. If you want to advertise esi other than 0, it must be set as dictionary type. If esi is dictionary type, 'type' key must be set and specifies ESI type. For the supported ESI type, see *os\_ken.lib.packet.bgp.EvpnEsi*. The remaining arguments are the same as that for the corresponding class.

*ethernet\_tag\_id* specifies the Ethernet Tag ID.

*mac\_addr* specifies a MAC address to advertise.

*ip\_addr* specifies an IPv4 or IPv6 address to advertise.

*ip\_prefix* specifies an IPv4 or IPv6 prefix to advertise.

`gw_ip_addr` specifies an IPv4 or IPv6 address of gateway to advertise.

`vni` specifies an Virtual Network Identifier for VXLAN or Virtual Subnet Identifier for NVGRE. If `tunnel_type` is not `TUNNEL_TYPE_VXLAN` or `TUNNEL_TYPE_NVGRE`, this field is ignored.

`next_hop` specifies the next hop address for this prefix.

`tunnel_type` specifies the data plane encapsulation type to advertise. By the default, this attribute is not advertised. The supported encapsulation types are following.

- `TUNNEL_TYPE_VXLAN = 'vxlan'`
- `TUNNEL_TYPE_NVGRE = 'nvgre'`

`pmsi_tunnel_type` specifies the type of the PMSI tunnel attribute used to encode the multicast tunnel identifier. This attribute is advertised only if `route_type` is `EVPN_MULTICAST_ETAG_ROUTE` and not advertised by the default. This attribute can also carry `vni` if `tunnel_type` is specified. The supported PMSI tunnel types are following.

- `PMSI_TYPE_NO_TUNNEL_INFO = 0`
- `PMSI_TYPE_INGRESS_REP = 6`

`redundancy_mode` specifies a redundancy mode type. This attribute is advertised only if `route_type` is `EVPN_ETH_AUTO_DISCOVERY` and not advertised by the default. The supported redundancy mode types are following.

- `REDUNDANCY_MODE_ALL_ACTIVE = 'all_active'`
- `REDUNDANCY_MODE_SINGLE_ACTIVE = 'single_active'`

`tunnel_endpoint_ip` specifies a VTEP IP address other than the local router ID. This attribute is advertised only if `route_type` is `EVPN_MULTICAST_ETAG_ROUTE`, and defaults to the local router ID.

`mac_mobility` specifies an optional integer sequence number to use in a MAC Mobility extended community field. The special value '-1' can be used to set the STATIC flag with a 0-value sequence number.

**`evpn_prefix_del`**(*route\_type, route\_dist, esi=0, ethernet\_tag\_id=None, mac\_addr=None, ip\_addr=None, ip\_prefix=None*)

This method deletes an advertised EVPN route.

`route_type` specifies one of the EVPN route type name.

`route_dist` specifies a route distinguisher value.

`esi` is an value to specify the Ethernet Segment Identifier.

`ethernet_tag_id` specifies the Ethernet Tag ID.

`mac_addr` specifies a MAC address to advertise.

`ip_addr` specifies an IPv4 or IPv6 address to advertise.

`ip_prefix` specifies an IPv4 or IPv6 prefix to advertise.

**`flowspec_prefix_add`**(*flowspec\_family, rules, route\_dist=None, actions=None*)

This method adds a new Flow Specification prefix to be advertised.

`flowspec_family` specifies one of the flowspec family name. This parameter must be one of the following.

- `FLWSPEC_FAMILY_IPV4 = 'ipv4fs'`
- `FLWSPEC_FAMILY_IPV6 = 'ipv6fs'`
- `FLWSPEC_FAMILY_VPNV4 = 'vpnv4fs'`
- `FLWSPEC_FAMILY_VPNV6 = 'vpnv6fs'`
- `FLWSPEC_FAMILY_L2VPN = 'l2vp nfs'`

`rules` specifies NLRIs of Flow Specification as a dictionary type value. For the supported NLRI types and arguments, see `from_user()` method of the following classes.

- `os_ken.lib.packet.bgp.FlowSpecIPv4NLRI`
- `os_ken.lib.packet.bgp.FlowSpecIPv6NLRI`
- `os_ken.lib.packet.bgp.FlowSpecVPNv4NLRI`
- `os_ken.lib.packet.bgp.FlowSpecVPNv6NLRI`
- `os_ken.lib.packet.bgp.FlowSpecL2VPNLRRI`

`route_dist` specifies a route distinguisher value. This parameter is required only if `flowspec_family` is one of the following address family.

- `FLWSPEC_FAMILY_VPNV4 = 'vpnv4fs'`
- `FLWSPEC_FAMILY_VPNV6 = 'vpnv6fs'`
- `FLWSPEC_FAMILY_L2VPN = 'l2vp nfs'`

`actions` specifies Traffic Filtering Actions of Flow Specification as a dictionary type value. The keys are "ACTION\_NAME" for each action class and values are used for the arguments to that class. For the supported "ACTION\_NAME" and arguments, see the following table.

AC-TION_NAME	Action Class
<code>traffic_rate</code>	<code>os_ken.lib.packet.bgp.BGPFlowSpecTrafficRateCommunity</code>
<code>traffic_action</code>	<code>os_ken.lib.packet.bgp.BGPFlowSpecTrafficActionCommunity</code>
<code>redirect</code>	<code>os_ken.lib.packet.bgp.BGPFlowSpecRedirectCommunity</code>
<code>traffic_marking</code>	<code>os_ken.lib.packet.bgp.BGPFlowSpecTrafficMarkingCommunity</code>
<code>vlan_action</code>	<code>os_ken.lib.packet.bgp.BGPFlowSpecVlanActionCommunity</code>
<code>tpid_action</code>	<code>os_ken.lib.packet.bgp.BGPFlowSpecTPIDActionCommunity</code>

Example(IPv4):

```
>>> speaker = BGPSpeaker(as_number=65001, router_id='172.17.0.1')
>>> speaker.neighbor_add(address='172.17.0.2',
...                       remote_as=65002,
...                       enable_ipv4fs=True)
>>> speaker.flowspec_prefix_add(
...     flowspec_family=FLWSPEC_FAMILY_IPV4,
...     rules={
```

(continues on next page)



(continued from previous page)

```

...         'dst_prefix': '10.60.1.0/24'
...     },
...     actions={
...         'traffic_marking': {
...             'dscp': 24
...         }
...     }
... )

```

Example(VPNv4):

```

>>> speaker = BGPSpeaker(as_number=65001, router_id='172.17.0.1')
>>> speaker.neighbor_add(address='172.17.0.2',
...                       remote_as=65002,
...                       enable_vpnv4fs=True)
>>> speaker.vrf_add(route_dist='65001:100',
...                  import_rts=['65001:100'],
...                  export_rts=['65001:100'],
...                  route_family=RF_VPNV4_FLOWSPEC)
>>> speaker.flowspec_prefix_add(
...     flowspec_family=FLOWSPEC_FAMILY_VPNV4,
...     route_dist='65000:100',
...     rules={
...         'dst_prefix': '10.60.1.0/24'
...     },
...     actions={
...         'traffic_marking': {
...             'dscp': 24
...         }
...     }
... )

```

**flowspec\_prefix\_del**(*flowspec\_family, rules, route\_dist=None*)

This method deletes an advertised Flow Specification route.

*flowspec\_family* specifies one of the flowspec family name.

*rules* specifies NLRIs of Flow Specification as a dictionary type value.

*route\_dist* specifies a route distinguisher value.

**in\_filter\_get**(*address*)

This method gets in-bound filters of the specified neighbor.

*address* specifies the IP address of the neighbor.

Returns a list object containing an instance of Filter sub-class

**in\_filter\_set**(*address, filters*)

This method sets in-bound filters to a neighbor.

*address* specifies the IP address of the neighbor

`filters` specifies filter list applied before advertised paths are imported to the global rib. All the items in the list must be an instance of Filter sub-class.

**neighbor\_add**(*address, remote\_as, remote\_port=179, enable\_ipv4=True, enable\_ipv6=False, enable\_vpnv4=False, enable\_vpnv6=False, enable\_evpn=False, enable\_ipv4fs=False, enable\_ipv6fs=False, enable\_vpnv4fs=False, enable\_vpnv6fs=False, enable\_l2vpnfs=False, enable\_enhanced\_refresh=False, enable\_four\_octet\_as\_number=True, next\_hop=None, password=None, multi\_exit\_disc=None, site\_of\_origins=None, is\_route\_server\_client=False, is\_route\_reflector\_client=False, is\_next\_hop\_self=False, local\_address=None, local\_port=None, local\_as=None, connect\_mode='both'*)

This method registers a new neighbor. The BGP speaker tries to establish a bgp session with the peer (accepts a connection from the peer and also tries to connect to it).

`address` specifies the IP address of the peer. It must be the string representation of an IP address. Only IPv4 is supported now.

`remote_as` specifies the AS number of the peer. It must be an integer between 1 and 65535.

`remote_port` specifies the TCP port number of the peer.

`enable_ipv4` enables IPv4 address family for this neighbor.

`enable_ipv6` enables IPv6 address family for this neighbor.

`enable_vpnv4` enables VPNv4 address family for this neighbor.

`enable_vpnv6` enables VPNv6 address family for this neighbor.

`enable_evpn` enables Ethernet VPN address family for this neighbor.

`enable_ipv4fs` enables IPv4 Flow Specification address family for this neighbor.

`enable_ipv6fs` enables IPv6 Flow Specification address family for this neighbor.

`enable_vpnv4fs` enables VPNv4 Flow Specification address family for this neighbor.

`enable_vpnv6fs` enables VPNv6 Flow Specification address family for this neighbor.

`enable_l2vpnfs` enables L2VPN Flow Specification address family for this neighbor.

`enable_enhanced_refresh` enables Enhanced Route Refresh for this neighbor.

`enable_four_octet_as_number` enables Four-Octet AS Number capability for this neighbor.

`next_hop` specifies the next hop IP address. If not specified, host's ip address to access to a peer is used.

`password` is used for the MD5 authentication if it's specified. By default, the MD5 authentication is disabled.

`multi_exit_disc` specifies multi exit discriminator (MED) value as an int type value. If omitted, MED is not sent to the neighbor.

`site_of_origins` specifies `site_of_origin` values. This parameter must be a list of string.

`is_route_server_client` specifies whether this neighbor is a router server's client or not.

`is_route_reflector_client` specifies whether this neighbor is a router reflector's client or not.

`is_next_hop_self` specifies whether the BGP speaker announces its own ip address to iBGP neighbor or not as path's next\_hop address.

`local_address` specifies Loopback interface address for iBGP peering.

`local_port` specifies source TCP port for iBGP peering.

`local_as` specifies local AS number per-peer. If omitted, the AS number of BGPSpeaker instance is used.

`connect_mode` specifies how to connect to this neighbor. This parameter must be one of the following.

- `CONNECT_MODE_ACTIVE` = 'active'
- `CONNECT_MODE_PASSIVE` = 'passive'
- `CONNECT_MODE_BOTH` (default) = 'both'

#### **neighbor\_del**(*address*)

This method unregister the registered neighbor. If a session with the peer exists, the session will be closed.

`address` specifies the IP address of the peer. It must be the string representation of an IP address.

#### **neighbor\_get**(*route\_type, address, format='json'*)

This method returns the BGP adj-RIB-in/adj-RIB-out information in a json format.

`route_type` This parameter is necessary for only received-routes and sent-routes.

- received-routes : paths received and not withdrawn by given peer
- sent-routes : paths sent and not withdrawn to given peer

`address` specifies the IP address of the peer. It must be the string representation of an IP address.

`format` specifies the format of the response. This parameter must be one of the following.

- 'json' (default)
- 'cli'

#### **neighbor\_reset**(*address*)

This method reset the registered neighbor.

`address` specifies the IP address of the peer. It must be the string representation of an IP address.

#### **neighbor\_state\_get**(*address=None, format='json'*)

This method returns the state of peer(s) in a json format.

`address` specifies the address of a peer. If not given, the state of all the peers return.

`format` specifies the format of the response. This parameter must be one of the following.

- 'json' (default)
- 'cli'

**neighbor\_update**(*address, conf\_type, conf\_value*)

This method changes the neighbor configuration.

*address* specifies the IP address of the peer.

*conf\_type* specifies configuration type which you want to change. Currently `os_ken.services.protocols.bgp.bgpspeaker.MULTI_EXIT_DISC` can be specified.

*conf\_value* specifies value for the configuration type.

**neighbors\_get**(*format='json'*)

This method returns a list of the BGP neighbors.

*format* specifies the format of the response. This parameter must be one of the following.

- 'json' (default)
- 'cli'

**out\_filter\_get**(*address*)

This method gets out-filter setting from the specified neighbor.

*address* specifies the IP address of the peer.

Returns a list object containing an instance of Filter sub-class

**out\_filter\_set**(*address, filters*)

This method sets out-filter to neighbor.

*address* specifies the IP address of the peer.

*filters* specifies a filter list to filter the path advertisement. The contents must be an instance of Filter sub-class

If you want to define out-filter that send only a particular prefix to neighbor, filters can be created as follows:

```
p = PrefixFilter('10.5.111.0/24',
                 policy=PrefixFilter.POLICY_PERMIT)

all = PrefixFilter('0.0.0.0/0',
                  policy=PrefixFilter.POLICY_DENY)

pList = [p, all]

self.bgpspeaker.out_filter_set(neighbor_address, pList)
```

#### Note

out-filter evaluates paths in the order of Filter in the pList.

**prefix\_add**(*prefix, next\_hop=None, route\_dist=None*)

This method adds a new prefix to be advertised.

*prefix* must be the string representation of an IP network (e.g., 10.1.1.0/24).

`next_hop` specifies the next hop address for this prefix. This parameter is necessary for only VPNv4 and VPNv6 address families.

`route_dist` specifies a route distinguisher value. This parameter is necessary for only VPNv4 and VPNv6 address families.

**prefix\_del**(*prefix*, *route\_dist=None*)

This method deletes a advertised prefix.

`prefix` must be the string representation of an IP network.

`route_dist` specifies a route distinguisher value.

**rib\_get**(*family='all'*, *format='json'*)

This method returns the BGP routing information in a json format. This will be improved soon.

`family` specifies the address family of the RIB (e.g. 'ipv4').

`format` specifies the format of the response. This parameter must be one of the following.

- 'json' (default)
- 'cli'

**shutdown**()

Shutdown BGP speaker

**vrf\_add**(*route\_dist*, *import\_rts*, *export\_rts*, *site\_of\_origins=None*, *route\_family='ipv4'*, *multi\_exit\_disc=None*)

This method adds a new vrf used for VPN.

`route_dist` specifies a route distinguisher value.

`import_rts` specifies a list of route targets to be imported.

`export_rts` specifies a list of route targets to be exported.

`site_of_origins` specifies `site_of_origin` values. This parameter must be a list of string.

`route_family` specifies route family of the VRF. This parameter must be one of the following.

- RF\_VPN\_V4 (default) = 'ipv4'
- RF\_VPN\_V6 = 'ipv6'
- RF\_L2\_EVPN = 'evpn'
- RF\_VPNV4\_FLOWSPEC = 'ipv4fs'
- RF\_VPNV6\_FLOWSPEC = 'ipv6fs'
- RF\_L2VPN\_FLOWSPEC = 'l2vpns'

`multi_exit_disc` specifies multi exit discriminator (MED) value. It must be an integer.

**vrf\_del**(*route\_dist*)

This method deletes the existing vrf.

`route_dist` specifies a route distinguisher value.

**vrf\_get**(*subcommand='routes', route\_dist=None, route\_family='all', format='json'*)

This method returns the existing vrfs.

*subcommand* specifies one of the following.

- 'routes': shows routes present for vrf
- 'summary': shows configuration and summary of vrf

*route\_dist* specifies a route distinguisher value. If *route\_family* is not 'all', this value must be specified.

*route\_family* specifies route family of the VRF. This parameter must be one of the following.

- RF\_VPN\_V4 = 'ipv4'
- RF\_VPN\_V6 = 'ipv6'
- RF\_L2\_EVPN = 'evpn'
- 'all' (default)

*format* specifies the format of the response. This parameter must be one of the following.

- 'json' (default)
- 'cli'

**class** `os_ken.services.protocols.bgp.bgpspeaker.EventPrefix`(*path, is\_withdraw*)

Used to pass an update on any best remote path to `best_path_change_handler`.

Attribute	Description
<code>remote_as</code>	The AS number of a peer that caused this change
<code>route_dist</code>	None in the case of IPv4 or IPv6 family
<code>prefix</code>	A prefix was changed
<code>nexthop</code>	The nexthop of the changed prefix
<code>label</code>	MPLS label for VPNv4, VPNv6 or EVPN prefix
<code>path</code>	An instance of <code>info_base.base.Path</code> subclass
<code>is_withdraw</code>	True if this prefix has gone otherwise False

**class** `os_ken.services.protocols.bgp.info_base.base.PrefixFilter`(*prefix, policy, ge=None, le=None*)

Used to specify a prefix for filter.

We can create PrefixFilter object as follows:

```
prefix_filter = PrefixFilter('10.5.111.0/24',
                             policy=PrefixFilter.POLICY_PERMIT)
```

Attribute	Description
prefix	A prefix used for this filter
policy	One of the following values.  PrefixFilter.POLICY.PERMIT PrefixFilter.POLICY_DENY
ge	Prefix length that will be applied to this filter. ge means greater than or equal.
le	Prefix length that will be applied to this filter. le means less than or equal.

For example, when PrefixFilter object is created as follows:

```
p = PrefixFilter('10.5.111.0/24',
                policy=PrefixFilter.POLICY_DENY,
                ge=26, le=28)
```

Prefixes which match 10.5.111.0/24 and its length matches from 26 to 28 will be filtered. When this filter is used as an out-filter, it will stop sending the path to neighbor because of POLICY\_DENY. When this filter is used as in-filter, it will stop importing the path to the global rib because of POLICY\_DENY. If you specify POLICY\_PERMIT, the path is sent to neighbor or imported to the global rib.

If you don't want to send prefixes 10.5.111.64/26 and 10.5.111.32/27 and 10.5.111.16/28, and allow to send other 10.5.111.0's prefixes, you can do it by specifying as follows:

```
p = PrefixFilter('10.5.111.0/24',
                policy=PrefixFilter.POLICY_DENY,
                ge=26, le=28).
```

### clone()

This method clones PrefixFilter object.

Returns PrefixFilter object that has the same values with the original one.

### evaluate(path)

This method evaluates the prefix.

Returns this object's policy and the result of matching. If the specified prefix matches this object's prefix and ge and le condition, this method returns True as the matching result.

path specifies the path that has prefix.

**class** `os_ken.services.protocols.bgp.info_base.base.ASPathFilter`(*as\_number*,  
*policy*)

Used to specify a prefix for AS\_PATH attribute.

We can create ASPathFilter object as follows:

```
as_path_filter = ASPathFilter(65000, policy=ASPathFilter.TOP)
```

Attribute	Description
as_number	A AS number used for this filter
policy	One of the following values.  ASPathFilter.POLICY_TOP ASPathFilter.POLICY_END ASPathFilter.POLICY_INCLUDE ASPathFilter.POLICY_NOT_INCLUDE

Meaning of each policy is as follows:

Policy	Description
POLICY_TOP	Filter checks if the specified AS number is at the top of AS_PATH attribute.
POLICY_END	Filter checks if the specified AS number is at the last of AS_PATH attribute.
POLICY_INCLUDE	Filter checks if specified AS number exists in AS_PATH attribute.
POLICY_NOT_INCLUDE	Opposite to POLICY_INCLUDE.

### clone()

This method clones ASPathFilter object.

Returns ASPathFilter object that has the same values with the original one.

### evaluate(path)

This method evaluates as\_path list.

Returns this object's policy and the result of matching. If the specified AS number matches this object's AS number according to the policy, this method returns True as the matching result.

path specifies the path.

**class** os\_ken.services.protocols.bgp.info\_base.base.**AttributeMap**(filters, attr\_type, attr\_value)

This class is used to specify an attribute to add if the path matches filters. We can create AttributeMap object as follows:

```
pref_filter = PrefixFilter('192.168.103.0/30',
                          PrefixFilter.POLICY_PERMIT)
attribute_map = AttributeMap([pref_filter],
                             AttributeMap.ATTR_LOCAL_PREF, 250)
speaker.attribute_map_set('192.168.50.102', [attribute_map])
```

AttributeMap.ATTR\_LOCAL\_PREF means that 250 is set as a local preference value if nlri in the path matches pref\_filter.



ASPathFilter is also available as a filter. ASPathFilter checks if AS\_PATH attribute in the path matches AS number in the filter.

At-tribute	Description
filters	A list of filter. Each object should be a Filter class or its sub-class
attr_type	A type of attribute to map on filters. Currently AttributeMap.ATTR_LOCAL_PREF is available.
attr_value	A attribute value

### clone()

This method clones AttributeMap object.

Returns AttributeMap object that has the same values with the original one.

### evaluate(*path*)

This method evaluates attributes of the path.

Returns the cause and result of matching. Both cause and result are returned from filters that this object contains.

*path* specifies the path.

## MRT file library

### Introduction

OS-Ken MRT file library helps you to read/write MRT (Multi-Threaded Routing Toolkit) Routing Information Export Format [RFC6396].

### Reading MRT file

For loading the routing information contained in MRT files, you can use `mrtlib.Reader`.

**class** `os_ken.lib.mrtlib.Reader(f)`

MRT format file reader.

Argument	Description
<code>f</code>	File object which reading MRT format file in binary mode.

Example of Usage:

```
import bz2
from os_ken.lib import mrtlib

count = 0
for record in mrtlib.Reader(
    bz2.BZ2File('rib.YYYYMMDD.hhmm.bz2', 'rb')):
    print("%d, %s" % (count, record))
    count += 1
```

## Writing MRT file

For dumping the routing information which your OSKenApp generated, you can use `mrtlib.Writer`.

**class** `os_ken.lib.mrtlib.Writer(f)`

MRT format file writer.

Argument	Description
<code>f</code>	File object which writing MRT format file in binary mode.

Example of usage:

```
import bz2
import time
from os_ken.lib import mrtlib
from os_ken.lib.packet import bgp

mrt_writer = mrtlib.Writer(
    bz2.BZ2File('rib.YYYYMMDD.hhmm.bz2', 'wb'))

prefix = bgp.IPAddrPrefix(24, '10.0.0.0')

rib_entry = mrtlib.MrtRibEntry(
    peer_index=0,
    originated_time=int(time.time()),
    bgp_attributes=[bgp.BGPPathAttributeOrigin(0)])

message = mrtlib.TableDump2RibIPv4UnicastMrtMessage(
    seq_num=0,
    prefix=prefix,
    rib_entries=[rib_entry])

record = mrtlib.TableDump2MrtRecord(
    message=message)

mrt_writer.write(record)
```

## OVSDB Manager library

Path: `os_ken.services.protocols.ovsdb`

### Introduction

OS-Ken OVSDB Manager library allows your code to interact with devices speaking the OVSDB protocol. This enables your code to perform remote management of the devices and react to topology changes on them.

Please note this library will spawn a server listening on the port 6640 (the IANA registered for OVSDB protocol), but does not initiate connections from controller side. Then, to make your devices connect to OS-Ken, you need to tell the controller IP address and port to your devices.

```
# Show current configuration
$ ovs-vsctl get-manager

# Set manager (controller) address
$ ovs-vsctl set-manager "tcp:127.0.0.1:6640"

# If you want to specify IPv6 address, wrap ip with brackets
$ ovs-vsctl set-manager "tcp:[::1]:6640"
```

Also this library identifies the devices by "system-id" which should be unique, persistent identifier among all devices connecting to a single controller. Please make sure "system-id" is configured before connecting.

```
# Show current configuration
$ ovs-vsctl get Open_vSwitch . external_ids:system-id

# Set system-id manually
$ ovs-vsctl set Open_vSwitch . external_ids:system-id=<SYSTEM-ID>
```

## Example

The following logs all new OVSDb connections in "handle\_new\_ovsdb\_connection" and also provides the API "create\_port" for creating a port on a bridge.

```
import uuid

from os_ken.base import app_manager
from os_ken.controller.handler import set_ev_cls
from os_ken.services.protocols.ovsdb import api as ovsdb
from os_ken.services.protocols.ovsdb import event as ovsdb_event

class MyApp(app_manager.OSKenApp):
    @set_ev_cls(ovsdb_event.EventNewOVSDbConnection)
    def handle_new_ovsdb_connection(self, ev):
        system_id = ev.system_id
        address = ev.client.address
        self.logger.info(
            'New OVSDb connection from system-id=%s, address=%s',
            system_id, address)

        # Example: If device has bridge "s1", add port "s1-eth99"
        if ovsdb.bridge_exists(self, system_id, "s1"):
            self.create_port(system_id, "s1", "s1-eth99")

    def create_port(self, system_id, bridge_name, name):
        new_iface_uuid = uuid.uuid4()
        new_port_uuid = uuid.uuid4()

        bridge = ovsdb.row_by_name(self, system_id, bridge_name)
```

(continues on next page)

(continued from previous page)

```

def _create_port(tables, insert):
    iface = insert(tables['Interface'], new_iface_uuid)
    iface.name = name
    iface.type = 'internal'

    port = insert(tables['Port'], new_port_uuid)
    port.name = name
    port.interfaces = [iface]

    bridge.ports = bridge.ports + [port]

    return new_port_uuid, new_iface_uuid

req = ovssdb_event.EventModifyRequest(system_id, _create_port)
rep = self.send_request(req)

if rep.status != 'success':
    self.logger.error('Error creating port %s on bridge %s: %s',
                      name, bridge, rep.status)

    return None

return rep.insert_uuids[new_port_uuid]

```

## OVSSDB library

Path: `os_ken.lib.ovs`

Similar to the *OVSSDB Manager library*, this library enables your application to speak the OVSSDB protocol (RFC7047), but differ from the *OVSSDB Manager library*, this library will initiate connections from controller side as `ovs-vsctl` command does. Please make sure that your devices are listening on either the Unix domain socket or TCP/SSL port before calling the APIs of this library.

```

# Show current configuration
$ ovs-vsctl get-manager

# Set TCP listen address
$ ovs-vsctl set-manager "ptcp:6640"

```

See manpage of `ovs-vsctl` command for more details.

## Basic Usage

1. Instantiate `os_ken.lib.ovs.vsctl.VSctl`.
2. Construct commands with `os_ken.lib.ovs.vsctl.VSctlCommand`. The syntax is almost the same as `ovs-vsctl` command.
3. Execute commands via `os_ken.lib.ovs.vsctl.VSctl.run_command`.

## Example

```

from os_ken.lib.ovs import vsctl

OVSDb_ADDR = 'tcp:127.0.0.1:6640'
ovs_vsctl = vsctl.VSctl(OVSDb_ADDR)

# Equivalent to
# $ ovs-vsctl show
command = vsctl.VSctlCommand('show')
ovs_vsctl.run_command([command])
print(command)
# >>> VSctlCommand(args=[], command='show', options=[], result='830d781f-c3c8-4b4f-
↳837e-106e1b33d058\n    ovs_version: "2.8.90"\n')

# Equivalent to
# $ ovs-vsctl list Port s1-eth1
command = vsctl.VSctlCommand('list', ('Port', 's1-eth1'))
ovs_vsctl.run_command([command])
print(command)
# >>> VSctlCommand(args=('Port', 's1-eth1'), command='list', options=[], result=[
↳<ovs.db.idl.Row object at 0x7f525fb682e8>])
print(command.result[0].name)
# >>> s1-eth1

```

## API Reference

### os\_ken.lib.ovs.vsctl

ovs-vsctl command like library to speak OVSDB protocol

**class** `os_ken.lib.ovs.vsctl.VSctl`(*remote*)

A class to describe an Open vSwitch instance.

*remote* specifies the address of the OVS instance. `os_ken.lib.ovs.vsctl.valid_ovsdb_addr` is a convenient function to validate this address.

**run\_command**(*commands*, *timeout\_sec=None*, *exception=None*)

Executes the given commands and sends OVSDB messages.

*commands* must be a list of `os_ken.lib.ovs.vsctl.VSctlCommand`.

If *timeout\_sec* is specified, raises exception after the given timeout [sec]. Additionally, if *exception* is specified, this function will wrap exception using the given exception class.

Retruns None but fills *result* attribute for each command instance.

**class** `os_ken.lib.ovs.vsctl.VSctlCommand`(*command*, *args=None*, *options=None*)

Class to describe arguments similar to those of ovs-vsctl command.

*command* specifies the command of ovs-vsctl.

*args* specifies a list or tuple of arguments for the given command.

*options* specifies a list or tuple of options for the given command. Please note that NOT all

options of `ovs-vsctl` are supported. For example, `--id` option is not yet supported. This class supports the followings.

Option	Description
<code>--may-exis</code>	Does nothing when the given port already exists. The supported commands are <code>add-port</code> and <code>add-bond</code> .
<code>--fake-ifa</code>	Creates a port as a fake interface. The supported command is <code>add-bond</code> .
<code>--must-exi</code>	Raises exception if the given port does not exist. The supported command is <code>del-port</code> .
<code>--with-ifa</code>	Takes effect to the interface which has the same name. The supported command is <code>del-port</code> .
<code>--if-exist</code>	Ignores exception when not found. The supported command is <code>get</code> .

`os_ken.lib.ovs.vsctl.valid_ovsdb_addr(addr)`

Returns True if the given `addr` is valid OVSDB server address, otherwise False.

The valid formats are:

- `unix:file`
- `tcp:ip:port`
- `ssl:ip:port`

If ip is IPv6 address, wrap ip with brackets (e.g., `ssl:[::1]:6640`).

#### Parameters

**addr** -- str value of OVSDB server address.

#### Returns

True if valid, otherwise False.

## os\_ken.lib.ovs.bridge

Wrapper utility library of `os_ken.lib.ovs.vsctl`

**class** `os_ken.lib.ovs.bridge.OVSBridge(CONF, datapath_id, ovsdb_addr, timeout=None, exception=None, br_name=None)`

Class to provide wrapper utilities of `os_ken.lib.ovs.vsctl.VSctl`

`CONF` is a instance of `oslo_config.cfg.ConfigOpts`. Mostly `self.CONF` is sufficient to instantiate this class from your OSKen application.

`datapath_id` specifies Datapath ID of the target OVS instance.

`ovsdb_addr` specifies the address of the OVS instance. Automatically validated when you call `init()` method. Refer to `os_ken.lib.ovs.vsctl.valid_ovsdb_addr` for the format of this address.

if `timeout` is omitted, `CONF.ovsdb_timeout` will be used as the default value.

Usage of `timeout` and `exception` is the same with `timeout_sec` and `exception` of `os_ken.lib.ovs.vsctl.VSctl.run_command`.

**add\_bond(name, ifaces, bond\_mode=None, lacp=None)**

Creates a bonded port.

**Parameters**

- **name** -- Port name to be created
- **ifaces** -- List of interfaces containing at least 2 interfaces
- **bond\_mode** -- Bonding mode (active-backup, balance-tcp or balance-slb)
- **lACP** -- LACP mode (active, passive or off)

**add\_db\_attribute**(*table, record, column, value, key=None*)

Adds ('key'=)'value' into 'column' in 'record' in 'table'.

This method is corresponding to the following ovs-vsctl command:

```
$ ovs-vsctl add TBL REC COL [KEY=]VALUE
```

**add\_gre\_port**(*name, remote\_ip, local\_ip=None, key=None, ofport=None*)

Creates a GRE tunnel port.

See the description of `add_tunnel_port()`.

**add\_tunnel\_port**(*name, tunnel\_type, remote\_ip, local\_ip=None, key=None, ofport=None*)

Creates a tunnel port.

**Parameters**

- **name** -- Port name to be created
- **tunnel\_type** -- Type of tunnel (gre or vxlan)
- **remote\_ip** -- Remote IP address of tunnel
- **local\_ip** -- Local IP address of tunnel
- **key** -- Key of GRE or VNI of VxLAN
- **ofport** -- Requested OpenFlow port number

**add\_vxlan\_port**(*name, remote\_ip, local\_ip=None, key=None, ofport=None*)

Creates a VxLAN tunnel port.

See the description of `add_tunnel_port()`.

**clear\_db\_attribute**(*table, record, column*)

Clears values from 'column' in 'record' in 'table'.

This method is corresponding to the following ovs-vsctl command:

```
$ ovs-vsctl clear TBL REC COL
```

**db\_get\_map**(*table, record, column*)

Gets dict type value of 'column' in 'record' in 'table'.

This method is corresponding to the following ovs-vsctl command:

```
$ ovs-vsctl get TBL REC COL
```

**db\_get\_val**(*table, record, column*)

Gets values of 'column' in 'record' in 'table'.

This method is corresponding to the following ovs-vsctl command:

```
$ ovs-vsctl get TBL REC COL
```

**del\_controller**()

Deletes the configured OpenFlow controller address.

This method is corresponding to the following ovs-vsctl command:

```
$ ovs-vsctl del-controller <bridge>
```

**del\_port**(*port\_name*)

Deletes a port on OVS instance.

This method is corresponding to the following ovs-vsctl command:

```
$ ovs-vsctl del-port <bridge> <port>
```

**del\_qos**(*port\_name*)

Deletes the Qos rule on the given port.

**delete\_port**(*port\_name*)

Deletes a port on the OVS instance.

This method is corresponding to the following ovs-vsctl command:

```
$ ovs-vsctl --if-exists del-port <bridge> <port>
```

**find\_db\_attributes**(*table, \*conditions*)

Lists records satisfying 'conditions' in 'table'.

This method is corresponding to the following ovs-vsctl command:

```
$ ovs-vsctl find TBL CONDITION...
```

**Note**

Currently, only '=' condition is supported. To support other condition is TODO.

**get\_controller**()

Gets the configured OpenFlow controller address.

This method is corresponding to the following ovs-vsctl command:

```
$ ovs-vsctl get-controller <bridge>
```

**get\_datapath\_id**()

Gets Datapath ID of OVS instance.

This method is corresponding to the following ovs-vsctl command:



```
$ ovs-vsctl get Bridge <bridge> datapath_id
```

**get\_db\_attribute**(*table, record, column, key=None*)

Gets values of 'column' in 'record' in 'table'.

This method is corresponding to the following ovs-vsctl command:

```
$ ovs-vsctl get TBL REC COL[:KEY]
```

**get\_ofport**(*port\_name*)

Gets the OpenFlow port number.

This method is corresponding to the following ovs-vsctl command:

```
$ ovs-vsctl get Interface <port> ofport
```

**get\_port\_name\_list**()

Gets a list of all ports on OVS instance.

This method is corresponding to the following ovs-vsctl command:

```
$ ovs-vsctl list-ports <bridge>
```

**get\_vif\_ports**()

Returns a VIF object for each VIF port

**init**()

Validates the given `ovsdb_addr` and connects to OVS instance.

If failed to connect to OVS instance or the given `datapath_id` does not match with the Datapath ID of the connected OVS instance, raises `os_ken.lib.ovs.bridge.OVSBridgeNotFound` exception.

**list\_db\_attributes**(*table, record=None*)

Lists 'record' (or all records) in 'table'.

This method is corresponding to the following ovs-vsctl command:

```
$ ovs-vsctl list TBL [REC]
```

**remove\_db\_attribute**(*table, record, column, value, key=None*)

Removes ('key'=)'value' into 'column' in 'record' in 'table'.

This method is corresponding to the following ovs-vsctl command:

```
$ ovs-vsctl remove TBL REC COL [KEY=]VALUE
```

**run\_command**(*commands*)

Executes the given commands and sends OVSDB messages.

`commands` must be a list of `os_ken.lib.ovs.vsctl.VSctlCommand`.

The given `timeout` and `exception` when instantiation will be used to call `os_ken.lib.ovs.vsctl.VSctl.run_command`.

**set\_controller**(*controllers*)

Sets the OpenFlow controller address.

This method is corresponding to the following ovs-vsctl command:

```
$ ovs-vsctl set-controller <bridge> <target>...
```

**set\_db\_attribute**(*table, record, column, value, key=None*)

Sets 'value' into 'column' in 'record' in 'table'.

This method is corresponding to the following ovs-vsctl command:

```
$ ovs-vsctl set TBL REC COL[:KEY]=VALUE
```

**set\_qos**(*port\_name, type='linux-htb', max\_rate=None, queues=None*)

Sets a Qos rule and creates Queues on the given port.

**exception** `os_ken.lib.ovs.bridge.OVSBridgeNotFound`(*msg=None, \*\*kwargs*)

## 8.1.5 OpenFlow protocol API Reference

### OpenFlow version independent classes and functions

#### Base class for OpenFlow messages

**class** `os_ken.ofproto.ofproto_parser.MsgBase`(*datapath*)

This is a base class for OpenFlow message classes.

An instance of this class has at least the following attributes.

Attribute	Description
<code>datapath</code>	A <code>os_ken.controller.controller.Datapath</code> instance for this message
<code>version</code>	OpenFlow protocol version
<code>msg_type</code>	Type of OpenFlow message
<code>msg_len</code>	Length of the message
<code>xid</code>	Transaction id
<code>buf</code>	Raw data

#### **\_TYPE**

`_TYPE` class attribute is used to annotate types of attributes.

This type information is used to find an appropriate conversion for a JSON style dictionary.

Currently the following types are implemented.

Type	Description
<code>ascii</code>	US-ASCII
<code>utf-8</code>	UTF-8

Example:

```

_TYPE = {
    'ascii': [
        'hw_addr',
    ],
    'utf-8': [
        'name',
    ]
}

```

**classmethod from\_jsondict**(*dict\_*, *decode\_string*=<function b64decode>, \*\**additional\_args*)

Create an instance from a JSON style dict.

Instantiate this class with parameters specified by the dict.

This method takes the following arguments.

Argument	Description
<i>dict_</i>	A dictionary which describes the parameters. For example, {"Param1": 100, "Param2": 200}
<i>decode_string</i>	(Optional) specify how to decode strings. The default is base64. This argument is used only for attributes which don't have explicit type annotations in <i>_TYPE</i> class attribute.
<i>additional_args</i>	(Optional) Additional kwargs for constructor.

**to\_jsondict**(*encode\_string*=<function b64encode>)

This method returns a JSON style dict to describe this object.

The returned dict is compatible with `json.dumps()` and `json.loads()`.

Suppose `ClassName` object inherits `StringifyMixin`. For an object like the following:

```
ClassName(Param1=100, Param2=200)
```

this method would produce:

```
{ "ClassName": {"Param1": 100, "Param2": 200} }
```

This method takes the following arguments.

Argument	Description
<i>encode_string</i>	(Optional) specify how to encode attributes which has python 'str' type. The default is base64. This argument is used only for attributes which don't have explicit type annotations in <i>_TYPE</i> class attribute.

## Functions

`os_ken.ofproto.ofproto_parser.ofp_msg_from_jsondict`(*dp*, *jsondict*)

This function instantiates an appropriate OpenFlow message class from the given JSON style dictionary. The objects created by following two code fragments are equivalent.

Code A:

```
jsonstr = '{ "OFPSetConfig": { "flags": 0, "miss_send_len": 128 } }'
jsondict = json.loads(jsonstr)
o = ofp_msg_from_jsondict(dp, jsondict)
```

Code B:

```
o = dp.ofproto_parser.OFPSetConfig(flags=0, miss_send_len=128)
```

This function takes the following arguments.

Argument	Description
dp	An instance of <code>os_ken.controller.Datapath</code> .
jsondict	A JSON style dict.

## OpenFlow v1.0 Messages and Structures

### Controller-to-Switch Messages

#### Handshake

**class** `os_ken.ofproto.ofproto_v1_0_parser.OFPFeaturesRequest`(*datapath*)

Features request message

The controller sends a feature request to the switch upon session establishment.

This message is handled by the OSKen framework, so the OSKen application do not need to process this typically.

Example:

```
def send_features_request(self, datapath):
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPFeaturesRequest(datapath)
    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPFeaturesRequest": {}
}
```

**class** `os_ken.ofproto.ofproto_v1_0_parser.OFPSwitchFeatures`(*datapath*,  
*datapath\_id=None*,  
*n\_buffers=None*,  
*n\_tables=None*,  
*capabilities=None*,  
*actions=None*,  
*ports=None*)

Features reply message

The switch responds with a features reply message to a features request.

This message is handled by the OSKen framework, so the OSKen application do not need to process this typically.

Attribute	Description
datapath_id	Datapath unique ID.
n_buffers	Max packets buffered at once.
n_tables	Number of tables supported by datapath.
capabilities	Bitmap of capabilities flag.  OFPC_FLOW_STATS OFPC_TABLE_STATS OFPC_PORT_STATS OFPC_STP OFPC_RESERVED OFPC_IP_REASM OFPC_QUEUE_STATS OFPC_ARP_MATCH_IP
actions	Bitmap of supported OFPAT_*.
ports	List of OFPPhyPort instances.

Example:

```
@set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
def switch_features_handler(self, ev):
    msg = ev.msg

    self.logger.debug('OFPSwitchFeatures received: '
                      'datapath_id=0x%016x n_buffers=%d '
                      'n_tables=%d capabilities=0x%08x ports=%s',
                      msg.datapath_id, msg.n_buffers, msg.n_tables,
                      msg.capabilities, msg.ports)
```

JSON Example:

```
{
  "OFPSwitchFeatures": {
    "actions": 2115,
    "capabilities": 169,
    "datapath_id": 1095522080376,
    "n_buffers": 0,
    "n_tables": 255,
    "ports": {
      "6": {
        "OFPPhyPort": {
          "advertised": 640,
          "config": 0,

```

(continues on next page)

(continued from previous page)

```

        "curr": 648,
        "hw_addr": "f2:0b:a4:7d:f8:ea",
        "name": "Port6",
        "peer": 648,
        "port_no": 6,
        "state": 2,
        "supported": 648
    },
    "7": {
        "OFPPhyPort": {
            "advertised": 640,
            "config": 0,
            "curr": 648,
            "hw_addr": "f2:0b:a4:d0:3f:70",
            "name": "Port7",
            "peer": 648,
            "port_no": 7,
            "state": 16,
            "supported": 648
        }
    }
}

```

## Switch Configuration

`class os_ken.ofproto.ofproto_v1_0_parser.OFPSetConfig(datapath, flags=None, miss_send_len=None)`

Set config request message

The controller sends a set config request message to set configuraion parameters.

Attribute	Description
flags	One of the following configuration flags.  OFPC_FRAG_NORMAL OFPC_FRAG_DROP OFPC_FRAG_REASM OFPC_FRAG_MASK
miss_send_len	Max bytes of new flow that datapath should send to the controller.

Example:

```
def send_set_config(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPSetConfig(datapath, ofp.OFPC_FRAG_NORMAL, 256)
    datapath.send_msg(req)
```

**class** `os_ken.ofproto.ofproto_v1_0_parser.OFPGetConfigRequest(datapath)`

Get config request message

The controller sends a get config request to query configuration parameters in the switch.

Example:

```
def send_get_config_request(self, datapath):
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPGetConfigRequest(datapath)
    datapath.send_msg(req)
```

**class** `os_ken.ofproto.ofproto_v1_0_parser.OFPGetConfigReply(datapath)`

Get config reply message

The switch responds to a configuration request with a get config reply message.

Attribute	Description
flags	One of the following configuration flags.  OFPC_FRAG_NORMAL OFPC_FRAG_DROP OFPC_FRAG_REASM OFPC_FRAG_MASK
miss_send_len	Max bytes of new flow that datapath should send to the controller.

Example:

```
@set_ev_cls(ofp_event.EventOFPGetConfigReply, MAIN_DISPATCHER)
def get_config_reply_handler(self, ev):
    msg = ev.msg
    dp = msg.datapath
    ofp = dp.ofproto

    if msg.flags == ofp.OFPC_FRAG_NORMAL:
        flags = 'NORMAL'
    elif msg.flags == ofp.OFPC_FRAG_DROP:
        flags = 'DROP'
    elif msg.flags == ofp.OFPC_FRAG_REASM:
        flags = 'REASM'
```

(continues on next page)

(continued from previous page)

```

elif msg.flags == ofp.OFPC_FRAG_MASK:
    flags = 'MASK'
else:
    flags = 'unknown'
self.logger.debug('OFPCGetConfigReply received: '
                  'flags=%s miss_send_len=%d',
                  flags, msg.miss_send_len)

```

## Modify State Messages

```

class os_ken.ofproto.ofproto_v1_0_parser.OFPFlowMod(datapath, match=None, cookie=0,
                                                    command=0, idle_timeout=0,
                                                    hard_timeout=0, priority=32768,
                                                    buffer_id=4294967295,
                                                    out_port=65535, flags=0,
                                                    actions=None)

```

Modify Flow entry message

The controller sends this message to modify the flow table.

Attribute	Description
match	Instance of OFPMatch.
cookie	Opaque controller-issued identifier.
command	One of the following values.  OFPFC_ADD OFPFC_MODIFY OFPFC_MODIFY_STRICT OFPFC_DELETE OFPFC_DELETE_STRICT
idle_timeout	Idle time before discarding (seconds).
hard_timeout	Max time before discarding (seconds).
priority	Priority level of flow entry.
buffer_id	Buffered packet to apply to (or 0xffffffff). Not meaningful for OFPFC_DELETE*.
out_port	For OFPFC_DELETE* commands, require matching entries to include this as an output port. A value of OFPP_NONE indicates no restriction.
flags	One of the following values.  OFPFF_SEND_FLOW_REM OFPFF_CHECK_OVERLAP OFPFF_EMERG
actions	List of OFPAction* instance.



Example:

```
def send_flow_mod(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    match = ofp_parser.OFPMatch(in_port=1)
    cookie = 0
    command = ofp.OFPFC_ADD
    idle_timeout = hard_timeout = 0
    priority = 32768
    buffer_id = 0xffffffff
    out_port = ofproto.OFPP_NONE
    flags = 0
    actions = [ofp_parser.OFPActionOutput(ofp.OFPP_NORMAL, 0)]
    req = ofp_parser.OFPFlowMod(
        datapath, match, cookie, command, idle_timeout, hard_timeout,
        priority, buffer_id, out_port, flags, actions)
    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPFlowMod": {
    "actions": [
      {
        "OFPActionOutput": {
          "max_len": 65535,
          "port": 6
        }
      }
    ],
    "buffer_id": 65535,
    "command": 0,
    "cookie": 0,
    "flags": 0,
    "hard_timeout": 0,
    "idle_timeout": 0,
    "match": {
      "OFPMatch": {
        "dl_dst": "f2:0b:a4:7d:f8:ea",
        "dl_src": "00:00:00:00:00:00",
        "dl_type": 0,
        "dl_vlan": 0,
        "dl_vlan_pcp": 0,
        "in_port": 0,
        "nw_dst": "0.0.0.0",
        "nw_proto": 0,
        "nw_src": "0.0.0.0",
        "nw_tos": 0,
        "tp_dst": 0,
```

(continues on next page)

(continued from previous page)

```
        "tp_src": 0,  
        "wildcards": 4194295  
    },  
    },  
    "out_port": 65532,  
    "priority": 123  
}
```

```
class os_ken.ofproto.ofproto_v1_0_parser.OFPPortMod(datapath, port_no=0,  
                                                    hw_addr='00:00:00:00:00:00',  
                                                    config=0, mask=0, advertise=0)
```

Port modification message

The controller send this message to modify the behavior of the port.

Attribute	Description
port_no	Port number to modify.
hw_addr	The hardware address that must be the same as hw_addr of OFPPhyPort of OFPSwitchFeatures.
config	Bitmap of configuration flags.  OFPPC_PORT_DOWN OFPPC_NO_STP OFPPC_NO_RECV OFPPC_NO_RECV_STP OFPPC_NO_FLOOD OFPPC_NO_FWD OFPPC_NO_PACKET_IN
mask	Bitmap of configuration flags above to be changed
advertise	Bitmap of the following flags.  OFPPF_10MB_HD OFPPF_10MB_FD OFPPF_100MB_HD OFPPF_100MB_FD OFPPF_1GB_HD OFPPF_1GB_FD OFPPF_10GB_FD OFPPF_COPPER OFPPF_FIBER OFPPF_AUTONEG OFPPF_PAUSE OFPPF_PAUSE_ASYM

Example:

```
def send_port_mod(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    port_no = 3
    hw_addr = 'fa:c8:e8:76:1d:7e'
    config = 0
    mask = (ofp.OFPPC_PORT_DOWN | ofp.OFPPC_NO_RECV |
            ofp.OFPPC_NO_FWD | ofp.OFPPC_NO_PACKET_IN)
    advertise = (ofp.OFPPF_10MB_HD | ofp.OFPPF_100MB_FD |
                ofp.OFPPF_1GB_FD | ofp.OFPPF_COPPER |
```

(continues on next page)

(continued from previous page)

```

        ofp.OFPPF_AUTONEG | ofp.OFPPF_PAUSE |
        ofp.OFPPF_PAUSE_ASYM)
    req = ofp_parser.OFPPortMod(datapath, port_no, hw_addr, config,
                               mask, advertise)
    datapath.send_msg(req)

```

## Queue Configuration Messages

**class** `os_ken.ofproto.ofproto_v1_0_parser.OFPQueueGetConfigRequest`(*datapath, port*)

Queue configuration request message

Attribute	Description
port	Port to be queried. Should refer to a valid physical port (i.e. < OFPP_MAX).

Example:

```

def send_queue_get_config_request(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPQueueGetConfigRequest(datapath,
                                              ofp.OFPP_NONE)
    datapath.send_msg(req)

```

**class** `os_ken.ofproto.ofproto_v1_0_parser.OFPQueueGetConfigReply`(*datapath*)

Queue configuration reply message

The switch responds with this message to a queue configuration request.

Attribute	Description
port	Port to be queried.
queues	List of OFPPacketQueue instance.

Example:

```

@set_ev_cls(ofp_event.EventOFPQueueGetConfigReply, MAIN_DISPATCHER)
def queue_get_config_reply_handler(self, ev):
    msg = ev.msg

    self.logger.debug('OFPQueueGetConfigReply received: '
                      'port=%s queues=%s',
                      msg.port, msg.queues)

```

## Read State Messages

**class** `os_ken.ofproto.ofproto_v1_0_parser.OFPDescStatsRequest(datapath, flags)`

Description statistics request message

The controller uses this message to query description of the switch.

Attribute	Description
flags	Zero (none yet defined in the spec).

Example:

```
def send_desc_stats_request(self, datapath):
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPDescStatsRequest(datapath)
    datapath.send_msg(req)
```

**class** `os_ken.ofproto.ofproto_v1_0_parser.OFPDescStatsReply(datapath)`

Description statistics reply message

The switch responds with a stats reply that include this message to a description statistics request.

Attribute	Description
mfr_desc	Manufacturer description.
hw_desc	Hardware description.
sw_desc	Software description.
serial_num	Serial number.
dp_desc	Human readable description of datapath.

Example:

```
@set_ev_cls(ofp_event.EventOFPDescStatsReply, MAIN_DISPATCHER)
def desc_stats_reply_handler(self, ev):
    msg = ev.msg
    ofp = msg.datapath.ofproto
    body = ev.msg.body

    self.logger.debug('DescStats: mfr_desc=%s hw_desc=%s sw_desc=%s '
                      'serial_num=%s dp_desc=%s',
                      body.mfr_desc, body.hw_desc, body.sw_desc,
                      body.serial_num, body.dp_desc)
```

**class** `os_ken.ofproto.ofproto_v1_0_parser.OFPFlowStatsRequest(datapath, flags, match, table_id, out_port)`

Individual flow statistics request message

The controller uses this message to query individual flow statistics.

Attribute	Description
flags	Zero (none yet defined in the spec).
match	Instance of <code>OFPMatch</code> .
table_id	ID of table to read (from <code>ofp_table_stats</code> ), <code>0xff</code> for all tables or <code>0xfe</code> for emergency.
out_port	Require matching entries to include this as an output port. A value of <code>OFPP_NONE</code> indicates no restriction.

Example:

```
def send_flow_stats_request(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    match = ofp_parser.OFPMatch(in_port=1)
    table_id = 0xff
    out_port = ofp.OFPP_NONE
    req = ofp_parser.OFPFlowStatsRequest(
        datapath, 0, match, table_id, out_port)

    datapath.send_msg(req)
```

**class** `os_ken.ofproto.ofproto_v1_0_parser.OFPFlowStatsReply(datapath)`

Individual flow statistics reply message

The switch responds with a stats reply that include this message to an individual flow statistics request.

Attribute	Description
table_id	ID of table flow came from.
match	Instance of <code>OFPMatch</code> .
duration_sec	Time flow has been alive in seconds.
duration_nsec	Time flow has been alive in nanoseconds beyond <code>duration_sec</code> .
priority	Priority of the entry. Only meaningful when this is not an exact-match entry.
idle_timeout	Number of seconds idle before expiration.
hard_timeout	Number of seconds before expiration.
cookie	Opaque controller-issued identifier.
packet_count	Number of packets in flow.
byte_count	Number of bytes in flow.
actions	List of <code>OFPACTION*</code> instance

Example:

```
@set_ev_cls(ofp_event.EventOFPFlowStatsReply, MAIN_DISPATCHER)
def flow_stats_reply_handler(self, ev):
    msg = ev.msg
    ofp = msg.datapath.ofproto
```

(continues on next page)

(continued from previous page)

```

body = ev.msg.body

flows = []
for stat in body:
    flows.append('table_id=%s match=%s '
                'duration_sec=%d duration_nsec=%d '
                'priority=%d '
                'idle_timeout=%d hard_timeout=%d '
                'cookie=%d packet_count=%d byte_count=%d '
                'actions=%s' %
                (stat.table_id, stat.match,
                 stat.duration_sec, stat.duration_nsec,
                 stat.priority,
                 stat.idle_timeout, stat.hard_timeout,
                 stat.cookie, stat.packet_count, stat.byte_count,
                 stat.actions))
self.logger.debug('FlowStats: %s', flows)

```

```

class os_ken.ofproto.ofproto_v1_0_parser.OFPAggregateStatsRequest(datapath, flags,
                                                                    match, table_id,
                                                                    out_port)

```

Aggregate flow statistics request message

The controller uses this message to query aggregate flow statistics.

At-tribute	Description
flags	Zero (none yet defined in the spec).
match	Fields to match.
table_id	ID of table to read (from <code>ofp_table_stats</code> ) 0xff for all tables or 0xfe for emergency.
out_port	Require matching entries to include this as an output port. A value of <code>OFPP_NONE</code> indicates no restriction.

Example:

```

def send_aggregate_stats_request(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    cookie = cookie_mask = 0
    match = ofp_parser.OFPMatch(in_port=1)
    req = ofp_parser.OFPAggregateStatsRequest(
        datapath, 0, match, 0xff, ofp.OFPP_NONE)

    datapath.send_msg(req)

```

```

class os_ken.ofproto.ofproto_v1_0_parser.OFPAggregateStatsReply(datapath)
    Aggregate flow statistics reply message

```

The switch responds with a stats reply that include this message to an aggregate flow statistics request.

Attribute	Description
packet_count	Number of packets in flows.
byte_count	Number of bytes in flows.
flow_count	Number of flows.

Example:

```
@set_ev_cls(ofp_event.EventOFPAggregateStatsReply, MAIN_DISPATCHER)
def aggregate_stats_reply_handler(self, ev):
    msg = ev.msg
    ofp = msg.datapath.ofproto
    body = ev.msg.body

    self.logger.debug('AggregateStats: packet_count=%d byte_count=%d '
                      'flow_count=%d',
                      body.packet_count, body.byte_count,
                      body.flow_count)
```

**class** `os_ken.ofproto.ofproto_v1_0_parser.OFPTableStatsRequest`(*datapath, flags*)

Table statistics request message

The controller uses this message to query flow table statistics.

Attribute	Description
flags	Zero (none yet defined in the spec).

Example:

```
def send_table_stats_request(self, datapath):
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPTableStatsRequest(datapath)
    datapath.send_msg(req)
```

**class** `os_ken.ofproto.ofproto_v1_0_parser.OFPTableStatsReply`(*datapath*)

Table statistics reply message

The switch responds with a stats reply that include this message to a table statistics request.



Attribute	Description
table_id	ID of table.
name	table name.
wildcards	Bitmap of OFPFW_* wildcards that are supported by the table.
max_entries	Max number of entries supported
active_count	Number of active entries
lookup_count	Number of packets looked up in table
matched_count	Number of packets that hit table

Example:

```
@set_ev_cls(ofp_event.EventOFPTableStatsReply, MAIN_DISPATCHER)
def stats_reply_handler(self, ev):
    msg = ev.msg
    ofp = msg.datapath.ofproto
    body = ev.msg.body

    tables = []
    for stat in body:
        tables.append('table_id=%d name=%s wildcards=0x%02x '
                      'max_entries=%d active_count=%d '
                      'lookup_count=%d matched_count=%d' %
                      (stat.table_id, stat.name, stat.wildcards,
                       stat.max_entries, stat.active_count,
                       stat.lookup_count, stat.matched_count))
    self.logger.debug('TableStats: %s', tables)
```

```
class os_ken.ofproto.ofproto_v1_0_parser.OFPPortStatsRequest(datapath, flags,
                                                             port_no)
```

Port statistics request message

The controller uses this message to query information about ports statistics.

Attribute	Description
flags	Zero (none yet defined in the spec).
port_no	Port number to read (OFPP_NONE to all ports).

Example:

```
def send_port_stats_request(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPPortStatsRequest(datapath, 0, ofp.OFPP_ANY)
    datapath.send_msg(req)
```

```
class os_ken.ofproto.ofproto_v1_0_parser.OFPPortStatsReply(datapath)
```

Port statistics reply message

The switch responds with a stats reply that include this message to a port statistics request.

Attribute	Description
port_no	Port number.
rx_packets	Number of received packets.
tx_packets	Number of transmitted packets.
rx_bytes	Number of received bytes.
tx_bytes	Number of transmitted bytes.
rx_dropped	Number of packets dropped by RX.
tx_dropped	Number of packets dropped by TX.
rx_errors	Number of receive errors.
tx_errors	Number of transmit errors.
rx_frame_err	Number of frame alignment errors.
rx_over_err	Number of packet with RX overrun.
rx_crc_err	Number of CRC errors.
collisions	Number of collisions.

Example:

```
@set_ev_cls(ofp_event.EventOFPPortStatsReply, MAIN_DISPATCHER)
def port_stats_reply_handler(self, ev):
    msg = ev.msg
    ofp = msg.datapath.ofproto
    body = ev.msg.body

    ports = []
    for stat in body:
        ports.append('port_no=%d '
                    'rx_packets=%d tx_packets=%d '
                    'rx_bytes=%d tx_bytes=%d '
                    'rx_dropped=%d tx_dropped=%d '
                    'rx_errors=%d tx_errors=%d '
                    'rx_frame_err=%d rx_over_err=%d rx_crc_err=%d '
                    'collisions=%d' %
                    (stat.port_no,
                     stat.rx_packets, stat.tx_packets,
                     stat.rx_bytes, stat.tx_bytes,
                     stat.rx_dropped, stat.tx_dropped,
                     stat.rx_errors, stat.tx_errors,
                     stat.rx_frame_err, stat.rx_over_err,
                     stat.rx_crc_err, stat.collisions))
    self.logger.debug('PortStats: %s', ports)
```

```
class os_ken.ofproto.ofproto_v1_0_parser.OFPQueueStatsRequest(datapath, flags,
                                                                port_no, queue_id)
```

Queue statistics request message

The controller uses this message to query queue statistics.

Attribute	Description
flags	Zero (none yet defined in the spec)
port_no	Port number to read (All ports if OFPT_ALL).
queue_id	ID of queue to read (All queues if OFPQ_ALL).

Example:

```
def send_queue_stats_request(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPQueueStatsRequest(datapath, 0, ofp.OFPT_ALL,
                                           ofp.OFPQ_ALL)

    datapath.send_msg(req)
```

**class** os\_ken.ofproto.ofproto\_v1\_0\_parser.OFPQueueStatsReply(datapath)

Queue statistics reply message

The switch responds with a stats reply that include this message to an aggregate flow statistics request.

Attribute	Description
port_no	Port number.
queue_id	ID of queue.
tx_bytes	Number of transmitted bytes.
tx_packets	Number of transmitted packets.
tx_errors	Number of packets dropped due to overrun.

Example:

```
@set_ev_cls(ofp_event.EventOFPQueueStatsReply, MAIN_DISPATCHER)
def stats_reply_handler(self, ev):
    msg = ev.msg
    ofp = msg.datapath.ofproto
    body = ev.msg.body

    queues = []
    for stat in body:
        queues.append('port_no=%d queue_id=%d '
                     'tx_bytes=%d tx_packets=%d tx_errors=%d ' %
                     (stat.port_no, stat.queue_id,
                      stat.tx_bytes, stat.tx_packets, stat.tx_errors))
    self.logger.debug('QueueStats: %s', queues)
```

**class** os\_ken.ofproto.ofproto\_v1\_0\_parser.OFPVendorStatsRequest(datapath, flags, vendor, specific\_data=None)

Vendor statistics request message

The controller uses this message to query vendor-specific information of a switch.

```
class os_ken.ofproto.ofproto_v1_0_parser.OFPVendorStatsReply(datapath)
```

Vendor statistics reply message

The switch responds with a stats reply that include this message to an vendor statistics request.

## Send Packet Message

```
class os_ken.ofproto.ofproto_v1_0_parser.OFPPacketOut(datapath, buffer_id=None,
                                                    in_port=None, actions=None,
                                                    data=None)
```

Packet-Out message

The controller uses this message to send a packet out through the switch.

Attribute	Description
buffer_id	ID assigned by datapath (0xffffffff if none).
in_port	Packet's input port (OFPP_NONE if none).
actions	ist of OFPAction* instance.
data	Packet data of a binary type value or an instances of packet.Packet.

Example:

```
def send_packet_out(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    buffer_id = 0xffffffff
    in_port = ofp.OFPP_NONE
    actions = [ofp_parser.OFPActionOutput(ofp.OFPP_FLOOD, 0)]
    req = ofp_parser.OFPPacketOut(datapath, buffer_id,
                                  in_port, actions)

    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPPacketOut": {
    "actions": [
      {
        "OFPActionOutput": {
          "max_len": 65535,
          "port": 65532
        }
      }
    ],
    "buffer_id": 4294967295,
    "data":
    ↪ "8guk0D9w8gukffjqCABFAABU+BoAAP8Br4sKAAABCgAAAggAAgj3YAAAMdYCAAAAAACrjS0xAAAAABAREhM
    ↪ ",
```

(continues on next page)

(continued from previous page)

```

        "in_port": 65533
    }
}

```

## Barrier Message

**class** `os_ken.ofproto.ofproto_v1_0_parser.OFPBarrierRequest(datapath)`

Barrier request message

The controller sends this message to ensure message dependencies have been met or receive notifications for completed operations.

Example:

```

def send_barrier_request(self, datapath):
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPBarrierRequest(datapath)
    datapath.send_msg(req)

```

**class** `os_ken.ofproto.ofproto_v1_0_parser.OFPBarrierReply(datapath)`

Barrier reply message

The switch responds with this message to a barrier request.

Example:

```

@set_ev_cls(ofp_event.EventOFPBarrierReply, MAIN_DISPATCHER)
def barrier_reply_handler(self, ev):
    self.logger.debug('OFPBarrierReply received')

```

## Asynchronous Messages

### Packet-In Message

**class** `os_ken.ofproto.ofproto_v1_0_parser.OFPPacketIn(datapath, buffer_id=None, total_len=None, in_port=None, reason=None, data=None)`

Packet-In message

The switch sends the packet that received to the controller by this message.

Attribute	Description
buffer_id	ID assigned by datapath.
total_len	Full length of frame.
in_port	Port on which frame was received.
reason	Reason packet is being sent.  OFPR_NO_MATCH OFPR_ACTION OFPR_INVALID_TTL
data	Ethernet frame.

Example:

```
@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def packet_in_handler(self, ev):
    msg = ev.msg
    dp = msg.datapath
    ofp = dp.ofproto

    if msg.reason == ofp.OFPR_NO_MATCH:
        reason = 'NO MATCH'
    elif msg.reason == ofp.OFPR_ACTION:
        reason = 'ACTION'
    elif msg.reason == ofp.OFPR_INVALID_TTL:
        reason = 'INVALID TTL'
    else:
        reason = 'unknown'

    self.logger.debug('OFPPacketIn received: '
                      'buffer_id=%x total_len=%d in_port=%d, '
                      'reason=%s data=%s',
                      msg.buffer_id, msg.total_len, msg.in_port,
                      reason, utils.hex_array(msg.data))
```

JSON Example:

```
{
  "OFPPacketIn": {
    "buffer_id": 2,
    "data": "/////////8gukffjqCAYAAQgABgQAAfILpH346goAAAEAAAAAAAAAKAAAD",
    "in_port": 99,
    "reason": 1,
    "total_len": 42
  }
}
```

## Flow Removed Message

**class** `os_ken.ofproto.ofproto_v1_0_parser.OFPFlowRemoved(datapath)`

Flow removed message

When flow entries time out or are deleted, the switch notifies controller with this message.

Attribute	Description
<code>match</code>	Instance of <code>OFPMatch</code> .
<code>cookie</code>	Opaque controller-issued identifier.
<code>priority</code>	Priority level of flow entry.
<code>reason</code>	One of the following values.  <code>OFPRR_IDLE_TIMEOUT</code> <code>OFPRR_HARD_TIMEOUT</code> <code>OFPRR_DELETE</code>
<code>duration_sec</code>	Time flow was alive in seconds.
<code>duration_nsec</code>	Time flow was alive in nanoseconds beyond <code>duration_sec</code> .
<code>idle_timeout</code>	Idle timeout from original flow mod.
<code>packet_count</code>	Number of packets that was associated with the flow.
<code>byte_count</code>	Number of bytes that was associated with the flow.

Example:

```
@set_ev_cls(ofp_event.EventOFPFlowRemoved, MAIN_DISPATCHER)
def flow_removed_handler(self, ev):
    msg = ev.msg
    dp = msg.datapath
    ofp = dp.ofproto

    if msg.reason == ofp.OFPRR_IDLE_TIMEOUT:
        reason = 'IDLE TIMEOUT'
    elif msg.reason == ofp.OFPRR_HARD_TIMEOUT:
        reason = 'HARD TIMEOUT'
    elif msg.reason == ofp.OFPRR_DELETE:
        reason = 'DELETE'
    elif msg.reason == ofp.OFPRR_GROUP_DELETE:
        reason = 'GROUP DELETE'
    else:
        reason = 'unknown'

    self.logger.debug('OFPFlowRemoved received: '
                      'match=%s cookie=%d priority=%d reason=%s '
                      'duration_sec=%d duration_nsec=%d '
                      'idle_timeout=%d packet_count=%d byte_count=%d',
                      msg.match, msg.cookie, msg.priority, reason,
                      msg.duration_sec, msg.duration_nsec,
                      msg.idle_timeout, msg.packet_count, msg.byte_count)
```

(continues on next page)

(continued from previous page)

```
msg.duration_sec, msg.duration_nsec,
msg.idle_timeout, msg.packet_count,
msg.byte_count)
```

## Port Status Message

```
class os_ken.ofproto.ofproto_v1_0_parser.OFPPortStatus(datapath, reason=None,
                                                         desc=None)
```

Port status message

The switch notifies controller of change of ports.

Attribute	Description
reason	One of the following values.  OFPPR_ADD OFPPR_DELETE OFPPR_MODIFY
desc	instance of OFPPhyPort

Example:

```
@set_ev_cls(ofp_event.EventOFPPortStatus, MAIN_DISPATCHER)
def port_status_handler(self, ev):
    msg = ev.msg
    dp = msg.datapath
    ofp = dp.ofproto

    if msg.reason == ofp.OFPPR_ADD:
        reason = 'ADD'
    elif msg.reason == ofp.OFPPR_DELETE:
        reason = 'DELETE'
    elif msg.reason == ofp.OFPPR_MODIFY:
        reason = 'MODIFY'
    else:
        reason = 'unknown'

    self.logger.debug('OFPPortStatus received: reason=%s desc=%s',
                      reason, msg.desc)
```

## Error Message

```
class os_ken.ofproto.ofproto_v1_0_parser.OFPErrormsg(datapath, type_=None,
                                                         code=None, data=None)
```

Error message

The switch notifies controller of problems by this message.



Attribute	Description
type	High level type of error
code	Details depending on the type
data	Variable length data depending on the type and code

type attribute corresponds to type\_ parameter of \_\_init\_\_.

Types and codes are defined in `os_ken.ofproto.ofproto`.

Type	Code
OFPET_HELLO_FAILED	OFPHFC_*
OFPET_BAD_REQUEST	OFPBRC_*
OFPET_BAD_ACTION	OFPBAC_*
OFPET_FLOW_MOD_FAILED	OFPFMFC_*
OFPET_PORT_MOD_FAILED	OFPPMFC_*
OFPET_QUEUE_OP_FAILED	OFPQOFC_*

Example:

```
@set_ev_cls(ofp_event.EventOFPErrormsg,
            [HANDSHAKE_DISPATCHER, CONFIG_DISPATCHER, MAIN_DISPATCHER])
def error_msg_handler(self, ev):
    msg = ev.msg

    self.logger.debug('OFPErrormsg received: type=0x%02x code=0x%02x '
                      'message=%s',
                      msg.type, msg.code, utils.hex_array(msg.data))
```

## Symmetric Messages

### Hello

`class os_ken.ofproto.ofproto_v1_0_parser.OFPHello(datapath)`

Hello message

When connection is started, the hello message is exchanged between a switch and a controller.

This message is handled by the OSKen framework, so the OSKen application do not need to process this typically.

### Echo Request

`class os_ken.ofproto.ofproto_v1_0_parser.OFPEchoRequest(datapath, data=None)`

Echo request message

This message is handled by the OSKen framework, so the OSKen application do not need to process this typically.

Attribute	Description
data	An arbitrary length data.

Example:

```
def send_echo_request(self, datapath, data):
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPEchoRequest(datapath, data)
    datapath.send_msg(req)
```

## Echo Reply

**class** `os_ken.ofproto.ofproto_v1_0_parser.OFPEchoReply(datapath, data=None)`

Echo reply message

This message is handled by the OSKen framework, so the OSKen application do not need to process this typically.

Attribute	Description
data	An arbitrary length data.

Example:

```
@set_ev_cls(ofp_event.EventOFPEchoReply,
            [HANDSHAKE_DISPATCHER, CONFIG_DISPATCHER, MAIN_DISPATCHER])
def echo_reply_handler(self, ev):
    self.logger.debug('OFPEchoReply received: data=%s',
                      utils.hex_array(ev.msg.data))
```

## Vendor

**class** `os_ken.ofproto.ofproto_v1_0_parser.OFPVendor(datapath)`

Vendor message

The controller send this message to send the vendor-specific information to a switch.

## Port Structures

**class** `os_ken.ofproto.ofproto_v1_0_parser.OFPPhyPort(port_no, hw_addr, name, config, state, curr, advertised, supported, peer)`

Description of a port

Attribute	Description
port_no	Port number and it uniquely identifies a port within a switch.
hw_addr	MAC address for the port.
name	Null-terminated string containing a human-readable name for the interface.
config	Bitmap of port configuration flags.  OFPPC_PORT_DOWN OFPPC_NO_STP OFPPC_NO_RECV OFPPC_NO_RECV_STP OFPPC_NO_FLOOD OFPPC_NO_FWD OFPPC_NO_PACKET_IN
state	Bitmap of port state flags.  OFPPS_LINK_DOWN OFPPS_STP_LISTEN OFPPS_STP_LEARN OFPPS_STP_FORWARD OFPPS_STP_BLOCK OFPPS_STP_MASK
curr	Current features.
advertised	Features being advertised by the port.
supported	Features supported by the port.
peer	Features advertised by peer.

## Flow Match Structure

```
class os_ken.ofproto.ofproto_v1_0_parser.OFPMatch(wildcards=None, in_port=None,
dl_src=None, dl_dst=None,
dl_vlan=None, dl_vlan_pcp=None,
dl_type=None, nw_tos=None,
nw_proto=None, nw_src=None,
nw_dst=None, tp_src=None,
tp_dst=None, nw_src_mask=32,
nw_dst_mask=32)
```

Flow Match Structure

This class is implementation of the flow match structure having compose/query API.

Attribute	Description
wildcards (match fields)	Wildcard fields. For the available match fields, please refer to the following.

Argument	Value	Description
in_port	Integer 16bit	Switch input port.
dl_src	MAC address	Ethernet source address.
dl_dst	MAC address	Ethernet destination address.
dl_vlan	Integer 16bit	Input VLAN id.
dl_vlan_pcp	Integer 8bit	Input VLAN priority.
dl_type	Integer 16bit	Ethernet frame type.
nw_tos	Integer 8bit	IP ToS (actually DSCP field, 6 bits).
nw_proto	Integer 8bit	IP protocol or lower 8 bits of ARP opcode.
nw_src	IPv4 address	IP source address.
nw_dst	IPv4 address	IP destination address.
tp_src	Integer 16bit	TCP/UDP source port.
tp_dst	Integer 16bit	TCP/UDP destination port.
nw_src_mask	Integer 8bit	IP source address mask specified as IPv4 address prefix.
nw_dst_mask	Integer 8bit	IP destination address mask specified as IPv4 address prefix.

Example:

```
>>> # compose
>>> match = parser.OFPMatch(
...     in_port=1,
...     dl_type=0x0800,
...     dl_src='aa:bb:cc:dd:ee:ff',
...     nw_src='192.168.0.1')
>>> # query
>>> if 'nw_src' in match:
...     print match['nw_src']
...
'192.168.0.1'
```

## Action Structures

```
class os_ken.ofproto.ofproto_v1_0_parser.OFPActionHeader(type_, len_)
```

```
class os_ken.ofproto.ofproto_v1_0_parser.OFPAction
```

```
class os_ken.ofproto.ofproto_v1_0_parser.OFPActionOutput(port, max_len=65509)
```

Output action

This action indicates output a packet to the switch port.

Attribute	Description
port	Output port.
max_len	Max length to send to controller.

### Note::

The reason of this magic number (0xffe5) is because there is no good constant in of1.0. The same value as OFPCML\_MAX of 1.2 and of1.3 is used.

**class** `os_ken.ofproto.ofproto_v1_0_parser.OFPActionVlanVid(vlan_vid)`

Set the 802.1q VLAN id action

This action indicates the 802.1q VLAN id to be set.

Attribute	Description
<code>vlan_vid</code>	VLAN id.

**class** `os_ken.ofproto.ofproto_v1_0_parser.OFPActionVlanPcp(vlan_pcp)`

Set the 802.1q priority action

This action indicates the 802.1q priority to be set.

Attribute	Description
<code>vlan_pcp</code>	VLAN priority.

**class** `os_ken.ofproto.ofproto_v1_0_parser.OFPActionStripVlan`

Strip the 802.1q header action

This action indicates the 802.1q priority to be striped.

**class** `os_ken.ofproto.ofproto_v1_0_parser.OFPActionDlAddr(dl_addr)`

**class** `os_ken.ofproto.ofproto_v1_0_parser.OFPActionSetDlSrc(dl_addr)`

Set the ethernet source address action

This action indicates the ethernet source address to be set.

Attribute	Description
<code>dl_addr</code>	Ethernet address.

**class** `os_ken.ofproto.ofproto_v1_0_parser.OFPActionSetDlDst(dl_addr)`

Set the ethernet destination address action

This action indicates the ethernet destination address to be set.

Attribute	Description
<code>dl_addr</code>	Ethernet address.

**class** `os_ken.ofproto.ofproto_v1_0_parser.OFPActionNwAddr(nw_addr)`

**class** `os_ken.ofproto.ofproto_v1_0_parser.OFPActionSetNwSrc(nw_addr)`

Set the IP source address action

This action indicates the IP source address to be set.

Attribute	Description
<code>nw_addr</code>	IP address.

**class** `os_ken.ofproto.ofproto_v1_0_parser.OFPActionSetNwDst(nw_addr)`

Set the IP destination address action

This action indicates the IP destination address to be set.

Attribute	Description
<code>nw_addr</code>	IP address.

**class** `os_ken.ofproto.ofproto_v1_0_parser.OFPActionSetNwTos(tos)`

Set the IP ToS action

This action indicates the IP ToS (DSCP field, 6 bits) to be set.

Attribute	Description
<code>tos</code>	IP ToS (DSCP field, 6 bits).

**class** `os_ken.ofproto.ofproto_v1_0_parser.OFPActionTpPort(tp)`

**class** `os_ken.ofproto.ofproto_v1_0_parser.OFPActionSetTpSrc(tp)`

Set the TCP/UDP source port action

This action indicates the TCP/UDP source port to be set.

Attribute	Description
<code>tp</code>	TCP/UDP port.

**class** `os_ken.ofproto.ofproto_v1_0_parser.OFPActionSetTpDst(tp)`

Set the TCP/UDP destination port action

This action indicates the TCP/UDP destination port to be set.

Attribute	Description
<code>tp</code>	TCP/UDP port.

**class** `os_ken.ofproto.ofproto_v1_0_parser.OFPActionEnqueue(port, queue_id)`

Output to queue action

This action indicates send packets to given queue on port.

Attribute	Description
<code>port</code>	Port that queue belongs.
<code>queue_id</code>	Where to enqueue the packets.

**class** `os_ken.ofproto.ofproto_v1_0_parser.OFPActionVendor(vendor=None)`

Vendor action

This action is an extensible action for the vendor.

## OpenFlow v1.2 Messages and Structures

### Controller-to-Switch Messages

#### Handshake

**class** os\_ken.ofproto.ofproto\_v1\_2\_parser.OFPFeaturesRequest(*datapath*)

Features request message

The controller sends a feature request to the switch upon session establishment.

This message is handled by the OSKen framework, so the OSKen application do not need to process this typically.

Example:

```
def send_features_request(self, datapath):
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPFeaturesRequest(datapath)
    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPFeaturesRequest": {}
}
```

**class** os\_ken.ofproto.ofproto\_v1\_2\_parser.OFPSwitchFeatures(*datapath*,  
*datapath\_id=None*,  
*n\_buffers=None*,  
*n\_tables=None*,  
*capabilities=None*,  
*ports=None*)

Features reply message

The switch responds with a features reply message to a features request.

This message is handled by the OSKen framework, so the OSKen application do not need to process this typically.

Example:

```
@set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
def switch_features_handler(self, ev):
    msg = ev.msg

    self.logger.debug('OFPSwitchFeatures received: '
                      'datapath_id=0x%016x n_buffers=%d '
                      'n_tables=%d capabilities=0x%08x ports=%s',
                      msg.datapath_id, msg.n_buffers, msg.n_tables,
                      msg.capabilities, msg.ports)
```

JSON Example:

```

{
  "OFPSwitchFeatures": {
    "capabilities": 79,
    "datapath_id": 9210263729383,
    "n_buffers": 0,
    "n_tables": 255,
    "ports": {
      "6": {
        "OFPPort": {
          "advertised": 10240,
          "config": 0,
          "curr": 10248,
          "curr_speed": 5000,
          "hw_addr": "f2:0b:a4:7d:f8:ea",
          "max_speed": 5000,
          "name": "Port6",
          "peer": 10248,
          "port_no": 6,
          "state": 4,
          "supported": 10248
        }
      },
      "7": {
        "OFPPort": {
          "advertised": 10240,
          "config": 0,
          "curr": 10248,
          "curr_speed": 5000,
          "hw_addr": "f2:0b:a4:d0:3f:70",
          "max_speed": 5000,
          "name": "Port7",
          "peer": 10248,
          "port_no": 7,
          "state": 4,
          "supported": 10248
        }
      }
    }
  }
}

```

## Switch Configuration

`class os_ken.ofproto.ofproto_v1_2_parser.OFPSetConfig(datapath, flags=0, miss_send_len=0)`

Set config request message

The controller sends a set config request message to set configuraion parameters.



Attribute	Description
flags	One of the following configuration flags.  OFPC_FRAG_NORMAL OFPC_FRAG_DROP OFPC_FRAG_REASM OFPC_FRAG_MASK OFPC_INVALID_TTL_TO_CONTROLLER
miss_send_len	Max bytes of new flow that datapath should send to the controller

Example:

```
def send_set_config(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPSetConfig(datapath, ofp.OFPC_FRAG_NORMAL, 256)
    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPSetConfig": {
    "flags": 0,
    "miss_send_len": 128
  }
}
```

**class** os\_ken.ofproto.ofproto\_v1\_2\_parser.OFPGetConfigRequest(datapath)

Get config request message

The controller sends a get config request to query configuration parameters in the switch.

Example:

```
def send_get_config_request(self, datapath):
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPGetConfigRequest(datapath)
    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPGetConfigRequest": {}
}
```

**class** os\_ken.ofproto.ofproto\_v1\_2\_parser.OFPGetConfigReply(datapath, flags=None, miss\_send\_len=None)

Get config reply message

The switch responds to a configuration request with a get config reply message.

Attribute	Description
flags	One of the following configuration flags.  OFPC_FRAG_NORMAL OFPC_FRAG_DROP OFPC_FRAG_REASM OFPC_FRAG_MASK OFPC_INVALID_TTL_TO_CONTROLLER
miss_send_len	Max bytes of new flow that datapath should send to the controller

Example:

```
@set_ev_cls(ofp_event.EventOFPCGetConfigReply, MAIN_DISPATCHER)
def get_config_reply_handler(self, ev):
    msg = ev.msg
    dp = msg.datapath
    ofp = dp.ofproto

    if msg.flags == ofp.OFPC_FRAG_NORMAL:
        flags = 'NORMAL'
    elif msg.flags == ofp.OFPC_FRAG_DROP:
        flags = 'DROP'
    elif msg.flags == ofp.OFPC_FRAG_REASM:
        flags = 'REASM'
    elif msg.flags == ofp.OFPC_FRAG_MASK:
        flags = 'MASK'
    elif msg.flags == ofp.OFPC_INVALID_TTL_TO_CONTROLLER:
        flags = 'INVALID TTL TO CONTROLLER'
    else:
        flags = 'unknown'
    self.logger.debug('OFPCGetConfigReply received: '
                      'flags=%s miss_send_len=%d',
                      flags, msg.miss_send_len)
```

JSON Example:

```
{
  "OFPCGetConfigReply": {
    "flags": 0,
    "miss_send_len": 128
  }
}
```

## Flow Table Configuration

**class** `os_ken.ofproto.ofproto_v1_2_parser.OFPTTableMod(datapath, table_id, config)`

Flow table configuration message

The controller sends this message to configure table state.

Attribute	Description
<code>table_id</code>	ID of the table (OFPTT_ALL indicates all tables)
<code>config</code>	Bitmap of the following flags.  OFPTC_TABLE_MISS_CONTROLLER OFPTC_TABLE_MISS_CONTINUE OFPTC_TABLE_MISS_DROP OFPTC_TABLE_MISS_MASK

Example:

```
def send_table_mod(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPTTableMod(datapath, ofp.OFPTT_ALL,
                                   ofp.OFPTC_TABLE_MISS_DROP)

    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPTTableMod": {
    "config": 0,
    "table_id": 255
  }
}
```

## Modify State Messages

**class** `os_ken.ofproto.ofproto_v1_2_parser.OFPFlowMod(datapath, cookie=0, cookie_mask=0, table_id=0, command=0, idle_timeout=0, hard_timeout=0, priority=0, buffer_id=4294967295, out_port=0, out_group=0, flags=0, match=None, instructions=None)`

Modify Flow entry message

The controller sends this message to modify the flow table.

Attribute	Description
cookie	Opaque controller-issued identifier
cookie_mask	Mask used to restrict the cookie bits that must match when the command is OFPFC_MODIFY* or OFPFC_DELETE*
table_id	ID of the table to put the flow in
command	One of the following values.  OFPFC_ADD OFPFC_MODIFY OFPFC_MODIFY_STRICT OFPFC_DELETE OFPFC_DELETE_STRICT
idle_timeout	Idle time before discarding (seconds)
hard_timeout	Max time before discarding (seconds)
priority	Priority level of flow entry
buffer_id	Buffered packet to apply to (or OFP_NO_BUFFER)
out_port	For OFPFC_DELETE* commands, require matching entries to include this as an output port
out_group	For OFPFC_DELETE* commands, require matching entries to include this as an output group
flags	One of the following values.  OFPFF_SEND_FLOW_REM OFPFF_CHECK_OVERLAP OFPFF_RESET_COUNTS
match	Instance of OFPMatch
instructions	list of OFPInstruction* instance

Example:

```
def send_flow_mod(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    cookie = cookie_mask = 0
    table_id = 0
    idle_timeout = hard_timeout = 0
    priority = 32768
    buffer_id = ofp.OFP_NO_BUFFER
    match = ofp_parser.OFPMatch(in_port=1, eth_dst='ff:ff:ff:ff:ff:ff')
    actions = [ofp_parser.OFPActionOutput(ofp.OFPP_NORMAL, 0)]
    inst = [ofp_parser.OFPInstructionActions(ofp.OFPIT_APPLY_ACTIONS,
```

(continues on next page)

(continued from previous page)

```

                                actions)]
req = ofp_parser.OFPFlowMod(datapath, cookie, cookie_mask,
                             table_id, ofp.OFPFC_ADD,
                             idle_timeout, hard_timeout,
                             priority, buffer_id,
                             ofp.OFPP_ANY, ofp.OFPG_ANY,
                             ofp.OFPFF_SEND_FLOW_REM,
                             match, inst)

datapath.send_msg(req)

```

**JSON Example:**

```

{
  "OFPFlowMod": {
    "buffer_id": 65535,
    "command": 0,
    "cookie": 0,
    "cookie_mask": 0,
    "flags": 0,
    "hard_timeout": 0,
    "idle_timeout": 0,
    "instructions": [
      {
        "OFPIInstructionActions": {
          "actions": [
            {
              "OFPActionSetField": {
                "field": {
                  "OXMTlv": {
                    "field": "vlan_vid",
                    "mask": null,
                    "value": 258
                  }
                }
              },
              "len": 16,
              "type": 25
            }
          ],
          "OFPAActionOutput": {
            "len": 16,
            "max_len": 65535,
            "port": 6,
            "type": 0
          }
        },
        "len": 40,
        "type": 3
      }
    ]
  }
}

```

(continues on next page)

(continued from previous page)

```

    },
    {
      "OFPIInstructionActions": {
        "actions": [
          {
            "OFPActionSetField": {
              "field": {
                "OXMTlv": {
                  "field": "eth_src",
                  "mask": null,
                  "value": "01:02:03:04:05:06"
                }
              }
            },
            "len": 16,
            "type": 25
          }
        ],
        "len": 24,
        "type": 4
      }
    },
    "match": {
      "OFPMatch": {
        "length": 14,
        "oxm_fields": [
          {
            "OXMTlv": {
              "field": "eth_dst",
              "mask": null,
              "value": "f2:0b:a4:7d:f8:ea"
            }
          }
        ],
        "type": 1
      }
    },
    "out_group": 4294967295,
    "out_port": 4294967295,
    "priority": 123,
    "table_id": 1
  }
}

```

```

{
  "OFPPFlowMod": {
    "buffer_id": 65535,
    "command": 0,

```

(continues on next page)

(continued from previous page)

```

"cookie": 0,
"cookie_mask": 0,
"flags": 0,
"hard_timeout": 0,
"idle_timeout": 0,
"instructions": [
  {
    "OFPIinstructionGotoTable": {
      "len": 8,
      "table_id": 1,
      "type": 1
    }
  }
],
"match": {
  "OFPMatch": {
    "length": 22,
    "oxm_fields": [
      {
        "OXMTlv": {
          "field": "in_port",
          "mask": null,
          "value": 6
        }
      },
      {
        "OXMTlv": {
          "field": "eth_src",
          "mask": null,
          "value": "f2:0b:a4:7d:f8:ea"
        }
      }
    ]
  },
  "type": 1
}
},
"out_group": 4294967295,
"out_port": 4294967295,
"priority": 123,
"table_id": 0
}
}

```

**class** `os_ken.ofproto.ofproto_v1_2_parser.OFPGroupMod`(*datapath*, *command=0*, *type\_=0*, *group\_id=0*, *buckets=None*)

Modify group entry message

The controller sends this message to modify the group table.

Attribute	Description
command	One of the following values.  OFPGC_ADD OFPGC_MODIFY OFPGC_DELETE
type	One of the following values.  OFPGT_ALL OFPGT_SELECT OFPGT_INDIRECT OFPGT_FF
group_id	Group identifier
buckets	list of OFPBucket

type attribute corresponds to type\_ parameter of \_\_init\_\_.

Example:

```
def send_group_mod(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    port = 1
    max_len = 2000
    actions = [ofp_parser.OFPActionOutput(port, max_len)]

    weight = 100
    watch_port = 0
    watch_group = 0
    buckets = [ofp_parser.OFPBucket(weight, watch_port, watch_group,
                                    actions)]

    group_id = 1
    req = ofp_parser.OFPGroupMod(datapath, ofp.OFPGC_ADD,
                                 ofp.OFPGT_SELECT, group_id, buckets)
    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPGroupMod": {
    "buckets": [
      {
        "OFPBucket": {
          "actions": [
            {
```

(continues on next page)



(continued from previous page)

```
        "OFPActionOutput": {
            "len": 16,
            "max_len": 65535,
            "port": 2,
            "type": 0
        }
    ],
    "len": 32,
    "watch_group": 1,
    "watch_port": 1,
    "weight": 1
}
},
"command": 0,
"group_id": 1,
"type": 0
}
```

```
class os_ken.ofproto.ofproto_v1_2_parser.OFPPortMod(datapath, port_no=0,
                                                    hw_addr='00:00:00:00:00:00',
                                                    config=0, mask=0, advertise=0)
```

Port modification message

The controller sends this message to modify the behavior of the port.

Attribute	Description
port_no	Port number to modify
hw_addr	The hardware address that must be the same as hw_addr of OFPPort of OFPSwitchFeatures
config	Bitmap of configuration flags.  OFPPC_PORT_DOWN OFPPC_NO_RECV OFPPC_NO_FWD OFPPC_NO_PACKET_IN
mask	Bitmap of configuration flags above to be changed
advertise	Bitmap of the following flags.  OFPPF_10MB_HD OFPPF_10MB_FD OFPPF_100MB_HD OFPPF_100MB_FD OFPPF_1GB_HD OFPPF_1GB_FD OFPPF_10GB_FD OFPPF_40GB_FD OFPPF_100GB_FD OFPPF_1TB_FD OFPPF_OTHER OFPPF_COPPER OFPPF_FIBER OFPPF_AUTONEG OFPPF_PAUSE OFPPF_PAUSE_ASYM

Example:

```
def send_port_mod(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    port_no = 3
    hw_addr = 'fa:c8:e8:76:1d:7e'
    config = 0
    mask = (ofp.OFPPC_PORT_DOWN | ofp.OFPPC_NO_RECV |
            ofp.OFPPC_NO_FWD | ofp.OFPPC_NO_PACKET_IN)
    advertise = (ofp.OFPPF_10MB_HD | ofp.OFPPF_100MB_FD |
```

(continues on next page)

(continued from previous page)

```

    ofp.OFPPF_1GB_FD | ofp.OFPPF_COPPER |
    ofp.OFPPF_AUTONEG | ofp.OFPPF_PAUSE |
    ofp.OFPPF_PAUSE_ASYM)
    req = ofp_parser.OFPPortMod(datapath, port_no, hw_addr, config,
                               mask, advertise)
    datapath.send_msg(req)

```

JSON Example:

```

{
  "OFPPortMod": {
    "advertise": 4096,
    "config": 0,
    "hw_addr": "00-11-00-00-11-11",
    "mask": 0,
    "port_no": 1
  }
}

```

## Read State Messages

**class** `os_ken.ofproto.ofproto_v1_2_parser.OFPDescStatsRequest`(*datapath*, *flags=0*)

Description statistics request message

The controller uses this message to query description of the switch.

Attribute	Description
flags	Zero (none yet defined in the spec)

Example:

```

def send_desc_stats_request(self, datapath):
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPDescStatsRequest(datapath)
    datapath.send_msg(req)

```

JSON Example:

```

{
  "OFPDescStatsRequest": {
    "flags": 0
  }
}

```

**class** `os_ken.ofproto.ofproto_v1_2_parser.OFPDescStats`(*mfr\_desc*, *hw\_desc*, *sw\_desc*,  
*serial\_num*, *dp\_desc*)

Description statistics reply message

The switch responds with a stats reply that include this message to a description statistics request.

Attribute	Description
mfr_desc	Manufacturer description
hw_desc	Hardware description
sw_desc	Software description
serial_num	Serial number
dp_desc	Human readable description of datapath

Example:

```
@set_ev_cls(ofp_event.EventOFPStatsReply, MAIN_DISPATCHER)
def stats_reply_handler(self, ev):
    msg = ev.msg
    ofp = msg.datapath.ofproto
    body = ev.msg.body

    if msg.type == ofp.OFPST_DESC:
        self.desc_stats_reply_handler(body)

def desc_stats_reply_handler(self, body):
    self.logger.debug('DescStats: mfr_desc=%s hw_desc=%s sw_desc=%s '
                      'serial_num=%s dp_desc=%s',
                      body.mfr_desc, body.hw_desc, body.sw_desc,
                      body.serial_num, body.dp_desc)
```

JSON Example:

```
{
  "OFPStatsReply": {
    "body": {
      "OFPDescStats": {
        "dp_desc": "dp",
        "hw_desc": "hw",
        "mfr_desc": "mfr",
        "serial_num": "serial",
        "sw_desc": "sw"
      }
    },
    "flags": 0,
    "type": 0
  }
}
```

```
class os_ken.ofproto.ofproto_v1_2_parser.OFPFlowStatsRequest(datapath,
                                                             table_id=255,
                                                             out_port=4294967295,
                                                             out_group=4294967295,
                                                             cookie=0,
                                                             cookie_mask=0,
                                                             match=None, flags=0)
```

Individual flow statistics request message

The controller uses this message to query individual flow statistics.

Attribute	Description
table_id	ID of table to read
out_port	Require matching entries to include this as an output port
out_group	Require matching entries to include this as an output group
cookie	Require matching entries to contain this cookie value
cookie_mask	Mask used to restrict the cookie bits that must match
match	Instance of OFPMatch
flags	Zero (none yet defined in the spec)

Example:

```
def send_flow_stats_request(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    cookie = cookie_mask = 0
    match = ofp_parser.OFPMatch(in_port=1)
    req = ofp_parser.OFPFlowStatsRequest(datapath,
                                         ofp.OFPTT_ALL,
                                         ofp.OFPP_ANY, ofp.OFPG_ANY,
                                         cookie, cookie_mask, match)

    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPFlowStatsRequest": {
    "cookie": 0,
    "cookie_mask": 0,
    "flags": 0,
    "match": {
      "OFPMatch": {
        "length": 4,
        "oxm_fields": [],
        "type": 1
      }
    },
    "out_group": 4294967295,
    "out_port": 4294967295,
    "table_id": 0
  }
}
```

```
class os_ken.ofproto.ofproto_v1_2_parser.OFPFlowStats(table_id, duration_sec,
                                                    duration_nsec, priority,
                                                    idle_timeout, hard_timeout,
                                                    cookie, packet_count,
                                                    byte_count, match,
                                                    instructions=None,
                                                    length=None)
```

Individual flow statistics reply message

The switch responds with a stats reply that include this message to an individual flow statistics request.

Attribute	Description
<code>table_id</code>	ID of table flow came from
<code>duration_sec</code>	Time flow has been alive in seconds
<code>duration_nsec</code>	Time flow has been alive in nanoseconds beyond <code>duration_sec</code>
<code>priority</code>	Priority of the entry
<code>idle_timeout</code>	Number of seconds idle before expiration
<code>hard_timeout</code>	Number of seconds before expiration
<code>cookie</code>	Opaque controller-issued identifier
<code>packet_count</code>	Number of packets in flow
<code>byte_count</code>	Number of bytes in flow
<code>match</code>	Instance of <code>OFPMatch</code>
<code>instructions</code>	list of <code>OFPInstruction*</code> instance

Example:

```
@set_ev_cls(ofp_event.EventOFPStatsReply, MAIN_DISPATCHER)
def stats_reply_handler(self, ev):
    msg = ev.msg
    ofp = msg.datapath.ofproto
    body = ev.msg.body

    if msg.type == ofp.OFPST_FLOW:
        self.flow_stats_reply_handler(body)

def flow_stats_reply_handler(self, body):
    flows = []
    for stat in body:
        flows.append('table_id=%s '
                    'duration_sec=%d duration_nsec=%d '
                    'priority=%d '
                    'idle_timeout=%d hard_timeout=%d '
                    'cookie=%d packet_count=%d byte_count=%d '
                    'match=%s instructions=%s' %
                    (stat.table_id,
                     stat.duration_sec, stat.duration_nsec,
                     stat.priority,
                     stat.idle_timeout, stat.hard_timeout,
                     stat.cookie, stat.packet_count, stat.byte_count,
```

(continues on next page)

(continued from previous page)

```

        stat.match, stat.instructions))
self.logger.debug('FlowStats: %s', flows)

```

JSON Example:

```

{
  "OFPSStatsReply": {
    "body": [
      {
        "OFPPFlowStats": {
          "byte_count": 0,
          "cookie": 0,
          "duration_nsec": 115277000,
          "duration_sec": 358,
          "hard_timeout": 0,
          "idle_timeout": 0,
          "instructions": [],
          "length": 56,
          "match": {
            "OFPMatch": {
              "length": 4,
              "oxm_fields": [],
              "type": 1
            }
          },
          "packet_count": 0,
          "priority": 65535,
          "table_id": 0
        }
      },
      {
        "OFPPFlowStats": {
          "byte_count": 0,
          "cookie": 0,
          "duration_nsec": 115055000,
          "duration_sec": 358,
          "hard_timeout": 0,
          "idle_timeout": 0,
          "instructions": [
            {
              "OFPInstructionActions": {
                "actions": [
                  {
                    "OFPActionOutput": {
                      "len": 16,
                      "max_len": 0,
                      "port": 4294967290,
                      "type": 0
                    }
                  }
                ]
              }
            }
          ]
        }
      }
    ]
  }
}

```

(continues on next page)

(continued from previous page)

```

    ],
    "len": 24,
    "type": 4
  }
}
],
"length": 88,
"match": {
  "OFPMatch": {
    "length": 10,
    "oxm_fields": [
      {
        "OXMTlv": {
          "field": "eth_type",
          "mask": null,
          "value": 2054
        }
      }
    ]
  },
  "type": 1
}
},
"packet_count": 0,
"priority": 65534,
"table_id": 0
},
{
  "OFPFLOWStats": {
    "byte_count": 238,
    "cookie": 0,
    "duration_nsec": 511582000,
    "duration_sec": 316220,
    "hard_timeout": 0,
    "idle_timeout": 0,
    "instructions": [
      {
        "OFPIInstructionGotoTable": {
          "len": 8,
          "table_id": 1,
          "type": 1
        }
      }
    ]
  },
  "length": 80,
  "match": {
    "OFPMatch": {
      "length": 22,
      "oxm_fields": [

```

(continues on next page)



(continued from previous page)

```

    {
      "OXMTlv": {
        "field": "in_port",
        "mask": null,
        "value": 6
      }
    },
    {
      "OXMTlv": {
        "field": "eth_src",
        "mask": null,
        "value": "f2:0b:a4:7d:f8:ea"
      }
    }
  ],
  "type": 1
},
"packet_count": 3,
"priority": 123,
"table_id": 0
},
{
  "OFPFFlowStats": {
    "byte_count": 98,
    "cookie": 0,
    "duration_nsec": 980901000,
    "duration_sec": 313499,
    "hard_timeout": 0,
    "idle_timeout": 0,
    "instructions": [
      {
        "OFPIInstructionActions": {
          "actions": [
            {
              "OFPAActionOutput": {
                "len": 16,
                "max_len": 65535,
                "port": 4294967293,
                "type": 0
              }
            }
          ]
        },
        "len": 24,
        "type": 3
      }
    ]
  }
},
],

```

(continues on next page)

(continued from previous page)

```

        "length": 80,
        "match": {
            "OFPMatch": {
                "length": 4,
                "oxm_fields": [],
                "type": 1
            }
        },
        "packet_count": 1,
        "priority": 0,
        "table_id": 0
    }
},
"flags": 0,
"type": 1
}
}

```

```

class os_ken.ofproto.ofproto_v1_2_parser.OFPAggregateStatsRequest(datapath,
                                                                    table_id=255,
                                                                    out_port=4294967295,
                                                                    out_group=4294967295,
                                                                    cookie=0,
                                                                    cookie_mask=0,
                                                                    match=None,
                                                                    flags=0)

```

Aggregate flow statistics request message

The controller uses this message to query aggregate flow statistics.

Attribute	Description
table_id	ID of table to read
out_port	Require matching entries to include this as an output port
out_group	Require matching entries to include this as an output group
cookie	Require matching entries to contain this cookie value
cookie_mask	Mask used to restrict the cookie bits that must match
match	Instance of OFPMatch
flags	Zero (none yet defined in the spec)

Example:

```

def send_aggregate_stats_request(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    cookie = cookie_mask = 0
    match = ofp_parser.OFPMatch(in_port=1)
    req = ofp_parser.OFPAggregateStatsRequest(datapath, 0,

```

(continues on next page)

(continued from previous page)

```

ofp.OFPTT_ALL,
ofp.OFPP_ANY,
ofp.OFPG_ANY,
cookie, cookie_mask,
match)

datapath.send_msg(req)

```

JSON Example:

```

{
  "OFPAggregateStatsRequest": {
    "cookie": 0,
    "cookie_mask": 0,
    "flags": 0,
    "match": {
      "OFPMatch": {
        "length": 4,
        "oxm_fields": [],
        "type": 1
      }
    },
    "out_group": 4294967295,
    "out_port": 4294967295,
    "table_id": 255
  }
}

```

```

class os_ken.ofproto.ofproto_v1_2_parser.OFPAggregateStatsReply(packet_count,
                                                                byte_count,
                                                                flow_count)

```

Aggregate flow statistics reply message

The switch responds with a stats reply that include this message to an aggregate flow statistics request.

Attribute	Description
packet_count	Number of packets in flows
byte_count	Number of bytes in flows
flow_count	Number of flows

Example:

```

@set_ev_cls(ofp_event.EventOFPSStatsReply, MAIN_DISPATCHER)
def stats_reply_handler(self, ev):
    msg = ev.msg
    ofp = msg.datapath.ofproto
    body = ev.msg.body

    if msg.type == ofp.OFPST_AGGREGATE:

```

(continues on next page)

(continued from previous page)

```

        self.aggregate_stats_reply_handler(body)

def aggregate_stats_reply_handler(self, body):
    self.logger.debug('AggregateStats: packet_count=%d byte_count=%d '
                      'flow_count=%d',
                      body.packet_count, body.byte_count,
                      body.flow_count)

```

JSON Example:

```

{
  "OFStatsReply": {
    "body": {
      "OFPAggregateStatsReply": {
        "byte_count": 574,
        "flow_count": 6,
        "packet_count": 7
      }
    },
    "flags": 0,
    "type": 2
  }
}

```

**class** `os_ken.ofproto.ofproto_v1_2_parser.OFPTableStatsRequest`(*datapath*, *flags=0*)

Table statistics request message

The controller uses this message to query flow table statistics.

Attribute	Description
flags	Zero (none yet defined in the spec)

Example:

```

def send_table_stats_request(self, datapath):
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPTableStatsRequest(datapath)
    datapath.send_msg(req)

```

JSON Example:

```

{
  "OFPTableStatsRequest": {
    "flags": 0
  }
}

```

```
class os_ken.ofproto.ofproto_v1_2_parser.OFPTableStats(table_id, name, match,
                                                    wildcards, write_actions,
                                                    apply_actions, write_setfields,
                                                    apply_setfields,
                                                    metadata_match,
                                                    metadata_write, instructions,
                                                    config, max_entries,
                                                    active_count, lookup_count,
                                                    matched_count)
```

Table statistics reply message

The switch responds with a stats reply that include this message to a table statistics request.

Attribute	Description
<code>table_id</code>	ID of table
<code>name</code>	table name
<code>match</code>	Bitmap of (1 « OFPXMT_*) that indicate the fields the table can match on
<code>wildcards</code>	Bitmap of (1 « OFPXMT_*) wildcards that are supported by the table
<code>write_actions</code>	Bitmap of OFPAT_* that are supported by the table with OFPIT_WRITE_ACTIONS
<code>apply_actions</code>	Bitmap of OFPAT_* that are supported by the table with OFPIT_APPLY_ACTIONS
<code>write_setfields</code>	Bitmap of (1 « OFPXMT_*) header fields that can be set with OFPIT_WRITE_ACTIONS
<code>apply_setfields</code>	Bitmap of (1 « OFPXMT_*) header fields that can be set with OFPIT_APPLY_ACTIONS
<code>meta-data_match</code>	Bits of metadata table can match
<code>meta-data_write</code>	Bits of metadata table can write
<code>instructions</code>	Bitmap of OFPIT_* values supported
<code>config</code>	Bitmap of OFPTC_* values
<code>max_entries</code>	Max number of entries supported
<code>active_count</code>	Number of active entries
<code>lookup_count</code>	Number of packets looked up in table
<code>matched_count</code>	Number of packets that hit table

Example:

```
@set_ev_cls(ofp_event.EventOFPStatsReply, MAIN_DISPATCHER)
def stats_reply_handler(self, ev):
    msg = ev.msg
    ofp = msg.datapath.ofproto
    body = ev.msg.body

    if msg.type == ofp.OFPST_TABLE:
        self.table_stats_reply_handler(body)

def table_stats_reply_handler(self, body):
    tables = []
```

(continues on next page)

(continued from previous page)

```

for stat in body:
    tables.append('table_id=%d active_count=%d lookup_count=%d '
                 ' matched_count=%d' %
                 (stat.table_id, stat.active_count,
                  stat.lookup_count, stat.matched_count))
self.logger.debug('TableStats: %s', tables)

```

```

class os_ken.ofproto.ofproto_v1_2_parser.OFPPortStatsRequest(datapath,
                                                             port_no=4294967295,
                                                             flags=0)

```

Port statistics request message

The controller uses this message to query information about ports statistics.

Attribute	Description
port_no	Port number to read (OFPP_ANY to all ports)
flags	Zero (none yet defined in the spec)

Example:

```

def send_port_stats_request(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPPortStatsRequest(datapath, ofp.OFPP_ANY)
    datapath.send_msg(req)

```

JSON Example:

```

{
  "OFPPortStatsRequest": {
    "flags": 0,
    "port_no": 4294967295
  }
}

```

```

class os_ken.ofproto.ofproto_v1_2_parser.OFPPortStats(port_no, rx_packets, tx_packets,
                                                       rx_bytes, tx_bytes, rx_dropped,
                                                       tx_dropped, rx_errors,
                                                       tx_errors, rx_frame_err,
                                                       rx_over_err, rx_crc_err,
                                                       collisions)

```

Port statistics reply message

The switch responds with a stats reply that include this message to a port statistics request.

Attribute	Description
port_no	Port number
rx_packets	Number of received packets
tx_packets	Number of transmitted packets
rx_bytes	Number of received bytes
tx_bytes	Number of transmitted bytes
rx_dropped	Number of packets dropped by RX
tx_dropped	Number of packets dropped by TX
rx_errors	Number of receive errors
tx_errors	Number of transmit errors
rx_frame_err	Number of frame alignment errors
rx_over_err	Number of packet with RX overrun
rx_crc_err	Number of CRC errors
collisions	Number of collisions

Example:

```
@set_ev_cls(ofp_event.EventOFPStatsReply, MAIN_DISPATCHER)
def stats_reply_handler(self, ev):
    msg = ev.msg
    ofp = msg.datapath.ofproto
    body = ev.msg.body

    if msg.type == ofp.OFPST_PORT:
        self.port_stats_reply_handler(body)

def port_stats_reply_handler(self, body):
    ports = []
    for stat in body:
        ports.append('port_no=%d '
                    'rx_packets=%d tx_packets=%d '
                    'rx_bytes=%d tx_bytes=%d '
                    'rx_dropped=%d tx_dropped=%d '
                    'rx_errors=%d tx_errors=%d '
                    'rx_frame_err=%d rx_over_err=%d rx_crc_err=%d '
                    'collisions=%d' %
                    (stat.port_no,
                     stat.rx_packets, stat.tx_packets,
                     stat.rx_bytes, stat.tx_bytes,
                     stat.rx_dropped, stat.tx_dropped,
                     stat.rx_errors, stat.tx_errors,
                     stat.rx_frame_err, stat.rx_over_err,
                     stat.rx_crc_err, stat.collisions))
    self.logger.debug('PortStats: %s', ports)
```

JSON Example:

```
{
  "OFPStatsReply": {
```

(continues on next page)

(continued from previous page)

```

"body": [
  {
    "OFPPortStats": {
      "collisions": 0,
      "port_no": 7,
      "rx_bytes": 0,
      "rx_crc_err": 0,
      "rx_dropped": 0,
      "rx_errors": 0,
      "rx_frame_err": 0,
      "rx_over_err": 0,
      "rx_packets": 0,
      "tx_bytes": 336,
      "tx_dropped": 0,
      "tx_errors": 0,
      "tx_packets": 4
    }
  },
  {
    "OFPPortStats": {
      "collisions": 0,
      "port_no": 6,
      "rx_bytes": 336,
      "rx_crc_err": 0,
      "rx_dropped": 0,
      "rx_errors": 0,
      "rx_frame_err": 0,
      "rx_over_err": 0,
      "rx_packets": 4,
      "tx_bytes": 336,
      "tx_dropped": 0,
      "tx_errors": 0,
      "tx_packets": 4
    }
  }
],
"flags": 0,
"type": 4
}

```

```

class os_ken.ofproto.ofproto_v1_2_parser.OFPQueueStatsRequest(datapath,
                                                                port_no=4294967295,
                                                                queue_id=4294967295,
                                                                flags=0)

```

Queue statistics request message

The controller uses this message to query queue statistics.



Attribute	Description
port_no	Port number to read
queue_id	ID of queue to read
flags	Zero (none yet defined in the spec)

Example:

```
def send_queue_stats_request(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPQueueStatsRequest(datapath, ofp.OFPP_ANY,
                                           ofp.OFPQ_ALL)

    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPQueueStatsRequest": {
    "flags": 0,
    "port_no": 4294967295,
    "queue_id": 4294967295
  }
}
```

**class** `os_ken.ofproto.ofproto_v1_2_parser.OFPQueueStats`(*port\_no*, *queue\_id*, *tx\_bytes*,  
*tx\_packets*, *tx\_errors*)

Queue statistics reply message

The switch responds with a stats reply that include this message to an aggregate flow statistics request.

Attribute	Description
port_no	Port number
queue_id	ID of queue
tx_bytes	Number of transmitted bytes
tx_packets	Number of transmitted packets
tx_errors	Number of packets dropped due to overrun

Example:

```
@set_ev_cls(ofp_event.EventOFPStatsReply, MAIN_DISPATCHER)
def stats_reply_handler(self, ev):
    msg = ev.msg
    ofp = msg.datapath.ofproto
    body = ev.msg.body

    if msg.type == ofp.OFPST_QUEUE:
```

(continues on next page)

(continued from previous page)

```

        self.queue_stats_reply_handler(body)

def queue_stats_reply_handler(self, body):
    queues = []
    for stat in body:
        queues.append('port_no=%d queue_id=%d '
                      'tx_bytes=%d tx_packets=%d tx_errors=%d ' %
                      (stat.port_no, stat.queue_id,
                       stat.tx_bytes, stat.tx_packets, stat.tx_errors))
    self.logger.debug('QueueStats: %s', queues)

```

JSON Example:

```

{
  "OFPPStatsReply": {
    "body": [
      {
        "OFPPQueueStats": {
          "port_no": 7,
          "queue_id": 1,
          "tx_bytes": 0,
          "tx_errors": 0,
          "tx_packets": 0
        }
      },
      {
        "OFPPQueueStats": {
          "port_no": 6,
          "queue_id": 1,
          "tx_bytes": 0,
          "tx_errors": 0,
          "tx_packets": 0
        }
      },
      {
        "OFPPQueueStats": {
          "port_no": 7,
          "queue_id": 2,
          "tx_bytes": 0,
          "tx_errors": 0,
          "tx_packets": 0
        }
      }
    ],
    "flags": 0,
    "type": 5
  }
}

```

```
class os_ken.ofproto.ofproto_v1_2_parser.OFPGGroupStatsRequest(datapath,
                                                                group_id=4294967292,
                                                                flags=0)
```

Group statistics request message

The controller uses this message to query statistics of one or more groups.

Attribute	Description
group_id	ID of group to read (OFPG_ALL to all groups)
flags	Zero (none yet defined in the spec)

Example:

```
def send_group_stats_request(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPGGroupStatsRequest(datapath, ofp.OFPG_ALL)
    datapath.send_msg(req)
```

```
class os_ken.ofproto.ofproto_v1_2_parser.OFPGGroupStats(group_id, ref_count,
                                                         packet_count, byte_count,
                                                         bucket_counters,
                                                         length=None)
```

Group statistics reply message

The switch responds with a stats reply that include this message to a group statistics request.

Attribute	Description
group_id	Group identifier
ref_count	Number of flows or groups that directly forward to this group
packet_count	Number of packets processed by group
byte_count	Number of bytes processed by group
bucket_counters	List of OFPBucketCounter instance

Example:

```
@set_ev_cls(ofp_event.EventOFPStatsReply, MAIN_DISPATCHER)
def stats_reply_handler(self, ev):
    msg = ev.msg
    ofp = msg.datapath.ofproto
    body = ev.msg.body

    if msg.type == ofp.OFPGST_GROUP:
        self.group_stats_reply_handler(body)

def group_stats_reply_handler(self, body):
    groups = []
    for stat in body:
```

(continues on next page)

(continued from previous page)

```

groups.append('group_id=%d ref_count=%d packet_count=%d '
              'byte_count=%d bucket_counters=%s' %
              (stat.group_id,
               stat.ref_count, stat.packet_count,
               stat.byte_count, stat.bucket_counters))

self.logger.debug('GroupStats: %s', groups)

```

```
class os_ken.ofproto.ofproto_v1_2_parser.OFPGroupDescStatsRequest(datapath,
                                                                    flags=0)
```

Group description request message

The controller uses this message to list the set of groups on a switch.

Attribute	Description
flags	Zero (none yet defined in the spec)

Example:

```

def send_group_desc_stats_request(self, datapath):
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPGroupDescStatsRequest(datapath)
    datapath.send_msg(req)

```

JSON Example:

```

{
  "OFPGroupDescStatsRequest": {
    "flags": 0
  }
}

```

```
class os_ken.ofproto.ofproto_v1_2_parser.OFPGroupDescStats(type_, group_id, buckets,
                                                            length=None)
```

Group description reply message

The switch responds with a stats reply that include this message to a group description request.

Attribute	Description
type	One of OFPGT_*
group_id	Group identifier
buckets	List of OFPBucket instance

type attribute corresponds to type\_ parameter of \_\_init\_\_.

Example:

```

@set_ev_cls(ofp_event.EventOFPStatsReply, MAIN_DISPATCHER)
def stats_reply_handler(self, ev):
    msg = ev.msg
    ofp = msg.datapath.ofproto
    body = ev.msg.body

    if msg.type == ofp.OFPST_GROUP_DESC:
        self.group_desc_stats_reply_handler(body)

def group_desc_stats_reply_handler(self, body):
    desc = []
    for stat in body:
        desc.append('type=%d group_id=%d buckets=%s' %
                   (stat.type, stat.group_id, stat.buckets))
    self.logger.debug('GroupDescStats: %s', desc)

```

JSON Example:

```

{
  "OFPStatsReply": {
    "body": [
      {
        "OFPGroupDescStats": {
          "buckets": [
            {
              "OFPBucket": {
                "actions": [
                  {
                    "OFPActionOutput": {
                      "len": 16,
                      "max_len": 65535,
                      "port": 2,
                      "type": 0
                    }
                  }
                ]
              }
            ],
            "len": 32,
            "watch_group": 1,
            "watch_port": 1,
            "weight": 1
          }
        ],
        "group_id": 1,
        "length": 40,
        "type": 0
      }
    ],
    "flags": 0,

```

(continues on next page)

(continued from previous page)

```

    "type": 7
  }
}

```

```

class os_ken.ofproto.ofproto_v1_2_parser.OFPGroupFeaturesStatsRequest(datapath,
                                                                    flags=0)

```

Group features request message

The controller uses this message to list the capabilities of groups on a switch.

Attribute	Description
flags	Zero (none yet defined in the spec)

Example:

```

def send_group_features_stats_request(self, datapath):
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPGroupFeaturesStatsRequest(datapath)
    datapath.send_msg(req)

```

JSON Example:

```

{
  "OFPGroupFeaturesStatsRequest": {
    "flags": 0
  }
}

```

```

class os_ken.ofproto.ofproto_v1_2_parser.OFPGroupFeaturesStats(types, capabilities,
                                                                max_groups,
                                                                actions,
                                                                length=None)

```

Group features reply message

The switch responds with a stats reply that include this message to a group features request.

Attribute	Description
types	Bitmap of OFPGT_* values supported
capabilities	Bitmap of OFPGFC_* capability supported
max_groups	Maximum number of groups for each type
actions	Bitmaps of OFPAT_* that are supported

Example:

```

@set_ev_cls(ofp_event.EventOFPSStatsReply, MAIN_DISPATCHER)
def stats_reply_handler(self, ev):
    msg = ev.msg

```

(continues on next page)

(continued from previous page)

```

ofp = msg.datapath.ofproto
body = ev.msg.body

if msg.type == ofp.OFPST_GROUP_FEATURES:
    self.group_features_stats_reply_handler(body)

def group_features_stats_reply_handler(self, body):
    self.logger.debug('GroupFeaturesStats: types=%d '
                      'capabilities=0x%08x max_groups=%s '
                      'actions=%s',
                      body.types, body.capabilities, body.max_groups,
                      body.actions)

```

JSON Example:

```

{
  "OFPPStatsReply": {
    "body": {
      "OFPPGroupFeaturesStats": {
        "actions": [
          67082241,
          67082241,
          67082241,
          67082241
        ],
        "capabilities": 5,
        "length": 40,
        "max_groups": [
          16777216,
          16777216,
          16777216,
          16777216
        ],
        "types": 15
      }
    },
    "flags": 0,
    "type": 8
  }
}

```

## Queue Configuration Messages

**class** `os_ken.ofproto.ofproto_v1_2_parser.OFPQueueGetConfigRequest` (*datapath, port*)

Queue configuration request message

Attribute	Description
port	Port to be queried (OFPP_ANY to all configured queues)

Example:

```
def send_queue_get_config_request(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPQueueGetConfigRequest(datapath, ofp.OFPP_ANY)
    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPQueueGetConfigRequest": {
    "port": 4294967295
  }
}
```

```
class os_ken.ofproto.ofproto_v1_2_parser.OFPQueueGetConfigReply(datapath,
                                                                port=None,
                                                                queues=None)
```

Queue configuration reply message

The switch responds with this message to a queue configuration request.

Attribute	Description
port	Port which was queried
queues	list of OFPPacketQueue instance

Example:

```
@set_ev_cls(ofp_event.EventOFPQueueGetConfigReply, MAIN_DISPATCHER)
def queue_get_config_reply_handler(self, ev):
    msg = ev.msg

    self.logger.debug('OFPQueueGetConfigReply received: '
                      'port=%s queues=%s',
                      msg.port, msg.queues)
```

JSON Example:

```
{
  "OFPQueueGetConfigReply": {
    "port": 4294967295,
    "queues": [
      {
        "OFPPacketQueue": {
          "len": 48,
          "port": 77,
          "properties": [
            {

```

(continues on next page)



(continued from previous page)

```

        "OFPPQueuePropMinRate": {
            "len": 16,
            "property": 1,
            "rate": 10
        },
        {
            "OFPPQueuePropMaxRate": {
                "len": 16,
                "property": 2,
                "rate": 900
            }
        }
    ],
    "queue_id": 99
},
{
    "OFPPacketQueue": {
        "len": 48,
        "port": 77,
        "properties": [
            {
                "OFPPQueuePropMinRate": {
                    "len": 16,
                    "property": 1,
                    "rate": 100
                }
            },
            {
                "OFPPQueuePropMaxRate": {
                    "len": 16,
                    "property": 2,
                    "rate": 200
                }
            }
        ],
        "queue_id": 88
    }
}
]
}
}

```

### Packet-Out Message

```

class os_ken.ofproto.ofproto_v1_2_parser.OFPPacketOut(datapath, buffer_id=None,
                                                       in_port=None, actions=None,
                                                       data=None, actions_len=None)

```

## Packet-Out message

The controller uses this message to send a packet out through the switch.

Attribute	Description
buffer_id	ID assigned by datapath (OFPP_NO_BUFFER if none)
in_port	Packet's input port or OFPP_CONTROLLER
actions	list of OpenFlow action class
data	Packet data of a binary type value or an instances of packet.Packet.

Example:

```
def send_packet_out(self, datapath, buffer_id, in_port):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    actions = [ofp_parser.OFPActionOutput(ofp.OFPP_FLOOD, 0)]
    req = ofp_parser.OFPPacketOut(datapath, buffer_id,
                                   in_port, actions)

    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPPacketOut": {
    "actions": [
      {
        "OFPActionOutput": {
          "len": 16,
          "max_len": 65535,
          "port": 4294967292,
          "type": 0
        }
      }
    ],
    "actions_len": 16,
    "buffer_id": 4294967295,
    "data":
    ↪ "8guk0D9w8gukffjqCABFAABU+BoAAP8Br4sKAAABCgAAAggAAgj3YAAAMdYCAAAAAACrjS0xAAAAABAREhM
    ↪",
    "in_port": 4294967293
  }
}
```

## Barrier Message

**class** os\_ken.ofproto.ofproto\_v1\_2\_parser.OFPBarrierRequest(datapath)

Barrier request message

The controller sends this message to ensure message dependencies have been met or receive notifications for completed operations.

Example:

```
def send_barrier_request(self, datapath):
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPBarrierRequest(datapath)
    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPBarrierRequest": {}
}
```

**class** os\_ken.ofproto.ofproto\_v1\_2\_parser.OFPBarrierReply(*datapath*)

Barrier reply message

The switch responds with this message to a barrier request.

Example:

```
@set_ev_cls(ofp_event.EventOFPBarrierReply, MAIN_DISPATCHER)
def barrier_reply_handler(self, ev):
    self.logger.debug('OFPBarrierReply received')
```

JSON Example:

```
{
  "OFPBarrierReply": {}
}
```

## Role Request Message

**class** os\_ken.ofproto.ofproto\_v1\_2\_parser.OFPRoleRequest(*datapath*, *role*,  
*generation\_id*)

Role request message

The controller uses this message to change its role.

Attribute	Description
role	One of the following values.  OFPCR_ROLE_NOCHANGE OFPCR_ROLE_EQUAL OFPCR_ROLE_MASTER OFPCR_ROLE_SLAVE
generation_id	Master Election Generation ID

Example:

```
def send_role_request(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPCRoleRequest(datapath, ofp.OFPCR_ROLE_EQUAL, 0)
    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPCRoleRequest": {
    "generation_id": 17294086455919964160,
    "role": 2
  }
}
```

```
class os_ken.ofproto.ofproto_v1_2_parser.OFPCRoleReply(datapath, role=None,
                                                       generation_id=None)
```

Role reply message

The switch responds with this message to a role request.

Attribute	Description
role	One of the following values.  OFPCR_ROLE_NOCHANGE OFPCR_ROLE_EQUAL OFPCR_ROLE_MASTER OFPCR_ROLE_SLAVE
generation_id	Master Election Generation ID

Example:

```
@set_ev_cls(ofp_event.EventOFPCRoleReply, MAIN_DISPATCHER)
def role_reply_handler(self, ev):
    msg = ev.msg
    dp = msg.datapath
    ofp = dp.ofproto

    if msg.role == ofp.OFPCR_ROLE_NOCHANGE:
        role = 'NOCHANGE'
    elif msg.role == ofp.OFPCR_ROLE_EQUAL:
        role = 'EQUAL'
    elif msg.role == ofp.OFPCR_ROLE_MASTER:
        role = 'MASTER'
    elif msg.role == ofp.OFPCR_ROLE_SLAVE:
        role = 'SLAVE'
    else:
```

(continues on next page)

(continued from previous page)

```

        role = 'unknown'

self.logger.debug('OFPPRoleReply received: '
                  'role=%s generation_id=%d',
                  role, msg.generation_id)

```

JSON Example:

```

{
  "OFPPRoleReply": {
    "generation_id": 17294086455919964160,
    "role": 3
  }
}

```

## Asynchronous Messages

### Packet-In Message

```

class os_ken.ofproto.ofproto_v1_2_parser.OFPPacketIn(datapath, buffer_id=None,
                                                    total_len=None, reason=None,
                                                    table_id=None, match=None,
                                                    data=None)

```

Packet-In message

The switch sends the packet that received to the controller by this message.

Attribute	Description
buffer_id	ID assigned by datapath
total_len	Full length of frame
reason	Reason packet is being sent.  OFPR_NO_MATCH OFPR_ACTION OFPR_INVALID_TTL
table_id	ID of the table that was looked up
match	Instance of OFPMatch
data	Ethernet frame

Example:

```

@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def packet_in_handler(self, ev):
    msg = ev.msg
    dp = msg.datapath
    ofp = dp.ofproto

```

(continues on next page)

(continued from previous page)

```

if msg.reason == ofp.OFPR_NO_MATCH:
    reason = 'NO MATCH'
elif msg.reason == ofp.OFPR_ACTION:
    reason = 'ACTION'
elif msg.reason == ofp.OFPR_INVALID_TTL:
    reason = 'INVALID TTL'
else:
    reason = 'unknown'

self.logger.debug('OFPPacketIn received: '
                  'buffer_id=%x total_len=%d reason=%s '
                  'table_id=%d match=%s data=%s',
                  msg.buffer_id, msg.total_len, reason,
                  msg.table_id, msg.match,
                  utils.hex_array(msg.data))

```

JSON Example:

```

{
  "OFPPacketIn": {
    "buffer_id": 2,
    "data": "/////////8gukffjqCAYAAQgABgQAAfILpH346goAAAEAAAAAAAAAKAAAD",
    "match": {
      "OFPMatch": {
        "length": 80,
        "oxm_fields": [
          {
            "OXMTlv": {
              "field": "in_port",
              "mask": null,
              "value": 6
            }
          },
          {
            "OXMTlv": {
              "field": "eth_type",
              "mask": null,
              "value": 2054
            }
          },
          {
            "OXMTlv": {
              "field": "eth_dst",
              "mask": null,
              "value": "ff:ff:ff:ff:ff:ff"
            }
          },
          {
            "OXMTlv": {
              "field": "eth_src",

```

(continues on next page)

(continued from previous page)

```
        "mask": null,
        "value": "f2:0b:a4:7d:f8:ea"
    },
    {
        "OXMTlv": {
            "field": "arp_op",
            "mask": null,
            "value": 1
        }
    },
    {
        "OXMTlv": {
            "field": "arp_spa",
            "mask": null,
            "value": "10.0.0.1"
        }
    },
    {
        "OXMTlv": {
            "field": "arp_tpa",
            "mask": null,
            "value": "10.0.0.3"
        }
    },
    {
        "OXMTlv": {
            "field": "arp_sha",
            "mask": null,
            "value": "f2:0b:a4:7d:f8:ea"
        }
    },
    {
        "OXMTlv": {
            "field": "arp_tha",
            "mask": null,
            "value": "00:00:00:00:00:00"
        }
    }
],
    "type": 1
}
},
"reason": 1,
"table_id": 1,
"total_len": 42
}
```

## Flow Removed Message

```
class os_ken.ofproto.ofproto_v1_2_parser.OFPFlowRemoved(datapath, cookie=None,
                                                         priority=None,
                                                         reason=None,
                                                         table_id=None,
                                                         duration_sec=None,
                                                         duration_nsec=None,
                                                         idle_timeout=None,
                                                         hard_timeout=None,
                                                         packet_count=None,
                                                         byte_count=None,
                                                         match=None)
```

Flow removed message

When flow entries time out or are deleted, the switch notifies controller with this message.

Attribute	Description
cookie	Opaque controller-issued identifier
priority	Priority level of flow entry
reason	One of the following values.  OFPRR_IDLE_TIMEOUT OFPRR_HARD_TIMEOUT OFPRR_DELETE OFPRR_GROUP_DELETE
table_id	ID of the table
duration_sec	Time flow was alive in seconds
duration_nsec	Time flow was alive in nanoseconds beyond duration_sec
idle_timeout	Idle timeout from original flow mod
hard_timeout	Hard timeout from original flow mod
packet_count	Number of packets that was associated with the flow
byte_count	Number of bytes that was associated with the flow
match	Instance of OFPMatch

Example:

```
@set_ev_cls(ofp_event.EventOFPFlowRemoved, MAIN_DISPATCHER)
def flow_removed_handler(self, ev):
    msg = ev.msg
    dp = msg.datapath
    ofp = dp.ofproto

    if msg.reason == ofp.OFPRR_IDLE_TIMEOUT:
        reason = 'IDLE TIMEOUT'
    elif msg.reason == ofp.OFPRR_HARD_TIMEOUT:
```

(continues on next page)



(continued from previous page)

```

        reason = 'HARD TIMEOUT'
    elif msg.reason == ofp.OFPRR_DELETE:
        reason = 'DELETE'
    elif msg.reason == ofp.OFPRR_GROUP_DELETE:
        reason = 'GROUP DELETE'
    else:
        reason = 'unknown'

    self.logger.debug('OFPPFlowRemoved received: '
                      'cookie=%d priority=%d reason=%s table_id=%d '
                      'duration_sec=%d duration_nsec=%d '
                      'idle_timeout=%d hard_timeout=%d '
                      'packet_count=%d byte_count=%d match.fields=%s',
                      msg.cookie, msg.priority, reason, msg.table_id,
                      msg.duration_sec, msg.duration_nsec,
                      msg.idle_timeout, msg.hard_timeout,
                      msg.packet_count, msg.byte_count, msg.match)

```

JSON Example:

```

{
  "OFPPFlowRemoved": {
    "byte_count": 86,
    "cookie": 0,
    "duration_nsec": 48825000,
    "duration_sec": 3,
    "hard_timeout": 0,
    "idle_timeout": 3,
    "match": {
      "OFPMatch": {
        "length": 14,
        "oxm_fields": [
          {
            "OXMTlv": {
              "field": "eth_dst",
              "mask": null,
              "value": "f2:0b:a4:7d:f8:ea"
            }
          }
        ],
        "type": 1
      }
    },
    "packet_count": 1,
    "priority": 65535,
    "reason": 0,
    "table_id": 0
  }
}

```

## Port Status Message

```
class os_ken.ofproto.ofproto_v1_2_parser.OFPPortStatus(datapath, reason=None,
                                                    desc=None)
```

Port status message

The switch notifies controller of change of ports.

Attribute	Description
reason	One of the following values.  OFPPR_ADD OFPPR_DELETE OFPPR_MODIFY
desc	instance of OFPPort

Example:

```
@set_ev_cls(ofp_event.EventOFPPortStatus, MAIN_DISPATCHER)
def port_status_handler(self, ev):
    msg = ev.msg
    dp = msg.datapath
    ofp = dp.ofproto

    if msg.reason == ofp.OFPPR_ADD:
        reason = 'ADD'
    elif msg.reason == ofp.OFPPR_DELETE:
        reason = 'DELETE'
    elif msg.reason == ofp.OFPPR_MODIFY:
        reason = 'MODIFY'
    else:
        reason = 'unknown'

    self.logger.debug('OFPPortStatus received: reason=%s desc=%s',
                      reason, msg.desc)
```

JSON Example:

```
{
  "OFPPortStatus": {
    "desc": {
      "OFPPort": {
        "advertised": 10240,
        "config": 0,
        "curr": 10248,
        "curr_speed": 5000,
        "hw_addr": "f2:0b:a4:d0:3f:70",
        "max_speed": 5000,
        "name": "\u79c1\u306e\u30dd\u30fc\u30c8",
```

(continues on next page)

(continued from previous page)

```

    "peer": 10248,
    "port_no": 7,
    "state": 4,
    "supported": 10248
  },
  "reason": 0
}

```

## Error Message

```

class os_ken.ofproto.ofproto_v1_2_parser.OFPErrormsg(datapath, type_=None,
                                                    code=None, data=None,
                                                    **kwargs)

```

Error message

The switch notifies controller of problems by this message.

Attribute	Description
type	High level type of error
code	Details depending on the type
data	Variable length data depending on the type and code

type attribute corresponds to type\_ parameter of \_\_init\_\_.

Types and codes are defined in os\_ken.ofproto.ofproto.

Type	Code
OFPET_HELLO_FAILED	OFPHFC_*
OFPET_BAD_REQUEST	OFPBRC_*
OFPET_BAD_ACTION	OFPBAC_*
OFPET_BAD_INSTRUCTION	OFPBIC_*
OFPET_BAD_MATCH	OFPBMC_*
OFPET_FLOW_MOD_FAILED	OFPFMFC_*
OFPET_GROUP_MOD_FAILED	OFPGMFC_*
OFPET_PORT_MOD_FAILED	OFPPMFC_*
OFPET_TABLE_MOD_FAILED	OFPTMFC_*
OFPET_QUEUE_OP_FAILED	OFPQOFC_*
OFPET_SWITCH_CONFIG_FAILED	OFPSCFC_*
OFPET_ROLE_REQUEST_FAILED	OFPRRFC_*
OFPET_EXPERIMENTER	N/A

If type == OFPET\_EXPERIMENTER, this message has also the following attributes.

Attribute	Description
exp_type	Experimenter defined type
experimenter	Experimenter ID

Example:

```
@set_ev_cls(ofp_event.EventOFPErrormsg,
            [HANDSHAKE_DISPATCHER, CONFIG_DISPATCHER, MAIN_DISPATCHER])
def error_msg_handler(self, ev):
    msg = ev.msg

    self.logger.debug('OFPErrormsg received: type=0x%02x code=0x%02x '
                      'message=%s',
                      msg.type, msg.code, utils.hex_array(msg.data))
```

JSON Example:

```
{
  "OFPErrormsg": {
    "code": 11,
    "data": "ZnVnYWZ1Z2E=",
    "type": 2
  }
}
```

```
{
  "OFPErrormsg": {
    "code": null,
    "data": "amlra2VuIGRhdGE=",
    "exp_type": 60000,
    "experimenter": 999999,
    "type": 65535
  }
}
```

## Symmetric Messages

### Hello

`class os_ken.ofproto.ofproto_v1_2_parser.OFPHello(datapath)`

Hello message

When connection is started, the hello message is exchanged between a switch and a controller.

This message is handled by the OSKen framework, so the OSKen application do not need to process this typically.

JSON Example:

```
{
  "OFPHello": {}
}
```

## Echo Request

**class** `os_ken.ofproto.ofproto_v1_2_parser.OFPEchoRequest` (*datapath*, *data=None*)

Echo request message

This message is handled by the OSKen framework, so the OSKen application do not need to process this typically.

Attribute	Description
<code>data</code>	An arbitrary length data

Example:

```
def send_echo_request(self, datapath, data):
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPEchoRequest(datapath, data)
    datapath.send_msg(req)

@set_ev_cls(ofp_event.EventOFPEchoRequest,
            [HANDSHAKE_DISPATCHER, CONFIG_DISPATCHER, MAIN_DISPATCHER])
def echo_request_handler(self, ev):
    self.logger.debug('OFPEchoRequest received: data=%s',
                      utils.hex_array(ev.msg.data))
```

JSON Example:

```
{
  "OFPEchoRequest": {
    "data": "aG9nZQ=="
  }
}
```

## Echo Reply

**class** `os_ken.ofproto.ofproto_v1_2_parser.OFPEchoReply` (*datapath*, *data=None*)

Echo reply message

This message is handled by the OSKen framework, so the OSKen application do not need to process this typically.

Attribute	Description
<code>data</code>	An arbitrary length data

Example:

```

def send_echo_reply(self, datapath, data):
    ofp_parser = datapath.ofproto_parser

    reply = ofp_parser.OFPEchoReply(datapath, data)
    datapath.send_msg(reply)

@set_ev_cls(ofp_event.EventOFPEchoReply,
            [HANDSHAKE_DISPATCHER, CONFIG_DISPATCHER, MAIN_DISPATCHER])
def echo_reply_handler(self, ev):
    self.logger.debug('OFPEchoReply received: data=%s',
                      utils.hex_array(ev.msg.data))

```

JSON Example:

```

{
  "OFPEchoReply": {
    "data": "aG9nZQ=="
  }
}

```

## Experimenter

```

class os_ken.ofproto.ofproto_v1_2_parser.OFPExperimenter(datapath,
                                                         experimenter=None,
                                                         exp_type=None,
                                                         data=None)

```

Experimenter extension message

Attribute	Description
experimenter	Experimenter ID
exp_type	Experimenter defined
data	Experimenter defined arbitrary additional data

JSON Example:

```

{
  "OFPExperimenter": {
    "data": "bmF6bW==",
    "exp_type": 123456789,
    "experimenter": 98765432
  }
}

```

## Port Structures

```

class os_ken.ofproto.ofproto_v1_2_parser.OFPPort(port_no, hw_addr, name, config, state,
                                                  curr, advertised, supported, peer,
                                                  curr_speed, max_speed)

```

Description of a port

Attribute	Description
port_no	Port number and it uniquely identifies a port within a switch.
hw_addr	MAC address for the port.
name	Null-terminated string containing a human-readable name for the interface.
config	Bitmap of port configuration flags.  OFPPC_PORT_DOWN OFPPC_NO_RECV OFPPC_NO_FWD OFPPC_NO_PACKET_IN
state	Bitmap of port state flags.  OFPPS_LINK_DOWN OFPPS_BLOCKED OFPPS_LIVE
curr	Current features.
advertised	Features being advertised by the port.
supported	Features supported by the port.
peer	Features advertised by peer.
curr_speed	Current port bitrate in kbps.
max_speed	Max port bitrate in kbps.

## Flow Match Structure

```
class os_ken.ofproto.ofproto_v1_2_parser.OFPMatch(type=None, length=None,  
                                                _ordered_fields=None, **kwargs)
```

Flow Match Structure

This class is implementation of the flow match structure having compose/query API. There are new API and old API for compatibility. the old API is supposed to be removed later.

You can define the flow match by the keyword arguments. The following arguments are available.

Argument	Value	Description
in_port	Integer 32bit	Switch input port
in_phy_port	Integer 32bit	Switch physical input port
metadata	Integer 64bit	Metadata passed between tables
eth_dst	MAC address	Ethernet destination address
eth_src	MAC address	Ethernet source address
eth_type	Integer 16bit	Ethernet frame type
vlan_vid	Integer 16bit	VLAN id
vlan_pcp	Integer 8bit	VLAN priority
ip_dscp	Integer 8bit	IP DSCP (6 bits in ToS field)
ip_ecn	Integer 8bit	IP ECN (2 bits in ToS field)

continues on next page

Table 1 – continued from previous page

Argument	Value	Description
ip_proto	Integer 8bit	IP protocol
ipv4_src	IPv4 address	IPv4 source address
ipv4_dst	IPv4 address	IPv4 destination address
tcp_src	Integer 16bit	TCP source port
tcp_dst	Integer 16bit	TCP destination port
udp_src	Integer 16bit	UDP source port
udp_dst	Integer 16bit	UDP destination port
sctp_src	Integer 16bit	SCTP source port
sctp_dst	Integer 16bit	SCTP destination port
icmpv4_type	Integer 8bit	ICMP type
icmpv4_code	Integer 8bit	ICMP code
arp_op	Integer 16bit	ARP opcode
arp_spa	IPv4 address	ARP source IPv4 address
arp_tpa	IPv4 address	ARP target IPv4 address
arp_sha	MAC address	ARP source hardware address
arp_tha	MAC address	ARP target hardware address
ipv6_src	IPv6 address	IPv6 source address
ipv6_dst	IPv6 address	IPv6 destination address
ipv6_flabel	Integer 32bit	IPv6 Flow Label
icmpv6_type	Integer 8bit	ICMPv6 type
icmpv6_code	Integer 8bit	ICMPv6 code
ipv6_nd_target	IPv6 address	Target address for ND
ipv6_nd_sll	MAC address	Source link-layer for ND
ipv6_nd_tll	MAC address	Target link-layer for ND
mpls_label	Integer 32bit	MPLS label
mpls_tc	Integer 8bit	MPLS TC
pbb_uca	Integer 8bit	PBB UCA header field (EXT-256 Old version of ONF Extension)
tcp_flags	Integer 16bit	TCP flags (EXT-109 ONF Extension)
actset_output	Integer 32bit	Output port from action set metadata (EXT-233 ONF Extension)

Example:

```
>>> # compose
>>> match = parser.OFPMatch(
...     in_port=1,
...     eth_type=0x86dd,
...     ipv6_src=('2001:db8:bd05:1d2:288a:1fc0:1:10ee',
...               'ffff:ffff:ffff:ffff:'),
...     ipv6_dst='2001:db8:bd05:1d2:288a:1fc0:1:10ee')
>>> # query
>>> if 'ipv6_src' in match:
...     print match['ipv6_src']
...
('2001:db8:bd05:1d2:288a:1fc0:1:10ee', 'ffff:ffff:ffff:ffff:')
```

#### Note



For the list of the supported Nicira experimenter matches, please refer to [os\\_ken.ofproto.nx\\_match](#).

## Note

For VLAN id match field, special values are defined in OpenFlow Spec.

### 1) Packets with and without a VLAN tag

- Example:

```
match = parser.OFPMatch()
```

- Packet Matching

non-VLAN-tagged	MATCH
VLAN-tagged(vlan_id=3)	MATCH
VLAN-tagged(vlan_id=5)	MATCH

### 2) Only packets without a VLAN tag

- Example:

```
match = parser.OFPMatch(vlan_vid=0x0000)
```

- Packet Matching

non-VLAN-tagged	MATCH
VLAN-tagged(vlan_id=3)	x
VLAN-tagged(vlan_id=5)	x

### 3) Only packets with a VLAN tag regardless of its value

- Example:

```
match = parser.OFPMatch(vlan_vid=(0x1000, 0x1000))
```

- Packet Matching

non-VLAN-tagged	x
VLAN-tagged(vlan_id=3)	MATCH
VLAN-tagged(vlan_id=5)	MATCH

### 4) Only packets with VLAN tag and VID equal

- Example:

```
match = parser.OFPMatch(vlan_vid=(0x1000 | 3))
```

- Packet Matching

non-VLAN-tagged	x
VLAN-tagged(vlan_id=3)	MATCH
VLAN-tagged(vlan_id=5)	x

## Flow Instruction Structures

```
class os_ken.ofproto.ofproto_v1_2_parser.OFPInstructionGotoTable(table_id,
                                                                type_=None,
                                                                len_=None)
```

Goto table instruction

This instruction indicates the next table in the processing pipeline.

Attribute	Description
<code>table_id</code>	Next table

```
class os_ken.ofproto.ofproto_v1_2_parser.OFPInstructionWriteMetadata(metadata,
                                                                      meta-
                                                                      data_mask,
                                                                      type_=None,
                                                                      len_=None)
```

Write metadata instruction

This instruction writes the masked metadata value into the metadata field.

Attribute	Description
<code>metadata</code>	Metadata value to write
<code>metadata_mask</code>	Metadata write bitmask

```
class os_ken.ofproto.ofproto_v1_2_parser.OFPInstructionActions(type_,
                                                                actions=None,
                                                                len_=None)
```

Actions instruction

This instruction writes/applies/clears the actions.

Attribute	Description
<code>type</code>	One of following values.  OFPIT_WRITE_ACTIONS OFPIT_APPLY_ACTIONS OFPIT_CLEAR_ACTIONS
<code>actions</code>	list of OpenFlow action class

`type` attribute corresponds to `type_` parameter of `__init__`.

## Action Structures

```
class os_ken.ofproto.ofproto_v1_2_parser.OFPActionOutput(port, max_len=65509,
                                                    type_=None, len_=None)
```

Output action

This action indicates output a packet to the switch port.

Attribute	Description
port	Output port
max_len	Max length to send to controller

```
class os_ken.ofproto.ofproto_v1_2_parser.OFPActionGroup(group_id=0, type_=None,
                                                    len_=None)
```

Group action

This action indicates the group used to process the packet.

Attribute	Description
group_id	Group identifier

```
class os_ken.ofproto.ofproto_v1_2_parser.OFPActionSetQueue(queue_id, type_=None,
                                                    len_=None)
```

Set queue action

This action sets the queue id that will be used to map a flow to an already-configured queue on a port.

Attribute	Description
queue_id	Queue ID for the packets

```
class os_ken.ofproto.ofproto_v1_2_parser.OFPActionSetMplsTtl(mpls_ttl, type_=None,
                                                    len_=None)
```

Set MPLS TTL action

This action sets the MPLS TTL.

Attribute	Description
mpls_ttl	MPLS TTL

```
class os_ken.ofproto.ofproto_v1_2_parser.OFPActionDecMplsTtl(type_=None,
                                                    len_=None)
```

Decrement MPLS TTL action

This action decrements the MPLS TTL.

```
class os_ken.ofproto.ofproto_v1_2_parser.OFPActionSetNwTtl(nw_ttl, type_=None,
                                                         len_=None)
```

Set IP TTL action

This action sets the IP TTL.

Attribute	Description
<code>nw_ttl</code>	IP TTL

```
class os_ken.ofproto.ofproto_v1_2_parser.OFPActionDecNwTtl(type_=None,
                                                            len_=None)
```

Decrement IP TTL action

This action decrements the IP TTL.

```
class os_ken.ofproto.ofproto_v1_2_parser.OFPActionCopyTtlOut(type_=None,
                                                             len_=None)
```

Copy TTL Out action

This action copies the TTL from the next-to-outermost header with TTL to the outermost header with TTL.

```
class os_ken.ofproto.ofproto_v1_2_parser.OFPActionCopyTtlIn(type_=None,
                                                            len_=None)
```

Copy TTL In action

This action copies the TTL from the outermost header with TTL to the next-to-outermost header with TTL.

```
class os_ken.ofproto.ofproto_v1_2_parser.OFPActionPushVlan(ethertype=33024,
                                                            type_=None,
                                                            len_=None)
```

Push VLAN action

This action pushes a new VLAN tag to the packet.

Attribute	Description
<code>ethertype</code>	Ether type. The default is 802.1Q. (0x8100)

```
class os_ken.ofproto.ofproto_v1_2_parser.OFPActionPushMpls(ethertype=34887,
                                                            type_=None,
                                                            len_=None)
```

Push MPLS action

This action pushes a new MPLS header to the packet.

Attribute	Description
<code>ethertype</code>	Ether type

```
class os_ken.ofproto.ofproto_v1_2_parser.OFPActionPopVlan(type_=None, len_=None)
```

Pop VLAN action

This action pops the outermost VLAN tag from the packet.

```
class os_ken.ofproto.ofproto_v1_2_parser.OFPActionPopMpls(ethertype=2048,  
type_=None, len_=None)
```

Pop MPLS action

This action pops the MPLS header from the packet.

```
class os_ken.ofproto.ofproto_v1_2_parser.OFPActionSetField(field=None, **kwargs)
```

Set field action

This action modifies a header field in the packet.

The set of keywords available for this is same as OFPMatch.

Example:

```
set_field = OFPActionSetField(eth_src="00:00:00:00:00:00")
```

```
class os_ken.ofproto.ofproto_v1_2_parser.OFPActionExperimenter(experimenter,  
type_=None,  
len_=None)
```

Experimenter action

This action is an extensible action for the experimenter.

Attribute	Description
experimenter	Experimenter ID

#### Note

For the list of the supported Nicira experimenter actions, please refer to [os\\_ken.ofproto.nx\\_actions](#).

## OpenFlow v1.3 Messages and Structures

### Controller-to-Switch Messages

#### Handshake

```
class os_ken.ofproto.ofproto_v1_3_parser.OFPFeaturesRequest(datapath)
```

Features request message

The controller sends a feature request to the switch upon session establishment.

This message is handled by the OSKen framework, so the OSKen application do not need to process this typically.

Example:

```
def send_features_request(self, datapath):
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPFeaturesRequest(datapath)
    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPFeaturesRequest": {}
}
```

```
class os_ken.ofproto.ofproto_v1_3_parser.OFPSwitchFeatures(datapath,
                                                           datapath_id=None,
                                                           n_buffers=None,
                                                           n_tables=None,
                                                           auxiliary_id=None,
                                                           capabilities=None)
```

Features reply message

The switch responds with a features reply message to a features request.

This message is handled by the OSKen framework, so the OSKen application do not need to process this typically.

Example:

```
@set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
def switch_features_handler(self, ev):
    msg = ev.msg

    self.logger.debug('OFPSwitchFeatures received: '
                      'datapath_id=0x%016x n_buffers=%d '
                      'n_tables=%d auxiliary_id=%d '
                      'capabilities=0x%08x',
                      msg.datapath_id, msg.n_buffers, msg.n_tables,
                      msg.auxiliary_id, msg.capabilities)
```

JSON Example:

```
{
  "OFPSwitchFeatures": {
    "auxiliary_id": 99,
    "capabilities": 79,
    "datapath_id": 9210263729383,
    "n_buffers": 0,
    "n_tables": 255
  }
}
```

## Switch Configuration

**class** `os_ken.ofproto.ofproto_v1_3_parser.OFPSetConfig`(*datapath*, *flags=0*,  
*miss\_send\_len=0*)

Set config request message

The controller sends a set config request message to set configuraion parameters.

Attribute	Description
<code>flags</code>	Bitmap of the following flags.  OFPC_FRAG_NORMAL OFPC_FRAG_DROP OFPC_FRAG_REASM
<code>miss_send_len</code>	Max bytes of new flow that datapath should send to the controller

Example:

```
def send_set_config(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPSetConfig(datapath, ofp.OFPC_FRAG_NORMAL, 256)
    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPSetConfig": {
    "flags": 0,
    "miss_send_len": 128
  }
}
```

**class** `os_ken.ofproto.ofproto_v1_3_parser.OFPGetConfigRequest`(*datapath*)

Get config request message

The controller sends a get config request to query configuration parameters in the switch.

Example:

```
def send_get_config_request(self, datapath):
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPGetConfigRequest(datapath)
    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPCGetConfigRequest": {}
}
```

**class** `os_ken.ofproto.ofproto_v1_3_parser.OFPCGetConfigReply`(*datapath*, *flags=None*, *miss\_send\_len=None*)

Get config reply message

The switch responds to a configuration request with a get config reply message.

Attribute	Description
flags	Bitmap of the following flags.  OFPC_FRAG_NORMAL OFPC_FRAG_DROP OFPC_FRAG_REASM OFPC_FRAG_MASK
miss_send_len	Max bytes of new flow that datapath should send to the controller

Example:

```
@set_ev_cls(ofp_event.EventOFPCGetConfigReply, MAIN_DISPATCHER)
def get_config_reply_handler(self, ev):
    msg = ev.msg
    dp = msg.datapath
    ofp = dp.ofproto
    flags = []

    if msg.flags & ofp.OFPC_FRAG_NORMAL:
        flags.append('NORMAL')
    if msg.flags & ofp.OFPC_FRAG_DROP:
        flags.append('DROP')
    if msg.flags & ofp.OFPC_FRAG_REASM:
        flags.append('REASM')
    self.logger.debug('OFPCGetConfigReply received: '
                      'flags=%s miss_send_len=%d',
                      ','.join(flags), msg.miss_send_len)
```

JSON Example:

```
{
  "OFPCGetConfigReply": {
    "flags": 0,
    "miss_send_len": 128
  }
}
```



## Flow Table Configuration

**class** `os_ken.ofproto.ofproto_v1_3_parser.OFPTTableMod(datapath, table_id, config)`

Flow table configuration message

The controller sends this message to configure table state.

Attribute	Description
<code>table_id</code>	ID of the table (OFPTT_ALL indicates all tables)
<code>config</code>	Bitmap of the following flags. OFPTC_DEPRECATED_MASK (3)

Example:

```
def send_table_mod(self, datapath):
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPTTableMod(datapath, 1, 3)
    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPTTableMod": {
    "config": 0,
    "table_id": 255
  }
}
```

## Modify State Messages

**class** `os_ken.ofproto.ofproto_v1_3_parser.OFPFlowMod(datapath, cookie=0, cookie_mask=0, table_id=0, command=0, idle_timeout=0, hard_timeout=0, priority=32768, buffer_id=4294967295, out_port=0, out_group=0, flags=0, match=None, instructions=None)`

Modify Flow entry message

The controller sends this message to modify the flow table.

Attribute	Description
cookie	Opaque controller-issued identifier
cookie_mask	Mask used to restrict the cookie bits that must match when the command is <code>OFPPFC_MODIFY*</code> or <code>OFPPFC_DELETE*</code>
table_id	ID of the table to put the flow in
command	One of the following values.  <code>OFPPFC_ADD</code> <code>OFPPFC_MODIFY</code> <code>OFPPFC_MODIFY_STRICT</code> <code>OFPPFC_DELETE</code> <code>OFPPFC_DELETE_STRICT</code>
idle_timeout	Idle time before discarding (seconds)
hard_timeout	Max time before discarding (seconds)
priority	Priority level of flow entry
buffer_id	Buffered packet to apply to (or <code>OFP_NO_BUFFER</code> )
out_port	For <code>OFPPFC_DELETE*</code> commands, require matching entries to include this as an output port
out_group	For <code>OFPPFC_DELETE*</code> commands, require matching entries to include this as an output group
flags	Bitmap of the following flags.  <code>OFPPFF_SEND_FLOW_REM</code> <code>OFPPFF_CHECK_OVERLAP</code> <code>OFPPFF_RESET_COUNTS</code> <code>OFPPFF_NO_PKT_COUNTS</code> <code>OFPPFF_NO_BYT_COUNTS</code>
match	Instance of <code>OFPMatch</code>
instructions	list of <code>OFPInstruction*</code> instance

Example:

```
def send_flow_mod(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    cookie = cookie_mask = 0
    table_id = 0
    idle_timeout = hard_timeout = 0
    priority = 32768
    buffer_id = ofp.OFP_NO_BUFFER
    match = ofp_parser.OFPMatch(in_port=1, eth_dst='ff:ff:ff:ff:ff:ff')
```

(continues on next page)

(continued from previous page)

```

actions = [ofp_parser.OFPActionOutput(ofp.OFPP_NORMAL, 0)]
inst = [ofp_parser.OFPInstructionActions(ofp.OFPIT_APPLY_ACTIONS,
                                         actions)]

req = ofp_parser.OFPFlowMod(datapath, cookie, cookie_mask,
                             table_id, ofp.OFPFC_ADD,
                             idle_timeout, hard_timeout,
                             priority, buffer_id,
                             ofp.OFPP_ANY, ofp.OFPG_ANY,
                             ofp.OFPFF_SEND_FLOW_REM,
                             match, inst)

datapath.send_msg(req)

```

JSON Example:

```

{
  "OFPFlowMod": {
    "buffer_id": 65535,
    "command": 0,
    "cookie": 0,
    "cookie_mask": 0,
    "flags": 0,
    "hard_timeout": 0,
    "idle_timeout": 0,
    "instructions": [
      {
        "OFPInstructionActions": {
          "actions": [
            {
              "OFPActionSetField": {
                "field": {
                  "OXMTlv": {
                    "field": "vlan_vid",
                    "mask": null,
                    "value": 258
                  }
                }
              },
              "len": 16,
              "type": 25
            }
          ],
          "OFPActionCopyTtlOut": {
            "len": 8,
            "type": 11
          },
          "OFPActionCopyTtlIn": {
            "len": 8,
            "type": 12
          }
        }
      }
    ]
  }
}

```

(continues on next page)

(continued from previous page)

```
    }
  },
  {
    "OFPActionCopyTtlIn": {
      "len": 8,
      "type": 12
    }
  },
  {
    "OFPActionPopPbb": {
      "len": 8,
      "type": 27
    }
  },
  {
    "OFPActionPushPbb": {
      "ethertype": 4660,
      "len": 8,
      "type": 26
    }
  },
  {
    "OFPActionPopMpls": {
      "ethertype": 39030,
      "len": 8,
      "type": 20
    }
  },
  {
    "OFPActionPushMpls": {
      "ethertype": 34887,
      "len": 8,
      "type": 19
    }
  },
  {
    "OFPActionPopVlan": {
      "len": 8,
      "type": 18
    }
  },
  {
    "OFPActionPushVlan": {
      "ethertype": 33024,
      "len": 8,
      "type": 17
    }
  },
  },
  {
```

(continues on next page)

(continued from previous page)

```
"OFPActionDecMplsTtl": {
  "len": 8,
  "type": 16
},
{
  "OFPActionSetMplsTtl": {
    "len": 8,
    "mpls_ttl": 10,
    "type": 15
  },
  {
    "OFPActionDecNwTtl": {
      "len": 8,
      "type": 24
    },
    {
      "OFPActionSetNwTtl": {
        "len": 8,
        "nw_ttl": 10,
        "type": 23
      },
      {
        "OFPActionExperimenterUnknown": {
          "data": "AAECAwQFBgc=",
          "experimenter": 101,
          "len": 16,
          "type": 65535
        },
        {
          "OFPActionSetQueue": {
            "len": 8,
            "queue_id": 3,
            "type": 21
          },
          {
            "OFPActionGroup": {
              "group_id": 99,
              "len": 8,
              "type": 22
            },
            {
              "OFPActionOutput": {
```

(continues on next page)

(continued from previous page)

```

        "len": 16,
        "max_len": 65535,
        "port": 6,
        "type": 0
    }
}
],
"len": 176,
"type": 3
},
{
    "OFPIInstructionActions": {
        "actions": [
            {
                "OFPActionSetField": {
                    "field": {
                        "OXMTlv": {
                            "field": "eth_src",
                            "mask": null,
                            "value": "01:02:03:04:05:06"
                        }
                    }
                },
                "len": 16,
                "type": 25
            }
        ],
        "len": 16,
        "type": 25
    }
},
{
    "OFPActionSetField": {
        "field": {
            "OXMTlv": {
                "field": "pbb_uca",
                "mask": null,
                "value": 1
            }
        }
    },
    "len": 16,
    "type": 25
}
],
"len": 40,
"type": 4
},
"match": {
    "OFPMatch": {
        "length": 14,

```

(continues on next page)

(continued from previous page)

```

        "oxm_fields": [
            {
                "OXMTlv": {
                    "field": "eth_dst",
                    "mask": null,
                    "value": "f2:0b:a4:7d:f8:ea"
                }
            }
        ],
        "type": 1
    }
},
"out_group": 4294967295,
"out_port": 4294967295,
"priority": 123,
"table_id": 1
}
}

```

```

{
  "OFPPFlowMod": {
    "buffer_id": 65535,
    "command": 0,
    "cookie": 0,
    "cookie_mask": 0,
    "flags": 0,
    "hard_timeout": 0,
    "idle_timeout": 0,
    "instructions": [
      {
        "OFPIInstructionGotoTable": {
          "len": 8,
          "table_id": 1,
          "type": 1
        }
      }
    ],
    "match": {
      "OFPMatch": {
        "length": 22,
        "oxm_fields": [
          {
            "OXMTlv": {
              "field": "in_port",
              "mask": null,
              "value": 6
            }
          }
        ],
      }
    }
  }
}

```

(continues on next page)

(continued from previous page)

```
        "OXMTlv": {
            "field": "eth_src",
            "mask": null,
            "value": "f2:0b:a4:7d:f8:ea"
        }
    ],
    "type": 1
}
},
"out_group": 4294967295,
"out_port": 4294967295,
"priority": 123,
"table_id": 0
}
```

```
{
  "OFPPFlowMod": {
    "buffer_id": 65535,
    "command": 0,
    "cookie": 0,
    "cookie_mask": 0,
    "flags": 0,
    "hard_timeout": 0,
    "idle_timeout": 0,
    "instructions": [
      {
        "OFPIInstructionMeter": {
          "len": 8,
          "meter_id": 1,
          "type": 6
        }
      },
      {
        "OFPIInstructionActions": {
          "actions": [
            {
              "OFPAActionOutput": {
                "len": 16,
                "max_len": 65535,
                "port": 6,
                "type": 0
              }
            }
          ],
          "len": 24,
          "type": 3
        }
      }
    ]
  }
}
```

(continues on next page)



(continued from previous page)

```

    }
  ],
  "match": {
    "OFPMatch": {
      "length": 14,
      "oxm_fields": [
        {
          "OXMTlv": {
            "field": "eth_dst",
            "mask": null,
            "value": "f2:0b:a4:7d:f8:ea"
          }
        }
      ]
    },
    "type": 1
  }
},
"out_group": 4294967295,
"out_port": 4294967295,
"priority": 123,
"table_id": 1
}
}

```

**class** `os_ken.ofproto.ofproto_v1_3_parser.OFPGGroupMod`(*datapath*, *command=0*, *type\_=0*, *group\_id=0*, *buckets=None*)

Modify group entry message

The controller sends this message to modify the group table.

Attribute	Description
<code>command</code>	One of the following values.  OFPGC_ADD OFPGC_MODIFY OFPGC_DELETE
<code>type</code>	One of the following values.  OFPGT_ALL OFPGT_SELECT OFPGT_INDIRECT OFPGT_FF
<code>group_id</code>	Group identifier
<code>buckets</code>	list of OFPBucket

`type` attribute corresponds to `type_` parameter of `__init__`.

Example:

```
def send_group_mod(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    port = 1
    max_len = 2000
    actions = [ofp_parser.OFPActionOutput(port, max_len)]

    weight = 100
    watch_port = 0
    watch_group = 0
    buckets = [ofp_parser.OFPBucket(weight, watch_port, watch_group,
                                     actions)]

    group_id = 1
    req = ofp_parser.OFPGroupMod(datapath, ofp.OFPGC_ADD,
                                  ofp.OFPGT_SELECT, group_id, buckets)

    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPGroupMod": {
    "buckets": [
      {
        "OFPBucket": {
          "actions": [
            {
              "OFPActionOutput": {
                "len": 16,
                "max_len": 65535,
                "port": 2,
                "type": 0
              }
            }
          ],
          "len": 32,
          "watch_group": 1,
          "watch_port": 1,
          "weight": 1
        }
      }
    ],
    "command": 0,
    "group_id": 1,
    "type": 0
  }
}
```

```
class os_ken.ofproto.ofproto_v1_3_parser.OFPPortMod(datapath, port_no=0,
                                                    hw_addr='00:00:00:00:00:00',
                                                    config=0, mask=0, advertise=0)
```

Port modification message

The controller sends this message to modify the behavior of the port.

Attribute	Description
port_no	Port number to modify
hw_addr	The hardware address that must be the same as hw_addr of OFPPort of OFPSwitchFeatures
config	Bitmap of configuration flags.  OFPPC_PORT_DOWN OFPPC_NO_RECV OFPPC_NO_FWD OFPPC_NO_PACKET_IN
mask	Bitmap of configuration flags above to be changed
advertise	Bitmap of the following flags.  OFPPF_10MB_HD OFPPF_10MB_FD OFPPF_100MB_HD OFPPF_100MB_FD OFPPF_1GB_HD OFPPF_1GB_FD OFPPF_10GB_FD OFPPF_40GB_FD OFPPF_100GB_FD OFPPF_1TB_FD OFPPF_OTHER OFPPF_COPPER OFPPF_FIBER OFPPF_AUTONEG OFPPF_PAUSE OFPPF_PAUSE_ASYM

Example:

```
def send_port_mod(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser
```

(continues on next page)

(continued from previous page)

```

port_no = 3
hw_addr = 'fa:c8:e8:76:1d:7e'
config = 0
mask = (ofp.OFPPC_PORT_DOWN | ofp.OFPPC_NO_RECV |
        ofp.OFPPC_NO_FWD | ofp.OFPPC_NO_PACKET_IN)
advertise = (ofp.OFPPF_10MB_FD | ofp.OFPPF_100MB_FD |
             ofp.OFPPF_1GB_FD | ofp.OFPPF_COPPER |
             ofp.OFPPF_AUTONEG | ofp.OFPPF_PAUSE |
             ofp.OFPPF_PAUSE_ASYM)
req = ofp_parser.OFPPortMod(datapath, port_no, hw_addr, config,
                             mask, advertise)
datapath.send_msg(req)

```

JSON Example:

```

{
  "OFPPortMod": {
    "advertise": 4096,
    "config": 0,
    "hw_addr": "00:11:00:00:11:11",
    "mask": 0,
    "port_no": 1
  }
}

```

**class** `os_ken.ofproto.ofproto_v1_3_parser.OFPMeterMod`(*datapath*, *command=0*, *flags=1*,  
*meter\_id=1*, *bands=None*)

Meter modification message

The controller sends this message to modify the meter.

Attribute	Description
command	One of the following values.  OFPMC_ADD OFPMC_MODIFY OFPMC_DELETE
flags	Bitmap of the following flags.  OFPMF_KBPS OFPMF_PKTPTS OFPMF_BURST OFPMF_STATS
meter_id	Meter instance
bands	list of the following class instance.  OFPMeterBandDrop OFPMeterBandDscpRemark OFPMeterBandExperimenter

JSON Example:

```
{
  "OFPMeterMod": {
    "bands": [
      {
        "OFPMeterBandDrop": {
          "burst_size": 10,
          "len": 16,
          "rate": 1000,
          "type": 1
        }
      },
      {
        "OFPMeterBandDscpRemark": {
          "burst_size": 10,
          "len": 16,
          "prec_level": 1,
          "rate": 1000,
          "type": 2
        }
      },
      {
        "OFPMeterBandExperimenter": {
          "burst_size": 10,
          "experimenter": 999,

```

(continues on next page)

(continued from previous page)

```

        "len": 16,
        "rate": 1000,
        "type": 65535
    }
}
],
"command": 0,
"flags": 14,
"meter_id": 100
}
}

```

## Multipart Messages

```
class os_ken.ofproto.ofproto_v1_3_parser.OFPDescStatsRequest(datapath, flags=0,
                                                             type_=None)
```

Description statistics request message

The controller uses this message to query description of the switch.

Attribute	Description
flags	Zero or OFPMPF_REQ_MORE

Example:

```
def send_desc_stats_request(self, datapath):
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPDescStatsRequest(datapath, 0)
    datapath.send_msg(req)
```

JSON Example:

```

{
  "OFPDescStatsRequest": {
    "flags": 0,
    "type": 0
  }
}

```

```
class os_ken.ofproto.ofproto_v1_3_parser.OFPDescStatsReply(datapath, type_=None,
                                                            **kwargs)
```

Description statistics reply message

The switch responds with this message to a description statistics request.

Attribute	Description
body	Instance of OFPDescStats

Example:

```
@set_ev_cls(ofp_event.EventOFPDescStatsReply, MAIN_DISPATCHER)
def desc_stats_reply_handler(self, ev):
    body = ev.msg.body

    self.logger.debug('DescStats: mfr_desc=%s hw_desc=%s sw_desc=%s '
                      'serial_num=%s dp_desc=%s',
                      body.mfr_desc, body.hw_desc, body.sw_desc,
                      body.serial_num, body.dp_desc)
```

JSON Example:

```
{
  "OFPDescStatsReply": {
    "body": {
      "OFPDescStats": {
        "dp_desc": "dp",
        "hw_desc": "hw",
        "mfr_desc": "mfr",
        "serial_num": "serial",
        "sw_desc": "sw"
      }
    },
    "flags": 0,
    "type": 0
  }
}
```

```
class os_ken.ofproto.ofproto_v1_3_parser.OFPFlowStatsRequest(datapath, flags=0,
                                                             table_id=255,
                                                             out_port=4294967295,
                                                             out_group=4294967295,
                                                             cookie=0,
                                                             cookie_mask=0,
                                                             match=None,
                                                             type_=None)
```

Individual flow statistics request message

The controller uses this message to query individual flow statistics.

Attribute	Description
flags	Zero or OFPMPF_REQ_MORE
table_id	ID of table to read
out_port	Require matching entries to include this as an output port
out_group	Require matching entries to include this as an output group
cookie	Require matching entries to contain this cookie value
cookie_mask	Mask used to restrict the cookie bits that must match
match	Instance of OFPMatch

Example:

```
def send_flow_stats_request(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    cookie = cookie_mask = 0
    match = ofp_parser.OFPMatch(in_port=1)
    req = ofp_parser.OFPFlowStatsRequest(datapath, 0,
                                          ofp.OFPTT_ALL,
                                          ofp.OFPP_ANY, ofp.OFPG_ANY,
                                          cookie, cookie_mask,
                                          match)

    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPFlowStatsRequest": {
    "cookie": 0,
    "cookie_mask": 0,
    "flags": 0,
    "match": {
      "OFPMatch": {
        "length": 4,
        "oxm_fields": [],
        "type": 1
      }
    },
    "out_group": 4294967295,
    "out_port": 4294967295,
    "table_id": 0,
    "type": 1
  }
}
```

**class** `os_ken.ofproto.ofproto_v1_3_parser.OFPFlowStatsReply`(*datapath*, *type\_=None*, *\*\*kwargs*)

Individual flow statistics reply message

The switch responds with this message to an individual flow statistics request.

Attribute	Description
body	List of OFPFlowStats instance

Example:

```
@set_ev_cls(ofp_event.EventOFPFlowStatsReply, MAIN_DISPATCHER)
def flow_stats_reply_handler(self, ev):
    flows = []
    for stat in ev.msg.body:
```

(continues on next page)



(continued from previous page)

```

flows.append('table_id=%s '
            'duration_sec=%d duration_nsec=%d '
            'priority=%d '
            'idle_timeout=%d hard_timeout=%d flags=0x%04x '
            'cookie=%d packet_count=%d byte_count=%d '
            'match=%s instructions=%s' %
            (stat.table_id,
             stat.duration_sec, stat.duration_nsec,
             stat.priority,
             stat.idle_timeout, stat.hard_timeout, stat.flags,
             stat.cookie, stat.packet_count, stat.byte_count,
             stat.match, stat.instructions))
self.logger.debug('FlowStats: %s', flows)

```

JSON Example:

```

{
  "OFPPFlowStatsReply": {
    "body": [
      {
        "OFPPFlowStats": {
          "byte_count": 0,
          "cookie": 0,
          "duration_nsec": 115277000,
          "duration_sec": 358,
          "flags": 0,
          "hard_timeout": 0,
          "idle_timeout": 0,
          "instructions": [],
          "length": 56,
          "match": {
            "OFPMatch": {
              "length": 4,
              "oxm_fields": [],
              "type": 1
            }
          },
          "packet_count": 0,
          "priority": 65535,
          "table_id": 0
        }
      },
      {
        "OFPPFlowStats": {
          "byte_count": 0,
          "cookie": 0,
          "duration_nsec": 115055000,
          "duration_sec": 358,
          "flags": 0,
          "hard_timeout": 0,

```

(continues on next page)

(continued from previous page)

```

        "idle_timeout": 0,
        "instructions": [
            {
                "OFPInstructionActions": {
                    "actions": [
                        {
                            "OFPActionOutput": {
                                "len": 16,
                                "max_len": 0,
                                "port": 4294967290,
                                "type": 0
                            }
                        }
                    ],
                    "len": 24,
                    "type": 4
                }
            }
        ],
        "length": 88,
        "match": {
            "OFPMatch": {
                "length": 10,
                "oxm_fields": [
                    {
                        "OXMTlv": {
                            "field": "eth_type",
                            "mask": null,
                            "value": 2054
                        }
                    }
                ],
                "type": 1
            }
        },
        "packet_count": 0,
        "priority": 65534,
        "table_id": 0
    },
    {
        "OFPFlowStats": {
            "byte_count": 238,
            "cookie": 0,
            "duration_nsec": 511582000,
            "duration_sec": 316220,
            "flags": 0,
            "hard_timeout": 0,
            "idle_timeout": 0,

```

(continues on next page)

(continued from previous page)

```

    "instructions": [
      {
        "OFPInstructionGotoTable": {
          "len": 8,
          "table_id": 1,
          "type": 1
        }
      }
    ],
    "length": 80,
    "match": {
      "OFPMatch": {
        "length": 22,
        "oxm_fields": [
          {
            "OXMTlv": {
              "field": "in_port",
              "mask": null,
              "value": 6
            }
          },
          {
            "OXMTlv": {
              "field": "eth_src",
              "mask": null,
              "value": "f2:0b:a4:7d:f8:ea"
            }
          }
        ]
      },
      "type": 1
    },
    "packet_count": 3,
    "priority": 123,
    "table_id": 0
  },
  {
    "OFPFlowStats": {
      "byte_count": 98,
      "cookie": 0,
      "duration_nsec": 980901000,
      "duration_sec": 313499,
      "flags": 0,
      "hard_timeout": 0,
      "idle_timeout": 0,
      "instructions": [
        {
          "OFPInstructionActions": {

```

(continues on next page)

(continued from previous page)

```
"actions": [  
  {  
    "OFPActionSetField": {  
      "field": {  
        "OXMTlv": {  
          "field": "vlan_vid",  
          "mask": null,  
          "value": 258  
        }  
      },  
      "len": 16,  
      "type": 25  
    }  
  },  
  {  
    "OFPActionCopyTtlOut": {  
      "len": 8,  
      "type": 11  
    }  
  },  
  {  
    "OFPActionCopyTtlIn": {  
      "len": 8,  
      "type": 12  
    }  
  },  
  {  
    "OFPActionCopyTtlIn": {  
      "len": 8,  
      "type": 12  
    }  
  },  
  {  
    "OFPActionPopPbb": {  
      "len": 8,  
      "type": 27  
    }  
  },  
  {  
    "OFPActionPushPbb": {  
      "ethertype": 4660,  
      "len": 8,  
      "type": 26  
    }  
  },  
  {  
    "OFPActionPopMpls": {  
      "ethertype": 39030,  
      "len": 8,  
      "type": 28  
    }  
  }  
]
```

(continues on next page)

(continued from previous page)

```
        "type": 20
    },
    {
        "OFPActionPushMpls": {
            "ethertype": 34887,
            "len": 8,
            "type": 19
        }
    },
    {
        "OFPActionPopVlan": {
            "len": 8,
            "type": 18
        }
    },
    {
        "OFPActionPushVlan": {
            "ethertype": 33024,
            "len": 8,
            "type": 17
        }
    },
    {
        "OFPActionDecMplsTtl": {
            "len": 8,
            "type": 16
        }
    },
    {
        "OFPActionSetMplsTtl": {
            "len": 8,
            "mpls_ttl": 10,
            "type": 15
        }
    },
    {
        "OFPActionDecNwTtl": {
            "len": 8,
            "type": 24
        }
    },
    {
        "OFPActionSetNwTtl": {
            "len": 8,
            "nw_ttl": 10,
            "type": 23
        }
    }
},
```

(continues on next page)

(continued from previous page)

```

    {
      "OFPActionSetQueue": {
        "len": 8,
        "queue_id": 3,
        "type": 21
      }
    },
    {
      "OFPActionGroup": {
        "group_id": 99,
        "len": 8,
        "type": 22
      }
    },
    {
      "OFPActionOutput": {
        "len": 16,
        "max_len": 65535,
        "port": 6,
        "type": 0
      }
    },
    {
      "OFPActionExperimenterUnknown": {
        "len": 16,
        "data": "ZXhwX2RhdGE=",
        "experimenter": 98765432,
        "type": 65535
      }
    },
    {
      "NXActionUnknown": {
        "len": 16,
        "data": "cF9kYXRh",
        "experimenter": 8992,
        "type": 65535,
        "subtype": 25976
      }
    }
  ],
  "len": 192,
  "type": 3
}
},
{
  "OFPIstructionActions": {
    "actions": [
      {
        "OFPActionSetField": {

```

(continues on next page)

(continued from previous page)

```

        "field": {
            "OXMTlv": {
                "field": "eth_src",
                "mask": null,
                "value": "01:02:03:04:05:06"
            }
        },
        "len": 16,
        "type": 25
    },
    {
        "OFPActionSetField": {
            "field": {
                "OXMTlv": {
                    "field": "pbb_uca",
                    "mask": null,
                    "value": 1
                }
            },
            "len": 16,
            "type": 25
        }
    },
    ],
    "len": 40,
    "type": 4
},
{
    "OFPIstructionActions": {
        "actions": [
            {
                "OFPActionOutput": {
                    "len": 16,
                    "max_len": 65535,
                    "port": 4294967293,
                    "type": 0
                }
            }
        ],
        "len": 24,
        "type": 3
    }
},
],
"length": 312,
"match": {
    "OFPMatch": {

```

(continues on next page)

(continued from previous page)

```

        "length": 4,
        "oxm_fields": [],
        "type": 1
    }
},
    "packet_count": 1,
    "priority": 0,
    "table_id": 0
}
],
"flags": 0,
"type": 1
}
}

```

```

class os_ken.ofproto.ofproto_v1_3_parser.OFPAggregateStatsRequest(datapath, flags,
                                                                    table_id,
                                                                    out_port,
                                                                    out_group,
                                                                    cookie,
                                                                    cookie_mask,
                                                                    match,
                                                                    type_=None)

```

Aggregate flow statistics request message

The controller uses this message to query aggregate flow statistics.

Attribute	Description
flags	Zero or OFPMPF_REQ_MORE
table_id	ID of table to read
out_port	Require matching entries to include this as an output port
out_group	Require matching entries to include this as an output group
cookie	Require matching entries to contain this cookie value
cookie_mask	Mask used to restrict the cookie bits that must match
match	Instance of OFPMatch

Example:

```

def send_aggregate_stats_request(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    cookie = cookie_mask = 0
    match = ofp_parser.OFPMatch(in_port=1)
    req = ofp_parser.OFPAggregateStatsRequest(datapath, 0,
                                              ofp.OFPTT_ALL,
                                              ofp.OFPP_ANY,
                                              ofp.OFPG_ANY,

```

(continues on next page)



(continued from previous page)

```

        cookie, cookie_mask,
        match)
datapath.send_msg(req)

```

JSON Example:

```

{
  "OFPAggregateStatsRequest": {
    "cookie": 0,
    "cookie_mask": 0,
    "flags": 0,
    "match": {
      "OFPMatch": {
        "length": 4,
        "oxm_fields": [],
        "type": 1
      }
    },
    "out_group": 4294967295,
    "out_port": 4294967295,
    "table_id": 255,
    "type": 2
  }
}

```

```

class os_ken.ofproto.ofproto_v1_3_parser.OFPAggregateStatsReply(datapath,
                                                                type_=None,
                                                                **kwargs)

```

Aggregate flow statistics reply message

The switch responds with this message to an aggregate flow statistics request.

Attribute	Description
body	Instance of OFPAggregateStats

Example:

```

@set_ev_cls(ofp_event.EventOFPAggregateStatsReply, MAIN_DISPATCHER)
def aggregate_stats_reply_handler(self, ev):
    body = ev.msg.body

    self.logger.debug('AggregateStats: packet_count=%d byte_count=%d '
                      'flow_count=%d',
                      body.packet_count, body.byte_count,
                      body.flow_count)

```

JSON Example:

```
{
  "OFPAggregateStatsReply": {
    "body": {
      "OFPAggregateStats": {
        "byte_count": 574,
        "flow_count": 6,
        "packet_count": 7
      }
    },
    "flags": 0,
    "type": 2
  }
}
```

**class** `os_ken.ofproto.ofproto_v1_3_parser.OFPTableStatsRequest`(*datapath*, *flags=0*, *type\_=None*)

Table statistics request message

The controller uses this message to query flow table statistics.

Attribute	Description
flags	Zero or OFPMPF_REQ_MORE

Example:

```
def send_table_stats_request(self, datapath):
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPTableStatsRequest(datapath, 0)
    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPTableStatsRequest": {
    "flags": 0,
    "type": 3
  }
}
```

**class** `os_ken.ofproto.ofproto_v1_3_parser.OFPTableStatsReply`(*datapath*, *type\_=None*, *\*\*kwargs*)

Table statistics reply message

The switch responds with this message to a table statistics request.

Attribute	Description
body	List of OFPTableStats instance

Example:

```
@set_ev_cls(ofp_event.EventOFPTableStatsReply, MAIN_DISPATCHER)
def table_stats_reply_handler(self, ev):
    tables = []
    for stat in ev.msg.body:
        tables.append('table_id=%d active_count=%d lookup_count=%d '
                    ' matched_count=%d' %
                    (stat.table_id, stat.active_count,
                     stat.lookup_count, stat.matched_count))
    self.logger.debug('TableStats: %s', tables)
```

JSON Example:

```
{
  "OFPTableStatsReply": {
    "body": [
      {
        "OFPTableStats": {
          "active_count": 4,
          "lookup_count": 4,
          "matched_count": 4,
          "table_id": 0
        }
      },
      {
        "OFPTableStats": {
          "active_count": 4,
          "lookup_count": 4,
          "matched_count": 4,
          "table_id": 1
        }
      }
    ],
    "flags": 0,
    "type": 3
  }
}
```

```
class os_ken.ofproto.ofproto_v1_3_parser.OFPPortStatsRequest(datapath, flags=0,
                                                             port_no=4294967295,
                                                             type_=None)
```

Port statistics request message

The controller uses this message to query information about ports statistics.

Attribute	Description
flags	Zero or OFPMPF_REQ_MORE
port_no	Port number to read (OFPP_ANY to all ports)

Example:

```
def send_port_stats_request(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPPortStatsRequest(datapath, 0, ofp.OFPP_ANY)
    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPPortStatsRequest": {
    "flags": 0,
    "port_no": 4294967295,
    "type": 4
  }
}
```

```
class os_ken.ofproto.ofproto_v1_3_parser.OFPPortStatsReply(datapath, type_=None,
                                                           **kwargs)
```

Port statistics reply message

The switch responds with this message to a port statistics request.

Attribute	Description
body	List of OFPPortStats instance

Example:

```
@set_ev_cls(ofp_event.EventOFPPortStatsReply, MAIN_DISPATCHER)
def port_stats_reply_handler(self, ev):
    ports = []
    for stat in ev.msg.body:
        ports.append('port_no=%d '
                    'rx_packets=%d tx_packets=%d '
                    'rx_bytes=%d tx_bytes=%d '
                    'rx_dropped=%d tx_dropped=%d '
                    'rx_errors=%d tx_errors=%d '
                    'rx_frame_err=%d rx_over_err=%d rx_crc_err=%d '
                    'collisions=%d duration_sec=%d duration_nsec=%d' %
                    (stat.port_no,
                     stat.rx_packets, stat.tx_packets,
                     stat.rx_bytes, stat.tx_bytes,
                     stat.rx_dropped, stat.tx_dropped,
                     stat.rx_errors, stat.tx_errors,
                     stat.rx_frame_err, stat.rx_over_err,
                     stat.rx_crc_err, stat.collisions,
                     stat.duration_sec, stat.duration_nsec))
    self.logger.debug('PortStats: %s', ports)
```

JSON Example:

```

{
  "OFPPortStatsReply": {
    "body": [
      {
        "OFPPortStats": {
          "collisions": 0,
          "duration_nsec": 0,
          "duration_sec": 0,
          "port_no": 7,
          "rx_bytes": 0,
          "rx_crc_err": 0,
          "rx_dropped": 0,
          "rx_errors": 0,
          "rx_frame_err": 0,
          "rx_over_err": 0,
          "rx_packets": 0,
          "tx_bytes": 336,
          "tx_dropped": 0,
          "tx_errors": 0,
          "tx_packets": 4
        }
      },
      {
        "OFPPortStats": {
          "collisions": 0,
          "duration_nsec": 0,
          "duration_sec": 0,
          "port_no": 6,
          "rx_bytes": 336,
          "rx_crc_err": 0,
          "rx_dropped": 0,
          "rx_errors": 0,
          "rx_frame_err": 0,
          "rx_over_err": 0,
          "rx_packets": 4,
          "tx_bytes": 336,
          "tx_dropped": 0,
          "tx_errors": 0,
          "tx_packets": 4
        }
      }
    ],
    "flags": 0,
    "type": 4
  }
}

```

```

class os_ken.ofproto.ofproto_v1_3_parser.OFPPortDescStatsRequest(datapath,
                                                                    flags=0,
                                                                    type_=None)

```

Port description request message

The controller uses this message to query description of all the ports.

Attribute	Description
flags	Zero or OFPMPF_REQ_MORE

Example:

```
def send_port_desc_stats_request(self, datapath):
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPPortDescStatsRequest(datapath, 0)
    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPPortDescStatsRequest": {
    "flags": 0,
    "type": 13
  }
}
```

```
class os_ken.ofproto.ofproto_v1_3_parser.OFPPortDescStatsReply(datapath,
                                                                type_=None,
                                                                **kwargs)
```

Port description reply message

The switch responds with this message to a port description request.

Attribute	Description
body	List of OFPPort instance

Example:

```
@set_ev_cls(ofp_event.EventOFPPortDescStatsReply, MAIN_DISPATCHER)
def port_desc_stats_reply_handler(self, ev):
    ports = []
    for p in ev.msg.body:
        ports.append('port_no=%d hw_addr=%s name=%s config=0x%08x '
                    'state=0x%08x curr=0x%08x advertised=0x%08x '
                    'supported=0x%08x peer=0x%08x curr_speed=%d '
                    'max_speed=%d' %
                    (p.port_no, p.hw_addr,
                     p.name, p.config,
                     p.state, p.curr, p.advertised,
                     p.supported, p.peer, p.curr_speed,
                     p.max_speed))
    self.logger.debug('OFPPortDescStatsReply received: %s', ports)
```

JSON Example:

```
{
  "OFPPortDescStatsReply": {
    "body": [
      {
        "OFPPort": {
          "advertised": 10240,
          "config": 0,
          "curr": 10248,
          "curr_speed": 5000,
          "hw_addr": "f2:0b:a4:d0:3f:70",
          "max_speed": 5000,
          "name": "Port7",
          "peer": 10248,
          "port_no": 7,
          "state": 4,
          "supported": 10248
        }
      },
      {
        "OFPPort": {
          "advertised": 10240,
          "config": 0,
          "curr": 10248,
          "curr_speed": 5000,
          "hw_addr": "f2:0b:a4:7d:f8:ea",
          "max_speed": 5000,
          "name": "Port6",
          "peer": 10248,
          "port_no": 6,
          "state": 4,
          "supported": 10248
        }
      }
    ],
    "flags": 0,
    "type": 13
  }
}
```

```
class os_ken.ofproto.ofproto_v1_3_parser.OFPQueueStatsRequest(datapath, flags=0,
                                                                port_no=4294967295,
                                                                queue_id=4294967295,
                                                                type_=None)
```

Queue statistics request message

The controller uses this message to query queue statistics.

Attribute	Description
flags	Zero or OFPMPF_REQ_MORE
port_no	Port number to read
queue_id	ID of queue to read

Example:

```
def send_queue_stats_request(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPQueueStatsRequest(datapath, 0, ofp.OFPP_ANY,
                                           ofp.OFPQ_ALL)

    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPQueueStatsRequest": {
    "flags": 0,
    "port_no": 4294967295,
    "queue_id": 4294967295,
    "type": 5
  }
}
```

```
class os_ken.ofproto.ofproto_v1_3_parser.OFPQueueStatsReply(datapath, type_=None,
                                                             **kwargs)
```

Queue statistics reply message

The switch responds with this message to an aggregate flow statistics request.

Attribute	Description
body	List of OFPQueueStats instance

Example:

```
@set_ev_cls(ofp_event.EventOFPQueueStatsReply, MAIN_DISPATCHER)
def queue_stats_reply_handler(self, ev):
    queues = []
    for stat in ev.msg.body:
        queues.append('port_no=%d queue_id=%d '
                     'tx_bytes=%d tx_packets=%d tx_errors=%d '
                     'duration_sec=%d duration_nsec=%d' %
                     (stat.port_no, stat.queue_id,
                      stat.tx_bytes, stat.tx_packets, stat.tx_errors,
                      stat.duration_sec, stat.duration_nsec))
    self.logger.debug('QueueStats: %s', queues)
```



JSON Example:

```
{
  "OFPQueueStatsReply": {
    "body": [
      {
        "OFPQueueStats": {
          "duration_nsec": 0,
          "duration_sec": 0,
          "port_no": 7,
          "queue_id": 1,
          "tx_bytes": 0,
          "tx_errors": 0,
          "tx_packets": 0
        }
      },
      {
        "OFPQueueStats": {
          "duration_nsec": 0,
          "duration_sec": 0,
          "port_no": 6,
          "queue_id": 1,
          "tx_bytes": 0,
          "tx_errors": 0,
          "tx_packets": 0
        }
      },
      {
        "OFPQueueStats": {
          "duration_nsec": 0,
          "duration_sec": 0,
          "port_no": 7,
          "queue_id": 2,
          "tx_bytes": 0,
          "tx_errors": 0,
          "tx_packets": 0
        }
      }
    ],
    "flags": 0,
    "type": 5
  }
}
```

```
class os_ken.ofproto.ofproto_v1_3_parser.OFPGroupStatsRequest(datapath, flags=0,
                                                                group_id=4294967292,
                                                                type_=None)
```

Group statistics request message

The controller uses this message to query statistics of one or more groups.

Attribute	Description
flags	Zero or OFPMPF_REQ_MORE
group_id	ID of group to read (OFPG_ALL to all groups)

Example:

```
def send_group_stats_request(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPGroupStatsRequest(datapath, 0, ofp.OFPG_ALL)
    datapath.send_msg(req)
```

```
class os_ken.ofproto.ofproto_v1_3_parser.OFPGroupStatsReply(datapath, type_=None,
                                                            **kwargs)
```

Group statistics reply message

The switch responds with this message to a group statistics request.

Attribute	Description
body	List of OFPGroupStats instance

Example:

```
@set_ev_cls(ofp_event.EventOFPGroupStatsReply, MAIN_DISPATCHER)
def group_stats_reply_handler(self, ev):
    groups = []
    for stat in ev.msg.body:
        groups.append('length=%d group_id=%d '
                    'ref_count=%d packet_count=%d byte_count=%d '
                    'duration_sec=%d duration_nsec=%d' %
                    (stat.length, stat.group_id,
                     stat.ref_count, stat.packet_count,
                     stat.byte_count, stat.duration_sec,
                     stat.duration_nsec))
    self.logger.debug('GroupStats: %s', groups)
```

```
class os_ken.ofproto.ofproto_v1_3_parser.OFPGroupDescStatsRequest(datapath,
                                                                    flags=0,
                                                                    type_=None)
```

Group description request message

The controller uses this message to list the set of groups on a switch.

Attribute	Description
flags	Zero or OFPMPF_REQ_MORE

Example:

```
def send_group_desc_stats_request(self, datapath):
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPGroupDescStatsRequest(datapath, 0)
    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPGroupDescStatsRequest": {
    "flags": 0,
    "type": 7
  }
}
```

```
class os_ken.ofproto.ofproto_v1_3_parser.OFPGroupDescStatsReply(datapath,
                                                                type_=None,
                                                                **kwargs)
```

Group description reply message

The switch responds with this message to a group description request.

Attribute	Description
body	List of OFPGroupDescStats instance

Example:

```
@set_ev_cls(ofp_event.EventOFPGroupDescStatsReply, MAIN_DISPATCHER)
def group_desc_stats_reply_handler(self, ev):
    descs = []
    for stat in ev.msg.body:
        descs.append('length=%d type=%d group_id=%d '
                    'buckets=%s' %
                    (stat.length, stat.type, stat.group_id,
                     stat.bucket))
    self.logger.debug('GroupDescStats: %s', descs)
```

JSON Example:

```
{
  "OFPGroupDescStatsReply": {
    "body": [
      {
        "OFPGroupDescStats": {
          "buckets": [
            {
              "OFPBucket": {
                "actions": [
                  {
                    "OFPActionOutput": {
```

(continues on next page)

(continued from previous page)

```

        "len": 16,
        "max_len": 65535,
        "port": 2,
        "type": 0
    }
}
],
"len": 32,
"watch_group": 1,
"watch_port": 1,
"weight": 1
}
}
],
"group_id": 1,
"length": 40,
"type": 0
}
}
],
"flags": 0,
"type": 7
}
}

```

```

class os_ken.ofproto.ofproto_v1_3_parser.OFPGroupFeaturesStatsRequest(datapath,
                                                                    flags=0,
                                                                    type_=None)

```

Group features request message

The controller uses this message to list the capabilities of groups on a switch.

Attribute	Description
flags	Zero or OFPMPF_REQ_MORE

Example:

```

def send_group_features_stats_request(self, datapath):
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPGroupFeaturesStatsRequest(datapath, 0)
    datapath.send_msg(req)

```

JSON Example:

```

{
  "OFPGroupFeaturesStatsRequest": {
    "flags": 0,
    "type": 8
  }
}

```

(continues on next page)

(continued from previous page)

```

    }
}

```

```

class os_ken.ofproto.ofproto_v1_3_parser.OFPGroupFeaturesStatsReply(datapath,
                                                                    type_=None,
                                                                    **kwargs)

```

Group features reply message

The switch responds with this message to a group features request.

Attribute	Description
body	Instance of OFPGroupFeaturesStats

Example:

```

@set_ev_cls(ofp_event.EventOFPGroupFeaturesStatsReply, MAIN_DISPATCHER)
def group_features_stats_reply_handler(self, ev):
    body = ev.msg.body

    self.logger.debug('GroupFeaturesStats: types=%d '
                      'capabilities=0x%08x max_groups=%s '
                      'actions=%s',
                      body.types, body.capabilities,
                      body.max_groups, body.actions)

```

JSON Example:

```

{
  "OFPGroupFeaturesStatsReply": {
    "body": {
      "OFPGroupFeaturesStats": {
        "actions": [
          67082241,
          67082241,
          67082241,
          67082241
        ],
        "capabilities": 5,
        "max_groups": [
          16777216,
          16777216,
          16777216,
          16777216
        ],
        "types": 15
      }
    },
    "flags": 0,
    "type": 8
  }
}

```

(continues on next page)

(continued from previous page)

```
}
}
```

```
class os_ken.ofproto.ofproto_v1_3_parser.OFPMeterStatsRequest(datapath, flags=0,
                                                                meter_id=4294967295,
                                                                type_=None)
```

Meter statistics request message

The controller uses this message to query statistics for one or more meters.

Attribute	Description
flags	Zero or OFPMPF_REQ_MORE
meter_id	ID of meter to read (OFPM_ALL to all meters)

Example:

```
def send_meter_stats_request(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPMeterStatsRequest(datapath, 0, ofp.OFPM_ALL)
    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPMeterStatsRequest": {
    "flags": 0,
    "meter_id": 4294967295,
    "type": 9
  }
}
```

```
class os_ken.ofproto.ofproto_v1_3_parser.OFPMeterStatsReply(datapath, type_=None,
                                                             **kwargs)
```

Meter statistics reply message

The switch responds with this message to a meter statistics request.

Attribute	Description
body	List of OFPMeterStats instance

Example:

```
@set_ev_cls(ofp_event.EventOFPMeterStatsReply, MAIN_DISPATCHER)
def meter_stats_reply_handler(self, ev):
    meters = []
```

(continues on next page)

(continued from previous page)

```

for stat in ev.msg.body:
    meters.append('meter_id=0x%08x len=%d flow_count=%d '
                 'packet_in_count=%d byte_in_count=%d '
                 'duration_sec=%d duration_nsec=%d '
                 'band_stats=%s' %
                 (stat.meter_id, stat.len, stat.flow_count,
                  stat.packet_in_count, stat.byte_in_count,
                  stat.duration_sec, stat.duration_nsec,
                  stat.band_stats))
self.logger.debug('MeterStats: %s', meters)

```

JSON Example:

```

{
  "OFPMeterStatsReply": {
    "body": [
      {
        "OFPMeterStats": {
          "band_stats": [
            {
              "OFPMeterBandStats": {
                "byte_band_count": 0,
                "packet_band_count": 0
              }
            }
          ],
          "byte_in_count": 0,
          "duration_nsec": 480000,
          "duration_sec": 0,
          "flow_count": 0,
          "len": 56,
          "meter_id": 100,
          "packet_in_count": 0
        }
      }
    ],
    "flags": 0,
    "type": 9
  }
}

```

```

class os_ken.ofproto.ofproto_v1_3_parser.OFPMeterConfigStatsRequest(datapath,
                                                                    flags=0, me-
                                                                    ter_id=4294967295,
                                                                    type_=None)

```

Meter configuration statistics request message

The controller uses this message to query configuration for one or more meters.

Attribute	Description
flags	Zero or OFPMPF_REQ_MORE
meter_id	ID of meter to read (OFPM_ALL to all meters)

Example:

```
def send_meter_config_stats_request(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPMeterConfigStatsRequest(datapath, 0,
                                                ofp.OFPM_ALL)

    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPMeterConfigStatsRequest": {
    "flags": 0,
    "meter_id": 4294967295,
    "type": 10
  }
}
```

```
class os_ken.ofproto.ofproto_v1_3_parser.OFPMeterConfigStatsReply(datapath,
                                                                    type_=None,
                                                                    **kwargs)
```

Meter configuration statistics reply message

The switch responds with this message to a meter configuration statistics request.

Attribute	Description
body	List of OFPMeterConfigStats instance

Example:

```
@set_ev_cls(ofp_event.EventOFPMeterConfigStatsReply, MAIN_DISPATCHER)
def meter_config_stats_reply_handler(self, ev):
    configs = []
    for stat in ev.msg.body:
        configs.append('length=%d flags=0x%04x meter_id=0x%08x '
                      'bands=%s' %
                      (stat.length, stat.flags, stat.meter_id,
                       stat.bands))
    self.logger.debug('MeterConfigStats: %s', configs)
```

JSON Example:



```

{
  "OFPMeterConfigStatsReply": {
    "body": [
      {
        "OFPMeterConfigStats": {
          "bands": [
            {
              "OFPMeterBandDrop": {
                "burst_size": 10,
                "len": 16,
                "rate": 1000,
                "type": 1
              }
            }
          ],
          "flags": 14,
          "length": 24,
          "meter_id": 100
        }
      }
    ],
    "flags": 0,
    "type": 10
  }
}

```

```

class os_ken.ofproto.ofproto_v1_3_parser.OFPMeterFeaturesStatsRequest(datapath,
                                                                    flags=0,
                                                                    type_=None)

```

Meter features statistics request message

The controller uses this message to query the set of features of the metering subsystem.

Attribute	Description
flags	Zero or OFPMPF_REQ_MORE

Example:

```

def send_meter_features_stats_request(self, datapath):
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPMeterFeaturesStatsRequest(datapath, 0)
    datapath.send_msg(req)

```

JSON Example:

```

{
  "OFPMeterFeaturesStatsRequest": {
    "flags": 0,

```

(continues on next page)

(continued from previous page)

```

        "type": 11
    }
}

```

```

class os_ken.ofproto.ofproto_v1_3_parser.OFPMeterFeaturesStatsReply(datapath,
                                                                    type_=None,
                                                                    **kwargs)

```

Meter features statistics reply message

The switch responds with this message to a meter features statistics request.

Attribute	Description
body	List of OFPMeterFeaturesStats instance

Example:

```

@set_ev_cls(ofp_event.EventOFPMeterFeaturesStatsReply, MAIN_DISPATCHER)
def meter_features_stats_reply_handler(self, ev):
    features = []
    for stat in ev.msg.body:
        features.append('max_meter=%d band_types=0x%08x '
                       'capabilities=0x%08x max_bands=%d '
                       'max_color=%d' %
                       (stat.max_meter, stat.band_types,
                        stat.capabilities, stat.max_bands,
                        stat.max_color))
    self.logger.debug('MeterFeaturesStats: %s', features)

```

JSON Example:

```

{
  "OFPMeterFeaturesStatsReply": {
    "body": [
      {
        "OFPMeterFeaturesStats": {
          "band_types": 2147483654,
          "capabilities": 15,
          "max_bands": 255,
          "max_color": 0,
          "max_meter": 16777216
        }
      }
    ],
    "flags": 0,
    "type": 11
  }
}

```

```
class os_ken.ofproto.ofproto_v1_3_parser.OFPTableFeaturesStatsRequest(datapath,
                                                                    flags=0,
                                                                    body=None,
                                                                    type_=None)
```

Table features statistics request message

The controller uses this message to query table features.

Attribute	Description
body	List of OFPTableFeaturesStats instances. The default is [].

```
class os_ken.ofproto.ofproto_v1_3_parser.OFPTableFeaturesStatsReply(datapath,
                                                                    type_=None,
                                                                    **kwargs)
```

Table features statistics reply message

The switch responds with this message to a table features statistics request.

Attribute	Description
body	List of OFPTableFeaturesStats instance

JSON Example:

See an example in:

```
os_ken/tests/unit/ofproto/json/of13/4-56-ofp_table_features_reply.
packet.json
```

## Queue Configuration Messages

```
class os_ken.ofproto.ofproto_v1_3_parser.OFPQueueGetConfigRequest(datapath, port)
Queue configuration request message
```

Attribute	Description
port	Port to be queried (OFPP_ANY to all configured queues)

Example:

```
def send_queue_get_config_request(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPQueueGetConfigRequest(datapath, ofp.OFPP_ANY)
    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPPQueueGetConfigRequest": {
    "port": 4294967295
  }
}
```

```
class os_ken.ofproto.ofproto_v1_3_parser.OFPPQueueGetConfigReply(datapath,
                                                                    queues=None,
                                                                    port=None)
```

Queue configuration reply message

The switch responds with this message to a queue configuration request.

Attribute	Description
queues	list of OFPPacketQueue instance
port	Port which was queried

Example:

```
@set_ev_cls(ofp_event.EventOFPPQueueGetConfigReply, MAIN_DISPATCHER)
def queue_get_config_reply_handler(self, ev):
    msg = ev.msg

    self.logger.debug('OFPPQueueGetConfigReply received: '
                      'port=%s queues=%s',
                      msg.port, msg.queues)
```

JSON Example:

```
{
  "OFPPQueueGetConfigReply": {
    "port": 4294967295,
    "queues": [
      {
        "OFPPacketQueue": {
          "len": 64,
          "port": 77,
          "properties": [
            {
              "OFPPQueuePropMinRate": {
                "len": 16,
                "property": 1,
                "rate": 10
              }
            },
            {
              "OFPPQueuePropMaxRate": {
                "len": 16,
                "property": 2,
```

(continues on next page)

(continued from previous page)

```

        "rate": 900
    }
},
{
    "OFPPacketQueue": {
        "len": 65,
        "port": 77,
        "properties": [
            {
                "OFPPacketQueuePropMinRate": {
                    "len": 16,
                    "property": 1,
                    "rate": 100
                }
            },
            {
                "OFPPacketQueuePropMaxRate": {
                    "len": 16,
                    "property": 2,
                    "rate": 200
                }
            }
        ],
        "OFPPacketQueuePropExperimenter": {
            "experimenter": 999,
            "data": [
                1
            ],
            "len": 17,
            "property": 65535
        }
    },
    "queue_id": 88
},
{

```

(continues on next page)

(continued from previous page)

```

    "OFPPacketQueue": {
        "len": 66,
        "port": 77,
        "properties": [
            {
                "OFPPQueuePropMinRate": {
                    "len": 16,
                    "property": 1,
                    "rate": 200
                }
            },
            {
                "OFPPQueuePropMaxRate": {
                    "len": 16,
                    "property": 2,
                    "rate": 400
                }
            },
            {
                "OFPPQueuePropExperimenter": {
                    "experimenter": 999,
                    "data": [
                        1,
                        2
                    ],
                    "len": 18,
                    "property": 65535
                }
            }
        ],
        "queue_id": 77
    }
}

```

### Packet-Out Message

```

class os_ken.ofproto.ofproto_v1_3_parser.OFPacketOut(datapath, buffer_id=None,
                                                       in_port=None, actions=None,
                                                       data=None, actions_len=None)

```

Packet-Out message

The controller uses this message to send a packet out through the switch.

Attribute	Description
buffer_id	ID assigned by datapath (OFP_NO_BUFFER if none)
in_port	Packet's input port or OFPP_CONTROLLER
actions	list of OpenFlow action class
data	Packet data of a binary type value or an instances of packet.Packet.

Example:

```
def send_packet_out(self, datapath, buffer_id, in_port):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    actions = [ofp_parser.OFPActionOutput(ofp.OFPP_FLOOD, 0)]
    req = ofp_parser.OFPPacketOut(datapath, buffer_id,
                                   in_port, actions)

    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPPacketOut": {
    "actions": [
      {
        "OFPActionOutput": {
          "len": 16,
          "max_len": 65535,
          "port": 4294967292,
          "type": 0
        }
      }
    ],
    "actions_len": 16,
    "buffer_id": 4294967295,
    "data":
    ↪ "8guk0D9w8gukffjqCABFAABU+BoAAP8Br4sKAAABCgAAAggAAgj3YAAAMdYCAAAAAACrjS0xAAAAABAREhMU
    ↪",
    "in_port": 4294967293
  }
}
```

## Barrier Message

**class** `os_ken.ofproto.ofproto_v1_3_parser.OFPBarrierRequest` (*datapath*)

Barrier request message

The controller sends this message to ensure message dependencies have been met or receive notifications for completed operations.

Example:

```
def send_barrier_request(self, datapath):
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPBarrierRequest(datapath)
    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPBarrierRequest": {}
}
```

**class** os\_ken.ofproto.ofproto\_v1\_3\_parser.OFPBarrierReply(*datapath*)

Barrier reply message

The switch responds with this message to a barrier request.

Example:

```
@set_ev_cls(ofp_event.EventOFPBarrierReply, MAIN_DISPATCHER)
def barrier_reply_handler(self, ev):
    self.logger.debug('OFPBarrierReply received')
```

JSON Example:

```
{
  "OFPBarrierReply": {}
}
```

## Role Request Message

**class** os\_ken.ofproto.ofproto\_v1\_3\_parser.OFPRoleRequest(*datapath*, *role=None*, *generation\_id=None*)

Role request message

The controller uses this message to change its role.

Attribute	Description
role	One of the following values.  OFPCR_ROLE_NOCHANGE OFPCR_ROLE_EQUAL OFPCR_ROLE_MASTER OFPCR_ROLE_SLAVE
generation_id	Master Election Generation ID

Example:



```
def send_role_request(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPCRoleRequest(datapath, ofp.OFPCR_ROLE_EQUAL, 0)
    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPCRoleRequest": {
    "generation_id": 17294086455919964160,
    "role": 2
  }
}
```

```
class os_ken.ofproto.ofproto_v1_3_parser.OFPCRoleReply(datapath, role=None,
                                                       generation_id=None)
```

Role reply message

The switch responds with this message to a role request.

Attribute	Description
role	One of the following values.  OFPCR_ROLE_NOCHANGE OFPCR_ROLE_EQUAL OFPCR_ROLE_MASTER OFPCR_ROLE_SLAVE
generation_id	Master Election Generation ID

Example:

```
@set_ev_cls(ofp_event.EventOFPCRoleReply, MAIN_DISPATCHER)
def role_reply_handler(self, ev):
    msg = ev.msg
    dp = msg.datapath
    ofp = dp.ofproto

    if msg.role == ofp.OFPCR_ROLE_NOCHANGE:
        role = 'NOCHANGE'
    elif msg.role == ofp.OFPCR_ROLE_EQUAL:
        role = 'EQUAL'
    elif msg.role == ofp.OFPCR_ROLE_MASTER:
        role = 'MASTER'
    elif msg.role == ofp.OFPCR_ROLE_SLAVE:
        role = 'SLAVE'
    else:
```

(continues on next page)

(continued from previous page)

```
role = 'unknown'  
  
self.logger.debug('OFPRoleReply received: '  
                  'role=%s generation_id=%d',  
                  role, msg.generation_id)
```

JSON Example:

```
{  
  "OFPRoleReply": {  
    "generation_id": 17294086455919964160,  
    "role": 3  
  }  
}
```

### Set Asynchronous Configuration Message

```
class os_ken.ofproto.ofproto_v1_3_parser.OFPSetAsync(datapath, packet_in_mask,  
                                                    port_status_mask,  
                                                    flow_removed_mask)
```

Set asynchronous configuration message

The controller sends this message to set the asynchronous messages that it wants to receive on a given OpenFlow channel.

Attribute	Description
packet_in_mask	2-element array: element 0, when the controller has a OFPCR_ROLE_EQUAL or OFPCR_ROLE_MASTER role. element 1, OFPCR_ROLE_SLAVE role controller. Bitmasks of following values.  OFPR_NO_MATCH OFPR_ACTION OFPR_INVALID_TTL
port_status_mask	2-element array. Bitmasks of following values.  OFPPR_ADD OFPPR_DELETE OFPPR_MODIFY
flow_removed_mask	2-element array. Bitmasks of following values.  OFPRR_IDLE_TIMEOUT OFPRR_HARD_TIMEOUT OFPRR_DELETE OFPRR_GROUP_DELETE

Example:

```
def send_set_async(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    packet_in_mask = 1 << ofp.OFPR_ACTION | 1 << ofp.OFPR_INVALID_TTL
    port_status_mask = (1 << ofp.OFPPR_ADD
                       | 1 << ofp.OFPPR_DELETE
                       | 1 << ofp.OFPPR_MODIFY)
    flow_removed_mask = (1 << ofp.OFPRR_IDLE_TIMEOUT
                       | 1 << ofp.OFPRR_HARD_TIMEOUT
                       | 1 << ofp.OFPRR_DELETE)
    req = ofp_parser.OFPSetAsync(datapath,
                                 [packet_in_mask, 0],
                                 [port_status_mask, 0],
                                 [flow_removed_mask, 0])

    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPSyncRequest": {
    "flow_removed_mask": [
      15,
      3
    ],
    "packet_in_mask": [
      5,
      1
    ],
    "port_status_mask": [
      7,
      3
    ]
  }
}
```

**class** `os_ken.ofproto.ofproto_v1_3_parser.OFPSyncRequest(datapath)`

Get asynchronous configuration request message

The controller uses this message to query the asynchronous message.

Example:

```
def send_get_async_request(self, datapath):
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPSyncRequest(datapath)
    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPSyncRequest": {}
}
```

**class** `os_ken.ofproto.ofproto_v1_3_parser.OFPSyncReply(datapath, packet_in_mask=None, port_status_mask=None, flow_removed_mask=None)`

Get asynchronous configuration reply message

The switch responds with this message to a get asynchronous configuration request.

Attribute	Description
packet_in_mask	2-element array: element 0, when the controller has a OFPCR_ROLE_EQUAL or OFPCR_ROLE_MASTER role. element 1, OFPCR_ROLE_SLAVE role controller. Bitmasks of following values.  OFPR_NO_MATCH OFPR_ACTION OFPR_INVALID_TTL
port_status_mask	2-element array. Bitmasks of following values.  OFPPR_ADD OFPPR_DELETE OFPPR_MODIFY
flow_removed_mask	2-element array. Bitmasks of following values.  OFPRR_IDLE_TIMEOUT OFPRR_HARD_TIMEOUT OFPRR_DELETE OFPRR_GROUP_DELETE

Example:

```
@set_ev_cls(ofp_event.EventOFPGetAsyncReply, MAIN_DISPATCHER)
def get_async_reply_handler(self, ev):
    msg = ev.msg

    self.logger.debug('OFPGetAsyncReply received: '
                      'packet_in_mask=0x%08x:0x%08x '
                      'port_status_mask=0x%08x:0x%08x '
                      'flow_removed_mask=0x%08x:0x%08x',
                      msg.packet_in_mask[0],
                      msg.packet_in_mask[1],
                      msg.port_status_mask[0],
                      msg.port_status_mask[1],
                      msg.flow_removed_mask[0],
                      msg.flow_removed_mask[1])
```

JSON Example:

```
{
  "OFPGetAsyncReply": {
```

(continues on next page)

(continued from previous page)

```

    "flow_removed_mask": [
        15,
        3
    ],
    "packet_in_mask": [
        5,
        1
    ],
    "port_status_mask": [
        7,
        3
    ]
}
}

```

## Asynchronous Messages

### Packet-In Message

```

class os_ken.ofproto.ofproto_v1_3_parser.OFPPacketIn(datapath, buffer_id=None,
                                                    total_len=None, reason=None,
                                                    table_id=None, cookie=None,
                                                    match=None, data=None)

```

Packet-In message

The switch sends the packet that received to the controller by this message.

Attribute	Description
buffer_id	ID assigned by datapath
total_len	Full length of frame
reason	Reason packet is being sent.  OFPR_NO_MATCH OFPR_ACTION OFPR_INVALID_TTL
table_id	ID of the table that was looked up
cookie	Cookie of the flow entry that was looked up
match	Instance of OFPMatch
data	Ethernet frame

Example:

```

@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def packet_in_handler(self, ev):
    msg = ev.msg
    dp = msg.datapath
    ofp = dp.ofproto

```

(continues on next page)

(continued from previous page)

```

if msg.reason == ofp.OFPR_NO_MATCH:
    reason = 'NO MATCH'
elif msg.reason == ofp.OFPR_ACTION:
    reason = 'ACTION'
elif msg.reason == ofp.OFPR_INVALID_TTL:
    reason = 'INVALID TTL'
else:
    reason = 'unknown'

self.logger.debug('OFPPacketIn received: '
                  'buffer_id=%x total_len=%d reason=%s '
                  'table_id=%d cookie=%d match=%s data=%s',
                  msg.buffer_id, msg.total_len, reason,
                  msg.table_id, msg.cookie, msg.match,
                  utils.hex_array(msg.data))

```

JSON Example:

```

{
  "OFPPacketIn": {
    "buffer_id": 2,
    "cookie": 2836868884868096,
    "data": "/////////8gukffjqCAYAAQgABgQAAfILpH346goAAAEAAAAAAAAAKAAAD",
    "match": {
      "OFPMatch": {
        "length": 80,
        "oxm_fields": [
          {
            "OXMTlv": {
              "field": "in_port",
              "mask": null,
              "value": 6
            }
          },
          {
            "OXMTlv": {
              "field": "eth_type",
              "mask": null,
              "value": 2054
            }
          },
          {
            "OXMTlv": {
              "field": "eth_dst",
              "mask": null,
              "value": "ff:ff:ff:ff:ff:ff"
            }
          }
        ]
      }
    }
  }
}

```

(continues on next page)

(continued from previous page)

```
        "OXMTlv": {
            "field": "eth_src",
            "mask": null,
            "value": "f2:0b:a4:7d:f8:ea"
        }
    },
    {
        "OXMTlv": {
            "field": "arp_op",
            "mask": null,
            "value": 1
        }
    },
    {
        "OXMTlv": {
            "field": "arp_spa",
            "mask": null,
            "value": "10.0.0.1"
        }
    },
    {
        "OXMTlv": {
            "field": "arp_tpa",
            "mask": null,
            "value": "10.0.0.3"
        }
    },
    {
        "OXMTlv": {
            "field": "arp_sha",
            "mask": null,
            "value": "f2:0b:a4:7d:f8:ea"
        }
    },
    {
        "OXMTlv": {
            "field": "arp_tha",
            "mask": null,
            "value": "00:00:00:00:00:00"
        }
    }
],
    "type": 1
}
},
"reason": 1,
"table_id": 1,
"total_len": 42
}
}
```



## Flow Removed Message

```
class os_ken.ofproto.ofproto_v1_3_parser.OFPFlowRemoved(datapath, cookie=None,
                                                         priority=None,
                                                         reason=None,
                                                         table_id=None,
                                                         duration_sec=None,
                                                         duration_nsec=None,
                                                         idle_timeout=None,
                                                         hard_timeout=None,
                                                         packet_count=None,
                                                         byte_count=None,
                                                         match=None)
```

Flow removed message

When flow entries time out or are deleted, the switch notifies controller with this message.

Attribute	Description
cookie	Opaque controller-issued identifier
priority	Priority level of flow entry
reason	One of the following values.  OFPRR_IDLE_TIMEOUT OFPRR_HARD_TIMEOUT OFPRR_DELETE OFPRR_GROUP_DELETE
table_id	ID of the table
duration_sec	Time flow was alive in seconds
duration_nsec	Time flow was alive in nanoseconds beyond duration_sec
idle_timeout	Idle timeout from original flow mod
hard_timeout	Hard timeout from original flow mod
packet_count	Number of packets that was associated with the flow
byte_count	Number of bytes that was associated with the flow
match	Instance of OFPMatch

Example:

```
@set_ev_cls(ofp_event.EventOFPFlowRemoved, MAIN_DISPATCHER)
def flow_removed_handler(self, ev):
    msg = ev.msg
    dp = msg.datapath
    ofp = dp.ofproto

    if msg.reason == ofp.OFPRR_IDLE_TIMEOUT:
        reason = 'IDLE TIMEOUT'
    elif msg.reason == ofp.OFPRR_HARD_TIMEOUT:
```

(continues on next page)

(continued from previous page)

```

        reason = 'HARD TIMEOUT'
    elif msg.reason == ofp.OFPRR_DELETE:
        reason = 'DELETE'
    elif msg.reason == ofp.OFPRR_GROUP_DELETE:
        reason = 'GROUP DELETE'
    else:
        reason = 'unknown'

    self.logger.debug('OFPPFlowRemoved received: '
                     'cookie=%d priority=%d reason=%s table_id=%d '
                     'duration_sec=%d duration_nsec=%d '
                     'idle_timeout=%d hard_timeout=%d '
                     'packet_count=%d byte_count=%d match.fields=%s',
                     msg.cookie, msg.priority, reason, msg.table_id,
                     msg.duration_sec, msg.duration_nsec,
                     msg.idle_timeout, msg.hard_timeout,
                     msg.packet_count, msg.byte_count, msg.match)

```

JSON Example:

```

{
  "OFPPFlowRemoved": {
    "byte_count": 86,
    "cookie": 0,
    "duration_nsec": 48825000,
    "duration_sec": 3,
    "hard_timeout": 0,
    "idle_timeout": 3,
    "match": {
      "OFPMatch": {
        "length": 14,
        "oxm_fields": [
          {
            "OXMTlv": {
              "field": "eth_dst",
              "mask": null,
              "value": "f2:0b:a4:7d:f8:ea"
            }
          }
        ],
        "type": 1
      }
    },
    "packet_count": 1,
    "priority": 65535,
    "reason": 0,
    "table_id": 0
  }
}

```

## Port Status Message

```
class os_ken.ofproto.ofproto_v1_3_parser.OFPPortStatus(datapath, reason=None,
                                                       desc=None)
```

Port status message

The switch notifies controller of change of ports.

Attribute	Description
reason	One of the following values.  OFPPR_ADD OFPPR_DELETE OFPPR_MODIFY
desc	instance of OFPPort

Example:

```
@set_ev_cls(ofp_event.EventOFPPortStatus, MAIN_DISPATCHER)
def port_status_handler(self, ev):
    msg = ev.msg
    dp = msg.datapath
    ofp = dp.ofproto

    if msg.reason == ofp.OFPPR_ADD:
        reason = 'ADD'
    elif msg.reason == ofp.OFPPR_DELETE:
        reason = 'DELETE'
    elif msg.reason == ofp.OFPPR_MODIFY:
        reason = 'MODIFY'
    else:
        reason = 'unknown'

    self.logger.debug('OFPPortStatus received: reason=%s desc=%s',
                      reason, msg.desc)
```

JSON Example:

```
{
  "OFPPortStatus": {
    "desc": {
      "OFPPort": {
        "advertised": 10240,
        "config": 0,
        "curr": 10248,
        "curr_speed": 5000,
        "hw_addr": "f2:0b:a4:d0:3f:70",
        "max_speed": 5000,
        "name": "\u79c1\u306e\u30dd\u30fc\u30c8",
```

(continues on next page)

(continued from previous page)

```

        "peer": 10248,
        "port_no": 7,
        "state": 4,
        "supported": 10248
    },
    "reason": 0
}

```

## Error Message

```

class os_ken.ofproto.ofproto_v1_3_parser.OFPErrormsg(datapath, type_=None,
                                                    code=None, data=None,
                                                    **kwargs)

```

Error message

The switch notifies controller of problems by this message.

Attribute	Description
type	High level type of error
code	Details depending on the type
data	Variable length data depending on the type and code

type attribute corresponds to type\_ parameter of \_\_init\_\_.

Types and codes are defined in os\_ken.ofproto.ofproto.

Type	Code
OFPET_HELLO_FAILED	OFPHFC_*
OFPET_BAD_REQUEST	OFPBRC_*
OFPET_BAD_ACTION	OFPBAC_*
OFPET_BAD_INSTRUCTION	OFPBIC_*
OFPET_BAD_MATCH	OFPBMC_*
OFPET_FLOW_MOD_FAILED	OFPFMFC_*
OFPET_GROUP_MOD_FAILED	OFPGMFC_*
OFPET_PORT_MOD_FAILED	OFPPMFC_*
OFPET_TABLE_MOD_FAILED	OFPTMFC_*
OFPET_QUEUE_OP_FAILED	OFPQOFC_*
OFPET_SWITCH_CONFIG_FAILED	OFPSCFC_*
OFPET_ROLE_REQUEST_FAILED	OFPRRFC_*
OFPET_METER_MOD_FAILED	OFPMMFC_*
OFPET_TABLE_FEATURES_FAILED	OFPTFFC_*
OFPET_EXPERIMENTER	N/A

If type == OFPET\_EXPERIMENTER, this message has also the following attributes.

Attribute	Description
exp_type	Experimenter defined type
experimenter	Experimenter ID

Example:

```
@set_ev_cls(ofp_event.EventOFPErrormsg,
            [HANDSHAKE_DISPATCHER, CONFIG_DISPATCHER, MAIN_DISPATCHER])
def error_msg_handler(self, ev):
    msg = ev.msg

    self.logger.debug('OFPErrormsg received: type=0x%02x code=0x%02x '
                      'message=%s',
                      msg.type, msg.code, utils.hex_array(msg.data))
```

JSON Example:

```
{
  "OFPErrormsg": {
    "code": 11,
    "data": "ZnVnYWZ1Z2E=",
    "type": 2
  }
}
```

## Symmetric Messages

### Hello

**class** os\_ken.ofproto.ofproto\_v1\_3\_parser.OFPHello(*datapath*, *elements=None*)

Hello message

When connection is started, the hello message is exchanged between a switch and a controller.

This message is handled by the OSKen framework, so the OSKen application do not need to process this typically.

Attribute	Description
elements	list of OFPHelloElemVersionBitmap instance

JSON Example:

```
{
  "OFPHello": {
    "elements": [
      {
        "OFPHelloElemVersionBitmap": {
          "length": 8,
          "type": 1,

```

(continues on next page)

(continued from previous page)

```

        "versions": [
            1,
            2,
            3,
            9,
            10,
            30
        ]
    }
}
}
}

```

```

class os_ken.ofproto.ofproto_v1_3_parser.OFPHelloElemVersionBitmap(versions,
                                                                    type_=None,
                                                                    length=None)

```

Version bitmap Hello Element

Attribute	Description
versions	list of versions of OpenFlow protocol a device supports

## Echo Request

```

class os_ken.ofproto.ofproto_v1_3_parser.OFPEchoRequest(datapath, data=None)

```

Echo request message

This message is handled by the OSKen framework, so the OSKen application do not need to process this typically.

Attribute	Description
data	An arbitrary length data

Example:

```

def send_echo_request(self, datapath, data):
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPEchoRequest(datapath, data)
    datapath.send_msg(req)

@set_ev_cls(ofp_event.EventOFPEchoRequest,
            [HANDSHAKE_DISPATCHER, CONFIG_DISPATCHER, MAIN_DISPATCHER])
def echo_request_handler(self, ev):
    self.logger.debug('OFPEchoRequest received: data=%s',
                      utils.hex_array(ev.msg.data))

```

JSON Example:

```
{
  "OFPEchoRequest": {
    "data": "aG9nZQ=="
  }
}
```

## Echo Reply

**class** os\_ken.ofproto.ofproto\_v1\_3\_parser.OFPEchoReply(datapath, data=None)

Echo reply message

This message is handled by the OSKen framework, so the OSKen application do not need to process this typically.

Attribute	Description
data	An arbitrary length data

Example:

```
def send_echo_reply(self, datapath, data):
    ofp_parser = datapath.ofproto_parser

    reply = ofp_parser.OFPEchoReply(datapath, data)
    datapath.send_msg(reply)

    @set_ev_cls(ofp_event.EventOFPEchoReply,
                [HANDSHAKE_DISPATCHER, CONFIG_DISPATCHER, MAIN_DISPATCHER])
    def echo_reply_handler(self, ev):
        self.logger.debug('OFPEchoReply received: data=%s',
                          utils.hex_array(ev.msg.data))
```

JSON Example:

```
{
  "OFPEchoReply": {
    "data": "aG9nZQ=="
  }
}
```

## Experimenter

**class** os\_ken.ofproto.ofproto\_v1\_3\_parser.OFPEExperimenter(datapath,  
*experimenter=None,*  
*exp\_type=None,*  
*data=None*)

Experimenter extension message

Attribute	Description
experimenter	Experimenter ID
exp_type	Experimenter defined
data	Experimenter defined arbitrary additional data

JSON Example:

```
{
  "OFPExperimenter": {
    "data": "bmF6bw==",
    "exp_type": 123456789,
    "experimenter": 98765432
  }
}
```

## Port Structures

**class** `os_ken.ofproto.ofproto_v1_3_parser.OFPPort`(*port\_no*, *hw\_addr*, *name*, *config*, *state*, *curr*, *advertised*, *supported*, *peer*, *curr\_speed*, *max\_speed*)

Description of a port

Attribute	Description
<code>port_no</code>	Port number and it uniquely identifies a port within a switch.
<code>hw_addr</code>	MAC address for the port.
<code>name</code>	Null-terminated string containing a human-readable name for the interface.
<code>config</code>	Bitmap of port configuration flags.  OFPPC_PORT_DOWN OFPPC_NO_RECV OFPPC_NO_FWD OFPPC_NO_PACKET_IN
<code>state</code>	Bitmap of port state flags.  OFPPS_LINK_DOWN OFPPS_BLOCKED OFPPS_LIVE
<code>curr</code>	Current features.
<code>advertised</code>	Features being advertised by the port.
<code>supported</code>	Features supported by the port.
<code>peer</code>	Features advertised by peer.
<code>curr_speed</code>	Current port bitrate in kbps.
<code>max_speed</code>	Max port bitrate in kbps.



## Flow Match Structure

```
class os_ken.ofproto.ofproto_v1_3_parser.OFPMatch(type_=None, length=None,  
                                                _ordered_fields=None, **kwargs)
```

Flow Match Structure

This class is implementation of the flow match structure having compose/query API. There are new API and old API for compatibility. the old API is supposed to be removed later.

You can define the flow match by the keyword arguments. The following arguments are available.

Argument	Value	Description
in_port	Integer 32bit	Switch input port
in_phy_port	Integer 32bit	Switch physical input port
metadata	Integer 64bit	Metadata passed between tables
eth_dst	MAC address	Ethernet destination address
eth_src	MAC address	Ethernet source address
eth_type	Integer 16bit	Ethernet frame type
vlan_vid	Integer 16bit	VLAN id
vlan_pcp	Integer 8bit	VLAN priority
ip_dscp	Integer 8bit	IP DSCP (6 bits in ToS field)
ip_ecn	Integer 8bit	IP ECN (2 bits in ToS field)
ip_proto	Integer 8bit	IP protocol
ipv4_src	IPv4 address	IPv4 source address
ipv4_dst	IPv4 address	IPv4 destination address
tcp_src	Integer 16bit	TCP source port
tcp_dst	Integer 16bit	TCP destination port
udp_src	Integer 16bit	UDP source port
udp_dst	Integer 16bit	UDP destination port
sctp_src	Integer 16bit	SCTP source port
sctp_dst	Integer 16bit	SCTP destination port
icmpv4_type	Integer 8bit	ICMP type
icmpv4_code	Integer 8bit	ICMP code
arp_op	Integer 16bit	ARP opcode
arp_spa	IPv4 address	ARP source IPv4 address
arp_tpa	IPv4 address	ARP target IPv4 address
arp_sha	MAC address	ARP source hardware address
arp_tha	MAC address	ARP target hardware address
ipv6_src	IPv6 address	IPv6 source address
ipv6_dst	IPv6 address	IPv6 destination address
ipv6_flabel	Integer 32bit	IPv6 Flow Label
icmpv6_type	Integer 8bit	ICMPv6 type
icmpv6_code	Integer 8bit	ICMPv6 code
ipv6_nd_target	IPv6 address	Target address for ND
ipv6_nd_sll	MAC address	Source link-layer for ND
ipv6_nd_tll	MAC address	Target link-layer for ND
mpls_label	Integer 32bit	MPLS label
mpls_tc	Integer 8bit	MPLS TC
mpls_bos	Integer 8bit	MPLS BoS bit
pbb_isid	Integer 24bit	PBB I-SID

continues on next page

Table 2 – continued from previous page

Argument	Value	Description
tunnel_id	Integer 64bit	Logical Port Metadata
ipv6_exthdr	Integer 16bit	IPv6 Extension Header pseudo-field
pbb_uca	Integer 8bit	PBB UCA header field (EXT-256 Old version of ONF Extension)
tcp_flags	Integer 16bit	TCP flags (EXT-109 ONF Extension)
actset_output	Integer 32bit	Output port from action set metadata (EXT-233 ONF Extension)

Example:

```
>>> # compose
>>> match = parser.OFPMatch(
...     in_port=1,
...     eth_type=0x86dd,
...     ipv6_src=('2001:db8:bd05:1d2:288a:1fc0:1:10ee',
...               'ffff:ffff:ffff:ffff:'),
...     ipv6_dst='2001:db8:bd05:1d2:288a:1fc0:1:10ee')
>>> # query
>>> if 'ipv6_src' in match:
...     print match['ipv6_src']
...
('2001:db8:bd05:1d2:288a:1fc0:1:10ee', 'ffff:ffff:ffff:ffff:')
```

#### Note

For the list of the supported Nicira experimenter matches, please refer to *os\_ken.ofproto.nx\_match*.

#### Note

For VLAN id match field, special values are defined in OpenFlow Spec.

##### 1) Packets with and without a VLAN tag

- Example:

```
match = parser.OFPMatch()
```

- Packet Matching

non-VLAN-tagged	MATCH
VLAN-tagged(vlan_id=3)	MATCH
VLAN-tagged(vlan_id=5)	MATCH

##### 2) Only packets without a VLAN tag

- Example:

```
match = parser.OFPMatch(vlan_vid=0x0000)
```

- Packet Matching

non-VLAN-tagged	MATCH
VLAN-tagged(vlan_id=3)	x
VLAN-tagged(vlan_id=5)	x

3) Only packets with a VLAN tag regardless of its value

- Example:

```
match = parser.OFPMatch(vlan_vid=(0x1000, 0x1000))
```

- Packet Matching

non-VLAN-tagged	x
VLAN-tagged(vlan_id=3)	MATCH
VLAN-tagged(vlan_id=5)	MATCH

4) Only packets with VLAN tag and VID equal

- Example:

```
match = parser.OFPMatch(vlan_vid=(0x1000 | 3))
```

- Packet Matching

non-VLAN-tagged	x
VLAN-tagged(vlan_id=3)	MATCH
VLAN-tagged(vlan_id=5)	x

## Flow Instruction Structures

```
class os_ken.ofproto.ofproto_v1_3_parser.OFPInstructionGotoTable(table_id,
                                                                type_=None,
                                                                len_=None)
```

Goto table instruction

This instruction indicates the next table in the processing pipeline.

Attribute	Description
table_id	Next table

```
class os_ken.ofproto.ofproto_v1_3_parser.OFPInstructionWriteMetadata(metadata,
                                                                    meta-
                                                                    data_mask,
                                                                    type_=None,
                                                                    len_=None)
```

Write metadata instruction

This instruction writes the masked metadata value into the metadata field.

Attribute	Description
metadata	Metadata value to write
metadata_mask	Metadata write bitmask

```
class os_ken.ofproto.ofproto_v1_3_parser.OFPInstructionActions(type_,
                                                                actions=None,
                                                                len_=None)
```

Actions instruction

This instruction writes/applies/clears the actions.

Attribute	Description
type	One of following values.  OFPIT_WRITE_ACTIONS OFPIT_APPLY_ACTIONS OFPIT_CLEAR_ACTIONS
actions	list of OpenFlow action class

type attribute corresponds to type\_ parameter of \_\_init\_\_.

```
class os_ken.ofproto.ofproto_v1_3_parser.OFPInstructionMeter(meter_id=1,
                                                                type_=None,
                                                                len_=None)
```

Meter instruction

This instruction applies the meter.

Attribute	Description
meter_id	Meter instance

## Action Structures

```
class os_ken.ofproto.ofproto_v1_3_parser.OFPActionOutput(port, max_len=65509,
                                                         type_=None, len_=None)
```

Output action

This action indicates output a packet to the switch port.

Attribute	Description
port	Output port
max_len	Max length to send to controller

```
class os_ken.ofproto.ofproto_v1_3_parser.OFPActionGroup(group_id=0, type_=None,
                                                         len_=None)
```

### Group action

This action indicates the group used to process the packet.

Attribute	Description
group_id	Group identifier

```
class os_ken.ofproto.ofproto_v1_3_parser.OFPActionSetQueue(queue_id, type_=None,  
                                                         len_=None)
```

### Set queue action

This action sets the queue id that will be used to map a flow to an already-configured queue on a port.

Attribute	Description
queue_id	Queue ID for the packets

```
class os_ken.ofproto.ofproto_v1_3_parser.OFPActionSetMplsTtl(mpls_ttl, type_=None,  
                                                            len_=None)
```

### Set MPLS TTL action

This action sets the MPLS TTL.

Attribute	Description
mpls_ttl	MPLS TTL

```
class os_ken.ofproto.ofproto_v1_3_parser.OFPActionDecMplsTtl(type_=None,  
                                                             len_=None)
```

### Decrement MPLS TTL action

This action decrements the MPLS TTL.

```
class os_ken.ofproto.ofproto_v1_3_parser.OFPActionSetNwTtl(nw_ttl, type_=None,  
                                                            len_=None)
```

### Set IP TTL action

This action sets the IP TTL.

Attribute	Description
nw_ttl	IP TTL

```
class os_ken.ofproto.ofproto_v1_3_parser.OFPActionDecNwTtl(type_=None,  
                                                            len_=None)
```

### Decrement IP TTL action

This action decrements the IP TTL.

```
class os_ken.ofproto.ofproto_v1_3_parser.OFPActionCopyTtlOut(type_=None,  
                                                         len_=None)
```

Copy TTL Out action

This action copies the TTL from the next-to-outermost header with TTL to the outermost header with TTL.

```
class os_ken.ofproto.ofproto_v1_3_parser.OFPActionCopyTtlIn(type_=None,  
                                                            len_=None)
```

Copy TTL In action

This action copies the TTL from the outermost header with TTL to the next-to-outermost header with TTL.

```
class os_ken.ofproto.ofproto_v1_3_parser.OFPActionPushVlan(ethertype=33024,  
                                                            type_=None,  
                                                            len_=None)
```

Push VLAN action

This action pushes a new VLAN tag to the packet.

Attribute	Description
ethertype	Ether type. The default is 802.1Q. (0x8100)

```
class os_ken.ofproto.ofproto_v1_3_parser.OFPActionPushMpls(ethertype=34887,  
                                                            type_=None,  
                                                            len_=None)
```

Push MPLS action

This action pushes a new MPLS header to the packet.

Attribute	Description
ethertype	Ether type

```
class os_ken.ofproto.ofproto_v1_3_parser.OFPActionPopVlan(type_=None, len_=None)
```

Pop VLAN action

This action pops the outermost VLAN tag from the packet.

```
class os_ken.ofproto.ofproto_v1_3_parser.OFPActionPopMpls(ethertype=2048,  
                                                           type_=None, len_=None)
```

Pop MPLS action

This action pops the MPLS header from the packet.

```
class os_ken.ofproto.ofproto_v1_3_parser.OFPActionSetField(field=None, **kwargs)
```

Set field action

This action modifies a header field in the packet.

The set of keywords available for this is same as OFPMatch.

Example:

```
set_field = OFPActionSetField(eth_src="00:00:00:00:00:00")
```

**class** `os_ken.ofproto.ofproto_v1_3_parser.OFPActionExperimenter(experimenter)`

Experimenter action

This action is an extensible action for the experimenter.

Attribute	Description
experimenter	Experimenter ID

#### Note

For the list of the supported Nicira experimenter actions, please refer to *os\_ken.ofproto.nx\_actions*.

## OpenFlow v1.4 Messages and Structures

### Controller-to-Switch Messages

#### Handshake

**class** `os_ken.ofproto.ofproto_v1_4_parser.OFPFeaturesRequest(datapath)`

Features request message

The controller sends a feature request to the switch upon session establishment.

This message is handled by the OSKen framework, so the OSKen application do not need to process this typically.

Example:

```
def send_features_request(self, datapath):
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPFeaturesRequest(datapath)
    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPFeaturesRequest": {}
}
```

**class** `os_ken.ofproto.ofproto_v1_4_parser.OFPSwitchFeatures(datapath, datapath_id=None, n_buffers=None, n_tables=None, auxiliary_id=None, capabilities=None)`

Features reply message

The switch responds with a features reply message to a features request.

This message is handled by the OSKen framework, so the OSKen application do not need to process this typically.

Example:

```
@set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
def switch_features_handler(self, ev):
    msg = ev.msg

    self.logger.debug('OFPSwitchFeatures received: '
                      'datapath_id=0x%016x n_buffers=%d '
                      'n_tables=%d auxiliary_id=%d '
                      'capabilities=0x%08x',
                      msg.datapath_id, msg.n_buffers, msg.n_tables,
                      msg.auxiliary_id, msg.capabilities)
```

JSON Example:

```
{
  "OFPSwitchFeatures": {
    "auxiliary_id": 99,
    "capabilities": 79,
    "datapath_id": 9210263729383,
    "n_buffers": 0,
    "n_tables": 255
  }
}
```

## Switch Configuration

```
class os_ken.ofproto.ofproto_v1_4_parser.OFPSetConfig(datapath, flags=0,
                                                       miss_send_len=0)
```

Set config request message

The controller sends a set config request message to set configuraion parameters.

Attribute	Description
flags	Bitmap of the following flags.  OFPC_FRAG_NORMAL OFPC_FRAG_DROP OFPC_FRAG_REASM
miss_send_len	Max bytes of new flow that datapath should send to the controller

Example:



```
def send_set_config(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPSetConfig(datapath, ofp.OFPC_FRAG_NORMAL, 256)
    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPSetConfig": {
    "flags": 0,
    "miss_send_len": 128
  }
}
```

**class** `os_ken.ofproto.ofproto_v1_4_parser.OFPGetConfigRequest(datapath)`

Get config request message

The controller sends a get config request to query configuration parameters in the switch.

Example:

```
def send_get_config_request(self, datapath):
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPGetConfigRequest(datapath)
    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPGetConfigRequest": {}
}
```

**class** `os_ken.ofproto.ofproto_v1_4_parser.OFPGetConfigReply(datapath, flags=None, miss_send_len=None)`

Get config reply message

The switch responds to a configuration request with a get config reply message.

Attribute	Description
flags	Bitmap of the following flags.  OFPC_FRAG_NORMAL OFPC_FRAG_DROP OFPC_FRAG_REASM
miss_send_len	Max bytes of new flow that datapath should send to the controller

Example:

```
@set_ev_cls(ofp_event.EventOFPCGetConfigReply, MAIN_DISPATCHER)
def get_config_reply_handler(self, ev):
    msg = ev.msg
    dp = msg.datapath
    ofp = dp.ofproto
    flags = []

    if msg.flags & ofp.OFPC_FRAG_NORMAL:
        flags.append('NORMAL')
    if msg.flags & ofp.OFPC_FRAG_DROP:
        flags.append('DROP')
    if msg.flags & ofp.OFPC_FRAG_REASM:
        flags.append('REASM')
    self.logger.debug('OFPCGetConfigReply received: '
                      'flags=%s miss_send_len=%d',
                      ','.join(flags), msg.miss_send_len)
```

JSON Example:

```
{
  "OFPCGetConfigReply": {
    "flags": 0,
    "miss_send_len": 128
  }
}
```

## Modify State Messages

```
class os_ken.ofproto.ofproto_v1_4_parser.OFPTTableMod(datapath, table_id, config,
                                                       properties)
```

Flow table configuration message

The controller sends this message to configure table state.

Attribute	Description
table_id	ID of the table (OFPTT_ALL indicates all tables)
config	Bitmap of configuration flags.  OFPTC_EVICTION OFPTC_VACANCY_EVENTS
properties	List of OFPTTableModProp subclass instance

Example:

```
def send_table_mod(self, datapath):
    ofp = datapath.ofproto
```

(continues on next page)

(continued from previous page)

```

ofp_parser = datapath.ofproto_parser

req = ofp_parser.OFPTTableMod(datapath, 1, 3)
flags = ofp.OFPTC_VACANCY_EVENTS
properties = [ofp_parser.OFPTTableModPropEviction(flags)]
req = ofp_parser.OFPTTableMod(datapath, 1, 3, properties)
datapath.send_msg(req)

```

**JSON Example:**

```

{
  "OFPTTableMod": {
    "config": 0,
    "properties": [
      {
        "OFPTTableModPropEviction": {
          "flags": 0,
          "length": 8,
          "type": 2
        }
      },
      {
        "OFPTTableModPropVacancy": {
          "length": 8,
          "type": 3,
          "vacancy": 0,
          "vacancy_down": 0,
          "vacancy_up": 0
        }
      },
      {
        "OFPTTableModPropExperimenter": {
          "data": [],
          "exp_type": 0,
          "experimenter": 101,
          "length": 12,
          "type": 65535
        }
      },
      {
        "OFPTTableModPropExperimenter": {
          "data": [
            1
          ],
          "exp_type": 1,
          "experimenter": 101,
          "length": 16,
          "type": 65535
        }
      }
    ]
  }
}

```

(continues on next page)

(continued from previous page)

```

    {
      "OFPTTableModPropExperimenter": {
        "data": [
          1,
          2
        ],
        "exp_type": 2,
        "experimenter": 101,
        "length": 20,
        "type": 65535
      }
    }
  ],
  "table_id": 255
}

```

```

class os_ken.ofproto.ofproto_v1_4_parser.OFPFlowMod(datapath, cookie=0,
                                                    cookie_mask=0, table_id=0,
                                                    command=0, idle_timeout=0,
                                                    hard_timeout=0, priority=32768,
                                                    buffer_id=4294967295,
                                                    out_port=0, out_group=0,
                                                    flags=0, importance=0,
                                                    match=None, instructions=None)

```

Modify Flow entry message

The controller sends this message to modify the flow table.

Attribute	Description
cookie	Opaque controller-issued identifier
cookie_mask	Mask used to restrict the cookie bits that must match when the command is <code>OPFFC_MODIFY*</code> or <code>OPFFC_DELETE*</code>
table_id	ID of the table to put the flow in
command	One of the following values.  <code>OPFFC_ADD</code> <code>OPFFC_MODIFY</code> <code>OPFFC_MODIFY_STRICT</code> <code>OPFFC_DELETE</code> <code>OPFFC_DELETE_STRICT</code>
idle_timeout	Idle time before discarding (seconds)
hard_timeout	Max time before discarding (seconds)
priority	Priority level of flow entry
buffer_id	Buffered packet to apply to (or <code>OPF_NO_BUFFER</code> )
out_port	For <code>OPFFC_DELETE*</code> commands, require matching entries to include this as an output port
out_group	For <code>OPFFC_DELETE*</code> commands, require matching entries to include this as an output group
flags	Bitmap of the following flags.  <code>OPFFF_SEND_FLOW_REM</code> <code>OPFFF_CHECK_OVERLAP</code> <code>OPFFF_RESET_COUNTS</code> <code>OPFFF_NO_PKT_COUNTS</code> <code>OPFFF_NO_BYT_COUNTS</code>
importance	Eviction precedence
match	Instance of <code>OFPMatch</code>
instructions	list of <code>OFPIInstruction*</code> instance

Example:

```
def send_flow_mod(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    cookie = cookie_mask = 0
    table_id = 0
    idle_timeout = hard_timeout = 0
    priority = 32768
    buffer_id = ofp.OPF_NO_BUFFER
```

(continues on next page)

(continued from previous page)

```

importance = 0
match = ofp_parser.OFPMatch(in_port=1, eth_dst='ff:ff:ff:ff:ff:ff')
actions = [ofp_parser.OFPActionOutput(ofp.OFPP_NORMAL, 0)]
inst = [ofp_parser.OFPInstructionActions(ofp.OFPIT_APPLY_ACTIONS,
                                         actions)]

req = ofp_parser.OFPFlowMod(datapath, cookie, cookie_mask,
                             table_id, ofp.OFPFC_ADD,
                             idle_timeout, hard_timeout,
                             priority, buffer_id,
                             ofp.OFPP_ANY, ofp.OFPG_ANY,
                             ofp.OFPFF_SEND_FLOW_REM,
                             importance,
                             match, inst)

datapath.send_msg(req)

```

JSON Example:

```

{
  "OFPFlowMod": {
    "buffer_id": 65535,
    "command": 0,
    "cookie": 0,
    "cookie_mask": 0,
    "flags": 0,
    "hard_timeout": 0,
    "idle_timeout": 0,
    "importance": 0,
    "instructions": [
      {
        "OFPInstructionActions": {
          "actions": [
            {
              "OFPActionSetField": {
                "field": {
                  "OXMTlv": {
                    "field": "vlan_vid",
                    "mask": null,
                    "value": 258
                  }
                }
              },
              "len": 16,
              "type": 25
            },
            {
              "OFPActionCopyTtlOut": {
                "len": 8,
                "type": 11
              }
            }
          ]
        }
      }
    ]
  }
}

```

(continues on next page)

(continued from previous page)

```
{
  "OFPAActionCopyTtlIn": {
    "len": 8,
    "type": 12
  }
},
{
  "OFPAActionCopyTtlIn": {
    "len": 8,
    "type": 12
  }
},
{
  "OFPAActionPopPbb": {
    "len": 8,
    "type": 27
  }
},
{
  "OFPAActionPushPbb": {
    "ethertype": 4660,
    "len": 8,
    "type": 26
  }
},
{
  "OFPAActionPopMpls": {
    "ethertype": 39030,
    "len": 8,
    "type": 20
  }
},
{
  "OFPAActionPushMpls": {
    "ethertype": 34887,
    "len": 8,
    "type": 19
  }
},
{
  "OFPAActionPopVlan": {
    "len": 8,
    "type": 18
  }
},
{
  "OFPAActionPushVlan": {
    "ethertype": 33024,
    "len": 8,
```

(continues on next page)

(continued from previous page)

```
        "type": 17
    }
},
{
    "OFPActionDecMplsTtl": {
        "len": 8,
        "type": 16
    }
},
{
    "OFPActionSetMplsTtl": {
        "len": 8,
        "mpls_ttl": 10,
        "type": 15
    }
},
{
    "OFPActionDecNwTtl": {
        "len": 8,
        "type": 24
    }
},
{
    "OFPActionSetNwTtl": {
        "len": 8,
        "nw_ttl": 10,
        "type": 23
    }
},
{
    "OFPActionExperimenterUnknown": {
        "data": "AAECAwQFBgc=",
        "experimenter": 101,
        "len": 16,
        "type": 65535
    }
},
{
    "OFPActionSetQueue": {
        "len": 8,
        "queue_id": 3,
        "type": 21
    }
},
{
    "OFPActionGroup": {
        "group_id": 99,
        "len": 8,
        "type": 22
    }
}
```

(continues on next page)



(continued from previous page)

```
    }
  },
  {
    "OFPActionOutput": {
      "len": 16,
      "max_len": 65535,
      "port": 6,
      "type": 0
    }
  }
],
"len": 176,
"type": 3
},
{
  "OFPIInstructionActions": {
    "actions": [
      {
        "OFPActionSetField": {
          "field": {
            "OXMTlv": {
              "field": "eth_src",
              "mask": null,
              "value": "01:02:03:04:05:06"
            }
          }
        },
        "len": 16,
        "type": 25
      }
    ],
    "OFPActionSetField": {
      "field": {
        "OXMTlv": {
          "field": "pbb_uca",
          "mask": null,
          "value": 1
        }
      },
      "len": 16,
      "type": 25
    }
  },
  "len": 40,
  "type": 4
}
```

(continues on next page)

(continued from previous page)

```

    ],
    "match": {
      "OFPMatch": {
        "length": 14,
        "oxm_fields": [
          {
            "OXMTlv": {
              "field": "eth_dst",
              "mask": null,
              "value": "f2:0b:a4:7d:f8:ea"
            }
          }
        ]
      },
      "type": 1
    }
  },
  "out_group": 4294967295,
  "out_port": 4294967295,
  "priority": 123,
  "table_id": 1
}
}

```

```

{
  "OFPPFlowMod": {
    "buffer_id": 65535,
    "command": 0,
    "cookie": 0,
    "cookie_mask": 0,
    "flags": 0,
    "hard_timeout": 0,
    "idle_timeout": 0,
    "importance": 0,
    "instructions": [
      {
        "OFPIInstructionGotoTable": {
          "len": 8,
          "table_id": 1,
          "type": 1
        }
      }
    ]
  },
  "match": {
    "OFPMatch": {
      "length": 22,
      "oxm_fields": [
        {
          "OXMTlv": {
            "field": "in_port",

```

(continues on next page)

(continued from previous page)

```

        "mask": null,
        "value": 6
    }
},
{
    "OXMTlv": {
        "field": "eth_src",
        "mask": null,
        "value": "f2:0b:a4:7d:f8:ea"
    }
}
],
"type": 1
}
},
"out_group": 4294967295,
"out_port": 4294967295,
"priority": 123,
"table_id": 0
}
}
}

```

```

{
    "OFPFLOWMod": {
        "buffer_id": 65535,
        "command": 0,
        "cookie": 0,
        "cookie_mask": 0,
        "flags": 0,
        "hard_timeout": 0,
        "idle_timeout": 0,
        "importance": 0,
        "instructions": [
            {
                "OFPIInstructionMeter": {
                    "len": 8,
                    "meter_id": 1,
                    "type": 6
                }
            },
            {
                "OFPIInstructionActions": {
                    "actions": [
                        {
                            "OFPAActionOutput": {
                                "len": 16,
                                "max_len": 65535,
                                "port": 6,
                                "type": 0
                            }
                        }
                    ]
                }
            }
        ]
    }
}

```

(continues on next page)

(continued from previous page)

```

        }
    },
    ],
    "len": 24,
    "type": 3
}
},
],
"match": {
  "OFPMatch": {
    "length": 14,
    "oxm_fields": [
      {
        "OXMTlv": {
          "field": "eth_dst",
          "mask": null,
          "value": "f2:0b:a4:7d:f8:ea"
        }
      }
    ],
    "type": 1
  }
},
"out_group": 4294967295,
"out_port": 4294967295,
"priority": 123,
"table_id": 1
}
}
}

```

```

{
  "OFPFLOWMod": {
    "buffer_id": 65535,
    "command": 0,
    "cookie": 0,
    "cookie_mask": 0,
    "flags": 0,
    "hard_timeout": 0,
    "idle_timeout": 0,
    "importance": 0,
    "instructions": [],
    "match": {
      "OFPMatch": {
        "length": 329,
        "oxm_fields": [
          {
            "OXMTlv": {
              "field": "in_port",
              "mask": null,

```

(continues on next page)

(continued from previous page)

```
        "value": 84281096
    },
    {
        "OXMTlv": {
            "field": "in_phy_port",
            "mask": null,
            "value": 16909060
        }
    },
    {
        "OXMTlv": {
            "field": "metadata",
            "mask": null,
            "value": 283686952306183
        }
    },
    {
        "OXMTlv": {
            "field": "eth_type",
            "mask": null,
            "value": 2054
        }
    },
    {
        "OXMTlv": {
            "field": "eth_dst",
            "mask": null,
            "value": "ff:ff:ff:ff:ff:ff"
        }
    },
    {
        "OXMTlv": {
            "field": "eth_src",
            "mask": null,
            "value": "f2:0b:a4:7d:f8:ea"
        }
    },
    {
        "OXMTlv": {
            "field": "vlan_vid",
            "mask": null,
            "value": 999
        }
    },
    {
        "OXMTlv": {
            "field": "ip_dscp",
            "mask": null,
```

(continues on next page)

(continued from previous page)

```
        "value": 9
    },
    {
        "OXMTlv": {
            "field": "ip_ecn",
            "mask": null,
            "value": 3
        }
    },
    {
        "OXMTlv": {
            "field": "ip_proto",
            "mask": null,
            "value": 99
        }
    },
    {
        "OXMTlv": {
            "field": "ipv4_src",
            "mask": null,
            "value": "1.2.3.4"
        }
    },
    {
        "OXMTlv": {
            "field": "ipv4_dst",
            "mask": null,
            "value": "1.2.3.4"
        }
    },
    {
        "OXMTlv": {
            "field": "tcp_src",
            "mask": null,
            "value": 8080
        }
    },
    {
        "OXMTlv": {
            "field": "tcp_dst",
            "mask": null,
            "value": 18080
        }
    },
    {
        "OXMTlv": {
            "field": "udp_src",
            "mask": null,
```

(continues on next page)

(continued from previous page)

```
        "value": 28080
    },
    {
        "OXMTlv": {
            "field": "udp_dst",
            "mask": null,
            "value": 55936
        }
    },
    {
        "OXMTlv": {
            "field": "sctp_src",
            "mask": null,
            "value": 48080
        }
    },
    {
        "OXMTlv": {
            "field": "sctp_dst",
            "mask": null,
            "value": 59328
        }
    },
    {
        "OXMTlv": {
            "field": "icmpv4_type",
            "mask": null,
            "value": 100
        }
    },
    {
        "OXMTlv": {
            "field": "icmpv4_code",
            "mask": null,
            "value": 101
        }
    },
    {
        "OXMTlv": {
            "field": "arp_op",
            "mask": null,
            "value": 1
        }
    },
    {
        "OXMTlv": {
            "field": "arp_spa",
            "mask": null,
```

(continues on next page)

(continued from previous page)

```
        "value": "10.0.0.1"
    }
},
{
    "OXMTlv": {
        "field": "arp_tpa",
        "mask": null,
        "value": "10.0.0.3"
    }
},
{
    "OXMTlv": {
        "field": "arp_sha",
        "mask": null,
        "value": "f2:0b:a4:7d:f8:ea"
    }
},
{
    "OXMTlv": {
        "field": "arp_tha",
        "mask": null,
        "value": "00:00:00:00:00:00"
    }
},
{
    "OXMTlv": {
        "field": "ipv6_src",
        "mask": null,
        "value": "fe80::f00b:a4ff:fe48:28a5"
    }
},
{
    "OXMTlv": {
        "field": "ipv6_dst",
        "mask": null,
        "value": "fe80::f00b:a4ff:fe05:b7dc"
    }
},
{
    "OXMTlv": {
        "field": "ipv6_flabel",
        "mask": null,
        "value": 541473
    }
},
{
    "OXMTlv": {
        "field": "icmpv6_type",
        "mask": null,
```

(continues on next page)



(continued from previous page)

```
        "value": 200
    },
    {
        "OXMTlv": {
            "field": "icmpv6_code",
            "mask": null,
            "value": 201
        }
    },
    {
        "OXMTlv": {
            "field": "ipv6_nd_target",
            "mask": null,
            "value": "fe80::a60:6eff:fe7f:74e7"
        }
    },
    {
        "OXMTlv": {
            "field": "ipv6_nd_sll",
            "mask": null,
            "value": "00:00:00:00:02:9a"
        }
    },
    {
        "OXMTlv": {
            "field": "ipv6_nd_tll",
            "mask": null,
            "value": "00:00:00:00:02:2b"
        }
    },
    {
        "OXMTlv": {
            "field": "mpls_label",
            "mask": null,
            "value": 624485
        }
    },
    {
        "OXMTlv": {
            "field": "mpls_tc",
            "mask": null,
            "value": 5
        }
    },
    {
        "OXMTlv": {
            "field": "mpls_bos",
            "mask": null,
```

(continues on next page)

(continued from previous page)

```

        "value": 1
    },
    {
        "OXMTlv": {
            "field": "pbb_isid",
            "mask": null,
            "value": 11259375
        }
    },
    {
        "OXMTlv": {
            "field": "tunnel_id",
            "mask": null,
            "value": 651061555542690057
        }
    },
    {
        "OXMTlv": {
            "field": "ipv6_exthdr",
            "mask": null,
            "value": 500
        }
    },
    {
        "OXMTlv": {
            "field": "pbb_uca",
            "mask": null,
            "value": 1
        }
    }
],
    "type": 1
}
},
    "out_group": 4294967295,
    "out_port": 4294967295,
    "priority": 123,
    "table_id": 1
}
}

```

**class** `os_ken.ofproto.ofproto_v1_4_parser.OFPGroupMod`(*datapath*, *command=0*, *type\_=0*, *group\_id=0*, *buckets=None*)

Modify group entry message

The controller sends this message to modify the group table.

Attribute	Description
command	One of the following values.  OFPGC_ADD OFPGC_MODIFY OFPGC_DELETE
type	One of the following values.  OFPGT_ALL OFPGT_SELECT OFPGT_INDIRECT OFPGT_FF
group_id	Group identifier
buckets	list of OFPBucket

type attribute corresponds to type\_ parameter of \_\_init\_\_.

Example:

```
def send_group_mod(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    port = 1
    max_len = 2000
    actions = [ofp_parser.OFPActionOutput(port, max_len)]

    weight = 100
    watch_port = 0
    watch_group = 0
    buckets = [ofp_parser.OFPBucket(weight, watch_port, watch_group,
                                    actions)]

    group_id = 1
    req = ofp_parser.OFPGroupMod(datapath, ofp.OFPGC_ADD,
                                 ofp.OFPGT_SELECT, group_id, buckets)
    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPGroupMod": {
    "buckets": [
      {
        "OFPBucket": {
          "actions": [
            {
```

(continues on next page)

(continued from previous page)

```

        "OFPActionOutput": {
            "len": 16,
            "max_len": 65535,
            "port": 2,
            "type": 0
        }
    ],
    "len": 32,
    "watch_group": 1,
    "watch_port": 1,
    "weight": 1
}
},
"command": 0,
"group_id": 1,
"type": 0
}
}

```

```

class os_ken.ofproto.ofproto_v1_4_parser.OFPPortMod(datapath, port_no=0,
                                                    hw_addr='00:00:00:00:00:00',
                                                    config=0, mask=0,
                                                    properties=None)

```

Port modification message

The controller sends this message to modify the behavior of the port.

Attribute	Description
port_no	Port number to modify
hw_addr	The hardware address that must be the same as hw_addr of OFPPort of OFPSwitchFeatures
config	Bitmap of configuration flags.  OFPPC_PORT_DOWN OFPPC_NO_RECV OFPPC_NO_FWD OFPPC_NO_PACKET_IN
mask	Bitmap of configuration flags above to be changed
properties	List of OFPPortModProp subclass instance

Example:

```

def send_port_mod(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    port_no = 3
    hw_addr = 'fa:c8:e8:76:1d:7e'
    config = 0
    mask = (ofp.OFPPC_PORT_DOWN | ofp.OFPPC_NO_RECV |
            ofp.OFPPC_NO_FWD | ofp.OFPPC_NO_PACKET_IN)
    advertise = (ofp.OFPPF_10MB_FD | ofp.OFPPF_100MB_FD |
                 ofp.OFPPF_1GB_FD | ofp.OFPPF_COPPER |
                 ofp.OFPPF_AUTONEG | ofp.OFPPF_PAUSE |
                 ofp.OFPPF_PAUSE_ASYM)
    properties = [ofp_parser.OFPPortModPropEthernet(advertise)]
    req = ofp_parser.OFPPortMod(datapath, port_no, hw_addr, config,
                                mask, properties)

    datapath.send_msg(req)

```

JSON Example:

```

{
  "OFPPortMod": {
    "config": 0,
    "hw_addr": "00:11:00:00:11:11",
    "mask": 0,
    "port_no": 1,
    "properties": [
      {
        "OFPPortModPropEthernet": {
          "advertise": 4096,
          "length": 8,
          "type": 0
        }
      },
      {
        "OFPPortModPropOptical": {
          "configure": 3,
          "fl_offset": 2000,
          "freq_lmda": 1500,
          "grid_span": 3000,
          "length": 24,
          "tx_pwr": 300,
          "type": 1
        }
      },
      {
        "OFPPortModPropExperimenter": {
          "data": [],
          "exp_type": 0,
          "experimenter": 101,

```

(continues on next page)

(continued from previous page)

```

        "length": 12,
        "type": 65535
    },
    {
        "OFPPortModPropExperimenter": {
            "data": [
                1
            ],
            "exp_type": 1,
            "experimenter": 101,
            "length": 16,
            "type": 65535
        }
    },
    {
        "OFPPortModPropExperimenter": {
            "data": [
                1,
                2
            ],
            "exp_type": 2,
            "experimenter": 101,
            "length": 20,
            "type": 65535
        }
    }
]
}
}

```

**class** `os_ken.ofproto.ofproto_v1_4_parser.OFPMeterMod`(*datapath*, *command=0*, *flags=1*, *meter\_id=1*, *bands=None*)

Meter modification message

The controller sends this message to modify the meter.

Attribute	Description
command	One of the following values.  OFPMC_ADD OFPMC_MODIFY OFPMC_DELETE
flags	Bitmap of the following flags.  OFPMF_KBPS OFPMF_PKTPTS OFPMF_BURST OFPMF_STATS
meter_id	Meter instance
bands	list of the following class instance.  OFPMeterBandDrop OFPMeterBandDscpRemark OFPMeterBandExperimenter

JSON Example:

```
{
  "OFPMeterMod": {
    "bands": [
      {
        "OFPMeterBandDrop": {
          "burst_size": 10,
          "len": 16,
          "rate": 1000,
          "type": 1
        }
      },
      {
        "OFPMeterBandDscpRemark": {
          "burst_size": 10,
          "len": 16,
          "prec_level": 1,
          "rate": 1000,
          "type": 2
        }
      },
      {
        "OFPMeterBandExperimenter": {
          "burst_size": 10,
          "experimenter": 999,

```

(continues on next page)

(continued from previous page)

```

        "len": 16,
        "rate": 1000,
        "type": 65535
    }
}
],
"command": 0,
"flags": 14,
"meter_id": 100
}
}

```

## Multipart Messages

```
class os_ken.ofproto.ofproto_v1_4_parser.OFPDescStatsRequest(datapath, flags=0,
                                                             type_=None)
```

Description statistics request message

The controller uses this message to query description of the switch.

Attribute	Description
flags	Zero or OFPMPF_REQ_MORE

Example:

```
def send_desc_stats_request(self, datapath):
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPDescStatsRequest(datapath, 0)
    datapath.send_msg(req)
```

JSON Example:

```

{
  "OFPDescStatsRequest": {
    "flags": 0,
    "type": 0
  }
}

```

```
class os_ken.ofproto.ofproto_v1_4_parser.OFPDescStatsReply(datapath, type_=None,
                                                            **kwargs)
```

Description statistics reply message

The switch responds with this message to a description statistics request.

Attribute	Description
body	Instance of OFPDescStats



Example:

```
@set_ev_cls(ofp_event.EventOFPDescStatsReply, MAIN_DISPATCHER)
def desc_stats_reply_handler(self, ev):
    body = ev.msg.body

    self.logger.debug('DescStats: mfr_desc=%s hw_desc=%s sw_desc=%s '
                      'serial_num=%s dp_desc=%s',
                      body.mfr_desc, body.hw_desc, body.sw_desc,
                      body.serial_num, body.dp_desc)
```

JSON Example:

```
{
  "OFPDescStatsReply": {
    "body": {
      "OFPDescStats": {
        "dp_desc": "dp",
        "hw_desc": "hw",
        "mfr_desc": "mfr",
        "serial_num": "serial",
        "sw_desc": "sw"
      }
    },
    "flags": 0,
    "type": 0
  }
}
```

```
class os_ken.ofproto.ofproto_v1_4_parser.OFPFlowStatsRequest(datapath, flags=0,
                                                             table_id=255,
                                                             out_port=4294967295,
                                                             out_group=4294967295,
                                                             cookie=0,
                                                             cookie_mask=0,
                                                             match=None,
                                                             type_=None)
```

Individual flow statistics request message

The controller uses this message to query individual flow statistics.

Attribute	Description
flags	Zero or OFPMPF_REQ_MORE
table_id	ID of table to read
out_port	Require matching entries to include this as an output port
out_group	Require matching entries to include this as an output group
cookie	Require matching entries to contain this cookie value
cookie_mask	Mask used to restrict the cookie bits that must match
match	Instance of OFPMatch

Example:

```
def send_flow_stats_request(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    cookie = cookie_mask = 0
    match = ofp_parser.OFPMatch(in_port=1)
    req = ofp_parser.OFPFlowStatsRequest(datapath, 0,
                                         ofp.OFPTT_ALL,
                                         ofp.OFPP_ANY, ofp.OFPG_ANY,
                                         cookie, cookie_mask,
                                         match)

    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPFlowStatsRequest": {
    "cookie": 0,
    "cookie_mask": 0,
    "flags": 0,
    "match": {
      "OFPMatch": {
        "length": 4,
        "oxm_fields": [],
        "type": 1
      }
    },
    "out_group": 4294967295,
    "out_port": 4294967295,
    "table_id": 0,
    "type": 1
  }
}
```

**class** `os_ken.ofproto.ofproto_v1_4_parser.OFPFlowStatsReply`(*datapath*, *type\_=None*, *\*\*kwargs*)

Individual flow statistics reply message

The switch responds with this message to an individual flow statistics request.

Attribute	Description
body	List of OFPFlowStats instance

Example:

```
@set_ev_cls(ofp_event.EventOFPFlowStatsReply, MAIN_DISPATCHER)
def flow_stats_reply_handler(self, ev):
    flows = []
    for stat in ev.msg.body:
```

(continues on next page)

(continued from previous page)

```

flows.append('table_id=%s '
            'duration_sec=%d duration_nsec=%d '
            'priority=%d '
            'idle_timeout=%d hard_timeout=%d flags=0x%04x '
            'importance=%d cookie=%d packet_count=%d '
            'byte_count=%d match=%s instructions=%s' %
            (stat.table_id,
             stat.duration_sec, stat.duration_nsec,
             stat.priority,
             stat.idle_timeout, stat.hard_timeout,
             stat.flags, stat.importance,
             stat.cookie, stat.packet_count, stat.byte_count,
             stat.match, stat.instructions))
self.logger.debug('FlowStats: %s', flows)

```

JSON Example:

```

{
  "OFPPFlowStatsReply": {
    "body": [
      {
        "OFPPFlowStats": {
          "byte_count": 0,
          "cookie": 0,
          "duration_nsec": 115277000,
          "duration_sec": 358,
          "flags": 0,
          "hard_timeout": 0,
          "idle_timeout": 0,
          "importance": 0,
          "instructions": [],
          "length": 56,
          "match": {
            "OFPMatch": {
              "length": 4,
              "oxm_fields": [],
              "type": 1
            }
          },
          "packet_count": 0,
          "priority": 65535,
          "table_id": 0
        }
      },
      {
        "OFPPFlowStats": {
          "byte_count": 0,
          "cookie": 0,
          "duration_nsec": 115055000,
          "duration_sec": 358,

```

(continues on next page)

(continued from previous page)

```
"flags": 0,
"hard_timeout": 0,
"idle_timeout": 0,
"importance": 0,
"instructions": [
  {
    "OFPInstructionActions": {
      "actions": [
        {
          "OFPActionOutput": {
            "len": 16,
            "max_len": 0,
            "port": 4294967290,
            "type": 0
          }
        }
      ],
      "len": 24,
      "type": 4
    }
  },
  {
    "length": 88,
    "match": {
      "OFPMatch": {
        "length": 10,
        "oxm_fields": [
          {
            "OXMTlv": {
              "field": "eth_type",
              "mask": null,
              "value": 2054
            }
          }
        ],
        "type": 1
      }
    },
    "packet_count": 0,
    "priority": 65534,
    "table_id": 0
  }
],
{
  "OFPFlowStats": {
    "byte_count": 238,
    "cookie": 0,
    "duration_nsec": 511582000,
    "duration_sec": 316220,
```

(continues on next page)

(continued from previous page)

```

"flags": 0,
"hard_timeout": 0,
"idle_timeout": 0,
"importance": 0,
"instructions": [
  {
    "OFPIInstructionGotoTable": {
      "len": 8,
      "table_id": 1,
      "type": 1
    }
  }
],
"length": 80,
"match": {
  "OFPMatch": {
    "length": 22,
    "oxm_fields": [
      {
        "OXMTlv": {
          "field": "in_port",
          "mask": null,
          "value": 6
        }
      },
      {
        "OXMTlv": {
          "field": "eth_src",
          "mask": null,
          "value": "f2:0b:a4:7d:f8:ea"
        }
      }
    ],
    "type": 1
  }
},
"packet_count": 3,
"priority": 123,
"table_id": 0
},
{
  "OFPFFlowStats": {
    "byte_count": 98,
    "cookie": 0,
    "duration_nsec": 980901000,
    "duration_sec": 313499,
    "flags": 0,
    "hard_timeout": 0,

```

(continues on next page)

(continued from previous page)

```
"idle_timeout": 0,
"importance": 0,
"instructions": [
  {
    "OFPInstructionActions": {
      "actions": [
        {
          "OFPActionSetField": {
            "field": {
              "OXMTlv": {
                "field": "vlan_vid",
                "mask": null,
                "value": 258
              }
            }
          },
          "len": 16,
          "type": 25
        }
      ],
      "OFPActionCopyTtlOut": {
        "len": 8,
        "type": 11
      },
      "OFPActionCopyTtlIn": {
        "len": 8,
        "type": 12
      },
      "OFPActionCopyTtlIn": {
        "len": 8,
        "type": 12
      },
      "OFPActionPopPbb": {
        "len": 8,
        "type": 27
      },
      "OFPActionPushPbb": {
        "ethertype": 4660,
        "len": 8,
        "type": 26
      }
    }
  }
]
```

(continues on next page)

(continued from previous page)

```
    },
    {
      "OFPACTIONPopMpls": {
        "ethertype": 39030,
        "len": 8,
        "type": 20
      }
    },
    {
      "OFPACTIONPushMpls": {
        "ethertype": 34887,
        "len": 8,
        "type": 19
      }
    },
    {
      "OFPACTIONPopVlan": {
        "len": 8,
        "type": 18
      }
    },
    {
      "OFPACTIONPushVlan": {
        "ethertype": 33024,
        "len": 8,
        "type": 17
      }
    },
    {
      "OFPACTIONDecMplsTtl": {
        "len": 8,
        "type": 16
      }
    },
    {
      "OFPACTIONSetMplsTtl": {
        "len": 8,
        "mpls_ttl": 10,
        "type": 15
      }
    },
    {
      "OFPACTIONDecNwTtl": {
        "len": 8,
        "type": 24
      }
    },
    {
      "OFPACTIONSetNwTtl": {
```

(continues on next page)

(continued from previous page)

```
        "len": 8,
        "nw_ttl": 10,
        "type": 23
    }
},
{
    "OFPActionSetQueue": {
        "len": 8,
        "queue_id": 3,
        "type": 21
    }
},
{
    "OFPActionGroup": {
        "group_id": 99,
        "len": 8,
        "type": 22
    }
},
{
    "OFPActionOutput": {
        "len": 16,
        "max_len": 65535,
        "port": 6,
        "type": 0
    }
},
{
    "OFPActionExperimenterUnknown": {
        "len": 16,
        "data": "ZXhwX2RhdGE=",
        "experimenter": 98765432,
        "type": 65535
    }
},
{
    "NXActionUnknown": {
        "len": 16,
        "data": "cF9kYXRh",
        "experimenter": 8992,
        "type": 65535,
        "subtype": 25976
    }
}
],
"len": 192,
"type": 3
}
},
```

(continues on next page)



(continued from previous page)

```

{
  "OFPInstructionActions": {
    "actions": [
      {
        "OFPActionSetField": {
          "field": {
            "OXMTlv": {
              "field": "eth_src",
              "mask": null,
              "value": "01:02:03:04:05:06"
            }
          }
        },
        "len": 16,
        "type": 25
      }
    ],
    "len": 40,
    "type": 4
  },
  {
    "OFPInstructionActions": {
      "actions": [
        {
          "OFPActionSetField": {
            "field": {
              "OXMTlv": {
                "field": "pbb_uca",
                "mask": null,
                "value": 1
              }
            }
          },
          "len": 16,
          "type": 25
        }
      ],
      "len": 40,
      "type": 4
    },
    {
      "OFPInstructionActions": {
        "actions": [
          {
            "OFPActionOutput": {
              "len": 16,
              "max_len": 65535,
              "port": 4294967293,
              "type": 0
            }
          }
        ],
        "len": 24,
        "type": 3
      }
    }
  }
}

```

(continues on next page)

(continued from previous page)

```

    },
    ],
    "length": 312,
    "match": {
        "OFPMatch": {
            "length": 4,
            "oxm_fields": [],
            "type": 1
        }
    },
    "packet_count": 1,
    "priority": 0,
    "table_id": 0
}
},
],
"flags": 0,
"type": 1
}
}

```

```

class os_ken.ofproto.ofproto_v1_4_parser.OFPAggregateStatsRequest(datapath, flags,
                                                                    table_id,
                                                                    out_port,
                                                                    out_group,
                                                                    cookie,
                                                                    cookie_mask,
                                                                    match,
                                                                    type_=None)

```

Aggregate flow statistics request message

The controller uses this message to query aggregate flow statistics.

Attribute	Description
flags	Zero or OFPMPF_REQ_MORE
table_id	ID of table to read
out_port	Require matching entries to include this as an output port
out_group	Require matching entries to include this as an output group
cookie	Require matching entries to contain this cookie value
cookie_mask	Mask used to restrict the cookie bits that must match
match	Instance of OFPMatch

Example:

```

def send_aggregate_stats_request(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    cookie = cookie_mask = 0

```

(continues on next page)

(continued from previous page)

```

match = ofp_parser.OFPMatch(in_port=1)
req = ofp_parser.OFPAggregateStatsRequest(datapath, 0,
                                           ofp.OFPTT_ALL,
                                           ofp.OFPP_ANY,
                                           ofp.OFPG_ANY,
                                           cookie, cookie_mask,
                                           match)

datapath.send_msg(req)

```

JSON Example:

```

{
  "OFPAggregateStatsRequest": {
    "cookie": 0,
    "cookie_mask": 0,
    "flags": 0,
    "match": {
      "OFPMatch": {
        "length": 4,
        "oxm_fields": [],
        "type": 1
      }
    },
    "out_group": 4294967295,
    "out_port": 4294967295,
    "table_id": 255,
    "type": 2
  }
}

```

```

class os_ken.ofproto.ofproto_v1_4_parser.OFPAggregateStatsReply(datapath,
                                                                type_=None,
                                                                **kwargs)

```

Aggregate flow statistics reply message

The switch responds with this message to an aggregate flow statistics request.

Attribute	Description
body	Instance of OFPAggregateStats

Example:

```

@set_ev_cls(ofp_event.EventOFPAggregateStatsReply, MAIN_DISPATCHER)
def aggregate_stats_reply_handler(self, ev):
    body = ev.msg.body

    self.logger.debug('AggregateStats: packet_count=%d byte_count=%d '
                      'flow_count=%d',

```

(continues on next page)

(continued from previous page)

```
body.packet_count, body.byte_count,
body.flow_count)
```

JSON Example:

```
{
  "OFPAggregateStatsReply": {
    "body": {
      "OFPAggregateStats": {
        "byte_count": 574,
        "flow_count": 6,
        "packet_count": 7
      }
    },
    "flags": 0,
    "type": 2
  }
}
```

```
class os_ken.ofproto.ofproto_v1_4_parser.OFPTableStatsRequest(datapath, flags,
                                                                type_=None)
```

Table statistics request message

The controller uses this message to query flow table statistics.

Attribute	Description
flags	Zero or OFPMPF_REQ_MORE

Example:

```
def send_table_stats_request(self, datapath):
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPTableStatsRequest(datapath, 0)
    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPTableStatsRequest": {
    "flags": 0,
    "type": 3
  }
}
```

```
class os_ken.ofproto.ofproto_v1_4_parser.OFPTableStatsReply(datapath, type_=None,
                                                             **kwargs)
```

Table statistics reply message

The switch responds with this message to a table statistics request.

Attribute	Description
body	List of OFPTableStats instance

Example:

```
@set_ev_cls(ofp_event.EventOFPTableStatsReply, MAIN_DISPATCHER)
def table_stats_reply_handler(self, ev):
    tables = []
    for stat in ev.msg.body:
        tables.append('table_id=%d active_count=%d lookup_count=%d '
                    ' matched_count=%d' %
                    (stat.table_id, stat.active_count,
                     stat.lookup_count, stat.matched_count))
    self.logger.debug('TableStats: %s', tables)
```

JSON Example:

```
{
  "OFPTableStatsReply": {
    "body": [
      {
        "OFPTableStats": {
          "active_count": 4,
          "lookup_count": 4,
          "matched_count": 4,
          "table_id": 0
        }
      },
      {
        "OFPTableStats": {
          "active_count": 4,
          "lookup_count": 4,
          "matched_count": 4,
          "table_id": 1
        }
      }
    ],
    "flags": 0,
    "type": 3
  }
}
```

```
class os_ken.ofproto.ofproto_v1_4_parser.OFPTableDescStatsRequest(datapath,
                                                                    flags=0,
                                                                    type_=None)
```

Table description request message

The controller uses this message to query description of all the tables.

Attribute	Description
flags	Zero or OFPMPF_REQ_MORE

Example:

```
def send_table_desc_stats_request(self, datapath):
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPTableDescStatsRequest(datapath, 0)
    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPTableDescStatsRequest": {
    "flags": 0,
    "type": 14
  }
}
```

```
class os_ken.ofproto.ofproto_v1_4_parser.OFPTableDescStatsReply(datapath,
                                                                type_=None,
                                                                **kwargs)
```

Table description reply message

The switch responds with this message to a table description request.

Attribute	Description
body	List of OFPTableDesc instance

Example:

```
@set_ev_cls(ofp_event.EventOFPTableDescStatsReply, MAIN_DISPATCHER)
def table_desc_stats_reply_handler(self, ev):
    tables = []
    for p in ev.msg.body:
        tables.append('table_id=%d config=0x%08x properties=%s' %
                    (p.table_id, p.config, repr(p.properties)))
    self.logger.debug('OFPTableDescStatsReply received: %s', tables)
```

JSON Example:

```
{
  "OFPTableDescStatsReply": {
    "body": [
      {
        "OFPTableDesc": {
          "config": 0,
          "length": 24,

```

(continues on next page)

(continued from previous page)

```

        "properties": [
            {
                "OFPTableModPropExperimenter": {
                    "data": [],
                    "exp_type": 0,
                    "experimenter": 101,
                    "length": 12,
                    "type": 65535
                }
            }
        ],
        "table_id": 7
    },
    {
        "OFPTableDesc": {
            "config": 0,
            "length": 80,
            "properties": [
                {
                    "OFPTableModPropEviction": {
                        "flags": 0,
                        "length": 8,
                        "type": 2
                    }
                },
                {
                    "OFPTableModPropVacancy": {
                        "length": 8,
                        "type": 3,
                        "vacancy": 0,
                        "vacancy_down": 0,
                        "vacancy_up": 0
                    }
                }
            ],
            {
                "OFPTableModPropExperimenter": {
                    "data": [],
                    "exp_type": 0,
                    "experimenter": 101,
                    "length": 12,
                    "type": 65535
                }
            },
            {
                "OFPTableModPropExperimenter": {
                    "data": [
                        1
                    ],

```

(continues on next page)

(continued from previous page)

```

        "exp_type": 1,
        "experimenter": 101,
        "length": 16,
        "type": 65535
    }
},
{
    "OFPTableModPropExperimenter": {
        "data": [
            1,
            2
        ],
        "exp_type": 2,
        "experimenter": 101,
        "length": 20,
        "type": 65535
    }
},
],
"table_id": 8
}
},
],
"flags": 0,
"type": 14
}
}

```

```

class os_ken.ofproto.ofproto_v1_4_parser.OFPTableFeaturesStatsRequest(datapath,
                                                                    flags=0,
                                                                    body=None,
                                                                    type_=None)

```

Table features statistics request message

The controller uses this message to query table features.

Attribute	Description
body	List of OFPTableFeaturesStats instances. The default is [].

JSON Example:

See an example in:

```

os_ken/tests/unit/ofproto/json/of14/5-53-ofp_table_features_request.
packet.json

```

```

class os_ken.ofproto.ofproto_v1_4_parser.OFPTableFeaturesStatsReply(datapath,
                                                                    type_=None,
                                                                    **kwargs)

```

Table features statistics reply message



The switch responds with this message to a table features statistics request.

Attribute	Description
body	List of OFPTableFeaturesStats instance

JSON Example:

See an example in:

```
os_ken/tests/unit/ofproto/json/of14/5-54-ofp_table_features_reply.
packet.json
```

```
class os_ken.ofproto.ofproto_v1_4_parser.OFPPortStatsRequest(datapath, flags,
                                                             port_no, type_=None)
```

Port statistics request message

The controller uses this message to query information about ports statistics.

Attribute	Description
flags	Zero or OFPMPF_REQ_MORE
port_no	Port number to read (OFPP_ANY to all ports)

Example:

```
def send_port_stats_request(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPPortStatsRequest(datapath, 0, ofp.OFPP_ANY)
    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPPortStatsRequest": {
    "flags": 0,
    "port_no": 4294967295,
    "type": 4
  }
}
```

```
class os_ken.ofproto.ofproto_v1_4_parser.OFPPortStatsReply(datapath, type_=None,
                                                            **kwargs)
```

Port statistics reply message

The switch responds with this message to a port statistics request.

Attribute	Description
body	List of OFPPortStats instance

Example:

```
@set_ev_cls(ofp_event.EventOFPPortStatsReply, MAIN_DISPATCHER)
def port_stats_reply_handler(self, ev):
    ports = []
    for stat in ev.msg.body:
        ports.append(stat.length, stat.port_no,
                    stat.duration_sec, stat.duration_nsec,
                    stat.rx_packets, stat.tx_packets,
                    stat.rx_bytes, stat.tx_bytes,
                    stat.rx_dropped, stat.tx_dropped,
                    stat.rx_errors, stat.tx_errors,
                    repr(stat.properties))
    self.logger.debug('PortStats: %s', ports)
```

JSON Example:

```
{
  "OFPPortStatsReply": {
    "body": [
      {
        "OFPPortStats": {
          "duration_nsec": 0,
          "duration_sec": 0,
          "length": 224,
          "port_no": 7,
          "properties": [
            {
              "OFPPortStatsPropEthernet": {
                "collisions": 0,
                "length": 40,
                "rx_crc_err": 0,
                "rx_frame_err": 0,
                "rx_over_err": 0,
                "type": 0
              }
            },
            {
              "OFPPortStatsPropOptical": {
                "bias_current": 300,
                "flags": 3,
                "length": 44,
                "rx_freq_lmda": 1500,
                "rx_grid_span": 500,
                "rx_offset": 700,
                "rx_pwr": 2000,
                "temperature": 273,
                "tx_freq_lmda": 1500,
                "tx_grid_span": 500,
                "tx_offset": 700,
                "tx_pwr": 2000,

```

(continues on next page)

(continued from previous page)

```

        "type": 1
    }
},
{
    "OFPPortStatsPropExperimenter": {
        "data": [],
        "exp_type": 0,
        "experimenter": 101,
        "length": 12,
        "type": 65535
    }
},
{
    "OFPPortStatsPropExperimenter": {
        "data": [
            1
        ],
        "exp_type": 1,
        "experimenter": 101,
        "length": 16,
        "type": 65535
    }
},
{
    "OFPPortStatsPropExperimenter": {
        "data": [
            1,
            2
        ],
        "exp_type": 2,
        "experimenter": 101,
        "length": 20,
        "type": 65535
    }
}
],
"rx_bytes": 0,
"rx_dropped": 0,
"rx_errors": 0,
"rx_packets": 0,
"tx_bytes": 336,
"tx_dropped": 0,
"tx_errors": 0,
"tx_packets": 4
}
},
{
    "OFPPortStats": {
        "duration_nsec": 0,

```

(continues on next page)

(continued from previous page)

```

    "duration_sec": 0,
    "length": 120,
    "port_no": 6,
    "properties": [
        {
            "OFPPortStatsPropEthernet": {
                "collisions": 0,
                "length": 40,
                "rx_crc_err": 0,
                "rx_frame_err": 0,
                "rx_over_err": 0,
                "type": 0
            }
        }
    ],
    "rx_bytes": 336,
    "rx_dropped": 0,
    "rx_errors": 0,
    "rx_packets": 4,
    "tx_bytes": 336,
    "tx_dropped": 0,
    "tx_errors": 0,
    "tx_packets": 4
}
},
"flags": 0,
"type": 4
}
}
}

```

```

class os_ken.ofproto.ofproto_v1_4_parser.OFPPortDescStatsRequest(datapath,
                                                                    flags=0,
                                                                    type_=None)

```

Port description request message

The controller uses this message to query description of all the ports.

Attribute	Description
flags	Zero or OFPMPF_REQ_MORE

Example:

```

def send_port_desc_stats_request(self, datapath):
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPPortDescStatsRequest(datapath, 0)
    datapath.send_msg(req)

```

JSON Example:

```
{
  "OFPPortDescStatsRequest": {
    "flags": 0,
    "type": 13
  }
}
```

```
class os_ken.ofproto.ofproto_v1_4_parser.OFPPortDescStatsReply(datapath,
                                                                type_=None,
                                                                **kwargs)
```

Port description reply message

The switch responds with this message to a port description request.

Attribute	Description
body	List of OFPPort instance

Example:

```
@set_ev_cls(ofp_event.EventOFPPortDescStatsReply, MAIN_DISPATCHER)
def port_desc_stats_reply_handler(self, ev):
    ports = []
    for p in ev.msg.body:
        ports.append('port_no=%d hw_addr=%s name=%s config=0x%08x '
                    'state=0x%08x properties=%s' %
                    (p.port_no, p.hw_addr,
                     p.name, p.config, p.state, repr(p.properties)))
    self.logger.debug('OFPPortDescStatsReply received: %s', ports)
```

JSON Example:

```
{
  "OFPPortDescStatsReply": {
    "body": [
      {
        "OFPPort": {
          "config": 0,
          "hw_addr": "f2:0b:a4:d0:3f:70",
          "length": 168,
          "name": "Port7",
          "port_no": 7,
          "properties": [
            {
              "OFPPortDescPropEthernet": {
                "advertised": 10240,
                "curr": 10248,
                "curr_speed": 5000,
                "length": 32,
```

(continues on next page)

(continued from previous page)

```

        "max_speed": 5000,
        "peer": 10248,
        "supported": 10248,
        "type": 0
    }
},
{
    "OFPPortDescPropOptical": {
        "length": 40,
        "rx_grid_freq_lmda": 1500,
        "rx_max_freq_lmda": 2000,
        "rx_min_freq_lmda": 1000,
        "supported": 1,
        "tx_grid_freq_lmda": 1500,
        "tx_max_freq_lmda": 2000,
        "tx_min_freq_lmda": 1000,
        "tx_pwr_max": 2000,
        "tx_pwr_min": 1000,
        "type": 1
    }
},
{
    "OFPPortDescPropExperimenter": {
        "data": [],
        "exp_type": 0,
        "experimenter": 101,
        "length": 12,
        "type": 65535
    }
},
{
    "OFPPortDescPropExperimenter": {
        "data": [
            1
        ],
        "exp_type": 1,
        "experimenter": 101,
        "length": 16,
        "type": 65535
    }
},
{
    "OFPPortDescPropExperimenter": {
        "data": [
            1,
            2
        ],
        "exp_type": 2,
        "experimenter": 101,

```

(continues on next page)

(continued from previous page)

```

        "length": 20,
        "type": 65535
    }
    },
    ],
    "state": 4
}
},
{
    "OFPPort": {
        "config": 0,
        "hw_addr": "f2:0b:a4:7d:f8:ea",
        "length": 72,
        "name": "Port6",
        "port_no": 6,
        "properties": [
            {
                "OFPPortDescPropEthernet": {
                    "advertised": 10240,
                    "curr": 10248,
                    "curr_speed": 5000,
                    "length": 32,
                    "max_speed": 5000,
                    "peer": 10248,
                    "supported": 10248,
                    "type": 0
                }
            }
        ],
        "state": 4
    }
},
],
"flags": 0,
"type": 13
}
}
}

```

```

class os_ken.ofproto.ofproto_v1_4_parser.OFPQueueStatsRequest(datapath, flags=0,
                                                                port_no=4294967295,
                                                                queue_id=4294967295,
                                                                type_=None)

```

Queue statistics request message

The controller uses this message to query queue statistics.

Attribute	Description
flags	Zero or OFPMPF_REQ_MORE
port_no	Port number to read
queue_id	ID of queue to read

Example:

```
def send_queue_stats_request(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPQueueStatsRequest(datapath, 0, ofp.OFPP_ANY,
                                           ofp.OFPQ_ALL)

    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPQueueStatsRequest": {
    "flags": 0,
    "port_no": 4294967295,
    "queue_id": 4294967295,
    "type": 5
  }
}
```

```
class os_ken.ofproto.ofproto_v1_4_parser.OFPQueueStatsReply(datapath, type_=None,
                                                             **kwargs)
```

Queue statistics reply message

The switch responds with this message to an aggregate flow statistics request.

Attribute	Description
body	List of OFPQueueStats instance

Example:

```
@set_ev_cls(ofp_event.EventOFPQueueStatsReply, MAIN_DISPATCHER)
def queue_stats_reply_handler(self, ev):
    queues = []
    for stat in ev.msg.body:
        queues.append('port_no=%d queue_id=%d '
                     'tx_bytes=%d tx_packets=%d tx_errors=%d '
                     'duration_sec=%d duration_nsec=%d'
                     'properties=%s' %
                     (stat.port_no, stat.queue_id,
                      stat.tx_bytes, stat.tx_packets, stat.tx_errors,
                      stat.duration_sec, stat.duration_nsec,
```

(continues on next page)



(continued from previous page)

```
repr(stat.properties)))
self.logger.debug('QueueStats: %s', queues)
```

JSON Example:

```
{
  "OFPQueueStatsReply": {
    "body": [
      {
        "OFPQueueStats": {
          "duration_nsec": 0,
          "duration_sec": 0,
          "length": 104,
          "port_no": 7,
          "properties": [
            {
              "OFPQueueStatsPropExperimenter": {
                "data": [],
                "exp_type": 0,
                "experimenter": 101,
                "length": 12,
                "type": 65535
              }
            },
            {
              "OFPQueueStatsPropExperimenter": {
                "data": [
                  1
                ],
                "exp_type": 1,
                "experimenter": 101,
                "length": 16,
                "type": 65535
              }
            },
            {
              "OFPQueueStatsPropExperimenter": {
                "data": [
                  1,
                  2
                ],
                "exp_type": 2,
                "experimenter": 101,
                "length": 20,
                "type": 65535
              }
            }
          ],
          "queue_id": 1,
          "tx_bytes": 0,
```

(continues on next page)

(continued from previous page)

```

        "tx_errors": 0,
        "tx_packets": 0
    },
    {
        "OFPQueueStats": {
            "duration_nsec": 0,
            "duration_sec": 0,
            "length": 48,
            "port_no": 6,
            "properties": [],
            "queue_id": 1,
            "tx_bytes": 0,
            "tx_errors": 0,
            "tx_packets": 0
        }
    },
    {
        "OFPQueueStats": {
            "duration_nsec": 0,
            "duration_sec": 0,
            "length": 48,
            "port_no": 7,
            "properties": [],
            "queue_id": 2,
            "tx_bytes": 0,
            "tx_errors": 0,
            "tx_packets": 0
        }
    }
],
"flags": 0,
"type": 5
}
}

```

```

class os_ken.ofproto.ofproto_v1_4_parser.OFPQueueDescStatsRequest(datapath,
                                                                    flags=0,
                                                                    port_no=4294967295,
                                                                    queue_id=4294967295,
                                                                    type_=None)

```

Queue description request message

The controller uses this message to query description of all the queues.

Attribute	Description
flags	Zero or OFPMPF_REQ_MORE
port_no	Port number to read (OFPP_ANY for all ports)
queue_id	ID of queue to read (OFPQ_ALL for all queues)

Example:

```
def send_queue_desc_stats_request(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser
    req = ofp_parser.OFPQueueDescStatsRequest(datapath, 0,
                                               ofp.OFPP_ANY,
                                               ofp.OFPQ_ALL)

    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPQueueDescStatsRequest": {
    "flags": 0,
    "port_no": 7,
    "queue_id": 4294967295,
    "type": 15
  }
}
```

```
class os_ken.ofproto.ofproto_v1_4_parser.OFPQueueDescStatsReply(datapath,
                                                                type_=None,
                                                                **kwargs)
```

Queue description reply message

The switch responds with this message to a queue description request.

Attribute	Description
body	List of OFPQueueDesc instance

Example:

```
@set_ev_cls(ofp_event.EventOFPQueueDescStatsReply, MAIN_DISPATCHER)
def queue_desc_stats_reply_handler(self, ev):
    queues = []
    for q in ev.msg.body:
        queues.append('port_no=%d queue_id=0x%08x properties=%s' %
                     (q.port_no, q.queue_id, repr(q.properties)))
    self.logger.debug('OFPQueueDescStatsReply received: %s', queues)
```

JSON Example:

```
{
  "OFPQueueDescStatsReply": {
    "body": [
      {
        "OFPQueueDesc": {
          "len": 32,
          "port_no": 7,

```

(continues on next page)

(continued from previous page)

```

        "properties": [
            {
                "OFPQueueDescPropExperimenter": {
                    "data": [],
                    "exp_type": 0,
                    "experimenter": 101,
                    "length": 12,
                    "type": 65535
                }
            }
        ],
        "queue_id": 0
    },
    {
        "OFPQueueDesc": {
            "len": 88,
            "port_no": 8,
            "properties": [
                {
                    "OFPQueueDescPropMinRate": {
                        "length": 8,
                        "rate": 300,
                        "type": 1
                    }
                },
                {
                    "OFPQueueDescPropMaxRate": {
                        "length": 8,
                        "rate": 900,
                        "type": 2
                    }
                },
                {
                    "OFPQueueDescPropExperimenter": {
                        "data": [],
                        "exp_type": 0,
                        "experimenter": 101,
                        "length": 12,
                        "type": 65535
                    }
                }
            ],
            {
                "OFPQueueDescPropExperimenter": {
                    "data": [
                        1
                    ],
                    "exp_type": 1,
                    "experimenter": 101,

```

(continues on next page)

(continued from previous page)

```

        "length": 16,
        "type": 65535
    },
    {
        "OFPQueueDescPropExperimenter": {
            "data": [
                1,
                2
            ],
            "exp_type": 2,
            "experimenter": 101,
            "length": 20,
            "type": 65535
        }
    },
    "queue_id": 1
}
],
"flags": 0,
"type": 15
}
}

```

```

class os_ken.ofproto.ofproto_v1_4_parser.OFPGroupStatsRequest(datapath, flags=0,
                                                                group_id=4294967292,
                                                                type_=None)

```

Group statistics request message

The controller uses this message to query statistics of one or more groups.

Attribute	Description
flags	Zero or OFPMPF_REQ_MORE
group_id	ID of group to read (OFPG_ALL to all groups)

Example:

```

def send_group_stats_request(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPGroupStatsRequest(datapath, 0, ofp.OFPG_ALL)
    datapath.send_msg(req)

```

JSON Example:

```
{
  "OFPGroupStatsRequest": {
    "flags": 0,
    "group_id": 4294967292,
    "type": 6
  }
}
```

```
class os_ken.ofproto.ofproto_v1_4_parser.OFPGroupStatsReply(datapath, type_=None,
                                                            **kwargs)
```

Group statistics reply message

The switch responds with this message to a group statistics request.

Attribute	Description
body	List of OFPGroupStats instance

Example:

```
@set_ev_cls(ofp_event.EventOFPGroupStatsReply, MAIN_DISPATCHER)
def group_stats_reply_handler(self, ev):
    groups = []
    for stat in ev.msg.body:
        groups.append('length=%d group_id=%d '
                     'ref_count=%d packet_count=%d byte_count=%d '
                     'duration_sec=%d duration_nsec=%d' %
                     (stat.length, stat.group_id,
                      stat.ref_count, stat.packet_count,
                      stat.byte_count, stat.duration_sec,
                      stat.duration_nsec))
    self.logger.debug('GroupStats: %s', groups)
```

JSON Example:

```
{
  "OFPGroupStatsReply": {
    "body": [
      {
        "OFPGroupStats": {
          "bucket_stats": [
            {
              "OFPBucketCounter": {
                "byte_count": 2345,
                "packet_count": 234
              }
            }
          ],
          "byte_count": 12345,
          "duration_nsec": 609036000,

```

(continues on next page)

(continued from previous page)

```

        "duration_sec": 9,
        "group_id": 1,
        "length": 56,
        "packet_count": 123,
        "ref_count": 2
    }
}
],
"flags": 0,
"type": 6
}
}

```

```

class os_ken.ofproto.ofproto_v1_4_parser.OFPGroupDescStatsRequest(datapath,
                                                                    flags=0,
                                                                    type_=None)

```

Group description request message

The controller uses this message to list the set of groups on a switch.

Attribute	Description
flags	Zero or OFPMPF_REQ_MORE

Example:

```

def send_group_desc_stats_request(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPGroupDescStatsRequest(datapath, 0)
    datapath.send_msg(req)

```

JSON Example:

```

{
  "OFPGroupDescStatsRequest": {
    "flags": 0,
    "type": 7
  }
}

```

```

class os_ken.ofproto.ofproto_v1_4_parser.OFPGroupDescStatsReply(datapath,
                                                                    type_=None,
                                                                    **kwargs)

```

Group description reply message

The switch responds with this message to a group description request.

Attribute	Description
body	List of OFPGroupDescStats instance

Example:

```
@set_ev_cls(ofp_event.EventOFPGroupDescStatsReply, MAIN_DISPATCHER)
def group_desc_stats_reply_handler(self, ev):
    descs = []
    for stat in ev.msg.body:
        descs.append('length=%d type=%d group_id=%d '
                    'buckets=%s' %
                    (stat.length, stat.type, stat.group_id,
                     stat.bucket))
    self.logger.debug('GroupDescStats: %s', descs)
```

JSON Example:

```
{
  "OFPGroupDescStatsReply": {
    "body": [
      {
        "OFPGroupDescStats": {
          "buckets": [
            {
              "OFPBucket": {
                "actions": [
                  {
                    "OFPActionOutput": {
                      "len": 16,
                      "max_len": 65535,
                      "port": 2,
                      "type": 0
                    }
                  }
                ]
              }
            ],
            "len": 32,
            "watch_group": 1,
            "watch_port": 1,
            "weight": 1
          }
        },
        "group_id": 1,
        "length": 40,
        "type": 0
      }
    ],
    "flags": 0,
```

(continues on next page)



(continued from previous page)

```

        "type": 7
    }
}

```

```

class os_ken.ofproto.ofproto_v1_4_parser.OFPGroupFeaturesStatsRequest(datapath,
                                                                    flags=0,
                                                                    type_=None)

```

Group features request message

The controller uses this message to list the capabilities of groups on a switch.

Attribute	Description
flags	Zero or OFPMPF_REQ_MORE

Example:

```

def send_group_features_stats_request(self, datapath):
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPGroupFeaturesStatsRequest(datapath, 0)
    datapath.send_msg(req)

```

JSON Example:

```

{
  "OFPGroupFeaturesStatsRequest": {
    "flags": 0,
    "type": 8
  }
}

```

```

class os_ken.ofproto.ofproto_v1_4_parser.OFPGroupFeaturesStatsReply(datapath,
                                                                    type_=None,
                                                                    **kwargs)

```

Group features reply message

The switch responds with this message to a group features request.

Attribute	Description
body	Instance of OFPGroupFeaturesStats

Example:

```

@set_ev_cls(ofp_event.EventOFPGroupFeaturesStatsReply, MAIN_DISPATCHER)
def group_features_stats_reply_handler(self, ev):
    body = ev.msg.body

    self.logger.debug('GroupFeaturesStats: types=%d '

```

(continues on next page)

(continued from previous page)

```
'capabilities=0x%08x max_groups=%s '
'actions=%s',
body.types, body.capabilities,
body.max_groups, body.actions)
```

JSON Example:

```
{
  "OFPGroupFeaturesStatsReply": {
    "body": {
      "OFPGroupFeaturesStats": {
        "actions": [
          67082241,
          67082241,
          67082241,
          67082241
        ],
        "capabilities": 5,
        "max_groups": [
          16777216,
          16777216,
          16777216,
          16777216
        ],
        "types": 15
      }
    },
    "flags": 0,
    "type": 8
  }
}
```

```
class os_ken.ofproto.ofproto_v1_4_parser.OFPMeterStatsRequest(datapath, flags=0,
                                                                meter_id=4294967295,
                                                                type_=None)
```

Meter statistics request message

The controller uses this message to query statistics for one or more meters.

Attribute	Description
flags	Zero or OFPMPF_REQ_MORE
meter_id	ID of meter to read (OFPM_ALL to all meters)

Example:

```
def send_meter_stats_request(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser
```

(continues on next page)

(continued from previous page)

```
req = ofp_parser.OFPMeterStatsRequest(datapath, 0, ofp.OFPM_ALL)
datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPMeterStatsRequest": {
    "flags": 0,
    "meter_id": 4294967295,
    "type": 9
  }
}
```

```
class os_ken.ofproto.ofproto_v1_4_parser.OFPMeterStatsReply(datapath, type_=None,
                                                            **kwargs)
```

Meter statistics reply message

The switch responds with this message to a meter statistics request.

Attribute	Description
body	List of OFPMeterStats instance

Example:

```
@set_ev_cls(ofp_event.EventOFPMeterStatsReply, MAIN_DISPATCHER)
def meter_stats_reply_handler(self, ev):
    meters = []
    for stat in ev.msg.body:
        meters.append('meter_id=0x%08x len=%d flow_count=%d '
                     'packet_in_count=%d byte_in_count=%d '
                     'duration_sec=%d duration_nsec=%d '
                     'band_stats=%s' %
                     (stat.meter_id, stat.len, stat.flow_count,
                      stat.packet_in_count, stat.byte_in_count,
                      stat.duration_sec, stat.duration_nsec,
                      stat.band_stats))
    self.logger.debug('MeterStats: %s', meters)
```

JSON Example:

```
{
  "OFPMeterStatsReply": {
    "body": [
      {
        "OFPMeterStats": {
          "band_stats": [
            {
              "OFPMeterBandStats": {
```

(continues on next page)

(continued from previous page)

```

        "byte_band_count": 0,
        "packet_band_count": 0
    }
}
],
"byte_in_count": 0,
"duration_nsec": 480000,
"duration_sec": 0,
"flow_count": 0,
"len": 56,
"meter_id": 100,
"packet_in_count": 0
}
}
],
"flags": 0,
"type": 9
}
}

```

```

class os_ken.ofproto.ofproto_v1_4_parser.OFPMeterConfigStatsRequest(datapath,
                                                                    flags=0, meter_id=4294967295,
                                                                    type_=None)

```

Meter configuration statistics request message

The controller uses this message to query configuration for one or more meters.

Attribute	Description
flags	Zero or OFPMPF_REQ_MORE
meter_id	ID of meter to read (OFPM_ALL to all meters)

Example:

```

def send_meter_config_stats_request(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPMeterConfigStatsRequest(datapath, 0,
                                                ofp.OFPM_ALL)

    datapath.send_msg(req)

```

JSON Example:

```

{
  "OFPMeterConfigStatsRequest": {
    "flags": 0,
    "meter_id": 4294967295,
    "type": 10
  }
}

```

(continues on next page)

(continued from previous page)

```
}
}
```

```
class os_ken.ofproto.ofproto_v1_4_parser.OFPMeterConfigStatsReply(datapath,
                                                                    type_=None,
                                                                    **kwargs)
```

Meter configuration statistics reply message

The switch responds with this message to a meter configuration statistics request.

Attribute	Description
body	List of OFPMeterConfigStats instance

Example:

```
@set_ev_cls(ofp_event.EventOFPMeterConfigStatsReply, MAIN_DISPATCHER)
def meter_config_stats_reply_handler(self, ev):
    configs = []
    for stat in ev.msg.body:
        configs.append('length=%d flags=0x%04x meter_id=0x%08x '
                      'bands=%s' %
                      (stat.length, stat.flags, stat.meter_id,
                       stat.bands))
    self.logger.debug('MeterConfigStats: %s', configs)
```

JSON Example:

```
{
  "OFPMeterConfigStatsReply": {
    "body": [
      {
        "OFPMeterConfigStats": {
          "bands": [
            {
              "OFPMeterBandDrop": {
                "burst_size": 10,
                "len": 16,
                "rate": 1000,
                "type": 1
              }
            }
          ],
          "flags": 14,
          "length": 24,
          "meter_id": 100
        }
      }
    ],
    "flags": 0,
```

(continues on next page)

(continued from previous page)

```

        "type": 10
    }
}

```

```

class os_ken.ofproto.ofproto_v1_4_parser.OFPMeterFeaturesStatsRequest(datapath,
                                                                    flags=0,
                                                                    type_=None)

```

Meter features statistics request message

The controller uses this message to query the set of features of the metering subsystem.

Attribute	Description
flags	Zero or OFPMPF_REQ_MORE

Example:

```

def send_meter_features_stats_request(self, datapath):
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPMeterFeaturesStatsRequest(datapath, 0)
    datapath.send_msg(req)

```

JSON Example:

```

{
  "OFPMeterFeaturesStatsRequest": {
    "flags": 0,
    "type": 11
  }
}

```

```

class os_ken.ofproto.ofproto_v1_4_parser.OFPMeterFeaturesStatsReply(datapath,
                                                                    type_=None,
                                                                    **kwargs)

```

Meter features statistics reply message

The switch responds with this message to a meter features statistics request.

Attribute	Description
body	List of OFPMeterFeaturesStats instance

Example:

```

@set_ev_cls(ofp_event.EventOFPMeterFeaturesStatsReply, MAIN_DISPATCHER)
def meter_features_stats_reply_handler(self, ev):
    features = []
    for stat in ev.msg.body:
        features.append('max_meter=%d band_types=0x%08x '

```

(continues on next page)

(continued from previous page)

```

        'capabilities=0x%08x max_bands=%d '
        'max_color=%d' %
        (stat.max_meter, stat.band_types,
         stat.capabilities, stat.max_bands,
         stat.max_color))
    self.logger.debug('MeterFeaturesStats: %s', features)

```

JSON Example:

```

{
  "OFPMeterFeaturesStatsReply": {
    "body": [
      {
        "OFPMeterFeaturesStats": {
          "band_types": 2147483654,
          "capabilities": 15,
          "max_bands": 255,
          "max_color": 0,
          "max_meter": 16777216
        }
      }
    ],
    "flags": 0,
    "type": 11
  }
}

```

```

class os_ken.ofproto.ofproto_v1_4_parser.OFPFlowMonitorRequest(datapath, flags=0,
                                                                monitor_id=0,
                                                                out_port=4294967295,
                                                                out_group=4294967295,
                                                                monitor_flags=0,
                                                                table_id=255,
                                                                command=0,
                                                                match=None,
                                                                type_=None)

```

Flow monitor request message

The controller uses this message to query flow monitors.

Attribute	Description
flags	Zero or OFPMPF_REQ_MORE
monitor_id	Controller-assigned ID for this monitor
out_port	Require matching entries to include this as an output port
out_group	Require matching entries to include this as an output group
monitor_flags	Bitmap of the following flags.  OFPFMF_INITIAL OFPFMF_ADD OFPFMF_REMOVED OFPFMF_MODIFY OFPFMF_INSTRUCTIONS OFPFMF_NO_ABBREV OFPFMF_ONLY_OWN
table_id	ID of table to monitor
command	One of the following values.  OFPFMC_ADD OFPFMC_MODIFY OFPFMC_DELETE
match	Instance of OFPMatch

Example:

```
def send_flow_monitor_request(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    monitor_flags = [ofp.OFPFMF_INITIAL, ofp.OFPFMF_ONLY_OWN]
    match = ofp_parser.OFPMatch(in_port=1)
    req = ofp_parser.OFPFlowMonitorRequest(datapath, 0, 10000,
                                           ofp.OFPP_ANY, ofp.OFPG_ANY,
                                           monitor_flags,
                                           ofp.OFPTT_ALL,
                                           ofp.OFPFMC_ADD, match)

    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPFlowMonitorRequest": {
    "command": 0,
    "flags": 0,
    "match": {
```

(continues on next page)



(continued from previous page)

```

        "OFPMatch": {
            "length": 14,
            "oxm_fields": [
                {
                    "OXMTlv": {
                        "field": "eth_dst",
                        "mask": null,
                        "value": "f2:0b:a4:7d:f8:ea"
                    }
                }
            ],
            "type": 1
        }
    },
    "monitor_flags": 15,
    "monitor_id": 1000000000,
    "out_group": 4294967295,
    "out_port": 22,
    "table_id": 33,
    "type": 16
}
}
}

```

```

class os_ken.ofproto.ofproto_v1_4_parser.OFPFlowMonitorReply(datapath, type_=None,
                                                             **kwargs)

```

Flow monitor reply message

The switch responds with this message to a flow monitor request.

Attribute	Description
body	List of list of the following class instance.  OFPFlowMonitorFull OFPFlowMonitorAbbrev OFPFlowMonitorPaused

Example:

```

@set_ev_cls(ofp_event.EventOFPFlowMonitorReply, MAIN_DISPATCHER)
def flow_monitor_reply_handler(self, ev):
    msg = ev.msg
    dp = msg.datapath
    ofp = dp.ofproto
    flow_updates = []

    for update in msg.body:
        update_str = 'length=%d event=%d' %

```

(continues on next page)

(continued from previous page)

```

        (update.length, update.event)
    if (update.event == ofp.OFPFME_INITIAL or
        update.event == ofp.OFPFME_ADDED or
        update.event == ofp.OFPFME_REMOVED or
        update.event == ofp.OFPFME_MODIFIED):
        update_str += 'table_id=%d reason=%d idle_timeout=%d '
                    'hard_timeout=%d priority=%d cookie=%d '
                    'match=%d instructions=%s' %
                    (update.table_id, update.reason,
                     update.idle_timeout, update.hard_timeout,
                     update.priority, update.cookie,
                     update.match, update.instructions)
    elif update.event == ofp.OFPFME_ABBREV:
        update_str += 'xid=%d' % (update.xid)
    flow_updates.append(update_str)
    self.logger.debug('FlowUpdates: %s', flow_updates)

```

JSON Example:

```

{
  "OFPPFlowMonitorReply": {
    "body": [
      {
        "OFPPFlowUpdateFull": {
          "cookie": 0,
          "event": 0,
          "hard_timeout": 700,
          "idle_timeout": 600,
          "instructions": [
            {
              "OFPInstructionActions": {
                "actions": [
                  {
                    "OFPActionOutput": {
                      "len": 16,
                      "max_len": 0,
                      "port": 4294967290,
                      "type": 0
                    }
                  }
                ],
                "len": 24,
                "type": 4
              }
            }
          ],
          "length": 64,
          "match": {
            "OFPMatch": {
              "length": 10,

```

(continues on next page)

(continued from previous page)

```

        "oxm_fields": [
            {
                "OXMTlv": {
                    "field": "eth_type",
                    "mask": null,
                    "value": 2054
                }
            }
        ],
        "type": 1
    },
    "priority": 3,
    "reason": 0,
    "table_id": 0
},
{
    "OFPPFlowUpdateAbbrev": {
        "event": 4,
        "length": 8,
        "xid": 1234
    }
},
{
    "OFPPFlowUpdatePaused": {
        "event": 5,
        "length": 8
    }
},
],
"flags": 0,
"type": 16
}
}

```

```

class os_ken.ofproto.ofproto_v1_4_parser.OFPExperimentStatsRequest(datapath,
                                                                    flags, exper-
                                                                    imenter,
                                                                    exp_type,
                                                                    data,
                                                                    type_=None)

```

Experimenter multipart request message

Attribute	Description
flags	Zero or OFPMPF_REQ_MORE
experimenter	Experimenter ID
exp_type	Experimenter defined
data	Experimenter defined additional data

JSON Example:

```
{
  "OFPExperimenterStatsRequest": {
    "data": "aG9nZWlvZ2U=",
    "exp_type": 3405678728,
    "experimenter": 3735928495,
    "flags": 0,
    "type": 65535
  }
}
```

```
class os_ken.ofproto.ofproto_v1_4_parser.OFPExperimenterStatsReply(datapath,
                                                                    type_=None,
                                                                    **kwargs)
```

Experimenter multipart reply message

Attribute	Description
body	An OFPExperimenterMultipart instance

JSON Example:

```
{
  "OFPExperimenterStatsReply": {
    "body": {
      "OFPExperimenterMultipart": {
        "data": "dGVzdGRhdGE50Tk50Tk50Q==",
        "exp_type": 3405674359,
        "experimenter": 3735928495
      }
    },
    "flags": 0,
    "type": 65535
  }
}
```

## Packet-Out Message

```
class os_ken.ofproto.ofproto_v1_4_parser.OFPPacketOut(datapath, buffer_id=None,
                                                       in_port=None, actions=None,
                                                       data=None, actions_len=None)
```

Packet-Out message

The controller uses this message to send a packet out through the switch.

Attribute	Description
buffer_id	ID assigned by datapath (OFP_NO_BUFFER if none)
in_port	Packet's input port or OFPP_CONTROLLER
actions	list of OpenFlow action class
data	Packet data of a binary type value or an instances of packet.Packet.

Example:

```
def send_packet_out(self, datapath, buffer_id, in_port):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    actions = [ofp_parser.OFPActionOutput(ofp.OFPP_FLOOD, 0)]
    req = ofp_parser.OFPPacketOut(datapath, buffer_id,
                                  in_port, actions)

    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPPacketOut": {
    "actions": [
      {
        "OFPActionOutput": {
          "len": 16,
          "max_len": 65535,
          "port": 4294967292,
          "type": 0
        }
      }
    ],
    "actions_len": 16,
    "buffer_id": 4294967295,
    "data":
    ↪ "8guk0D9w8gukffjqCABFAABU+BoAAP8Br4sKAAABCGAAAggAAgj3YAAAMdYCAAAAAACrjS0xAAAAABAREhMU
    ↪",
    "in_port": 4294967293
  }
}
```

## Barrier Message

**class** os\_ken.ofproto.ofproto\_v1\_4\_parser.OFPBarrierRequest(*datapath*)

Barrier request message

The controller sends this message to ensure message dependencies have been met or receive notifications for completed operations.

Example:

```
def send_barrier_request(self, datapath):
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPBarrierRequest(datapath)
    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPBarrierRequest": {}
}
```

**class** `os_ken.ofproto.ofproto_v1_4_parser.OFPBarrierReply(datapath)`

Barrier reply message

The switch responds with this message to a barrier request.

Example:

```
@set_ev_cls(ofp_event.EventOFPBarrierReply, MAIN_DISPATCHER)
def barrier_reply_handler(self, ev):
    self.logger.debug('OFPBarrierReply received')
```

JSON Example:

```
{
  "OFPBarrierReply": {}
}
```

## Role Request Message

**class** `os_ken.ofproto.ofproto_v1_4_parser.OFPRoleRequest(datapath, role=None, generation_id=None)`

Role request message

The controller uses this message to change its role.

Attribute	Description
role	One of the following values.  OFPCR_ROLE_NOCHANGE OFPCR_ROLE_EQUAL OFPCR_ROLE_MASTER OFPCR_ROLE_SLAVE
generation_id	Master Election Generation ID

Example:

```
def send_role_request(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPCRoleRequest(datapath, ofp.OFPCR_ROLE_EQUAL, 0)
    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPCRoleRequest": {
    "generation_id": 17294086455919964160,
    "role": 2
  }
}
```

```
class os_ken.ofproto.ofproto_v1_4_parser.OFPCRoleReply(datapath, role=None,
                                                       generation_id=None)
```

Role reply message

The switch responds with this message to a role request.

Attribute	Description
role	One of the following values.  OFPCR_ROLE_NOCHANGE OFPCR_ROLE_EQUAL OFPCR_ROLE_MASTER OFPCR_ROLE_SLAVE
generation_id	Master Election Generation ID

Example:

```
@set_ev_cls(ofp_event.EventOFPCRoleReply, MAIN_DISPATCHER)
def role_reply_handler(self, ev):
    msg = ev.msg
    dp = msg.datapath
    ofp = dp.ofproto

    if msg.role == ofp.OFPCR_ROLE_NOCHANGE:
        role = 'NOCHANGE'
    elif msg.role == ofp.OFPCR_ROLE_EQUAL:
        role = 'EQUAL'
    elif msg.role == ofp.OFPCR_ROLE_MASTER:
        role = 'MASTER'
    elif msg.role == ofp.OFPCR_ROLE_SLAVE:
        role = 'SLAVE'
    else:
```

(continues on next page)

(continued from previous page)

```

    role = 'unknown'

    self.logger.debug('OFPRoleReply received: '
                      'role=%s generation_id=%d',
                      role, msg.generation_id)

```

JSON Example:

```

{
  "OFPRoleReply": {
    "generation_id": 17294086455919964160,
    "role": 3
  }
}

```

## Bundle Messages

```

class os_ken.ofproto.ofproto_v1_4_parser.OFPBundleCtrlMsg(datapath,
                                                           bundle_id=None,
                                                           type_=None, flags=None,
                                                           properties=None)

```

Bundle control message

The controller uses this message to create, destroy and commit bundles

Attribute	Description
bundle_id	Id of the bundle
type	One of the following values.  OFPBCT_OPEN_REQUEST OFPBCT_OPEN_REPLY OFPBCT_CLOSE_REQUEST OFPBCT_CLOSE_REPLY OFPBCT_COMMIT_REQUEST OFPBCT_COMMIT_REPLY OFPBCT_DISCARD_REQUEST OFPBCT_DISCARD_REPLY
flags	Bitmap of the following flags.  OFPBF_ATOMIC OFPBF_ORDERED
properties	List of OFPBundleProp subclass instance

Example:



```
def send_bundle_control(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPBundleCtrlMsg(datapath, 7,
                                      ofp.OFPBCT_OPEN_REQUEST,
                                      ofp.OFPBF_ATOMIC, [])

    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPBundleCtrlMsg": {
    "bundle_id": 1234,
    "flags": 1,
    "properties": [
      {
        "OFPBundlePropExperimenter": {
          "data": [],
          "exp_type": 0,
          "experimenter": 101,
          "length": 12,
          "type": 65535
        }
      },
      {
        "OFPBundlePropExperimenter": {
          "data": [
            1
          ],
          "exp_type": 1,
          "experimenter": 101,
          "length": 16,
          "type": 65535
        }
      },
      {
        "OFPBundlePropExperimenter": {
          "data": [
            1,
            2
          ],
          "exp_type": 2,
          "experimenter": 101,
          "length": 20,
          "type": 65535
        }
      }
    ],
    "type": 0
  }
}
```

(continues on next page)

(continued from previous page)

```
}
}
```

**class** `os_ken.ofproto.ofproto_v1_4_parser.OFPBundleAddMsg`(*datapath, bundle\_id, flags, message, properties*)

Bundle add message

The controller uses this message to add a message to a bundle

Attribute	Description
<code>bundle_id</code>	Id of the bundle
<code>flags</code>	Bitmap of the following flags.  OFPBF_ATOMIC OFPBF_ORDERED
<code>message</code>	MsgBase subclass instance
<code>properties</code>	List of OFPBundleProp subclass instance

Example:

```
def send_bundle_add_message(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    msg = ofp_parser.OFPRoleRequest(datapath, ofp.OFPCR_ROLE_EQUAL, 0)

    req = ofp_parser.OFPBundleAddMsg(datapath, 7, ofp.OFPBF_ATOMIC,
                                     msg, [])

    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPBundleAddMsg": {
    "bundle_id": 1234,
    "flags": 1,
    "message": {
      "OFPEchoRequest": {
        "data": null
      }
    },
    "properties": [
      {
        "OFPBundlePropExperimenter": {
          "data": [],
          "exp_type": 0,
          "experimenter": 101,
          "length": 12,

```

(continues on next page)

(continued from previous page)

```

        "type": 65535
    },
    {
        "OFPBundlePropExperimenter": {
            "data": [
                1
            ],
            "exp_type": 1,
            "experimenter": 101,
            "length": 16,
            "type": 65535
        }
    },
    {
        "OFPBundlePropExperimenter": {
            "data": [
                1,
                2
            ],
            "exp_type": 2,
            "experimenter": 101,
            "length": 20,
            "type": 65535
        }
    }
]
}
}

```

## Set Asynchronous Configuration Message

**class** `os_ken.ofproto.ofproto_v1_4_parser.OFPSetAsync(datapath, properties=None)`

Set asynchronous configuration message

The controller sends this message to set the asynchronous messages that it wants to receive on a given OpenFlow channel.

Attribute	Description
<code>properties</code>	List of <code>OFPAsyncConfigProp</code> subclass instances

Example:

```

def send_set_async(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    properties = [

```

(continues on next page)

(continued from previous page)

```

    ofp_parser.OFPAsyncConfigPropReasons(
        ofp.OFPACPT_PACKET_IN_SLAVE, 8,
        (1 << ofp.OFPR_APPLY_ACTION
         | 1 << ofp.OFPR_INVALID_TTL))])
req = ofp_parser.OFPSetAsync(datapath, properties)
datapath.send_msg(req)

```

JSON Example:

```

{
  "OFPSetAsync": {
    "properties": [
      {
        "OFPAsyncConfigPropReasons": {
          "length": 8,
          "mask": 3,
          "type": 0
        }
      },
      {
        "OFPAsyncConfigPropReasons": {
          "length": 8,
          "mask": 3,
          "type": 1
        }
      },
      {
        "OFPAsyncConfigPropReasons": {
          "length": 8,
          "mask": 3,
          "type": 2
        }
      },
      {
        "OFPAsyncConfigPropReasons": {
          "length": 8,
          "mask": 3,
          "type": 3
        }
      },
      {
        "OFPAsyncConfigPropReasons": {
          "length": 8,
          "mask": 3,
          "type": 4
        }
      },
      {
        "OFPAsyncConfigPropReasons": {
          "length": 8,

```

(continues on next page)

(continued from previous page)

```
        "mask": 3,
        "type": 5
    }
},
{
    "OFPAsyncConfigPropReasons": {
        "length": 8,
        "mask": 3,
        "type": 6
    }
},
{
    "OFPAsyncConfigPropReasons": {
        "length": 8,
        "mask": 3,
        "type": 7
    }
},
{
    "OFPAsyncConfigPropReasons": {
        "length": 8,
        "mask": 24,
        "type": 8
    }
},
{
    "OFPAsyncConfigPropReasons": {
        "length": 8,
        "mask": 24,
        "type": 9
    }
},
{
    "OFPAsyncConfigPropReasons": {
        "length": 8,
        "mask": 3,
        "type": 10
    }
},
{
    "OFPAsyncConfigPropReasons": {
        "length": 8,
        "mask": 3,
        "type": 11
    }
},
{
    "OFPAsyncConfigPropExperimenter": {
        "data": [],
```

(continues on next page)

(continued from previous page)

```

        "exp_type": 0,
        "experimenter": 101,
        "length": 12,
        "type": 65534
    },
    {
        "OFPAsyncConfigPropExperimenter": {
            "data": [
                1
            ],
            "exp_type": 1,
            "experimenter": 101,
            "length": 16,
            "type": 65535
        }
    },
    {
        "OFPAsyncConfigPropExperimenter": {
            "data": [
                1,
                2
            ],
            "exp_type": 2,
            "experimenter": 101,
            "length": 20,
            "type": 65535
        }
    }
]
}

```

**class** `os_ken.ofproto.ofproto_v1_4_parser.OFPGetAsyncRequest` (*datapath*)

Get asynchronous configuration request message

The controller uses this message to query the asynchronous message.

Example:

```

def send_get_async_request(self, datapath):
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPGetAsyncRequest(datapath)
    datapath.send_msg(req)

```

JSON Example:

```

{
  "OFPGetAsyncRequest": {}
}

```

```
class os_ken.ofproto.ofproto_v1_4_parser.OFPGetAsyncReply(datapath,
                                                         properties=None)
```

Get asynchronous configuration reply message

The switch responds with this message to a get asynchronous configuration request.

Attribute	Description
properties	List of OFPAsyncConfigProp subclass instances

Example:

```
@set_ev_cls(ofp_event.EventOFPGetAsyncReply, MAIN_DISPATCHER)
def get_async_reply_handler(self, ev):
    msg = ev.msg

    self.logger.debug('OFPGetAsyncReply received: '
                      'properties=%s', repr(msg.properties))
```

JSON Example:

```
{
  "OFPGetAsyncReply": {
    "properties": [
      {
        "OFPAsyncConfigPropReasons": {
          "length": 8,
          "mask": 3,
          "type": 0
        }
      },
      {
        "OFPAsyncConfigPropReasons": {
          "length": 8,
          "mask": 3,
          "type": 1
        }
      },
      {
        "OFPAsyncConfigPropReasons": {
          "length": 8,
          "mask": 3,
          "type": 2
        }
      },
      {
        "OFPAsyncConfigPropReasons": {
          "length": 8,
          "mask": 3,
          "type": 3
        }
      }
    ]
  }
}
```

(continues on next page)

(continued from previous page)

```
    },
    {
      "OFPAsyncConfigPropReasons": {
        "length": 8,
        "mask": 3,
        "type": 4
      }
    },
    {
      "OFPAsyncConfigPropReasons": {
        "length": 8,
        "mask": 3,
        "type": 5
      }
    },
    {
      "OFPAsyncConfigPropReasons": {
        "length": 8,
        "mask": 3,
        "type": 6
      }
    },
    {
      "OFPAsyncConfigPropReasons": {
        "length": 8,
        "mask": 3,
        "type": 7
      }
    },
    {
      "OFPAsyncConfigPropReasons": {
        "length": 8,
        "mask": 24,
        "type": 8
      }
    },
    {
      "OFPAsyncConfigPropReasons": {
        "length": 8,
        "mask": 24,
        "type": 9
      }
    },
    {
      "OFPAsyncConfigPropReasons": {
        "length": 8,
        "mask": 3,
        "type": 10
      }
    }
  }
```

(continues on next page)



(continued from previous page)

```
    },
    {
      "OFPAsyncConfigPropReasons": {
        "length": 8,
        "mask": 3,
        "type": 11
      }
    },
    {
      "OFPAsyncConfigPropExperimenter": {
        "data": [],
        "exp_type": 0,
        "experimenter": 101,
        "length": 12,
        "type": 65534
      }
    },
    {
      "OFPAsyncConfigPropExperimenter": {
        "data": [
          1
        ],
        "exp_type": 1,
        "experimenter": 101,
        "length": 16,
        "type": 65535
      }
    },
    {
      "OFPAsyncConfigPropExperimenter": {
        "data": [
          1,
          2
        ],
        "exp_type": 2,
        "experimenter": 101,
        "length": 20,
        "type": 65535
      }
    }
  ]
}
```

## Asynchronous Messages

### Packet-In Message

```
class os_ken.ofproto.ofproto_v1_4_parser.OFPPacketIn(datapath, buffer_id=None,
                                                    total_len=None, reason=None,
                                                    table_id=None, cookie=None,
                                                    match=None, data=None)
```

Packet-In message

The switch sends the packet that received to the controller by this message.

Attribute	Description
buffer_id	ID assigned by datapath
total_len	Full length of frame
reason	Reason packet is being sent.  OFPR_TABLE_MISS OFPR_APPLY_ACTION OFPR_INVALID_TTL OFPR_ACTION_SET OFPR_GROUP OFPR_PACKET_OUT
table_id	ID of the table that was looked up
cookie	Cookie of the flow entry that was looked up
match	Instance of OFPMatch
data	Ethernet frame

Example:

```
@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def packet_in_handler(self, ev):
    msg = ev.msg
    dp = msg.datapath
    ofp = dp.ofproto

    if msg.reason == ofp.TABLE_MISS:
        reason = 'TABLE MISS'
    elif msg.reason == ofp.OFPR_APPLY_ACTION:
        reason = 'APPLY ACTION'
    elif msg.reason == ofp.OFPR_INVALID_TTL:
        reason = 'INVALID TTL'
    elif msg.reason == ofp.OFPR_ACTION_SET:
        reason = 'ACTION SET'
    elif msg.reason == ofp.OFPR_GROUP:
        reason = 'GROUP'
    elif msg.reason == ofp.OFPR_PACKET_OUT:
        reason = 'PACKET OUT'
```

(continues on next page)

(continued from previous page)

```

else:
    reason = 'unknown'

self.logger.debug('OFPPacketIn received: '
                  'buffer_id=%x total_len=%d reason=%s '
                  'table_id=%d cookie=%d match=%s data=%s',
                  msg.buffer_id, msg.total_len, reason,
                  msg.table_id, msg.cookie, msg.match,
                  utils.hex_array(msg.data))

```

JSON Example:

```

{
  "OFPPacketIn": {
    "buffer_id": 2,
    "cookie": 2836868884868096,
    "data": "/////////8gukffjqCAYAAQgABgQAAfILpH346goAAAEAAAAAAAAAKAAAD",
    "match": {
      "OFPMatch": {
        "length": 80,
        "oxm_fields": [
          {
            "OXMTlv": {
              "field": "in_port",
              "mask": null,
              "value": 6
            }
          },
          {
            "OXMTlv": {
              "field": "eth_type",
              "mask": null,
              "value": 2054
            }
          },
          {
            "OXMTlv": {
              "field": "eth_dst",
              "mask": null,
              "value": "ff:ff:ff:ff:ff:ff"
            }
          },
          {
            "OXMTlv": {
              "field": "eth_src",
              "mask": null,
              "value": "f2:0b:a4:7d:f8:ea"
            }
          }
        ]
      }
    }
  }
}

```

(continues on next page)

(continued from previous page)

```
        "OXMTlv": {
            "field": "arp_op",
            "mask": null,
            "value": 1
        }
    },
    {
        "OXMTlv": {
            "field": "arp_spa",
            "mask": null,
            "value": "10.0.0.1"
        }
    },
    {
        "OXMTlv": {
            "field": "arp_tpa",
            "mask": null,
            "value": "10.0.0.3"
        }
    },
    {
        "OXMTlv": {
            "field": "arp_sha",
            "mask": null,
            "value": "f2:0b:a4:7d:f8:ea"
        }
    },
    {
        "OXMTlv": {
            "field": "arp_tha",
            "mask": null,
            "value": "00:00:00:00:00:00"
        }
    }
],
    "type": 1
}
},
"reason": 3,
"table_id": 1,
"total_len": 42
}
}
```

```
{
    "OFPPacketIn": {
        "buffer_id": 4026531840,
        "cookie": 283686884868096,
        "data": "",
```

(continues on next page)

(continued from previous page)

```
"match": {
  "OFPMatch": {
    "length": 329,
    "oxm_fields": [
      {
        "OXMTlv": {
          "field": "in_port",
          "mask": null,
          "value": 84281096
        }
      },
      {
        "OXMTlv": {
          "field": "in_phy_port",
          "mask": null,
          "value": 16909060
        }
      },
      {
        "OXMTlv": {
          "field": "metadata",
          "mask": null,
          "value": 283686952306183
        }
      },
      {
        "OXMTlv": {
          "field": "eth_type",
          "mask": null,
          "value": 2054
        }
      },
      {
        "OXMTlv": {
          "field": "eth_dst",
          "mask": null,
          "value": "ff:ff:ff:ff:ff:ff"
        }
      },
      {
        "OXMTlv": {
          "field": "eth_src",
          "mask": null,
          "value": "f2:0b:a4:7d:f8:ea"
        }
      },
      {
        "OXMTlv": {
          "field": "vlan_vid",
```

(continues on next page)

(continued from previous page)

```
        "mask": null,
        "value": 999
    },
    {
        "OXMTlv": {
            "field": "ip_dscp",
            "mask": null,
            "value": 9
        }
    },
    {
        "OXMTlv": {
            "field": "ip_ecn",
            "mask": null,
            "value": 3
        }
    },
    {
        "OXMTlv": {
            "field": "ip_proto",
            "mask": null,
            "value": 99
        }
    },
    {
        "OXMTlv": {
            "field": "ipv4_src",
            "mask": null,
            "value": "1.2.3.4"
        }
    },
    {
        "OXMTlv": {
            "field": "ipv4_dst",
            "mask": null,
            "value": "1.2.3.4"
        }
    },
    {
        "OXMTlv": {
            "field": "tcp_src",
            "mask": null,
            "value": 8080
        }
    },
    {
        "OXMTlv": {
            "field": "tcp_dst",
```

(continues on next page)

(continued from previous page)

```
        "mask": null,
        "value": 18080
    }
},
{
    "OXMTlv": {
        "field": "udp_src",
        "mask": null,
        "value": 28080
    }
},
{
    "OXMTlv": {
        "field": "udp_dst",
        "mask": null,
        "value": 55936
    }
},
{
    "OXMTlv": {
        "field": "sctp_src",
        "mask": null,
        "value": 48080
    }
},
{
    "OXMTlv": {
        "field": "sctp_dst",
        "mask": null,
        "value": 59328
    }
},
{
    "OXMTlv": {
        "field": "icmpv4_type",
        "mask": null,
        "value": 100
    }
},
{
    "OXMTlv": {
        "field": "icmpv4_code",
        "mask": null,
        "value": 101
    }
},
{
    "OXMTlv": {
        "field": "arp_op",
```

(continues on next page)

(continued from previous page)

```
        "mask": null,
        "value": 1
    }
},
{
    "OXMTlv": {
        "field": "arp_spa",
        "mask": null,
        "value": "10.0.0.1"
    }
},
{
    "OXMTlv": {
        "field": "arp_tpa",
        "mask": null,
        "value": "10.0.0.3"
    }
},
{
    "OXMTlv": {
        "field": "arp_sha",
        "mask": null,
        "value": "f2:0b:a4:7d:f8:ea"
    }
},
{
    "OXMTlv": {
        "field": "arp_tha",
        "mask": null,
        "value": "00:00:00:00:00:00"
    }
},
{
    "OXMTlv": {
        "field": "ipv6_src",
        "mask": null,
        "value": "fe80::f00b:a4ff:fe48:28a5"
    }
},
{
    "OXMTlv": {
        "field": "ipv6_dst",
        "mask": null,
        "value": "fe80::f00b:a4ff:fe05:b7dc"
    }
},
{
    "OXMTlv": {
        "field": "ipv6_flabel",
```

(continues on next page)



(continued from previous page)

```
        "mask": null,
        "value": 541473
    },
    {
        "OXMTlv": {
            "field": "icmpv6_type",
            "mask": null,
            "value": 200
        }
    },
    {
        "OXMTlv": {
            "field": "icmpv6_code",
            "mask": null,
            "value": 201
        }
    },
    {
        "OXMTlv": {
            "field": "ipv6_nd_target",
            "mask": null,
            "value": "fe80::a60:6eff:fe7f:74e7"
        }
    },
    {
        "OXMTlv": {
            "field": "ipv6_nd_sll",
            "mask": null,
            "value": "00:00:00:00:02:9a"
        }
    },
    {
        "OXMTlv": {
            "field": "ipv6_nd_tll",
            "mask": null,
            "value": "00:00:00:00:02:2b"
        }
    },
    {
        "OXMTlv": {
            "field": "mpls_label",
            "mask": null,
            "value": 624485
        }
    },
    {
        "OXMTlv": {
            "field": "mpls_tc",
```

(continues on next page)

(continued from previous page)

```
        "mask": null,
        "value": 5
    }
},
{
    "OXMTlv": {
        "field": "mpls_bos",
        "mask": null,
        "value": 1
    }
},
{
    "OXMTlv": {
        "field": "pbb_isid",
        "mask": null,
        "value": 11259375
    }
},
{
    "OXMTlv": {
        "field": "tunnel_id",
        "mask": null,
        "value": 651061555542690057
    }
},
{
    "OXMTlv": {
        "field": "ipv6_exthdr",
        "mask": null,
        "value": 500
    }
},
{
    "OXMTlv": {
        "field": "pbb_uca",
        "mask": null,
        "value": 1
    }
}
],
    "type": 1
}
},
"reason": 0,
"table_id": 200,
"total_len": 0
}
}
```

## Flow Removed Message

```
class os_ken.ofproto.ofproto_v1_4_parser.OFPFlowRemoved(datapath, cookie=None,
                                                         priority=None,
                                                         reason=None,
                                                         table_id=None,
                                                         duration_sec=None,
                                                         duration_nsec=None,
                                                         idle_timeout=None,
                                                         hard_timeout=None,
                                                         packet_count=None,
                                                         byte_count=None,
                                                         match=None)
```

Flow removed message

When flow entries time out or are deleted, the switch notifies controller with this message.

Attribute	Description
cookie	Opaque controller-issued identifier
priority	Priority level of flow entry
reason	One of the following values.  OFPRR_IDLE_TIMEOUT OFPRR_HARD_TIMEOUT OFPRR_DELETE OFPRR_GROUP_DELETE OFPRR_METER_DELETE OFPRR_EVICTION
table_id	ID of the table
duration_sec	Time flow was alive in seconds
duration_nsec	Time flow was alive in nanoseconds beyond duration_sec
idle_timeout	Idle timeout from original flow mod
hard_timeout	Hard timeout from original flow mod
packet_count	Number of packets that was associated with the flow
byte_count	Number of bytes that was associated with the flow
match	Instance of OFPMatch

Example:

```
@set_ev_cls(ofp_event.EventOFPFlowRemoved, MAIN_DISPATCHER)
def flow_removed_handler(self, ev):
    msg = ev.msg
    dp = msg.datapath
    ofp = dp.ofproto
```

(continues on next page)

(continued from previous page)

```

if msg.reason == ofp.OFPRR_IDLE_TIMEOUT:
    reason = 'IDLE TIMEOUT'
elif msg.reason == ofp.OFPRR_HARD_TIMEOUT:
    reason = 'HARD TIMEOUT'
elif msg.reason == ofp.OFPRR_DELETE:
    reason = 'DELETE'
elif msg.reason == ofp.OFPRR_GROUP_DELETE:
    reason = 'GROUP DELETE'
else:
    reason = 'unknown'

self.logger.debug('OFPPFlowRemoved received: '
                  'cookie=%d priority=%d reason=%s table_id=%d '
                  'duration_sec=%d duration_nsec=%d '
                  'idle_timeout=%d hard_timeout=%d '
                  'packet_count=%d byte_count=%d match.fields=%s',
                  msg.cookie, msg.priority, reason, msg.table_id,
                  msg.duration_sec, msg.duration_nsec,
                  msg.idle_timeout, msg.hard_timeout,
                  msg.packet_count, msg.byte_count, msg.match)

```

JSON Example:

```

{
  "OFPPFlowRemoved": {
    "byte_count": 86,
    "cookie": 0,
    "duration_nsec": 48825000,
    "duration_sec": 3,
    "hard_timeout": 0,
    "idle_timeout": 3,
    "match": {
      "OFPMatch": {
        "length": 14,
        "oxm_fields": [
          {
            "OXMTlv": {
              "field": "eth_dst",
              "mask": null,
              "value": "f2:0b:a4:7d:f8:ea"
            }
          }
        ],
        "type": 1
      }
    },
    "packet_count": 1,
    "priority": 65535,
    "reason": 0,
    "table_id": 0
  }
}

```

(continues on next page)

(continued from previous page)

```

    }
}

```

## Port Status Message

```
class os_ken.ofproto.ofproto_v1_4_parser.OFPPortStatus(datapath, reason=None,
                                                       desc=None)
```

Port status message

The switch notifies controller of change of ports.

Attribute	Description
reason	One of the following values.  OFPPR_ADD OFPPR_DELETE OFPPR_MODIFY
desc	instance of OFPPort

Example:

```

@set_ev_cls(ofp_event.EventOFPPortStatus, MAIN_DISPATCHER)
def port_status_handler(self, ev):
    msg = ev.msg
    dp = msg.datapath
    ofp = dp.ofproto

    if msg.reason == ofp.OFPPR_ADD:
        reason = 'ADD'
    elif msg.reason == ofp.OFPPR_DELETE:
        reason = 'DELETE'
    elif msg.reason == ofp.OFPPR_MODIFY:
        reason = 'MODIFY'
    else:
        reason = 'unknown'

    self.logger.debug('OFPPortStatus received: reason=%s desc=%s',
                      reason, msg.desc)

```

JSON Example:

```

{
  "OFPPortStatus": {
    "desc": {
      "OFPPort": {
        "config": 0,
        "hw_addr": "f2:0b:a4:d0:3f:70",

```

(continues on next page)

(continued from previous page)

```

"length": 168,
"name": "\u79c1\u306e\u30dd\u30fc\u30c8",
"port_no": 7,
"properties": [
  {
    "OFPPortDescPropEthernet": {
      "advertised": 10240,
      "curr": 10248,
      "curr_speed": 5000,
      "length": 32,
      "max_speed": 5000,
      "peer": 10248,
      "supported": 10248,
      "type": 0
    }
  },
  {
    "OFPPortDescPropOptical": {
      "length": 40,
      "rx_grid_freq_lmda": 1500,
      "rx_max_freq_lmda": 2000,
      "rx_min_freq_lmda": 1000,
      "supported": 1,
      "tx_grid_freq_lmda": 1500,
      "tx_max_freq_lmda": 2000,
      "tx_min_freq_lmda": 1000,
      "tx_pwr_max": 2000,
      "tx_pwr_min": 1000,
      "type": 1
    }
  },
  {
    "OFPPortDescPropExperimenter": {
      "data": [],
      "exp_type": 0,
      "experimenter": 101,
      "length": 12,
      "type": 65535
    }
  },
  {
    "OFPPortDescPropExperimenter": {
      "data": [
        1
      ],
      "exp_type": 1,
      "experimenter": 101,
      "length": 16,
      "type": 65535
    }
  }
]

```

(continues on next page)

(continued from previous page)

```

    }
  },
  {
    "OFPPortDescPropExperimenter": {
      "data": [
        1,
        2
      ],
      "exp_type": 2,
      "experimenter": 101,
      "length": 20,
      "type": 65535
    }
  }
],
"state": 4
},
"reason": 0
}
}
}

```

### Controller Role Status Message

```

class os_ken.ofproto.ofproto_v1_4_parser.OFPCRStatus(datapath, role=None,
                                                    reason=None,
                                                    generation_id=None,
                                                    properties=None)

```

Role status message

The switch notifies controller of change of role.

Attribute	Description
role	One of the following values.  OFPCR_ROLE_NOCHANGE OFPCR_ROLE_EQUAL OFPCR_ROLE_MASTER
reason	One of the following values.  OFPCRR_MASTER_REQUEST OFPCRR_CONFIG OFPCRR_EXPERIMENTER
generation_id	Master Election Generation ID
properties	List of OFPCRLeProp subclass instance

Example:

```
@set_ev_cls(ofp_event.EventOFPCRoleStatus, MAIN_DISPATCHER)
def role_status_handler(self, ev):
    msg = ev.msg
    dp = msg.datapath
    ofp = dp.ofproto

    if msg.role == ofp.OFPCR_ROLE_NOCHANGE:
        role = 'ROLE NOCHANGE'
    elif msg.role == ofp.OFPCR_ROLE_EQUAL:
        role = 'ROLE EQUAL'
    elif msg.role == ofp.OFPCR_ROLE_MASTER:
        role = 'ROLE MASTER'
    else:
        role = 'unknown'

    if msg.reason == ofp.OFPCRR_MASTER_REQUEST:
        reason = 'MASTER REQUEST'
    elif msg.reason == ofp.OFPCRR_CONFIG:
        reason = 'CONFIG'
    elif msg.reason == ofp.OFPCRR_EXPERIMENTER:
        reason = 'EXPERIMENTER'
    else:
        reason = 'unknown'

    self.logger.debug('OFPCRoleStatus received: role=%s reason=%s '
                      'generation_id=%d properties=%s', role, reason,
                      msg.generation_id, repr(msg.properties))
```

JSON Example:

```
{
  "OFPCRoleStatus": {
    "generation_id": 7,
    "properties": [
      {
        "OFPCRolePropExperimenter": {
          "data": [],
          "exp_type": 0,
          "experimenter": 101,
          "length": 12,
          "type": 65535
        }
      },
      {
        "OFPCRolePropExperimenter": {
          "data": [
            1
          ],
          "exp_type": 1,

```

(continues on next page)



(continued from previous page)

```

        "experimenter": 101,
        "length": 16,
        "type": 65535
    }
},
{
    "OFPRolePropExperimenter": {
        "data": [
            1,
            2
        ],
        "exp_type": 2,
        "experimenter": 101,
        "length": 20,
        "type": 65535
    }
}
],
"reason": 0,
"role": 2
}
}

```

## Table Status Message

**class** `os_ken.ofproto.ofproto_v1_4_parser.OFPTableStatus`(*datapath*, *reason=None*, *table=None*)

Table status message

The switch notifies controller of change of table status.

Attribute	Description
reason	One of the following values.  OFPTR_VACANCY_DOWN OFPTR_VACANCY_UP
table	OFPTableDesc instance

Example:

```

@set_ev_cls(ofp_event.EventOFPTableStatus, MAIN_DISPATCHER)
def table(self, ev):
    msg = ev.msg
    dp = msg.datapath
    ofp = dp.ofproto

    if msg.reason == ofp.OFPTR_VACANCY_DOWN:

```

(continues on next page)

(continued from previous page)

```

    reason = 'VACANCY_DOWN'
elif msg.reason == ofp.OFPTR_VACANCY_UP:
    reason = 'VACANCY_UP'
else:
    reason = 'unknown'

self.logger.debug('OFPTableStatus received: reason=%s '
                  'table_id=%d config=0x%08x properties=%s',
                  reason, msg.table.table_id, msg.table.config,
                  repr(msg.table.properties))

```

JSON Example:

```

{
  "OFPTableStatus": {
    "reason": 3,
    "table": {
      "OFPTableDesc": {
        "config": 0,
        "length": 80,
        "properties": [
          {
            "OFPTableModPropEviction": {
              "flags": 0,
              "length": 8,
              "type": 2
            }
          },
          {
            "OFPTableModPropVacancy": {
              "length": 8,
              "type": 3,
              "vacancy": 0,
              "vacancy_down": 0,
              "vacancy_up": 0
            }
          },
          {
            "OFPTableModPropExperimenter": {
              "data": [],
              "exp_type": 0,
              "experimenter": 101,
              "length": 12,
              "type": 65535
            }
          },
          {
            "OFPTableModPropExperimenter": {
              "data": [
                1

```

(continues on next page)



(continued from previous page)

```

        msg.request.command, msg.request.type,
        msg.request.group_id, msg.request.buckets)
elif msg.request.msg_type == ofp.OFPT_METER_MOD:
    self.logger.debug(
        'OFPPRequestForward received: request=OFPMeterMod('
        'command=%d, flags=%d, meter_id=%d, bands=%s)',
        msg.request.command, msg.request.flags,
        msg.request.meter_id, msg.request.bands)
else:
    self.logger.debug(
        'OFPPRequestForward received: request=Unknown')

```

JSON Example:

```

{
  "OFPPRequestForward": {
    "request": {
      "OFPPGroupMod": {
        "buckets": [
          {
            "OFPPBucket": {
              "actions": [
                {
                  "OFPPActionOutput": {
                    "len": 16,
                    "max_len": 65535,
                    "port": 2,
                    "type": 0
                  }
                }
              ],
              "len": 32,
              "watch_group": 1,
              "watch_port": 1,
              "weight": 1
            }
          ]
        },
        "command": 0,
        "group_id": 1,
        "type": 0
      }
    }
  }
}

```

## Symmetric Messages

### Hello

`class os_ken.ofproto.ofproto_v1_4_parser.OFPHello(datapath, elements=None)`

Hello message

When connection is started, the hello message is exchanged between a switch and a controller.

This message is handled by the OSKen framework, so the OSKen application do not need to process this typically.

Attribute	Description
elements	list of OFPHelloElemVersionBitmap instance

JSON Example:

```
{
  "OFPHello": {
    "elements": [
      {
        "OFPHelloElemVersionBitmap": {
          "length": 8,
          "type": 1,
          "versions": [
            1,
            2,
            3,
            9,
            10,
            30
          ]
        }
      }
    ]
  }
}
```

`class os_ken.ofproto.ofproto_v1_4_parser.OFPHelloElemVersionBitmap(versions, type_=None, length=None)`

Version bitmap Hello Element

Attribute	Description
versions	list of versions of OpenFlow protocol a device supports

## Echo Request

**class** `os_ken.ofproto.ofproto_v1_4_parser.OFPEchoRequest` (*datapath*, *data=None*)

Echo request message

This message is handled by the OSKen framework, so the OSKen application do not need to process this typically.

Attribute	Description
<code>data</code>	An arbitrary length data

Example:

```
def send_echo_request(self, datapath, data):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPEchoRequest(datapath, data)
    datapath.send_msg(req)

@set_ev_cls(ofp_event.EventOFPEchoRequest,
            [HANDSHAKE_DISPATCHER, CONFIG_DISPATCHER, MAIN_DISPATCHER])
def echo_request_handler(self, ev):
    self.logger.debug('OFPEchoRequest received: data=%s',
                      utils.hex_array(ev.msg.data))
```

JSON Example:

```
{
  "OFPEchoRequest": {
    "data": "aG9nZQ=="
  }
}
```

## Echo Reply

**class** `os_ken.ofproto.ofproto_v1_4_parser.OFPEchoReply` (*datapath*, *data=None*)

Echo reply message

This message is handled by the OSKen framework, so the OSKen application do not need to process this typically.

Attribute	Description
<code>data</code>	An arbitrary length data

Example:

```
def send_echo_reply(self, datapath, data):
    ofp_parser = datapath.ofproto_parser
```

(continues on next page)

(continued from previous page)

```

reply = ofp_parser.OFPEchoReply(datapath, data)
datapath.send_msg(reply)

@set_ev_cls(ofp_event.EventOFPEchoReply,
            [HANDSHAKE_DISPATCHER, CONFIG_DISPATCHER, MAIN_DISPATCHER])
def echo_reply_handler(self, ev):
    self.logger.debug('OFPEchoReply received: data=%s',
                      utils.hex_array(ev.msg.data))

```

JSON Example:

```

{
  "OFPEchoReply": {
    "data": "aG9nZQ=="
  }
}

```

## Error Message

```

class os_ken.ofproto.ofproto_v1_4_parser.OFPErrormsg(datapath, type_=None,
                                                    code=None, data=None,
                                                    **kwargs)

```

Error message

The switch notifies controller of problems by this message.

Attribute	Description
type	High level type of error
code	Details depending on the type
data	Variable length data depending on the type and code

type attribute corresponds to type\_ parameter of \_\_init\_\_.

Types and codes are defined in os\_ken.ofproto.ofproto.

Type	Code
OFPET_HELLO_FAILED	OFPHFC_*
OFPET_BAD_REQUEST	OFPBRC_*
OFPET_BAD_ACTION	OFPBAC_*
OFPET_BAD_INSTRUCTION	OFPBIC_*
OFPET_BAD_MATCH	OFPBMC_*
OFPET_FLOW_MOD_FAILED	OFPFMFC_*
OFPET_GROUP_MOD_FAILED	OFPGMFC_*
OFPET_PORT_MOD_FAILED	OFPPMFC_*
OFPET_TABLE_MOD_FAILED	OFPTMFC_*
OFPET_QUEUE_OP_FAILED	OFPQOFC_*
OFPET_SWITCH_CONFIG_FAILED	OFPSCFC_*
OFPET_ROLE_REQUEST_FAILED	OFPRRFC_*
OFPET_METER_MOD_FAILED	OFPMMFC_*
OFPET_TABLE_FEATURES_FAILED	OFPTFFC_*
OFPET_EXPERIMENTER	N/A

If `type == OFPET_EXPERIMENTER`, this message has also the following attributes.

Attribute	Description
<code>exp_type</code>	Experimenter defined type
<code>experimenter</code>	Experimenter ID

Example:

```
@set_ev_cls(ofp_event.EventOFPErrormsg,
            [HANDSHAKE_DISPATCHER, CONFIG_DISPATCHER, MAIN_DISPATCHER])
def error_msg_handler(self, ev):
    msg = ev.msg

    self.logger.debug('OFPErrormsg received: type=0x%02x code=0x%02x '
                      'message=%s',
                      msg.type, msg.code, utils.hex_array(msg.data))
```

JSON Example:

```
{
  "OFPErrormsg": {
    "code": 11,
    "data": "ZnVnYWZ1Z2E=",
    "type": 2
  }
}
```



## Experimenter

```
class os_ken.ofproto.ofproto_v1_4_parser.OFPEExperimenter(datapath,
                                                         experimenter=None,
                                                         exp_type=None,
                                                         data=None)
```

Experimenter extension message

Attribute	Description
experimenter	Experimenter ID
exp_type	Experimenter defined
data	Experimenter defined arbitrary additional data

JSON Example:

```
{
  "OFPEExperimenter": {
    "data": "bmF6bw==",
    "exp_type": 123456789,
    "experimenter": 98765432
  }
}
```

## Port Structures

```
class os_ken.ofproto.ofproto_v1_4_parser.OFPPort(port_no=None, length=None,
                                                  hw_addr=None, name=None,
                                                  config=None, state=None,
                                                  properties=None)
```

Description of a port

Attribute	Description
port_no	Port number and it uniquely identifies a port within a switch.
length	Length of ofp_port (excluding padding).
hw_addr	MAC address for the port.
name	Null-terminated string containing a human-readable name for the interface.
config	Bitmap of port configuration flags.  OFPPC_PORT_DOWN OFPPC_NO_RECV OFPPC_NO_FWD OFPPC_NO_PACKET_IN
state	Bitmap of port state flags.  OFPPS_LINK_DOWN OFPPS_BLOCKED OFPPS_LIVE
properties	List of OFPPortDescProp subclass instance

## Flow Match Structure

```
class os_ken.ofproto.ofproto_v1_4_parser.OFPMatch(type_=None, length=None,
                                                _ordered_fields=None, **kwargs)
```

Flow Match Structure

This class is implementation of the flow match structure having compose/query API.

You can define the flow match by the keyword arguments. The following arguments are available.

Argument	Value	Description
in_port	Integer 32bit	Switch input port
in_phy_port	Integer 32bit	Switch physical input port
metadata	Integer 64bit	Metadata passed between tables
eth_dst	MAC address	Ethernet destination address
eth_src	MAC address	Ethernet source address
eth_type	Integer 16bit	Ethernet frame type
vlan_vid	Integer 16bit	VLAN id
vlan_pcp	Integer 8bit	VLAN priority
ip_dscp	Integer 8bit	IP DSCP (6 bits in ToS field)
ip_ecn	Integer 8bit	IP ECN (2 bits in ToS field)
ip_proto	Integer 8bit	IP protocol
ipv4_src	IPv4 address	IPv4 source address
ipv4_dst	IPv4 address	IPv4 destination address
tcp_src	Integer 16bit	TCP source port
tcp_dst	Integer 16bit	TCP destination port

continues on next page

Table 3 – continued from previous page

Argument	Value	Description
udp_src	Integer 16bit	UDP source port
udp_dst	Integer 16bit	UDP destination port
sctp_src	Integer 16bit	SCTP source port
sctp_dst	Integer 16bit	SCTP destination port
icmpv4_type	Integer 8bit	ICMP type
icmpv4_code	Integer 8bit	ICMP code
arp_op	Integer 16bit	ARP opcode
arp_spa	IPv4 address	ARP source IPv4 address
arp_tpa	IPv4 address	ARP target IPv4 address
arp_sha	MAC address	ARP source hardware address
arp_tha	MAC address	ARP target hardware address
ipv6_src	IPv6 address	IPv6 source address
ipv6_dst	IPv6 address	IPv6 destination address
ipv6_flabel	Integer 32bit	IPv6 Flow Label
icmpv6_type	Integer 8bit	ICMPv6 type
icmpv6_code	Integer 8bit	ICMPv6 code
ipv6_nd_target	IPv6 address	Target address for ND
ipv6_nd_sll	MAC address	Source link-layer for ND
ipv6_nd_tll	MAC address	Target link-layer for ND
mpls_label	Integer 32bit	MPLS label
mpls_tc	Integer 8bit	MPLS TC
mpls_bos	Integer 8bit	MPLS BoS bit
pbb_isid	Integer 24bit	PBB I-SID
tunnel_id	Integer 64bit	Logical Port Metadata
ipv6_exthdr	Integer 16bit	IPv6 Extension Header pseudo-field
pbb_uca	Integer 8bit	PBB UCA header field
tcp_flags	Integer 16bit	TCP flags (EXT-109 ONF Extension)
actset_output	Integer 32bit	Output port from action set metadata (EXT-233 ONF Extension)

Example:

```
>>> # compose
>>> match = parser.OFPMatch(
...     in_port=1,
...     eth_type=0x86dd,
...     ipv6_src=('2001:db8:bd05:1d2:288a:1fc0:1:10ee',
...               'ffff:ffff:ffff:ffff:'),
...     ipv6_dst='2001:db8:bd05:1d2:288a:1fc0:1:10ee')
>>> # query
>>> if 'ipv6_src' in match:
...     print match['ipv6_src']
...
('2001:db8:bd05:1d2:288a:1fc0:1:10ee', 'ffff:ffff:ffff:ffff:')
```

#### Note

For the list of the supported Nicira experimenter matches, please refer to

*os\_ken.ofproto.nx\_match.*

## Note

For VLAN id match field, special values are defined in OpenFlow Spec.

### 1) Packets with and without a VLAN tag

- Example:

```
match = parser.OFPMatch()
```

- Packet Matching

non-VLAN-tagged	MATCH
VLAN-tagged(vlan_id=3)	MATCH
VLAN-tagged(vlan_id=5)	MATCH

### 2) Only packets without a VLAN tag

- Example:

```
match = parser.OFPMatch(vlan_vid=0x0000)
```

- Packet Matching

non-VLAN-tagged	MATCH
VLAN-tagged(vlan_id=3)	x
VLAN-tagged(vlan_id=5)	x

### 3) Only packets with a VLAN tag regardless of its value

- Example:

```
match = parser.OFPMatch(vlan_vid=(0x1000, 0x1000))
```

- Packet Matching

non-VLAN-tagged	x
VLAN-tagged(vlan_id=3)	MATCH
VLAN-tagged(vlan_id=5)	MATCH

### 4) Only packets with VLAN tag and VID equal

- Example:

```
match = parser.OFPMatch(vlan_vid=(0x1000 | 3))
```

- Packet Matching

non-VLAN-tagged	x
VLAN-tagged(vlan_id=3)	MATCH
VLAN-tagged(vlan_id=5)	x

## Flow Instruction Structures

```
class os_ken.ofproto.ofproto_v1_4_parser.OFPInstructionGotoTable(table_id,
                                                                type_=None,
                                                                len_=None)
```

Goto table instruction

This instruction indicates the next table in the processing pipeline.

Attribute	Description
<code>table_id</code>	Next table

```
class os_ken.ofproto.ofproto_v1_4_parser.OFPInstructionWriteMetadata(metadata,
                                                                      meta-
                                                                      data_mask,
                                                                      type_=None,
                                                                      len_=None)
```

Write metadata instruction

This instruction writes the masked metadata value into the metadata field.

Attribute	Description
<code>metadata</code>	Metadata value to write
<code>metadata_mask</code>	Metadata write bitmask

```
class os_ken.ofproto.ofproto_v1_4_parser.OFPInstructionActions(type_,
                                                                actions=None,
                                                                len_=None)
```

Actions instruction

This instruction writes/applies/clears the actions.

Attribute	Description
<code>type</code>	One of following values.  OFPIT_WRITE_ACTIONS OFPIT_APPLY_ACTIONS OFPIT_CLEAR_ACTIONS
<code>actions</code>	list of OpenFlow action class

`type` attribute corresponds to `type_` parameter of `__init__`.

```
class os_ken.ofproto.ofproto_v1_4_parser.OFPInstructionMeter(meter_id=1,
                                                                type_=None,
                                                                len_=None)
```

Meter instruction

This instruction applies the meter.

Attribute	Description
meter_id	Meter instance

## Action Structures

```
class os_ken.ofproto.ofproto_v1_4_parser.OFFActionOutput(port, max_len=65509,
                                                         type_=None, len_=None)
```

Output action

This action indicates output a packet to the switch port.

Attribute	Description
port	Output port
max_len	Max length to send to controller

```
class os_ken.ofproto.ofproto_v1_4_parser.OFFActionCopyTtlOut(type_=None,
                                                                len_=None)
```

Copy TTL Out action

This action copies the TTL from the next-to-outermost header with TTL to the outermost header with TTL.

```
class os_ken.ofproto.ofproto_v1_4_parser.OFFActionCopyTtlIn(type_=None,
                                                                len_=None)
```

Copy TTL In action

This action copies the TTL from the outermost header with TTL to the next-to-outermost header with TTL.

```
class os_ken.ofproto.ofproto_v1_4_parser.OFFActionSetMplsTtl(mpls_ttl, type_=None,
                                                                len_=None)
```

Set MPLS TTL action

This action sets the MPLS TTL.

Attribute	Description
mpls_ttl	MPLS TTL

```
class os_ken.ofproto.ofproto_v1_4_parser.OFFActionDecMplsTtl(type_=None,
                                                                len_=None)
```

Decrement MPLS TTL action

This action decrements the MPLS TTL.

```
class os_ken.ofproto.ofproto_v1_4_parser.OFPActionPushVlan(ethertype=33024,
                                                         type_=None,
                                                         len_=None)
```

Push VLAN action

This action pushes a new VLAN tag to the packet.

Attribute	Description
ethertype	Ether type. The default is 802.1Q. (0x8100)

```
class os_ken.ofproto.ofproto_v1_4_parser.OFPActionPopVlan(type_=None, len_=None)
```

Pop VLAN action

This action pops the outermost VLAN tag from the packet.

```
class os_ken.ofproto.ofproto_v1_4_parser.OFPActionPushMpls(ethertype=34887,
                                                            type_=None,
                                                            len_=None)
```

Push MPLS action

This action pushes a new MPLS header to the packet.

Attribute	Description
ethertype	Ether type

```
class os_ken.ofproto.ofproto_v1_4_parser.OFPActionPopMpls(ethertype=2048,
                                                           type_=None, len_=None)
```

Pop MPLS action

This action pops the MPLS header from the packet.

```
class os_ken.ofproto.ofproto_v1_4_parser.OFPActionSetQueue(queue_id, type_=None,
                                                            len_=None)
```

Set queue action

This action sets the queue id that will be used to map a flow to an already-configured queue on a port.

Attribute	Description
queue_id	Queue ID for the packets

```
class os_ken.ofproto.ofproto_v1_4_parser.OFPActionGroup(group_id=0, type_=None,
                                                         len_=None)
```

Group action

This action indicates the group used to process the packet.

Attribute	Description
group_id	Group identifier

```
class os_ken.ofproto.ofproto_v1_4_parser.OFPActionSetNwTtl(nw_ttl, type_=None,
                                                         len_=None)
```

Set IP TTL action

This action sets the IP TTL.

Attribute	Description
<code>nw_ttl</code>	IP TTL

```
class os_ken.ofproto.ofproto_v1_4_parser.OFPActionDecNwTtl(type_=None,
                                                            len_=None)
```

Decrement IP TTL action

This action decrements the IP TTL.

```
class os_ken.ofproto.ofproto_v1_4_parser.OFPActionSetField(field=None, **kwargs)
```

Set field action

This action modifies a header field in the packet.

The set of keywords available for this is same as OFPMatch.

Example:

```
set_field = OFPActionSetField(eth_src="00:00:00:00:00:00")
```

```
class os_ken.ofproto.ofproto_v1_4_parser.OFPActionPushPbb(ethertype, type_=None,
                                                           len_=None)
```

Push PBB action

This action pushes a new PBB header to the packet.

Attribute	Description
<code>ethertype</code>	Ether type

```
class os_ken.ofproto.ofproto_v1_4_parser.OFPActionPopPbb(type_=None, len_=None)
```

Pop PBB action

This action pops the outermost PBB service instance header from the packet.

```
class os_ken.ofproto.ofproto_v1_4_parser.OFPActionExperimenter(experimenter)
```

Experimenter action

This action is an extensible action for the experimenter.

Attribute	Description
<code>experimenter</code>	Experimenter ID



**Note**

For the list of the supported Nicira experimenter actions, please refer to *os\_ken.ofproto.nx\_actions*.

**OpenFlow v1.5 Messages and Structures****Controller-to-Switch Messages****Handshake**

**class** `os_ken.ofproto.ofproto_v1_5_parser.OFPFeaturesRequest(datapath)`

Features request message

The controller sends a feature request to the switch upon session establishment.

This message is handled by the OSKen framework, so the OSKen application do not need to process this typically.

Example:

```
def send_features_request(self, datapath):
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPFeaturesRequest(datapath)
    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPFeaturesRequest": {}
}
```

**class** `os_ken.ofproto.ofproto_v1_5_parser.OFPSwitchFeatures(datapath, datapath_id=None, n_buffers=None, n_tables=None, auxiliary_id=None, capabilities=None)`

Features reply message

The switch responds with a features reply message to a features request.

This message is handled by the OSKen framework, so the OSKen application do not need to process this typically.

Example:

```
@set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
def switch_features_handler(self, ev):
    msg = ev.msg

    self.logger.debug('OFPSwitchFeatures received: '
```

(continues on next page)

(continued from previous page)

```
'datapath_id=0x%016x n_buffers=%d '
'n_tables=%d auxiliary_id=%d '
'capabilities=0x%08x',
msg.datapath_id, msg.n_buffers, msg.n_tables,
msg.auxiliary_id, msg.capabilities)
```

JSON Example:

```
{
  "OFPSwitchFeatures": {
    "auxiliary_id": 0,
    "capabilities": 79,
    "datapath_id": 1,
    "n_buffers": 255,
    "n_tables": 255
  }
}
```

## Switch Configuration

```
class os_ken.ofproto.ofproto_v1_5_parser.OFPSetConfig(datapath, flags=0,
                                                    miss_send_len=0)
```

Set config request message

The controller sends a set config request message to set configuraion parameters.

Attribute	Description
flags	Bitmap of the following flags.  OFPC_FRAG_NORMAL OFPC_FRAG_DROP OFPC_FRAG_REASM
miss_send_len	Max bytes of new flow that datapath should send to the controller

Example:

```
def send_set_config(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPSetConfig(datapath, ofp.OFPC_FRAG_NORMAL, 256)
    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPSetsConfig": {
    "flags": 0,
    "miss_send_len": 128
  }
}
```

**class** `os_ken.ofproto.ofproto_v1_5_parser.OFPGetConfigRequest(datapath)`

Get config request message

The controller sends a get config request to query configuration parameters in the switch.

Example:

```
def send_get_config_request(self, datapath):
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPGetConfigRequest(datapath)
    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPGetConfigRequest": {}
}
```

**class** `os_ken.ofproto.ofproto_v1_5_parser.OFPGetConfigReply(datapath, flags=None, miss_send_len=None)`

Get config reply message

The switch responds to a configuration request with a get config reply message.

Attribute	Description
flags	Bitmap of the following flags.  OFPC_FRAG_NORMAL OFPC_FRAG_DROP OFPC_FRAG_REASM
miss_send_len	Max bytes of new flow that datapath should send to the controller

Example:

```
@set_ev_cls(ofp_event.EventOFPGetConfigReply, MAIN_DISPATCHER)
def get_config_reply_handler(self, ev):
    msg = ev.msg
    dp = msg.datapath
    ofp = dp.ofproto
    flags = []
```

(continues on next page)

(continued from previous page)

```

if msg.flags & ofp.OFPC_FRAG_NORMAL:
    flags.append('NORMAL')
if msg.flags & ofp.OFPC_FRAG_DROP:
    flags.append('DROP')
if msg.flags & ofp.OFPC_FRAG_REASM:
    flags.append('REASM')
self.logger.debug('OFPCGetConfigReply received: '
                  'flags=%s miss_send_len=%d',
                  ','.join(flags), msg.miss_send_len)

```

JSON Example:

```

{
  "OFPCGetConfigReply": {
    "flags": 0,
    "miss_send_len": 128
  }
}

```

## Modify State Messages

**class** `os_ken.ofproto.ofproto_v1_5_parser.OFPTTableMod`(*datapath*, *table\_id*, *config*, *properties*)

Flow table configuration message

The controller sends this message to configure table state.

Attribute	Description
<code>table_id</code>	ID of the table (OFPTT_ALL indicates all tables)
<code>config</code>	Bitmap of configuration flags.  OFPTC_EVICTION OFPTC_VACANCY_EVENTS
<code>properties</code>	List of OFPTTableModProp subclass instance

Example:

```

def send_table_mod(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPTTableMod(datapath, 1, 3)
    flags = ofp.OFPTC_VACANCY_EVENTS
    properties = [ofp_parser.OFPTTableModPropEviction(flags)]
    req = ofp_parser.OFPTTableMod(datapath, 1, 3, properties)
    datapath.send_msg(req)

```

JSON Example:

```
{
  "OFPTableMod": {
    "config": 4,
    "properties": [
      {
        "OFPTableModPropEviction": {
          "flags": 2,
          "length": 8,
          "type": 2
        }
      }
    ],
    "table_id": 255
  }
}
```

```
class os_ken.ofproto.ofproto_v1_5_parser.OFPFlowMod(datapath, cookie=0,
                                                    cookie_mask=0, table_id=0,
                                                    command=0, idle_timeout=0,
                                                    hard_timeout=0, priority=32768,
                                                    buffer_id=4294967295,
                                                    out_port=0, out_group=0,
                                                    flags=0, importance=0,
                                                    match=None, instructions=None)
```

Modify Flow entry message

The controller sends this message to modify the flow table.

Attribute	Description
cookie	Opaque controller-issued identifier
cookie_mask	Mask used to restrict the cookie bits that must match when the command is <code>OPFFC_MODIFY*</code> or <code>OPFFC_DELETE*</code>
table_id	ID of the table to put the flow in
command	One of the following values.  <code>OPFFC_ADD</code> <code>OPFFC_MODIFY</code> <code>OPFFC_MODIFY_STRICT</code> <code>OPFFC_DELETE</code> <code>OPFFC_DELETE_STRICT</code>
idle_timeout	Idle time before discarding (seconds)
hard_timeout	Max time before discarding (seconds)
priority	Priority level of flow entry
buffer_id	Buffered packet to apply to (or <code>OPF_NO_BUFFER</code> )
out_port	For <code>OPFFC_DELETE*</code> commands, require matching entries to include this as an output port
out_group	For <code>OPFFC_DELETE*</code> commands, require matching entries to include this as an output group
flags	Bitmap of the following flags.  <code>OPPFF_SEND_FLOW_REM</code> <code>OPPFF_CHECK_OVERLAP</code> <code>OPPFF_RESET_COUNTS</code> <code>OPPFF_NO_PKT_COUNTS</code> <code>OPPFF_NO_BYT_COUNTS</code>
importance	Eviction precedence
match	Instance of <code>OFPMatch</code>
instructions	list of <code>OFPIInstruction*</code> instance

Example:

```
def send_flow_mod(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    cookie = cookie_mask = 0
    table_id = 0
    idle_timeout = hard_timeout = 0
    priority = 32768
    buffer_id = ofp.OPF_NO_BUFFER
```

(continues on next page)

(continued from previous page)

```

importance = 0
match = ofp_parser.OFPMatch(in_port=1, eth_dst='ff:ff:ff:ff:ff:ff')
actions = [ofp_parser.OFPActionOutput(ofp.OFPP_NORMAL, 0)]
inst = [ofp_parser.OFPInstructionActions(ofp.OFPIT_APPLY_ACTIONS,
                                         actions)]

req = ofp_parser.OFPFlowMod(datapath, cookie, cookie_mask,
                            table_id, ofp.OFPFC_ADD,
                            idle_timeout, hard_timeout,
                            priority, buffer_id,
                            ofp.OFPP_ANY, ofp.OFPG_ANY,
                            ofp.OFPFF_SEND_FLOW_REM,
                            importance,
                            match, inst)

datapath.send_msg(req)

```

JSON Example:

```

{
  "OFPFlowMod": {
    "buffer_id": 0,
    "command": 0,
    "cookie": 1311768467463790320,
    "cookie_mask": 18446744073709551615,
    "flags": 0,
    "hard_timeout": 0,
    "idle_timeout": 0,
    "importance": 39032,
    "instructions": [
      {
        "OFPInstructionActions": {
          "actions": [
            {
              "OFPActionPopVlan": {
                "len": 8,
                "type": 18
              }
            },
            {
              "OFPActionSetField": {
                "field": {
                  "OXMTlv": {
                    "field": "ipv4_dst",
                    "mask": null,
                    "value": "192.168.2.9"
                  }
                }
              }
            }
          ],
          "len": 16,
          "type": 25
        }
      }
    ]
  }
}

```

(continues on next page)

(continued from previous page)

```
{
  "NXActionLearn": {
    "cookie": 0,
    "experimenter": 8992,
    "fin_hard_timeout": 0,
    "fin_idle_timeout": 0,
    "flags": 0,
    "hard_timeout": 300,
    "idle_timeout": 0,
    "len": 96,
    "priority": 1,
    "specs": [
      {
        "NXFlowSpecMatch": {
          "dst": [
            "vlan_vid",
            0
          ],
          "n_bits": 12,
          "src": [
            "vlan_vid",
            0
          ]
        },
        {
          "NXFlowSpecMatch": {
            "dst": [
              "eth_dst_nxm",
              0
            ],
            "n_bits": 48,
            "src": [
              "eth_src_nxm",
              0
            ]
          }
        },
        {
          "NXFlowSpecLoad": {
            "dst": [
              "vlan_vid",
              0
            ],
            "n_bits": 12,
            "src": 0
          }
        }
      ]
    }
  }
}
```

(continues on next page)



(continued from previous page)

```

        "NXFlowSpecLoad": {
            "dst": [
                "tunnel_id_nxm",
                0
            ],
            "n_bits": 64,
            "src": [
                "tunnel_id_nxm",
                0
            ]
        },
        {
            "NXFlowSpecOutput": {
                "dst": "",
                "n_bits": 32,
                "src": [
                    "in_port",
                    0
                ]
            }
        },
        "subtype": 16,
        "table_id": 99,
        "type": 65535
    }
},
"len": 128,
"type": 4
},
{
    "OFPIInstructionGotoTable": {
        "len": 8,
        "table_id": 100,
        "type": 1
    }
},
"match": {
    "OFPMatch": {
        "length": 70,
        "oxm_fields": [
            {
                "OXMTlv": {
                    "field": "in_port",
                    "mask": null,

```

(continues on next page)

(continued from previous page)

```
        "value": 43981
    },
    {
        "OXMTlv": {
            "field": "eth_dst",
            "mask": null,
            "value": "aa:bb:cc:99:88:77"
        }
    },
    {
        "OXMTlv": {
            "field": "eth_type",
            "mask": null,
            "value": 2048
        }
    },
    {
        "OXMTlv": {
            "field": "vlan_vid",
            "mask": null,
            "value": 5095
        }
    },
    {
        "OXMTlv": {
            "field": "ipv4_dst",
            "mask": null,
            "value": "192.168.2.1"
        }
    },
    {
        "OXMTlv": {
            "field": "tunnel_id",
            "mask": null,
            "value": 50000
        }
    },
    {
        "OXMTlv": {
            "field": "tun_ipv4_src",
            "mask": null,
            "value": "192.168.2.3"
        }
    },
    {
        "OXMTlv": {
            "field": "tun_ipv4_dst",
            "mask": null,
```

(continues on next page)

(continued from previous page)

```

        "value": "192.168.2.4"
      }
    }
  ],
  "type": 1
}
},
"out_group": 0,
"out_port": 0,
"priority": 0,
"table_id": 2
}
}

```

```

{
  "OFPFlowMod": {
    "buffer_id": 0,
    "command": 0,
    "cookie": 1311768467463790320,
    "cookie_mask": 18446744073709551615,
    "flags": 0,
    "hard_timeout": 0,
    "idle_timeout": 0,
    "importance": 39032,
    "instructions": [
      {
        "OFPInstructionActions": {
          "actions": [
            {
              "NXActionConjunction": {
                "clause": 1,
                "experimenter": 8992,
                "id": 11259375,
                "len": 16,
                "n_clauses": 2,
                "subtype": 34,
                "type": 65535
              }
            }
          ],
          "len": 24,
          "type": 4
        }
      }
    ],
    "match": {
      "OFPMatch": {
        "length": 70,
        "oxm_fields": [

```

(continues on next page)

(continued from previous page)

```
{
  "OXMTlv": {
    "field": "in_port",
    "mask": null,
    "value": 43981
  }
},
{
  "OXMTlv": {
    "field": "eth_dst",
    "mask": null,
    "value": "aa:bb:cc:99:88:77"
  }
},
{
  "OXMTlv": {
    "field": "eth_type",
    "mask": null,
    "value": 2048
  }
},
{
  "OXMTlv": {
    "field": "vlan_vid",
    "mask": null,
    "value": 5095
  }
},
{
  "OXMTlv": {
    "field": "ipv4_dst",
    "mask": null,
    "value": "192.168.2.1"
  }
},
{
  "OXMTlv": {
    "field": "tunnel_id",
    "mask": null,
    "value": 50000
  }
},
{
  "OXMTlv": {
    "field": "tun_ipv4_src",
    "mask": null,
    "value": "192.168.2.3"
  }
},
},
```

(continues on next page)

(continued from previous page)

```

    {
      "OXMTlv": {
        "field": "tun_ipv4_dst",
        "mask": null,
        "value": "192.168.2.4"
      }
    }
  ],
  "type": 1
}
},
"out_group": 0,
"out_port": 0,
"priority": 0,
"table_id": 4
}
}

```

```

{
  "OFPPFlowMod": {
    "buffer_id": 0,
    "command": 0,
    "cookie": 1311768467463790320,
    "cookie_mask": 18446744073709551615,
    "flags": 0,
    "hard_timeout": 0,
    "idle_timeout": 0,
    "importance": 39032,
    "instructions": [
      {
        "OFPIInstructionActions": {
          "actions": [
            {
              "OFPAActionPopVlan": {
                "len": 8,
                "type": 18
              }
            },
            {
              "OFPAActionSetField": {
                "field": {
                  "OXMTlv": {
                    "field": "ipv4_dst",
                    "mask": null,
                    "value": "192.168.2.9"
                  }
                }
              },
              "len": 16,
              "type": 25
            }
          ]
        }
      }
    ]
  }
}

```

(continues on next page)

(continued from previous page)

```

        }
    ],
    "len": 32,
    "type": 4
}
},
{
    "OFPIInstructionGotoTable": {
        "len": 8,
        "table_id": 100,
        "type": 1
    }
}
],
"match": {
    "OFPMatch": {
        "length": 12,
        "oxm_fields": [
            {
                "OXMTlv": {
                    "field": "conj_id",
                    "mask": null,
                    "value": 11259375
                }
            }
        ],
        "type": 1
    }
},
"out_group": 0,
"out_port": 0,
"priority": 0,
"table_id": 3
}
}
}

```

```

class os_ken.ofproto.ofproto_v1_5_parser.OFPGroupMod(datapath, command=0, type_=0,
                                                    group_id=0, com-
                                                    mand_bucket_id=4294967295,
                                                    buckets=None, properties=None,
                                                    bucket_array_len=None)

```

Modify group entry message

The controller sends this message to modify the group table.

Attribute	Description
command	One of the following values.  OFPGC_ADD OFPGC_MODIFY OFPGC_DELETE OFPGC_INSERT_BUCKET OFPGC_REMOVE_BUCKET
type	One of the following values.  OFPGT_ALL OFPGT_SELECT OFPGT_INDIRECT OFPGT_FF
group_id	Group identifier.
command_bucket_id	Bucket Id used as part of OFPGC_INSERT_BUCKET and OFPGC_REMOVE_BUCKET commands execution.
buckets	List of OFPBucket instance
properties	List of OFPGroupProp instance

type attribute corresponds to type\_ parameter of \_\_init\_\_.

Example:

```
def send_group_mod(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    port = 1
    max_len = 2000
    actions = [ofp_parser.OFPActionOutput(port, max_len)]

    weight = 100
    watch_port = 0
    watch_group = 0
    buckets = [ofp_parser.OFPBucket(weight, watch_port, watch_group,
                                    actions)]

    group_id = 1
    command_bucket_id=1
    req = ofp_parser.OFPGroupMod(datapath, ofp.OFPGC_ADD,
                                 ofp.OFPGT_SELECT, group_id,
                                 command_bucket_id, buckets)

    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPGroupMod": {
    "bucket_array_len": 56,
    "buckets": [
      {
        "OFPBucket": {
          "action_array_len": 24,
          "actions": [
            {
              "OFPActionPopVlan": {
                "len": 8,
                "type": 18
              }
            },
            {
              "OFPActionSetField": {
                "field": {
                  "OXMTlv": {
                    "field": "ipv4_dst",
                    "mask": null,
                    "value": "192.168.2.9"
                  }
                }
              },
              "len": 16,
              "type": 25
            }
          ]
        },
        "bucket_id": 305419896,
        "len": 56,
        "properties": [
          {
            "OFPGroupBucketPropWeight": {
              "length": 8,
              "type": 0,
              "weight": 52428
            }
          },
          {
            "OFPGroupBucketPropWatch": {
              "length": 8,
              "type": 1,
              "watch": 56797
            }
          },
          {
            "OFPGroupBucketPropWatch": {
              "length": 8,
              "type": 2,
```

(continues on next page)



(continued from previous page)

```

        "watch": 4008636142
    }
}
],
"command": 3,
"command_bucket_id": 3149642683,
"group_id": 2863311530,
"properties": [],
"type": 1
}
}

```

```

class os_ken.ofproto.ofproto_v1_5_parser.OFPPortMod(datapath, port_no=0,
                                                    hw_addr='00:00:00:00:00:00',
                                                    config=0, mask=0,
                                                    properties=None)

```

Port modification message

The controller sends this message to modify the behavior of the port.

Attribute	Description
port_no	Port number to modify
hw_addr	The hardware address that must be the same as hw_addr of OFPPort of OFPSwitchFeatures
config	Bitmap of configuration flags.  OFPPC_PORT_DOWN OFPPC_NO_RECV OFPPC_NO_FWD OFPPC_NO_PACKET_IN
mask	Bitmap of configuration flags above to be changed
properties	List of OFPPortModProp subclass instance

Example:

```

def send_port_mod(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    port_no = 3
    hw_addr = 'fa:c8:e8:76:1d:7e'
    config = 0

```

(continues on next page)

(continued from previous page)

```

mask = (ofp.OFPPC_PORT_DOWN | ofp.OFPPC_NO_RECV |
        ofp.OFPPC_NO_FWD | ofp.OFPPC_NO_PACKET_IN)
advertise = (ofp.OFPPF_10MB_FD | ofp.OFPPF_100MB_FD |
            ofp.OFPPF_1GB_FD | ofp.OFPPF_COPPER |
            ofp.OFPPF_AUTONEG | ofp.OFPPF_PAUSE |
            ofp.OFPPF_PAUSE_ASYM)
properties = [ofp_parser.OFPPortModPropEthernet(advertise)]
req = ofp_parser.OFPPortMod(datapath, port_no, hw_addr, config,
                            mask, properties)

datapath.send_msg(req)

```

JSON Example:

```

{
  "OFPPortMod": {
    "config": 0,
    "hw_addr": "00:11:00:00:11:11",
    "mask": 0,
    "port_no": 1,
    "properties": [
      {
        "OFPPortModPropEthernet": {
          "advertise": 4096,
          "length": 8,
          "type": 0
        }
      },
      {
        "OFPPortModPropOptical": {
          "configure": 3,
          "fl_offset": 2000,
          "freq_lmda": 1500,
          "grid_span": 3000,
          "length": 24,
          "tx_pwr": 300,
          "type": 1
        }
      },
      {
        "OFPPortModPropExperimenter": {
          "data": [],
          "exp_type": 0,
          "experimenter": 101,
          "length": 12,
          "type": 65535
        }
      },
      {
        "OFPPortModPropExperimenter": {
          "data": [

```

(continues on next page)

(continued from previous page)

```
        1
    ],
    "exp_type": 1,
    "experimenter": 101,
    "length": 16,
    "type": 65535
}
},
{
    "OFPPortModPropExperimenter": {
        "data": [
            1,
            2
        ],
        "exp_type": 2,
        "experimenter": 101,
        "length": 20,
        "type": 65535
    }
}
]
}
}
```

**class** `os_ken.ofproto.ofproto_v1_5_parser.OFPMeterMod`(*datapath*, *command=0*, *flags=1*,  
*meter\_id=1*, *bands=None*)

Meter modification message

The controller sends this message to modify the meter.

Attribute	Description
command	One of the following values.  OFPMC_ADD OFPMC_MODIFY OFPMC_DELETE
flags	Bitmap of the following flags.  OFPMF_KBPS OFPMF_PKTPTS OFPMF_BURST OFPMF_STATS
meter_id	Meter instance
bands	list of the following class instance.  OFPMeterBandDrop OFPMeterBandDscpRemark OFPMeterBandExperimenter

JSON Example:

```
{
  "OFPMeterMod": {
    "bands": [
      {
        "OFPMeterBandDrop": {
          "burst_size": 10,
          "len": 16,
          "rate": 1000,
          "type": 1
        }
      },
      {
        "OFPMeterBandDscpRemark": {
          "burst_size": 10,
          "len": 16,
          "prec_level": 1,
          "rate": 1000,
          "type": 2
        }
      }
    ],
    "command": 0,
    "flags": 14,
    "meter_id": 100
  }
}
```

(continues on next page)

(continued from previous page)

```

    }
}

```

## Multipart Messages

**class** `os_ken.ofproto.ofproto_v1_5_parser.OFPDescStatsRequest`(*datapath*, *flags=0*, *type\_=None*)

Description statistics request message

The controller uses this message to query description of the switch.

Attribute	Description
flags	Zero or OFPMPF_REQ_MORE

Example:

```

def send_desc_stats_request(self, datapath):
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPDescStatsRequest(datapath, 0)
    datapath.send_msg(req)

```

JSON Example:

```

{
  "OFPDescStatsRequest": {
    "flags": 0,
    "type": 0
  }
}

```

**class** `os_ken.ofproto.ofproto_v1_5_parser.OFPDescStatsReply`(*datapath*, *type\_=None*, *\*\*kwargs*)

Description statistics reply message

The switch responds with this message to a description statistics request.

Attribute	Description
body	Instance of OFPDescStats

Example:

```

@set_ev_cls(ofp_event.EventOFPDescStatsReply, MAIN_DISPATCHER)
def desc_stats_reply_handler(self, ev):
    body = ev.msg.body

    self.logger.debug('DescStats: mfr_desc=%s hw_desc=%s sw_desc=%s '

```

(continues on next page)

(continued from previous page)

```
'serial_num=%s dp_desc=%s',
body.mfr_desc, body.hw_desc, body.sw_desc,
body.serial_num, body.dp_desc)
```

JSON Example:

```
{
  "OFPDescStatsReply": {
    "body": {
      "OFPDescStats": {
        "dp_desc": "dp",
        "hw_desc": "hw",
        "mfr_desc": "mfr",
        "serial_num": "serial",
        "sw_desc": "sw"
      }
    },
    "flags": 0,
    "type": 0
  }
}
```

```
class os_ken.ofproto.ofproto_v1_5_parser.OFPFlowDescStatsRequest(datapath,
                                                                    flags=0,
                                                                    table_id=255,
                                                                    out_port=4294967295,
                                                                    out_group=4294967295,
                                                                    cookie=0,
                                                                    cookie_mask=0,
                                                                    match=None,
                                                                    type_=None)
```

Individual flow descriptions request message

The controller uses this message to query individual flow descriptions.

Attribute	Description
flags	Zero or OFPMPF_REQ_MORE
table_id	ID of table to read
out_port	Require matching entries to include this as an output port
out_group	Require matching entries to include this as an output group
cookie	Require matching entries to contain this cookie value
cookie_mask	Mask used to restrict the cookie bits that must match
match	Instance of OFPMatch

Example:

```
def send_flow_desc_request(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser
```

(continues on next page)

(continued from previous page)

```

cookie = cookie_mask = 0
match = ofp_parser.OFPMatch(in_port=1)
req = ofp_parser.OFPFlowDescStatsRequest(datapath, 0,
                                         ofp.OFPTT_ALL,
                                         ofp.OFPP_ANY,
                                         ofp.OFPG_ANY,
                                         cookie, cookie_mask,
                                         match)

datapath.send_msg(req)

```

JSON Example:

```

{
  "OFPFlowDescStatsRequest": {
    "cookie": 1234605616436508552,
    "cookie_mask": 18446744073709551615,
    "flags": 0,
    "match": {
      "OFPMatch": {
        "length": 12,
        "oxm_fields": [
          {
            "OXMTlv": {
              "field": "in_port",
              "mask": null,
              "value": 1
            }
          }
        ],
        "type": 1
      }
    },
    "out_group": 4294967295,
    "out_port": 4294967295,
    "table_id": 1,
    "type": 1
  }
}

```

```

class os_ken.ofproto.ofproto_v1_5_parser.OFPFlowDescStatsReply(datapath,
                                                                type_=None,
                                                                **kwargs)

```

Individual flow descriptions reply message

The switch responds with this message to an individual flow descriptions request.

Attribute	Description
body	List of OFPFlowDesc instance

Example:

```
@set_ev_cls(ofp_event.EventOFPPFlowDescStatsReply, MAIN_DISPATCHER)
def flow_desc_reply_handler(self, ev):
    flows = []
    for stat in ev.msg.body:
        flows.append('table_id=%s priority=%d '
                    'idle_timeout=%d hard_timeout=%d flags=0x%04x '
                    'importance=%d cookie=%d match=%s '
                    'stats=%s instructions=%s' %
                    (stat.table_id, stat.priority,
                     stat.idle_timeout, stat.hard_timeout,
                     stat.flags, stat.importance,
                     stat.cookie, stat.match,
                     stat.stats, stat.instructions))
    self.logger.debug('FlowDesc: %s', flows)
```

JSON Example:

```
{
  "OFPPFlowDescStatsReply": {
    "body": [
      {
        "OFPPFlowDesc": {
          "cookie": 1234605616436508552,
          "flags": 1,
          "hard_timeout": 255,
          "idle_timeout": 255,
          "importance": 43690,
          "instructions": [
            {
              "OFPIInstructionGotoTable": {
                "len": 8,
                "table_id": 2,
                "type": 1
              }
            }
          ],
          "length": 64,
          "match": {
            "OFPMatch": {
              "length": 12,
              "oxm_fields": [
                {
                  "OXMTlv": {
                    "field": "in_port",
                    "mask": null,
                    "value": 1
                  }
                }
              ]
            }
          }
        }
      ]
    }
  }
}
```

(continues on next page)



(continued from previous page)

```

        "type": 1
    },
    "priority": 5,
    "stats": {
        "OFStats": {
            "length": 12,
            "oxs_fields": [
                {
                    "OXSTlv": {
                        "field": "flow_count",
                        "value": 1
                    }
                }
            ]
        }
    },
    "table_id": 1
}
],
"flags": 0,
"type": 1
}
}

```

```

class os_ken.ofproto.ofproto_v1_5_parser.OFPFlowStatsRequest(datapath, flags=0,
                                                                table_id=255,
                                                                out_port=4294967295,
                                                                out_group=4294967295,
                                                                cookie=0,
                                                                cookie_mask=0,
                                                                match=None,
                                                                type_=None)

```

Individual flow statistics request message

The controller uses this message to query individual flow statistics.

Attribute	Description
flags	Zero or OFPMPF_REQ_MORE
table_id	ID of table to read
out_port	Require matching entries to include this as an output port
out_group	Require matching entries to include this as an output group
cookie	Require matching entries to contain this cookie value
cookie_mask	Mask used to restrict the cookie bits that must match
match	Instance of OFPMatch

Example:

```
def send_flow_stats_request(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    cookie = cookie_mask = 0
    match = ofp_parser.OFPMatch(in_port=1)
    req = ofp_parser.OFPFlowStatsRequest(datapath, 0,
                                         ofp.OFPTT_ALL,
                                         ofp.OFPP_ANY, ofp.OFPG_ANY,
                                         cookie, cookie_mask,
                                         match)

    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPFlowStatsRequest": {
    "cookie": 0,
    "cookie_mask": 0,
    "flags": 0,
    "match": {
      "OFPMatch": {
        "length": 4,
        "oxm_fields": [],
        "type": 1
      }
    },
    "out_group": 4294967295,
    "out_port": 4294967295,
    "table_id": 0,
    "type": 17
  }
}
```

**class** `os_ken.ofproto.ofproto_v1_5_parser.OFPFlowStatsReply`(*datapath*, *type\_=None*, *\*\*kwargs*)

Individual flow statistics reply message

The switch responds with this message to an individual flow statistics request.

Attribute	Description
body	List of OFPFlowStats instance

Example:

```
@set_ev_cls(ofp_event.EventOFPFlowStatsReply, MAIN_DISPATCHER)
def flow_stats_reply_handler(self, ev):
    flows = []
    for stat in ev.msg.body:
```

(continues on next page)

(continued from previous page)

```
flows.append('table_id=%s reason=%d priority=%d '
            'match=%s stats=%s' %
            (stat.table_id, stat.reason, stat.priority,
             stat.match, stat.stats))
self.logger.debug('FlowStats: %s', flows)
```

JSON Example:

```
{
  "OFPPFlowStatsReply": {
    "body": [
      {
        "OFPPFlowStats": {
          "length": 40,
          "match": {
            "OFPMatch": {
              "length": 12,
              "oxm_fields": [
                {
                  "OXMTlv": {
                    "field": "in_port",
                    "mask": null,
                    "value": 1
                  }
                }
              ],
              "type": 1
            }
          },
          "priority": 1,
          "reason": 0,
          "stats": {
            "OFPPStats": {
              "length": 12,
              "oxs_fields": [
                {
                  "OXSTlv": {
                    "field": "flow_count",
                    "value": 1
                  }
                }
              ]
            }
          },
          "table_id": 1
        }
      ]
    },
    "flags": 0,
    "type": 17
  }
}
```

(continues on next page)

(continued from previous page)

```

    }
}

```

```

class os_ken.ofproto.ofproto_v1_5_parser.OFPAggregateStatsRequest(datapath, flags,
                                                                    table_id,
                                                                    out_port,
                                                                    out_group,
                                                                    cookie,
                                                                    cookie_mask,
                                                                    match,
                                                                    type_=None)

```

Aggregate flow statistics request message

The controller uses this message to query aggregate flow statistics.

Attribute	Description
flags	Zero or OFPMPF_REQ_MORE
table_id	ID of table to read
out_port	Require matching entries to include this as an output port
out_group	Require matching entries to include this as an output group
cookie	Require matching entries to contain this cookie value
cookie_mask	Mask used to restrict the cookie bits that must match
match	Instance of OFPMatch

Example:

```

def send_aggregate_stats_request(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    cookie = cookie_mask = 0
    match = ofp_parser.OFPMatch(in_port=1)
    req = ofp_parser.OFPAggregateStatsRequest(datapath, 0,
                                              ofp.OFPTT_ALL,
                                              ofp.OFPP_ANY,
                                              ofp.OFPG_ANY,
                                              cookie, cookie_mask,
                                              match)

    datapath.send_msg(req)

```

JSON Example:

```

{
  "OFPAggregateStatsRequest": {
    "cookie": 0,
    "cookie_mask": 0,
    "flags": 0,
    "match": {
      "OFPMatch": {

```

(continues on next page)

(continued from previous page)

```

        "length": 4,
        "oxm_fields": [],
        "type": 1
    },
    "out_group": 4294967295,
    "out_port": 4294967295,
    "table_id": 255,
    "type": 2
}
}

```

```

class os_ken.ofproto.ofproto_v1_5_parser.OFPAggregateStatsReply(datapath,
                                                                type_=None,
                                                                **kwargs)

```

Aggregate flow statistics reply message

The switch responds with this message to an aggregate flow statistics request.

Attribute	Description
body	Instance of OFPAggregateStats

Example:

```

@set_ev_cls(ofp_event.EventOFPAggregateStatsReply, MAIN_DISPATCHER)
def aggregate_stats_reply_handler(self, ev):
    body = ev.msg.body

    self.logger.debug('AggregateStats: stats=%s', body.stats)

```

JSON Example:

```

{
  "OFPAggregateStatsReply": {
    "body": {
      "OFPAggregateStats": {
        "length": 16,
        "stats": {
          "OFPStats": {
            "length": 12,
            "oxs_fields": [
              {
                "OXSTlv": {
                  "field": "flow_count",
                  "value": 1
                }
              }
            ]
          }
        }
      }
    }
  }
}

```

(continues on next page)

(continued from previous page)

```

    }
    },
    "flags": 0,
    "type": 2
  }
}

```

**class** `os_ken.ofproto.ofproto_v1_5_parser.OFPPortStatsRequest`(*datapath*, *flags*, *port\_no*, *type\_=None*)

Port statistics request message

The controller uses this message to query information about ports statistics.

Attribute	Description
<code>flags</code>	Zero or <code>OFPMPPF_REQ_MORE</code>
<code>port_no</code>	Port number to read ( <code>OFPP_ANY</code> to all ports)

Example:

```

def send_port_stats_request(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPPortStatsRequest(datapath, 0, ofp.OFPP_ANY)
    datapath.send_msg(req)

```

JSON Example:

```

{
  "OFPPortStatsRequest": {
    "flags": 0,
    "port_no": 4294967295,
    "type": 4
  }
}

```

**class** `os_ken.ofproto.ofproto_v1_5_parser.OFPPortStatsReply`(*datapath*, *type\_=None*, *\*\*kwargs*)

Port statistics reply message

The switch responds with this message to a port statistics request.

Attribute	Description
<code>body</code>	List of <code>OFPPortStats</code> instance

Example:

```

@set_ev_cls(ofp_event.EventOFPPortStatsReply, MAIN_DISPATCHER)
def port_stats_reply_handler(self, ev):
    ports = []
    for stat in ev.msg.body:
        ports.append(stat.length, stat.port_no,
                    stat.duration_sec, stat.duration_nsec,
                    stat.rx_packets, stat.tx_packets,
                    stat.rx_bytes, stat.tx_bytes,
                    stat.rx_dropped, stat.tx_dropped,
                    stat.rx_errors, stat.tx_errors,
                    repr(stat.properties))
    self.logger.debug('PortStats: %s', ports)

```

JSON Example:

```

{
  "OFPPortStatsReply": {
    "body": [
      {
        "OFPPortStats": {
          "duration_nsec": 0,
          "duration_sec": 0,
          "length": 224,
          "port_no": 7,
          "properties": [
            {
              "OFPPortStatsPropEthernet": {
                "collisions": 0,
                "length": 40,
                "rx_crc_err": 0,
                "rx_frame_err": 0,
                "rx_over_err": 0,
                "type": 0
              }
            },
            {
              "OFPPortStatsPropOptical": {
                "bias_current": 300,
                "flags": 3,
                "length": 44,
                "rx_freq_lmda": 1500,
                "rx_grid_span": 500,
                "rx_offset": 700,
                "rx_pwr": 2000,
                "temperature": 273,
                "tx_freq_lmda": 1500,
                "tx_grid_span": 500,
                "tx_offset": 700,
                "tx_pwr": 2000,
                "type": 1
              }
            }
          ]
        }
      }
    ]
  }
}

```

(continues on next page)

(continued from previous page)

```

    }
  },
  {
    "OFPPortStatsPropExperimenter": {
      "data": [],
      "exp_type": 0,
      "experimenter": 101,
      "length": 12,
      "type": 65535
    }
  },
  {
    "OFPPortStatsPropExperimenter": {
      "data": [
        1
      ],
      "exp_type": 1,
      "experimenter": 101,
      "length": 16,
      "type": 65535
    }
  },
  {
    "OFPPortStatsPropExperimenter": {
      "data": [
        1,
        2
      ],
      "exp_type": 2,
      "experimenter": 101,
      "length": 20,
      "type": 65535
    }
  }
],
"rx_bytes": 0,
"rx_dropped": 0,
"rx_errors": 0,
"rx_packets": 0,
"tx_bytes": 336,
"tx_dropped": 0,
"tx_errors": 0,
"tx_packets": 4
}
},
{
  "OFPPortStats": {
    "duration_nsec": 0,
    "duration_sec": 0,

```

(continues on next page)



(continued from previous page)

```

    "length": 120,
    "port_no": 6,
    "properties": [
        {
            "OFPPortStatsPropEthernet": {
                "collisions": 0,
                "length": 40,
                "rx_crc_err": 0,
                "rx_frame_err": 0,
                "rx_over_err": 0,
                "type": 0
            }
        }
    ],
    "rx_bytes": 336,
    "rx_dropped": 0,
    "rx_errors": 0,
    "rx_packets": 4,
    "tx_bytes": 336,
    "tx_dropped": 0,
    "tx_errors": 0,
    "tx_packets": 4
}
],
"flags": 0,
"type": 4
}
}

```

```

class os_ken.ofproto.ofproto_v1_5_parser.OFPPortDescStatsRequest(datapath,
                                                                    flags=0,
                                                                    port_no=4294967295,
                                                                    type_=None)

```

Port description request message

The controller uses this message to query description of one or all the ports.

Attribute	Description
flags	Zero or OFPMPF_REQ_MORE
port_no	Port number to read (OFPP_ANY to all ports)

Example:

```

def send_port_desc_stats_request(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

```

(continues on next page)

(continued from previous page)

```
req = ofp_parser.OFPPortDescStatsRequest(datapath, 0, ofp.OFPP_ANY)
datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPPortDescStatsRequest": {
    "flags": 0,
    "port_no": 48346,
    "type": 13
  }
}
```

```
class os_ken.ofproto.ofproto_v1_5_parser.OFPPortDescStatsReply(datapath,
                                                                type_=None,
                                                                **kwargs)
```

Port description reply message

The switch responds with this message to a port description request.

Attribute	Description
body	List of OFPPort instance

Example:

```
@set_ev_cls(ofp_event.EventOFPPortDescStatsReply, MAIN_DISPATCHER)
def port_desc_stats_reply_handler(self, ev):
    ports = []
    for p in ev.msg.body:
        ports.append('port_no=%d hw_addr=%s name=%s config=0x%08x '
                    'state=0x%08x properties=%s' %
                    (p.port_no, p.hw_addr,
                     p.name, p.config, p.state, repr(p.properties)))
    self.logger.debug('OFPPortDescStatsReply received: %s', ports)
```

JSON Example:

```
{
  "OFPPortDescStatsReply": {
    "body": [
      {
        "OFPPort": {
          "config": 0,
          "hw_addr": "f2:0b:a4:d0:3f:70",
          "length": 168,
          "name": "Port7",
          "port_no": 7,
          "properties": [
            {

```

(continues on next page)

(continued from previous page)

```

"OFPPortDescPropEthernet": {
    "advertised": 10240,
    "curr": 10248,
    "curr_speed": 5000,
    "length": 32,
    "max_speed": 5000,
    "peer": 10248,
    "supported": 10248,
    "type": 0
}
},
{
    "OFPPortDescPropOptical": {
        "length": 40,
        "rx_grid_freq_lmda": 1500,
        "rx_max_freq_lmda": 2000,
        "rx_min_freq_lmda": 1000,
        "supported": 1,
        "tx_grid_freq_lmda": 1500,
        "tx_max_freq_lmda": 2000,
        "tx_min_freq_lmda": 1000,
        "tx_pwr_max": 2000,
        "tx_pwr_min": 1000,
        "type": 1
    }
},
{
    "OFPPortDescPropExperimenter": {
        "data": [],
        "exp_type": 0,
        "experimenter": 101,
        "length": 12,
        "type": 65535
    }
},
{
    "OFPPortDescPropExperimenter": {
        "data": [
            1
        ],
        "exp_type": 1,
        "experimenter": 101,
        "length": 16,
        "type": 65535
    }
},
{
    "OFPPortDescPropExperimenter": {
        "data": [

```

(continues on next page)

(continued from previous page)

```

        1,
        2
    ],
    "exp_type": 2,
    "experimenter": 101,
    "length": 20,
    "type": 65535
}
}
],
"state": 4
}
},
{
    "OFPPort": {
        "config": 0,
        "hw_addr": "f2:0b:a4:7d:f8:ea",
        "length": 72,
        "name": "Port6",
        "port_no": 6,
        "properties": [
            {
                "OFPPortDescPropEthernet": {
                    "advertised": 10240,
                    "curr": 10248,
                    "curr_speed": 5000,
                    "length": 32,
                    "max_speed": 5000,
                    "peer": 10248,
                    "supported": 10248,
                    "type": 0
                }
            }
        ],
        "state": 4
    }
}
],
"flags": 0,
"type": 13
}
}
}

```

```

class os_ken.ofproto.ofproto_v1_5_parser.OFPQueueStatsRequest(datapath, flags=0,
                                                                port_no=4294967295,
                                                                queue_id=4294967295,
                                                                type_=None)

```

Queue statistics request message

The controller uses this message to query queue statistics.

Attribute	Description
flags	Zero or OFPMPF_REQ_MORE
port_no	Port number to read
queue_id	ID of queue to read

Example:

```
def send_queue_stats_request(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPQueueStatsRequest(datapath, 0, ofp.OFPP_ANY,
                                           ofp.OFPQ_ALL)

    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPQueueStatsRequest": {
    "flags": 0,
    "port_no": 43981,
    "queue_id": 4294967295,
    "type": 5
  }
}
```

**class** `os_ken.ofproto.ofproto_v1_5_parser.OFPQueueStatsReply`(*datapath*, *type\_=None*, *\*\*kwargs*)

Queue statistics reply message

The switch responds with this message to an aggregate flow statistics request.

Attribute	Description
body	List of OFPQueueStats instance

Example:

```
@set_ev_cls(ofp_event.EventOFPQueueStatsReply, MAIN_DISPATCHER)
def queue_stats_reply_handler(self, ev):
    queues = []
    for stat in ev.msg.body:
        queues.append('port_no=%d queue_id=%d '
                    'tx_bytes=%d tx_packets=%d tx_errors=%d '
                    'duration_sec=%d duration_nsec=%d'
                    'properties=%s' %
                    (stat.port_no, stat.queue_id,
                     stat.tx_bytes, stat.tx_packets, stat.tx_errors,
                     stat.duration_sec, stat.duration_nsec,
```

(continues on next page)

(continued from previous page)

```
repr(stat.properties)))
self.logger.debug('QueueStats: %s', queues)
```

JSON Example:

```
{
  "OFPQueueStatsReply": {
    "body": [
      {
        "OFPQueueStats": {
          "duration_nsec": 0,
          "duration_sec": 0,
          "length": 104,
          "port_no": 7,
          "properties": [
            {
              "OFPQueueStatsPropExperimenter": {
                "data": [],
                "exp_type": 0,
                "experimenter": 101,
                "length": 12,
                "type": 65535
              }
            },
            {
              "OFPQueueStatsPropExperimenter": {
                "data": [
                  1
                ],
                "exp_type": 1,
                "experimenter": 101,
                "length": 16,
                "type": 65535
              }
            },
            {
              "OFPQueueStatsPropExperimenter": {
                "data": [
                  1,
                  2
                ],
                "exp_type": 2,
                "experimenter": 101,
                "length": 20,
                "type": 65535
              }
            }
          ],
          "queue_id": 1,
          "tx_bytes": 0,
```

(continues on next page)

(continued from previous page)

```

        "tx_errors": 0,
        "tx_packets": 0
    },
    {
        "OFPQueueStats": {
            "duration_nsec": 0,
            "duration_sec": 0,
            "length": 48,
            "port_no": 6,
            "properties": [],
            "queue_id": 1,
            "tx_bytes": 0,
            "tx_errors": 0,
            "tx_packets": 0
        }
    },
    {
        "OFPQueueStats": {
            "duration_nsec": 0,
            "duration_sec": 0,
            "length": 48,
            "port_no": 7,
            "properties": [],
            "queue_id": 2,
            "tx_bytes": 0,
            "tx_errors": 0,
            "tx_packets": 0
        }
    }
],
"flags": 0,
"type": 5
}
}

```

```

class os_ken.ofproto.ofproto_v1_5_parser.OFPQueueDescStatsRequest(datapath,
                                                                    flags=0,
                                                                    port_no=4294967295,
                                                                    queue_id=4294967295,
                                                                    type_=None)

```

Queue description request message

The controller uses this message to query description of all the queues.

Attribute	Description
flags	Zero or OFPMPF_REQ_MORE
port_no	Port number to read (OFPP_ANY for all ports)
queue_id	ID of queue to read (OFPQ_ALL for all queues)

Example:

```
def send_queue_desc_stats_request(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPQueueDescStatsRequest(datapath, 0,
                                                ofp.OFPP_ANY,
                                                ofp.OFPQ_ALL)

    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPQueueDescStatsRequest": {
    "flags": 0,
    "port_no": 52651,
    "queue_id": 57020,
    "type": 15
  }
}
```

```
class os_ken.ofproto.ofproto_v1_5_parser.OFPQueueDescStatsReply(datapath,
                                                                type_=None,
                                                                **kwargs)
```

Queue description reply message

The switch responds with this message to a queue description request.

Attribute	Description
body	List of OFPQueueDesc instance

Example:

```
@set_ev_cls(ofp_event.EventOFPQueueDescStatsReply, MAIN_DISPATCHER)
def queue_desc_stats_reply_handler(self, ev):
    queues = []
    for q in ev.msg.body:
        queues.append('port_no=%d queue_id=0x%08x properties=%s' %
                     (q.port_no, q.queue_id, repr(q.properties)))
    self.logger.debug('OFPQueueDescStatsReply received: %s', queues)
```

JSON Example:

```
{
  "OFPQueueDescStatsReply": {
    "body": [
      {
        "OFPQueueDesc": {
          "len": 32,

```

(continues on next page)



(continued from previous page)

```

    "port_no": 7,
    "properties": [
      {
        "OFPQueueDescPropExperimenter": {
          "data": [],
          "exp_type": 0,
          "experimenter": 101,
          "length": 12,
          "type": 65535
        }
      }
    ],
    "queue_id": 0
  },
  {
    "OFPQueueDesc": {
      "len": 88,
      "port_no": 8,
      "properties": [
        {
          "OFPQueueDescPropMinRate": {
            "length": 8,
            "rate": 300,
            "type": 1
          }
        },
        {
          "OFPQueueDescPropMaxRate": {
            "length": 8,
            "rate": 900,
            "type": 2
          }
        },
        {
          "OFPQueueDescPropExperimenter": {
            "data": [],
            "exp_type": 0,
            "experimenter": 101,
            "length": 12,
            "type": 65535
          }
        }
      ]
    },
    {
      "OFPQueueDescPropExperimenter": {
        "data": [
          1
        ],
        "exp_type": 1,

```

(continues on next page)

(continued from previous page)

```

        "experimenter": 101,
        "length": 16,
        "type": 65535
    }
},
{
    "OFPQueueDescPropExperimenter": {
        "data": [
            1,
            2
        ],
        "exp_type": 2,
        "experimenter": 101,
        "length": 20,
        "type": 65535
    }
},
],
"queue_id": 1
}
}
],
"flags": 0,
"type": 15
}
}

```

```

class os_ken.ofproto.ofproto_v1_5_parser.OFPGroupStatsRequest(datapath, flags=0,
                                                                group_id=4294967292,
                                                                type_=None)

```

Group statistics request message

The controller uses this message to query statistics of one or more groups.

Attribute	Description
flags	Zero or OFPMPF_REQ_MORE
group_id	ID of group to read (OFPG_ALL to all groups)

Example:

```

def send_group_stats_request(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPGroupStatsRequest(datapath, 0, ofp.OFPG_ALL)
    datapath.send_msg(req)

```

JSON Example:

```
{
  "OFPGroupStatsRequest": {
    "flags": 0,
    "group_id": 4294967292,
    "type": 6
  }
}
```

```
class os_ken.ofproto.ofproto_v1_5_parser.OFPGroupStatsReply(datapath, type_=None,
                                                            **kwargs)
```

Group statistics reply message

The switch responds with this message to a group statistics request.

Attribute	Description
body	List of OFPGroupStats instance

Example:

```
@set_ev_cls(ofp_event.EventOFPGroupStatsReply, MAIN_DISPATCHER)
def group_stats_reply_handler(self, ev):
    groups = []
    for stat in ev.msg.body:
        groups.append('length=%d group_id=%d '
                     'ref_count=%d packet_count=%d byte_count=%d '
                     'duration_sec=%d duration_nsec=%d' %
                     (stat.length, stat.group_id,
                      stat.ref_count, stat.packet_count,
                      stat.byte_count, stat.duration_sec,
                      stat.duration_nsec))
    self.logger.debug('GroupStats: %s', groups)
```

JSON Example:

```
{
  "OFPGroupStatsReply": {
    "body": [
      {
        "OFPGroupStats": {
          "bucket_stats": [
            {
              "OFPBucketCounter": {
                "byte_count": 2345,
                "packet_count": 234
              }
            }
          ],
          "byte_count": 12345,
          "duration_nsec": 609036000,

```

(continues on next page)

(continued from previous page)

```

        "duration_sec": 9,
        "group_id": 1,
        "length": 56,
        "packet_count": 123,
        "ref_count": 2
    }
}
],
"flags": 0,
"type": 6
}
}

```

```

class os_ken.ofproto.ofproto_v1_5_parser.OFPGGroupDescStatsRequest(datapath,
                                                                    flags=0,
                                                                    group_id=4294967292,
                                                                    type_=None)

```

Group description request message

The controller uses this message to list the set of groups on a switch.

Attribute	Description
flags	Zero or OFPMPF_REQ_MORE
group_id	ID of group to read (OFPG_ALL to all groups)

Example:

```

def send_group_desc_stats_request(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPGGroupDescStatsRequest(datapath, 0, ofp.OFPG_ALL)
    datapath.send_msg(req)

```

JSON Example:

```

{
  "OFPGGroupDescStatsRequest": {
    "flags": 0,
    "group_id": 52651,
    "type": 7
  }
}

```

```

class os_ken.ofproto.ofproto_v1_5_parser.OFPGGroupDescStatsReply(datapath,
                                                                    type_=None,
                                                                    **kwargs)

```

Group description reply message

The switch responds with this message to a group description request.

Attribute	Description
body	List of OFPGroupDescStats instance

Example:

```
@set_ev_cls(ofp_event.EventOFPGroupDescStatsReply, MAIN_DISPATCHER)
def group_desc_stats_reply_handler(self, ev):
    descs = []
    for stat in ev.msg.body:
        descs.append('length=%d type=%d group_id=%d '
                    'buckets=%s properties=%s' %
                    (stat.length, stat.type, stat.group_id,
                     stat.bucket, repr(stat.properties)))
    self.logger.debug('GroupDescStats: %s', descs)
```

JSON Example:

```
{
  "OFPGroupDescStatsReply": {
    "body": [
      {
        "OFPGroupDescStats": {
          "bucket_array_len": 32,
          "buckets": [
            {
              "OFPBucket": {
                "action_array_len": 16,
                "actions": [
                  {
                    "OFPActionOutput": {
                      "len": 16,
                      "max_len": 65509,
                      "port": 1,
                      "type": 0
                    }
                  }
                ],
                "bucket_id": 65535,
                "len": 32,
                "properties": [
                  {
                    "OFPGroupBucketPropWeight": {
                      "length": 8,
                      "type": 0,
                      "weight": 65535
                    }
                  }
                ]
              }
            }
          ]
        }
      }
    ]
  }
}
```

(continues on next page)

(continued from previous page)

```

    }
    ],
    "group_id": 1,
    "length": 48,
    "properties": [],
    "type": 1
  }
}
],
"flags": 0,
"type": 7
}
}

```

```

class os_ken.ofproto.ofproto_v1_5_parser.OFPGroupFeaturesStatsRequest(datapath,
                                                                    flags=0,
                                                                    type_=None)

```

Group features request message

The controller uses this message to list the capabilities of groups on a switch.

Attribute	Description
flags	Zero or OFPMPF_REQ_MORE

Example:

```

def send_group_features_stats_request(self, datapath):
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPGroupFeaturesStatsRequest(datapath, 0)
    datapath.send_msg(req)

```

JSON Example:

```

{
  "OFPGroupFeaturesStatsRequest": {
    "flags": 0,
    "type": 8
  }
}

```

```

class os_ken.ofproto.ofproto_v1_5_parser.OFPGroupFeaturesStatsReply(datapath,
                                                                    type_=None,
                                                                    **kwargs)

```

Group features reply message

The switch responds with this message to a group features request.

Attribute	Description
body	Instance of OFPGroupFeaturesStats

Example:

```
@set_ev_cls(ofp_event.EventOFPGroupFeaturesStatsReply, MAIN_DISPATCHER)
def group_features_stats_reply_handler(self, ev):
    body = ev.msg.body

    self.logger.debug('GroupFeaturesStats: types=%d '
                      'capabilities=0x%08x max_groups=%s '
                      'actions=%s',
                      body.types, body.capabilities,
                      body.max_groups, body.actions)
```

JSON Example:

```
{
  "OFPGroupFeaturesStatsReply": {
    "body": {
      "OFPGroupFeaturesStats": {
        "actions": [
          67082241,
          67082241,
          67082241,
          67082241
        ],
        "capabilities": 5,
        "max_groups": [
          16777216,
          16777216,
          16777216,
          16777216
        ],
        "types": 15
      }
    },
    "flags": 0,
    "type": 8
  }
}
```

```
class os_ken.ofproto.ofproto_v1_5_parser.OFPMeterStatsRequest(datapath, flags=0,
                                                                meter_id=4294967295,
                                                                type_=None)
```

Meter statistics request message

The controller uses this message to query statistics for one or more meters.

Attribute	Description
flags	Zero or OFPMPF_REQ_MORE
meter_id	ID of meter to read (OFPM_ALL to all meters)

Example:

```
def send_meter_stats_request(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPMeterStatsRequest(datapath, 0, ofp.OFPM_ALL)
    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPMeterStatsRequest": {
    "flags": 0,
    "meter_id": 4294967295,
    "type": 9
  }
}
```

**class** `os_ken.ofproto.ofproto_v1_5_parser.OFPMeterStatsReply`(*datapath*, *type\_=None*, *\*\*kwargs*)

Meter statistics reply message

The switch responds with this message to a meter statistics request.

Attribute	Description
body	List of OFPMeterStats instance

Example:

```
@set_ev_cls(ofp_event.EventOFPMeterStatsReply, MAIN_DISPATCHER)
def meter_stats_reply_handler(self, ev):
    meters = []
    for stat in ev.msg.body:
        meters.append('meter_id=0x%08x len=%d ref_count=%d '
                     'packet_in_count=%d byte_in_count=%d '
                     'duration_sec=%d duration_nsec=%d '
                     'band_stats=%s' %
                     (stat.meter_id, stat.len, stat.ref_count,
                      stat.packet_in_count, stat.byte_in_count,
                      stat.duration_sec, stat.duration_nsec,
                      stat.band_stats))
    self.logger.debug('MeterStats: %s', meters)
```

JSON Example:



```

{
  "OFPMeterStatsReply": {
    "body": [
      {
        "OFPMeterStats": {
          "band_stats": [
            {
              "OFPMeterBandStats": {
                "byte_band_count": 0,
                "packet_band_count": 0
              }
            }
          ],
          "byte_in_count": 0,
          "duration_nsec": 480000,
          "duration_sec": 0,
          "ref_count": 0,
          "len": 56,
          "meter_id": 100,
          "packet_in_count": 0
        }
      }
    ],
    "flags": 0,
    "type": 9
  }
}

```

```

class os_ken.ofproto.ofproto_v1_5_parser.OFPMeterDescStatsRequest(datapath,
                                                                    flags=0, me-
                                                                    ter_id=4294967295,
                                                                    type_=None)

```

Meter description statistics request message

The controller uses this message to query configuration for one or more meters.

Attribute	Description
flags	Zero or OFPMPF_REQ_MORE
meter_id	ID of meter to read (OFPM_ALL to all meters)

Example:

```

def send_meter_desc_stats_request(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPMeterDescStatsRequest(datapath, 0,
                                              ofp.OFPM_ALL)

    datapath.send_msg(req)

```

JSON Example:

```
{
  "OFPMeterDescStatsRequest": {
    "flags": 0,
    "meter_id": 4294967295,
    "type": 10
  }
}
```

```
class os_ken.ofproto.ofproto_v1_5_parser.OFPMeterDescStatsReply(datapath,
                                                                type_=None,
                                                                **kwargs)
```

Meter description statistics reply message

The switch responds with this message to a meter description statistics request.

Attribute	Description
body	List of OFPMeterDescStats instance

Example:

```
@set_ev_cls(ofp_event.EventOFPMeterDescStatsReply, MAIN_DISPATCHER)
def meter_desc_stats_reply_handler(self, ev):
    configs = []
    for stat in ev.msg.body:
        configs.append('length=%d flags=0x%04x meter_id=0x%08x '
                      'bands=%s' %
                      (stat.length, stat.flags, stat.meter_id,
                       stat.bands))
    self.logger.debug('MeterDescStats: %s', configs)
```

JSON Example:

```
{
  "OFPMeterDescStatsReply": {
    "body": [
      {
        "OFPMeterDescStats": {
          "bands": [
            {
              "OFPMeterBandDrop": {
                "burst_size": 10,
                "len": 16,
                "rate": 1000,
                "type": 1
              }
            }
          ],
          "flags": 14,
```

(continues on next page)

(continued from previous page)

```

        "length": 24,
        "meter_id": 100
    }
}
],
"flags": 0,
"type": 10
}
}

```

```

class os_ken.ofproto.ofproto_v1_5_parser.OFPMeterFeaturesStatsRequest(datapath,
                                                                    flags=0,
                                                                    type_=None)

```

Meter features statistics request message

The controller uses this message to query the set of features of the metering subsystem.

Attribute	Description
flags	Zero or OFPMPF_REQ_MORE

Example:

```

def send_meter_features_stats_request(self, datapath):
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPMeterFeaturesStatsRequest(datapath, 0)
    datapath.send_msg(req)

```

JSON Example:

```

{
  "OFPMeterFeaturesStatsRequest": {
    "flags": 0,
    "type": 11
  }
}

```

```

class os_ken.ofproto.ofproto_v1_5_parser.OFPMeterFeaturesStatsReply(datapath,
                                                                    type_=None,
                                                                    **kwargs)

```

Meter features statistics reply message

The switch responds with this message to a meter features statistics request.

Attribute	Description
body	List of OFPMeterFeaturesStats instance

Example:

```

@set_ev_cls(ofp_event.EventOFPMeterFeaturesStatsReply, MAIN_DISPATCHER)
def meter_features_stats_reply_handler(self, ev):
    features = []
    for stat in ev.msg.body:
        features.append('max_meter=%d band_types=0x%08x '
                       'capabilities=0x%08x max_bands=%d '
                       'max_color=%d' %
                       (stat.max_meter, stat.band_types,
                        stat.capabilities, stat.max_bands,
                        stat.max_color))
    self.logger.debug('MeterFeaturesStats: %s', features)

```

JSON Example:

```

{
  "OFPMeterFeaturesStatsReply": {
    "body": [
      {
        "OFPMeterFeaturesStats": {
          "band_types": 2147483654,
          "capabilities": 15,
          "features": 3,
          "max_bands": 255,
          "max_color": 0,
          "max_meter": 16777216
        }
      }
    ],
    "flags": 0,
    "type": 11
  }
}

```

```

class os_ken.ofproto.ofproto_v1_5_parser.OFPControllerStatusStatsRequest(datapath,
                                                                           flags=0,
                                                                           type_=None)

```

Controller status multipart request message

The controller uses this message to request the status, the roles and the control channels of other controllers configured on the switch.

Attribute	Description
flags	Zero or OFPMPF_REQ_MORE

Example:

```

def send_controller_status_multipart_request(self, datapath):
    ofp_parser = datapath.ofproto_parser

```

(continues on next page)

(continued from previous page)

```
req = ofp_parser.OFPPortDescStatsRequest(datapath, 0)
datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPControllerStatusStatsRequest": {
    "flags": 0,
    "type": 18
  }
}
```

```
class os_ken.ofproto.ofproto_v1_5_parser.OFPControllerStatusStatsReply(datapath,
                                                                    type_=None,
                                                                    **kwargs)
```

Controller status multipart reply message

The switch responds with this message to a controller status multipart request.

Attribute	Description
body	List of OFPControllerStatus instance

Example:

```
@set_ev_cls(ofp_event.EventOFPControllerStatusStatsReply,
            MAIN_DISPATCHER)
def controller_status_multipart_reply_handler(self, ev):
    status = []
    for s in ev.msg.body:
        status.append('short_id=%d role=%d reason=%d '
                    'channel_status=%d properties=%s' %
                    (s.short_id, s.role, s.reason,
                     s.channel_status, repr(s.properties)))
    self.logger.debug('OFPControllerStatusStatsReply received: %s',
                    status)
```

JSON Example:

```
{
  "OFPControllerStatusStatsReply": {
    "body": [
      {
        "OFPControllerStatusStats": {
          "channel_status": 1,
          "length": 48,
          "properties": [
            {
              "OFPControllerStatusPropUri": {
```

(continues on next page)

(continued from previous page)

```

        "length": 26,
        "type": 0,
        "uri": "tls:192.168.34.23:6653"
    }
}
],
"reason": 1,
"role": 1,
"short_id": 65535
}
}
],
"flags": 0,
"type": 18
}
}

```

```
class os_ken.ofproto.ofproto_v1_5_parser.OFPTableStatsRequest(datapath, flags,
                                                             type_=None)
```

Table statistics request message

The controller uses this message to query flow table statistics.

Attribute	Description
flags	Zero or OFPMPF_REQ_MORE

Example:

```
def send_table_stats_request(self, datapath):
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPTableStatsRequest(datapath, 0)
    datapath.send_msg(req)
```

JSON Example:

```

{
  "OFPTableStatsRequest": {
    "flags": 0,
    "type": 3
  }
}

```

```
class os_ken.ofproto.ofproto_v1_5_parser.OFPTableStatsReply(datapath, type_=None,
                                                             **kwargs)
```

Table statistics reply message

The switch responds with this message to a table statistics request.

Attribute	Description
body	List of OFPTableStats instance

Example:

```
@set_ev_cls(ofp_event.EventOFPTableStatsReply, MAIN_DISPATCHER)
def table_stats_reply_handler(self, ev):
    tables = []
    for stat in ev.msg.body:
        tables.append('table_id=%d active_count=%d lookup_count=%d '
                    ' matched_count=%d' %
                    (stat.table_id, stat.active_count,
                     stat.lookup_count, stat.matched_count))
    self.logger.debug('TableStats: %s', tables)
```

JSON Example:

```
{
  "OFPTableStatsReply": {
    "body": [
      {
        "OFPTableStats": {
          "active_count": 4,
          "lookup_count": 4,
          "matched_count": 4,
          "table_id": 0
        }
      },
      {
        "OFPTableStats": {
          "active_count": 4,
          "lookup_count": 4,
          "matched_count": 4,
          "table_id": 1
        }
      }
    ],
    "flags": 0,
    "type": 3
  }
}
```

```
class os_ken.ofproto.ofproto_v1_5_parser.OFPTableDescStatsRequest(datapath,
                                                                    flags=0,
                                                                    type_=None)
```

Table description request message

The controller uses this message to query description of all the tables.

Attribute	Description
flags	Zero or OFPMPF_REQ_MORE

Example:

```
def send_table_desc_stats_request(self, datapath):
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPTableDescStatsRequest(datapath, 0)
    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPTableDescStatsRequest": {
    "flags": 0,
    "type": 14
  }
}
```

```
class os_ken.ofproto.ofproto_v1_5_parser.OFPTableDescStatsReply(datapath,
                                                                type_=None,
                                                                **kwargs)
```

Table description reply message

The switch responds with this message to a table description request.

Attribute	Description
body	List of OFPTableDesc instance

Example:

```
@set_ev_cls(ofp_event.EventOFPTableDescStatsReply, MAIN_DISPATCHER)
def table_desc_stats_reply_handler(self, ev):
    tables = []
    for p in ev.msg.body:
        tables.append('table_id=%d config=0x%08x properties=%s' %
                     (p.table_id, p.config, repr(p.properties)))
    self.logger.debug('OFPTableDescStatsReply received: %s', tables)
```

JSON Example:

```
{
  "OFPTableDescStatsReply": {
    "body": [
      {
        "OFPTableDesc": {
          "config": 0,
          "length": 24,

```

(continues on next page)



(continued from previous page)

```

        "properties": [
            {
                "OFPTableModPropExperimenter": {
                    "data": [],
                    "exp_type": 0,
                    "experimenter": 101,
                    "length": 12,
                    "type": 65535
                }
            }
        ],
        "table_id": 7
    },
    {
        "OFPTableDesc": {
            "config": 0,
            "length": 80,
            "properties": [
                {
                    "OFPTableModPropEviction": {
                        "flags": 0,
                        "length": 8,
                        "type": 2
                    }
                },
                {
                    "OFPTableModPropVacancy": {
                        "length": 8,
                        "type": 3,
                        "vacancy": 0,
                        "vacancy_down": 0,
                        "vacancy_up": 0
                    }
                }
            ],
            {
                "OFPTableModPropExperimenter": {
                    "data": [],
                    "exp_type": 0,
                    "experimenter": 101,
                    "length": 12,
                    "type": 65535
                }
            },
            {
                "OFPTableModPropExperimenter": {
                    "data": [
                        1
                    ],

```

(continues on next page)

(continued from previous page)

```

        "exp_type": 1,
        "experimenter": 101,
        "length": 16,
        "type": 65535
    }
},
{
    "OFPTableModPropExperimenter": {
        "data": [
            1,
            2
        ],
        "exp_type": 2,
        "experimenter": 101,
        "length": 20,
        "type": 65535
    }
},
],
"table_id": 8
}
},
],
"flags": 0,
"type": 14
}
}

```

```

class os_ken.ofproto.ofproto_v1_5_parser.OFPTableFeaturesStatsRequest(datapath,
                                                                    flags=0,
                                                                    body=None,
                                                                    type_=None)

```

Table features statistics request message

The controller uses this message to query table features.

Attribute	Description
body	List of OFPTableFeaturesStats instances. The default is [].

JSON Example:

```

{
  "OFPTableFeaturesStatsRequest": {
    "body": [
      {
        "OFPTableFeaturesStats": {
          "capabilities": 4,
          "command": 1,
          "features": 1,

```

(continues on next page)

(continued from previous page)

```

        "length": 80,
        "max_entries": 255,
        "metadata_match": 18446744073709551615,
        "metadata_write": 18446744073709551615,
        "name": "table1",
        "properties": [
            {
                "OFPTableFeaturePropOxmValues": {
                    "length": 14,
                    "oxm_values": [
                        {
                            "OXMTlv": {
                                "field": "eth_src",
                                "mask": null,
                                "value": "aa:bb:cc:dd:ee:ff"
                            }
                        }
                    ],
                    "type": 22
                }
            }
        ],
        "table_id": 1
    },
    "flags": 0,
    "type": 12
}
}

```

```

class os_ken.ofproto.ofproto_v1_5_parser.OFPTableFeaturesStatsReply(datapath,
                                                                    type_=None,
                                                                    **kwargs)

```

Table features statistics reply message

The switch responds with this message to a table features statistics request.

Attribute	Description
body	List of OFPTableFeaturesStats instance

JSON Example:

```

{
  "OFPTableFeaturesStatsReply": {
    "body": [
      {
        "OFPTableFeaturesStats": {
          "capabilities": 4,

```

(continues on next page)

(continued from previous page)

```

"command": 1,
"features": 1,
"length": 80,
"max_entries": 255,
"metadata_match": 18446744073709551615,
"metadata_write": 18446744073709551615,
"name": "table1",
"properties": [
  {
    "OFPTTableFeaturePropOxmValues": {
      "length": 14,
      "oxm_values": [
        {
          "OXMTlv": {
            "field": "eth_src",
            "mask": null,
            "value": "aa:bb:cc:dd:ee:ff"
          }
        }
      ],
      "type": 22
    }
  },
  {
    "table_id": 1
  }
],
"flags": 0,
"type": 12
}

```

```

class os_ken.ofproto.ofproto_v1_5_parser.OFPFlowMonitorRequest(datapath, flags=0,
                                                                monitor_id=0,
                                                                out_port=4294967295,
                                                                out_group=4294967295,
                                                                monitor_flags=0,
                                                                table_id=255,
                                                                command=0,
                                                                match=None,
                                                                type_=None)

```

Flow monitor request message

The controller uses this message to query flow monitors.

Attribute	Description
flags	Zero or OFPMPF_REQ_MORE
monitor_id	Controller-assigned ID for this monitor
out_port	Require matching entries to include this as an output port
out_group	Require matching entries to include this as an output group
monitor_flags	Bitmap of the following flags.  OFPFPMF_INITIAL OFPFPMF_ADD OFPFPMF_REMOVED OFPFPMF_MODIFY OFPFPMF_INSTRUCTIONS OFPFPMF_NO_ABBREV OFPFPMF_ONLY_OWN
table_id	ID of table to monitor
command	One of the following values.  OFPFMC_ADD OFPFMC_MODIFY OFPFMC_DELETE
match	Instance of OFPMatch

Example:

```
def send_flow_monitor_request(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    monitor_flags = [ofp.OFPFPMF_INITIAL, ofp.OFPFPMF_ONLY_OWN]
    match = ofp_parser.OFPMatch(in_port=1)
    req = ofp_parser.OFPFlowMonitorRequest(datapath, 0, 10000,
                                           ofp.OFPP_ANY, ofp.OFPG_ANY,
                                           monitor_flags,
                                           ofp.OFPTT_ALL,
                                           ofp.OFPFMC_ADD, match)

    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPFlowMonitorRequest": {
    "command": 0,
    "flags": 0,
    "match": {
```

(continues on next page)

(continued from previous page)

```

        "OFPMatch": {
            "length": 14,
            "oxm_fields": [
                {
                    "OXMTlv": {
                        "field": "eth_dst",
                        "mask": null,
                        "value": "f2:0b:a4:7d:f8:ea"
                    }
                }
            ],
            "type": 1
        }
    },
    "monitor_flags": 15,
    "monitor_id": 1000000000,
    "out_group": 4294967295,
    "out_port": 22,
    "table_id": 33,
    "type": 16
}
}
}

```

```

class os_ken.ofproto.ofproto_v1_5_parser.OFPFlowMonitorReply(datapath, type_=None,
                                                             **kwargs)

```

Flow monitor reply message

The switch responds with this message to a flow monitor request.

Attribute	Description
body	List of list of the following class instance.  OFPFlowMonitorFull OFPFlowMonitorAbbrev OFPFlowMonitorPaused

Example:

```

@set_ev_cls(ofp_event.EventOFPFlowMonitorReply, MAIN_DISPATCHER)
def flow_monitor_reply_handler(self, ev):
    msg = ev.msg
    dp = msg.datapath
    ofp = dp.ofproto
    flow_updates = []

    for update in msg.body:
        update_str = 'length=%d event=%d' %

```

(continues on next page)

(continued from previous page)

```

        (update.length, update.event)
    if (update.event == ofp.OFPFME_INITIAL or
        update.event == ofp.OFPFME_ADDED or
        update.event == ofp.OFPFME_REMOVED or
        update.event == ofp.OFPFME_MODIFIED):
        update_str += 'table_id=%d reason=%d idle_timeout=%d '
                    'hard_timeout=%d priority=%d cookie=%d '
                    'match=%d instructions=%s' %
                    (update.table_id, update.reason,
                     update.idle_timeout, update.hard_timeout,
                     update.priority, update.cookie,
                     update.match, update.instructions)
    elif update.event == ofp.OFPFME_ABBREV:
        update_str += 'xid=%d' % (update.xid)
    flow_updates.append(update_str)
    self.logger.debug('FlowUpdates: %s', flow_updates)

```

JSON Example:

```

{
  "OFPPFlowMonitorReply": {
    "body": [
      {
        "OFPPFlowUpdateFull": {
          "cookie": 0,
          "event": 0,
          "hard_timeout": 700,
          "idle_timeout": 600,
          "instructions": [
            {
              "OFPInstructionActions": {
                "actions": [
                  {
                    "OFPActionOutput": {
                      "len": 16,
                      "max_len": 0,
                      "port": 4294967290,
                      "type": 0
                    }
                  }
                ],
                "len": 24,
                "type": 4
              }
            }
          ],
          "length": 64,
          "match": {
            "OFPMatch": {
              "length": 10,

```

(continues on next page)

(continued from previous page)

```

        "oxm_fields": [
            {
                "OXMTlv": {
                    "field": "eth_type",
                    "mask": null,
                    "value": 2054
                }
            },
            ],
        "type": 1
    },
    "priority": 3,
    "reason": 0,
    "table_id": 0
},
{
    "OFPPFlowUpdateAbbrev": {
        "event": 4,
        "length": 8,
        "xid": 1234
    }
},
{
    "OFPPFlowUpdatePaused": {
        "event": 5,
        "length": 8
    }
},
],
"flags": 0,
"type": 16
}
}

```

```

class os_ken.ofproto.ofproto_v1_5_parser.OFPBundleFeaturesStatsRequest(datapath,
                                                                    flags=0,
                                                                    fea-
                                                                    ture_request_flags=0,
                                                                    proper-
                                                                    ties=None,
                                                                    type_=None)

```

Bundle features request message

The controller uses this message to query a switch about its bundle capabilities, including whether it supports atomic bundles, ordered bundles, and scheduled bundles.



Attribute	Description
flags	Zero or OFPMPF_REQ_MORE
feature_request_flags	Bitmap of the following flags.  OFPBF_TIMESTAMP OFPBF_TIME_SET_SCHED
properties	List of OFPBundleFeaturesProp subclass instance

Example:

```
def send_bundle_features_stats_request(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPBundleFeaturesStatsRequest(datapath, 0)
    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPBundleFeaturesStatsRequest": {
    "feature_request_flags": 3,
    "flags": 0,
    "properties": [
      {
        "OFPBundleFeaturesPropTime": {
          "length": 72,
          "sched_accuracy": {
            "OFPTIME": {
              "nanoseconds": 1717986918,
              "seconds": 6148914691236517205
            }
          },
          "sched_max_future": {
            "OFPTIME": {
              "nanoseconds": 2290649224,
              "seconds": 8608480567731124087
            }
          },
          "sched_max_past": {
            "OFPTIME": {
              "nanoseconds": 2863311530,
              "seconds": 11068046444225730969
            }
          }
        },
        "timestamp": {
          "OFPTIME": {
```

(continues on next page)

(continued from previous page)

```

        "nanoseconds": 3435973836,
        "seconds": 13527612320720337851
    },
    },
    "type": 1
},
}
],
"type": 19
}
}

```

```

class os_ken.ofproto.ofproto_v1_5_parser.OFPBundleFeaturesStatsReply(datapath,
                                                                    type_=None,
                                                                    **kwargs)

```

Bundle features reply message

The switch responds with this message to a bundle features request.

Attribute	Description
body	Instance of OFPBundleFeaturesStats

Example:

```

@set_ev_cls(ofp_event.EventOFPBundleFeaturesStatsReply, MAIN_DISPATCHER)
def bundle_features_stats_reply_handler(self, ev):
    body = ev.msg.body

    self.logger.debug('OFPBundleFeaturesStats: capabilities=%0x%08x '
                      'properties=%s',
                      body.capabilities, repr(body.properties))

```

JSON Example:

```

{
  "OFPBundleFeaturesStatsReply": {
    "body": {
      "OFPBundleFeaturesStats": {
        "capabilities": 7,
        "properties": [
          {
            "OFPBundleFeaturesPropTime": {
              "length": 72,
              "sched_accuracy": {
                "OFPTime": {
                  "nanoseconds": 1717986918,
                  "seconds": 6148914691236517205
                }
              }
            }
          }
        ],
      }
    }
  },
}

```

(continues on next page)



(continued from previous page)

```

    "exp_type": 3405678728,
    "experimenter": 3735928495,
    "flags": 0,
    "type": 65535
  }
}

```

```

class os_ken.ofproto.ofproto_v1_5_parser.OFPExperimenterStatsReply(datapath,
                                                                    type_=None,
                                                                    **kwargs)

```

Experimenter multipart reply message

Attribute	Description
body	An OFPExperimenterMultipart instance

JSON Example:

```

{
  "OFPExperimenterStatsReply": {
    "body": {
      "OFPExperimenterMultipart": {
        "data": "dGVzdGRhdGE50Tk50Tk50Q==",
        "exp_type": 3405674359,
        "experimenter": 3735928495
      }
    },
    "flags": 0,
    "type": 65535
  }
}

```

## Packet-Out Message

```

class os_ken.ofproto.ofproto_v1_5_parser.OFPPacketOut(datapath, buffer_id=None,
                                                       match=None, actions=None,
                                                       data=None, actions_len=None)

```

Packet-Out message

The controller uses this message to send a packet out through the switch.

Attribute	Description
buffer_id	ID assigned by datapath (OFP_NO_BUFFER if none)
match	Instance of OFPMatch ( <code>in_port</code> is mandatory in the match field)
actions	list of OpenFlow action class
data	Packet data of a binary type value or an instances of packet.Packet.

Example:

```
def send_packet_out(self, datapath, buffer_id, in_port):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    match = OFPMatch(in_port=in_port)
    actions = [ofp_parser.OFPActionOutput(ofp.OFPP_FLOOD, 0)]
    req = ofp_parser.OFPPacketOut(datapath, buffer_id,
                                  match, actions)

    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPPacketOut": {
    "actions": [
      {
        "OFPActionOutput": {
          "len": 16,
          "max_len": 65535,
          "port": 4294967291,
          "type": 0
        }
      }
    ],
    "actions_len": 16,
    "buffer_id": 4294967295,
    "data": "dGVzdA==",
    "match": {
      "OFPMatch": {
        "length": 12,
        "oxm_fields": [
          {
            "OXMTlv": {
              "field": "in_port",
              "mask": null,
              "value": 4294967040
            }
          }
        ]
      },
      "type": 1
    }
  }
}
```

## Barrier Message

**class** os\_ken.ofproto.ofproto\_v1\_5\_parser.OFPBarrierRequest(*datapath*)

Barrier request message

The controller sends this message to ensure message dependencies have been met or receive noti-

fications for completed operations.

Example:

```
def send_barrier_request(self, datapath):
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPBarrierRequest(datapath)
    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPBarrierRequest": {}
}
```

**class** `os_ken.ofproto.ofproto_v1_5_parser.OFPBarrierReply(datapath)`

Barrier reply message

The switch responds with this message to a barrier request.

Example:

```
@set_ev_cls(ofp_event.EventOFPBarrierReply, MAIN_DISPATCHER)
def barrier_reply_handler(self, ev):
    self.logger.debug('OFPBarrierReply received')
```

JSON Example:

```
{
  "OFPBarrierReply": {}
}
```

## Role Request Message

**class** `os_ken.ofproto.ofproto_v1_5_parser.OFPRoleRequest(datapath, role=None, short_id=None, generation_id=None)`

Role request message

The controller uses this message to change its role.

Attribute	Description
role	One of the following values.  OFPCR_ROLE_NOCHANGE OFPCR_ROLE_EQUAL OFPCR_ROLE_MASTER OFPCR_ROLE_SLAVE
short_id	ID number for the controller. The default is OFPCID_UNDEFINED.
generation_id	Master Election Generation ID

Example:

```
def send_role_request(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPRoleRequest(datapath, ofp.OFPCR_ROLE_EQUAL,
                                    ofp.OFPCID_UNDEFINED, 0)

    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPRoleRequest": {
    "generation_id": 1234605616436508552,
    "role": 1,
    "short_id": 43690
  }
}
```

```
class os_ken.ofproto.ofproto_v1_5_parser.OFPRoleReply(datapath, role=None,
                                                       short_id=None,
                                                       generation_id=None)
```

Role reply message

The switch responds with this message to a role request.

Attribute	Description
role	One of the following values.  OFPCR_ROLE_NOCHANGE OFPCR_ROLE_EQUAL OFPCR_ROLE_MASTER OFPCR_ROLE_SLAVE
short_id	ID number for the controller. The default is OFPCID_UNDEFINED.
generation_id	Master Election Generation ID

Example:

```
@set_ev_cls(ofp_event.EventOFPCRoleReply, MAIN_DISPATCHER)
def role_reply_handler(self, ev):
    msg = ev.msg
    dp = msg.datapath
    ofp = dp.ofproto

    if msg.role == ofp.OFPCR_ROLE_NOCHANGE:
        role = 'NOCHANGE'
    elif msg.role == ofp.OFPCR_ROLE_EQUAL:
        role = 'EQUAL'
    elif msg.role == ofp.OFPCR_ROLE_MASTER:
        role = 'MASTER'
    elif msg.role == ofp.OFPCR_ROLE_SLAVE:
        role = 'SLAVE'
    else:
        role = 'unknown'

    self.logger.debug('OFPCRoleReply received: '
                      'role=%s short_id=%d, generation_id=%d',
                      role, msg.short_id, msg.generation_id)
```

JSON Example:

```
{
  "OFPCRoleReply": {
    "generation_id": 1234605616436508552,
    "role": 1,
    "short_id": 43690
  }
}
```



## Bundle Messages

```
class os_ken.ofproto.ofproto_v1_5_parser.OFPBundleCtrlMsg(datapath,
                                                         bundle_id=None,
                                                         type_=None, flags=None,
                                                         properties=None)
```

Bundle control message

The controller uses this message to create, destroy and commit bundles

Attribute	Description
bundle_id	Id of the bundle
type	One of the following values.  OFPBCT_OPEN_REQUEST OFPBCT_OPEN_REPLY OFPBCT_CLOSE_REQUEST OFPBCT_CLOSE_REPLY OFPBCT_COMMIT_REQUEST OFPBCT_COMMIT_REPLY OFPBCT_DISCARD_REQUEST OFPBCT_DISCARD_REPLY
flags	Bitmap of the following flags.  OFPBF_ATOMIC OFPBF_ORDERED
properties	List of OFPBundleProp subclass instance

Example:

```
def send_bundle_control(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPBundleCtrlMsg(datapath, 7,
                                       ofp.OFPBCT_OPEN_REQUEST,
                                       ofp.OFPBF_ATOMIC, [])

    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPBundleCtrlMsg": {
    "bundle_id": 99999999,
    "flags": 1,
    "properties": [],
    "type": 1
  }
}
```

(continues on next page)

(continued from previous page)

```
}
}
```

**class** `os_ken.ofproto.ofproto_v1_5_parser.OFPBundleAddMsg`(*datapath, bundle\_id, flags, message, properties*)

Bundle add message

The controller uses this message to add a message to a bundle

Attribute	Description
<code>bundle_id</code>	Id of the bundle
<code>flags</code>	Bitmap of the following flags.  OFPBF_ATOMIC OFPBF_ORDERED
<code>message</code>	MsgBase subclass instance
<code>properties</code>	List of OFPBundleProp subclass instance

Example:

```
def send_bundle_add_message(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    msg = ofp_parser.OFPRoleRequest(datapath, ofp.OFPCR_ROLE_EQUAL, 0)

    req = ofp_parser.OFPBundleAddMsg(datapath, 7, ofp.OFPBF_ATOMIC,
                                     msg, [])

    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPBundleAddMsg": {
    "bundle_id": 99999999,
    "flags": 1,
    "message": {
      "OFPFlowMod": {
        "buffer_id": 0,
        "command": 0,
        "cookie": 1311768467463790320,
        "cookie_mask": 18446744073709551615,
        "flags": 0,
        "hard_timeout": 0,
        "idle_timeout": 0,
        "importance": 39032,
        "instructions": [
          {
```

(continues on next page)

(continued from previous page)

```

"OFPInstructionActions": {
  "actions": [
    {
      "OFPActionPopVlan": {
        "len": 8,
        "type": 18
      }
    },
    {
      "OFPActionSetField": {
        "field": {
          "OXMTlv": {
            "field": "ipv4_dst",
            "mask": null,
            "value": "192.168.2.9"
          }
        },
        "len": 16,
        "type": 25
      }
    },
    {
      "NXActionLearn": {
        "cookie": 0,
        "experimenter": 8992,
        "fin_hard_timeout": 0,
        "fin_idle_timeout": 0,
        "flags": 0,
        "hard_timeout": 300,
        "idle_timeout": 0,
        "len": 96,
        "priority": 1,
        "specs": [
          {
            "NXFlowSpecMatch": {
              "dst": [
                "vlan_vid",
                0
              ],
              "n_bits": 12,
              "src": [
                "vlan_vid",
                0
              ]
            }
          },
          {
            "NXFlowSpecMatch": {
              "dst": [

```

(continues on next page)

(continued from previous page)

```

        "eth_dst_nxm",
        0
    ],
    "n_bits": 48,
    "src": [
        "eth_src_nxm",
        0
    ]
}
},
{
    "NXFlowSpecLoad": {
        "dst": [
            "vlan_vid",
            0
        ],
        "n_bits": 12,
        "src": 0
    }
},
{
    "NXFlowSpecLoad": {
        "dst": [
            "tunnel_id_nxm",
            0
        ],
        "n_bits": 64,
        "src": [
            "tunnel_id_nxm",
            0
        ]
    }
},
{
    "NXFlowSpecOutput": {
        "dst": "",
        "n_bits": 32,
        "src": [
            "in_port",
            0
        ]
    }
}
],
"subtype": 16,
"table_id": 99,
"type": 65535
}
}

```

(continues on next page)

(continued from previous page)

```
    ],
    "len": 128,
    "type": 4
  }
},
{
  "OFPInstructionGotoTable": {
    "len": 8,
    "table_id": 100,
    "type": 1
  }
}
],
"match": {
  "OFPMatch": {
    "length": 70,
    "oxm_fields": [
      {
        "OXMTlv": {
          "field": "in_port",
          "mask": null,
          "value": 43981
        }
      },
      {
        "OXMTlv": {
          "field": "eth_dst",
          "mask": null,
          "value": "aa:bb:cc:99:88:77"
        }
      },
      {
        "OXMTlv": {
          "field": "eth_type",
          "mask": null,
          "value": 2048
        }
      },
      {
        "OXMTlv": {
          "field": "vlan_vid",
          "mask": null,
          "value": 5095
        }
      },
      {
        "OXMTlv": {
          "field": "ipv4_dst",
          "mask": null,
```

(continues on next page)

(continued from previous page)

```

        "value": "192.168.2.1"
    },
    {
        "OXMTlv": {
            "field": "tunnel_id",
            "mask": null,
            "value": 50000
        }
    },
    {
        "OXMTlv": {
            "field": "tun_ipv4_src",
            "mask": null,
            "value": "192.168.2.3"
        }
    },
    {
        "OXMTlv": {
            "field": "tun_ipv4_dst",
            "mask": null,
            "value": "192.168.2.4"
        }
    }
],
"type": 1
}
},
"out_group": 0,
"out_port": 0,
"priority": 0,
"table_id": 2
},
"properties": []
}
}
}

```

### Set Asynchronous Configuration Message

**class** `os_ken.ofproto.ofproto_v1_5_parser.OFPSetAsync(datapath, properties=None)`

Set asynchronous configuration message

The controller sends this message to set the asynchronous messages that it wants to receive on a given OpenFlow channel.

Attribute	Description
<code>properties</code>	List of <code>OFPAsyncConfigProp</code> subclass instances

Example:

```
def send_set_async(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    properties = [
        ofp_parser.OFPAsyncConfigPropReasons(
            ofp.OFPACPT_PACKET_IN_SLAVE, 8,
            (1 << ofp.OFPR_APPLY_ACTION
             | 1 << ofp.OFPR_INVALID_TTL))
    ]
    req = ofp_parser.OFPSetAsync(datapath, properties)
    datapath.send_msg(req)
```

JSON Example:

```
{
  "OFPSetAsync": {
    "properties": [
      {
        "OFPAsyncConfigPropReasons": {
          "length": 8,
          "mask": 3,
          "type": 0
        }
      },
      {
        "OFPAsyncConfigPropReasons": {
          "length": 8,
          "mask": 3,
          "type": 1
        }
      },
      {
        "OFPAsyncConfigPropReasons": {
          "length": 8,
          "mask": 3,
          "type": 2
        }
      },
      {
        "OFPAsyncConfigPropReasons": {
          "length": 8,
          "mask": 3,
          "type": 3
        }
      },
      {
        "OFPAsyncConfigPropReasons": {
          "length": 8,
          "mask": 3,
```

(continues on next page)

(continued from previous page)

```
        "type": 4
      }
    },
    {
      "OFPAsyncConfigPropReasons": {
        "length": 8,
        "mask": 3,
        "type": 5
      }
    },
    {
      "OFPAsyncConfigPropReasons": {
        "length": 8,
        "mask": 3,
        "type": 6
      }
    },
    {
      "OFPAsyncConfigPropReasons": {
        "length": 8,
        "mask": 3,
        "type": 7
      }
    },
    {
      "OFPAsyncConfigPropReasons": {
        "length": 8,
        "mask": 24,
        "type": 8
      }
    },
    {
      "OFPAsyncConfigPropReasons": {
        "length": 8,
        "mask": 24,
        "type": 9
      }
    },
    {
      "OFPAsyncConfigPropReasons": {
        "length": 8,
        "mask": 3,
        "type": 10
      }
    },
    {
      "OFPAsyncConfigPropReasons": {
        "length": 8,
        "mask": 3,
```

(continues on next page)



(continued from previous page)

```

        "type": 11
    },
    {
        "OFPAsyncConfigPropExperimenter": {
            "data": [],
            "exp_type": 0,
            "experimenter": 101,
            "length": 12,
            "type": 65534
        }
    },
    {
        "OFPAsyncConfigPropExperimenter": {
            "data": [
                1
            ],
            "exp_type": 1,
            "experimenter": 101,
            "length": 16,
            "type": 65535
        }
    },
    {
        "OFPAsyncConfigPropExperimenter": {
            "data": [
                1,
                2
            ],
            "exp_type": 2,
            "experimenter": 101,
            "length": 20,
            "type": 65535
        }
    }
]
}
}

```

**class** `os_ken.ofproto.ofproto_v1_5_parser.OFPGetAsyncRequest(datapath)`

Get asynchronous configuration request message

The controller uses this message to query the asynchronous message.

Example:

```

def send_get_async_request(self, datapath):
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPGetAsyncRequest(datapath)
    datapath.send_msg(req)

```

JSON Example:

```
{
  "OFPGetAsyncRequest": {}
}
```

**class** `os_ken.ofproto.ofproto_v1_5_parser.OFPGetAsyncReply`(*datapath*,  
*properties=None*)

Get asynchronous configuration reply message

The switch responds with this message to a get asynchronous configuration request.

Attribute	Description
<code>properties</code>	List of <code>OFPAsyncConfigProp</code> subclass instances

Example:

```
@set_ev_cls(ofp_event.EventOFPGetAsyncReply, MAIN_DISPATCHER)
def get_async_reply_handler(self, ev):
    msg = ev.msg

    self.logger.debug('OFPGetAsyncReply received: '
                      'properties=%s', repr(msg.properties))
```

JSON Example:

```
{
  "OFPGetAsyncReply": {
    "properties": [
      {
        "OFPAsyncConfigPropReasons": {
          "length": 8,
          "mask": 3,
          "type": 0
        }
      },
      {
        "OFPAsyncConfigPropReasons": {
          "length": 8,
          "mask": 3,
          "type": 1
        }
      },
      {
        "OFPAsyncConfigPropReasons": {
          "length": 8,
          "mask": 3,
          "type": 2
        }
      }
    ]
  }
}
```

(continues on next page)

(continued from previous page)

```
{
  "OFPAsyncConfigPropReasons": {
    "length": 8,
    "mask": 3,
    "type": 3
  }
},
{
  "OFPAsyncConfigPropReasons": {
    "length": 8,
    "mask": 3,
    "type": 4
  }
},
{
  "OFPAsyncConfigPropReasons": {
    "length": 8,
    "mask": 3,
    "type": 5
  }
},
{
  "OFPAsyncConfigPropReasons": {
    "length": 8,
    "mask": 3,
    "type": 6
  }
},
{
  "OFPAsyncConfigPropReasons": {
    "length": 8,
    "mask": 3,
    "type": 7
  }
},
{
  "OFPAsyncConfigPropReasons": {
    "length": 8,
    "mask": 24,
    "type": 8
  }
},
{
  "OFPAsyncConfigPropReasons": {
    "length": 8,
    "mask": 24,
    "type": 9
  }
},
},
```

(continues on next page)

(continued from previous page)

```

    {
      "OFPAsyncConfigPropReasons": {
        "length": 8,
        "mask": 3,
        "type": 10
      }
    },
    {
      "OFPAsyncConfigPropReasons": {
        "length": 8,
        "mask": 3,
        "type": 11
      }
    },
    {
      "OFPAsyncConfigPropExperimenter": {
        "data": [],
        "exp_type": 0,
        "experimenter": 101,
        "length": 12,
        "type": 65534
      }
    },
    {
      "OFPAsyncConfigPropExperimenter": {
        "data": [
          1
        ],
        "exp_type": 1,
        "experimenter": 101,
        "length": 16,
        "type": 65535
      }
    },
    {
      "OFPAsyncConfigPropExperimenter": {
        "data": [
          1,
          2
        ],
        "exp_type": 2,
        "experimenter": 101,
        "length": 20,
        "type": 65535
      }
    }
  ]
}

```

## Asynchronous Messages

### Packet-In Message

```
class os_ken.ofproto.ofproto_v1_5_parser.OFPPacketIn(datapath, buffer_id=None,
                                                    total_len=None, reason=None,
                                                    table_id=None, cookie=None,
                                                    match=None, data=None)
```

Packet-In message

The switch sends the packet that received to the controller by this message.

Attribute	Description
buffer_id	ID assigned by datapath
total_len	Full length of frame
reason	Reason packet is being sent.  OFPR_TABLE_MISS OFPR_APPLY_ACTION OFPR_INVALID_TTL OFPR_ACTION_SET OFPR_GROUP OFPR_PACKET_OUT
table_id	ID of the table that was looked up
cookie	Cookie of the flow entry that was looked up
match	Instance of OFPMatch
data	Ethernet frame

Example:

```
@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def packet_in_handler(self, ev):
    msg = ev.msg
    dp = msg.datapath
    ofp = dp.ofproto

    if msg.reason == ofp.TABLE_MISS:
        reason = 'TABLE MISS'
    elif msg.reason == ofp.OFPR_APPLY_ACTION:
        reason = 'APPLY ACTION'
    elif msg.reason == ofp.OFPR_INVALID_TTL:
        reason = 'INVALID TTL'
    elif msg.reason == ofp.OFPR_ACTION_SET:
        reason = 'ACTION SET'
    elif msg.reason == ofp.OFPR_GROUP:
        reason = 'GROUP'
    elif msg.reason == ofp.OFPR_PACKET_OUT:
        reason = 'PACKET OUT'
```

(continues on next page)

(continued from previous page)

```

else:
    reason = 'unknown'

self.logger.debug('OFPPacketIn received: '
                  'buffer_id=%x total_len=%d reason=%s '
                  'table_id=%d cookie=%d match=%s data=%s',
                  msg.buffer_id, msg.total_len, reason,
                  msg.table_id, msg.cookie, msg.match,
                  utils.hex_array(msg.data))

```

JSON Example:

```

{
  "OFPPacketIn": {
    "buffer_id": 200,
    "cookie": 0,
    "data": "aG9nZQ==",
    "match": {
      "OFPMatch": {
        "length": 40,
        "oxm_fields": [
          {
            "OXMTlv": {
              "field": "in_port",
              "mask": null,
              "value": 43981
            }
          },
          {
            "OXMTlv": {
              "field": "tunnel_id",
              "mask": null,
              "value": 50000
            }
          },
          {
            "OXMTlv": {
              "field": "tun_ipv4_src",
              "mask": null,
              "value": "192.168.2.3"
            }
          },
          {
            "OXMTlv": {
              "field": "tun_ipv4_dst",
              "mask": null,
              "value": "192.168.2.4"
            }
          }
        ]
      }
    }
  },
  ],

```

(continues on next page)

(continued from previous page)

```

        "type": 1
    }
},
"reason": 0,
"table_id": 100,
"total_len": 1000
}
}

```

## Flow Removed Message

```

class os_ken.ofproto.ofproto_v1_5_parser.OFPFlowRemoved(datapath, table_id=None,
                                                         reason=None,
                                                         priority=None,
                                                         idle_timeout=None,
                                                         hard_timeout=None,
                                                         cookie=None, match=None,
                                                         stats=None)

```

Flow removed message

When flow entries time out or are deleted, the switch notifies controller with this message.

Attribute	Description
table_id	ID of the table
reason	One of the following values.  OFPRR_IDLE_TIMEOUT OFPRR_HARD_TIMEOUT OFPRR_DELETE OFPRR_GROUP_DELETE OFPRR_METER_DELETE OFPRR_EVICTION
priority	Priority level of flow entry
idle_timeout	Idle timeout from original flow mod
hard_timeout	Hard timeout from original flow mod
cookie	Opaque controller-issued identifier
match	Instance of OFPMatch
stats	Instance of OFPStats

Example:

```

@set_ev_cls(ofp_event.EventOFPFlowRemoved, MAIN_DISPATCHER)
def flow_removed_handler(self, ev):
    msg = ev.msg
    dp = msg.datapath
    ofp = dp.ofproto

```

(continues on next page)

(continued from previous page)

```

if msg.reason == ofp.OFPRR_IDLE_TIMEOUT:
    reason = 'IDLE TIMEOUT'
elif msg.reason == ofp.OFPRR_HARD_TIMEOUT:
    reason = 'HARD TIMEOUT'
elif msg.reason == ofp.OFPRR_DELETE:
    reason = 'DELETE'
elif msg.reason == ofp.OFPRR_GROUP_DELETE:
    reason = 'GROUP DELETE'
elif msg.reason == ofp.OFPRR_METER_DELETE:
    reason = 'METER DELETE'
elif msg.reason == ofp.OFPRR_EVICTION:
    reason = 'EVICTION'
else:
    reason = 'unknown'

self.logger.debug('OFPPFlowRemoved received: '
                  'table_id=%d reason=%s priority=%d '
                  'idle_timeout=%d hard_timeout=%d cookie=%d '
                  'match=%s stats=%s',
                  msg.table_id, reason, msg.priority,
                  msg.idle_timeout, msg.hard_timeout, msg.cookie,
                  msg.match, msg.stats)

```

JSON Example:

```

{
  "OFPPFlowRemoved": {
    "cookie": 1234605616436508552,
    "hard_timeout": 255,
    "idle_timeout": 255,
    "match": {
      "OFPMatch": {
        "length": 12,
        "oxm_fields": [
          {
            "OXMTlv": {
              "field": "in_port",
              "mask": null,
              "value": 1
            }
          }
        ],
        "type": 1
      }
    },
    "priority": 1,
    "reason": 0,
    "stats": {
      "OFPPStats": {

```

(continues on next page)



(continued from previous page)

```

        "length": 12,
        "oxs_fields": [
            {
                "OXSTlv": {
                    "field": "flow_count",
                    "value": 1
                }
            }
        ]
    },
    "table_id": 1
}
}

```

## Port Status Message

```
class os_ken.ofproto.ofproto_v1_5_parser.OFPPortStatus(datapath, reason=None,
                                                    desc=None)
```

Port status message

The switch notifies controller of change of ports.

Attribute	Description
reason	One of the following values.  OFPPR_ADD OFPPR_DELETE OFPPR_MODIFY
desc	instance of OFPPort

Example:

```

@set_ev_cls(ofp_event.EventOFPPortStatus, MAIN_DISPATCHER)
def port_status_handler(self, ev):
    msg = ev.msg
    dp = msg.datapath
    ofp = dp.ofproto

    if msg.reason == ofp.OFPPR_ADD:
        reason = 'ADD'
    elif msg.reason == ofp.OFPPR_DELETE:
        reason = 'DELETE'
    elif msg.reason == ofp.OFPPR_MODIFY:
        reason = 'MODIFY'
    else:
        reason = 'unknown'

```

(continues on next page)

(continued from previous page)

```
self.logger.debug('OFPPortStatus received: reason=%s desc=%s',
                  reason, msg.desc)
```

JSON Example:

```
{
  "OFPPortStatus": {
    "desc": {
      "OFPPort": {
        "config": 0,
        "hw_addr": "f2:0b:a4:d0:3f:70",
        "length": 168,
        "name": "\u79c1\u306e\u30dd\u30fc\u30c8",
        "port_no": 7,
        "properties": [
          {
            "OFPPortDescPropEthernet": {
              "advertised": 10240,
              "curr": 10248,
              "curr_speed": 5000,
              "length": 32,
              "max_speed": 5000,
              "peer": 10248,
              "supported": 10248,
              "type": 0
            }
          },
          {
            "OFPPortDescPropOptical": {
              "length": 40,
              "rx_grid_freq_lmda": 1500,
              "rx_max_freq_lmda": 2000,
              "rx_min_freq_lmda": 1000,
              "supported": 1,
              "tx_grid_freq_lmda": 1500,
              "tx_max_freq_lmda": 2000,
              "tx_min_freq_lmda": 1000,
              "tx_pwr_max": 2000,
              "tx_pwr_min": 1000,
              "type": 1
            }
          },
          {
            "OFPPortDescPropExperimenter": {
              "data": [],
              "exp_type": 0,
              "experimenter": 101,
              "length": 12,
              "type": 65535
            }
          }
        ]
      }
    }
  }
}
```

(continues on next page)

(continued from previous page)

```

    }
  },
  {
    "OFPPortDescPropExperimenter": {
      "data": [
        1
      ],
      "exp_type": 1,
      "experimenter": 101,
      "length": 16,
      "type": 65535
    }
  },
  {
    "OFPPortDescPropExperimenter": {
      "data": [
        1,
        2
      ],
      "exp_type": 2,
      "experimenter": 101,
      "length": 20,
      "type": 65535
    }
  }
],
"state": 4
}
},
"reason": 0
}
}

```

### Controller Role Status Message

```

class os_ken.ofproto.ofproto_v1_5_parser.OFPRoleStatus(datapath, role=None,
                                                       reason=None,
                                                       generation_id=None,
                                                       properties=None)

```

Role status message

The switch notifies controller of change of role.

Attribute	Description
role	One of the following values.  OFPCR_ROLE_NOCHANGE OFPCR_ROLE_EQUAL OFPCR_ROLE_MASTER
reason	One of the following values.  OFPCRR_MASTER_REQUEST OFPCRR_CONFIG OFPCRR_EXPERIMENTER
generation_id	Master Election Generation ID
properties	List of OFPRoleProp subclass instance

Example:

```
@set_ev_cls(ofp_event.EventOFPRoleStatus, MAIN_DISPATCHER)
def role_status_handler(self, ev):
    msg = ev.msg
    dp = msg.datapath
    ofp = dp.ofproto

    if msg.role == ofp.OFPCR_ROLE_NOCHANGE:
        role = 'ROLE NOCHANGE'
    elif msg.role == ofp.OFPCR_ROLE_EQUAL:
        role = 'ROLE EQUAL'
    elif msg.role == ofp.OFPCR_ROLE_MASTER:
        role = 'ROLE MASTER'
    else:
        role = 'unknown'

    if msg.reason == ofp.OFPCRR_MASTER_REQUEST:
        reason = 'MASTER REQUEST'
    elif msg.reason == ofp.OFPCRR_CONFIG:
        reason = 'CONFIG'
    elif msg.reason == ofp.OFPCRR_EXPERIMENTER:
        reason = 'EXPERIMENTER'
    else:
        reason = 'unknown'

    self.logger.debug('OFPRoleStatus received: role=%s reason=%s '
                      'generation_id=%d properties=%s', role, reason,
                      msg.generation_id, repr(msg.properties))
```

JSON Example:

```
{
  "OFPRoleStatus": {
    "generation_id": 17356517385562371090,
    "properties": [],
    "reason": 0,
    "role": 3
  }
}
```

## Table Status Message

**class** os\_ken.ofproto.ofproto\_v1\_5\_parser.OFPTableStatus(*datapath*, *reason=None*, *table=None*)

Table status message

The switch notifies controller of change of table status.

Attribute	Description
reason	One of the following values.  OFPTR_VACANCY_DOWN OFPTR_VACANCY_UP
table	OFPTableDesc instance

Example:

```
@set_ev_cls(ofp_event.EventOFPTableStatus, MAIN_DISPATCHER)
def table(self, ev):
    msg = ev.msg
    dp = msg.datapath
    ofp = dp.ofproto

    if msg.reason == ofp.OFPTR_VACANCY_DOWN:
        reason = 'VACANCY_DOWN'
    elif msg.reason == ofp.OFPTR_VACANCY_UP:
        reason = 'VACANCY_UP'
    else:
        reason = 'unknown'

    self.logger.debug('OFPTableStatus received: reason=%s '
                      'table_id=%d config=0x%08x properties=%s',
                      reason, msg.table.table_id, msg.table.config,
                      repr(msg.table.properties))
```

JSON Example:

```
{
  "OFPTableStatus": {
```

(continues on next page)

(continued from previous page)

```

"reason": 3,
"table": {
  "OFPTableDesc": {
    "config": 0,
    "length": 80,
    "properties": [
      {
        "OFPTableModPropEviction": {
          "flags": 0,
          "length": 8,
          "type": 2
        }
      },
      {
        "OFPTableModPropVacancy": {
          "length": 8,
          "type": 3,
          "vacancy": 0,
          "vacancy_down": 0,
          "vacancy_up": 0
        }
      },
      {
        "OFPTableModPropExperimenter": {
          "data": [],
          "exp_type": 0,
          "experimenter": 101,
          "length": 12,
          "type": 65535
        }
      },
      {
        "OFPTableModPropExperimenter": {
          "data": [
            1
          ],
          "exp_type": 1,
          "experimenter": 101,
          "length": 16,
          "type": 65535
        }
      },
      {
        "OFPTableModPropExperimenter": {
          "data": [
            1,
            2
          ],
          "exp_type": 2,

```

(continues on next page)

(continued from previous page)

```

        "experimenter": 101,
        "length": 20,
        "type": 65535
    }
    },
    "table_id": 8
}
}
}

```

## Request Forward Message

`class os_ken.ofproto.ofproto_v1_5_parser.OFPRequestForward(datapath, request=None)`

Forwarded request message

The switch forwards request messages from one controller to other controllers.

Attribute	Description
request	OFPGroupMod or OFPMeterMod instance

Example:

```

@set_ev_cls(ofp_event.EventOFPRequestForward, MAIN_DISPATCHER)
def request_forward_handler(self, ev):
    msg = ev.msg
    dp = msg.datapath
    ofp = dp.ofproto

    if msg.request.msg_type == ofp.OFPT_GROUP_MOD:
        self.logger.debug(
            'OFPRequestForward received: request=OFPGroupMod('
            'command=%d, type=%d, group_id=%d, command_bucket_id=%d, '
            'buckets=%s, properties=%s)',
            msg.request.command, msg.request.type,
            msg.request.group_id, msg.request.command_bucket_id,
            msg.request.buckets, repr(msg.request.properties))
    elif msg.request.msg_type == ofp.OFPT_METER_MOD:
        self.logger.debug(
            'OFPRequestForward received: request=OFPMeterMod('
            'command=%d, flags=%d, meter_id=%d, bands=%s)',
            msg.request.command, msg.request.flags,
            msg.request.meter_id, msg.request.bands)
    else:
        self.logger.debug(
            'OFPRequestForward received: request=Unknown')

```

JSON Example:

```

{
  "OFPPRequestForward": {
    "request": {
      "OFPPGroupMod": {
        "bucket_array_len": 56,
        "buckets": [
          {
            "OFPPBucket": {
              "action_array_len": 24,
              "actions": [
                {
                  "OFPPActionPopVlan": {
                    "len": 8,
                    "type": 18
                  }
                },
                {
                  "OFPPActionSetField": {
                    "field": {
                      "OXMTlv": {
                        "field": "ipv4_dst",
                        "mask": null,
                        "value": "192.168.2.9"
                      }
                    }
                  },
                  "len": 16,
                  "type": 25
                }
              ]
            },
            "bucket_id": 305419896,
            "len": 56,
            "properties": [
              {
                "OFPPGroupBucketPropWeight": {
                  "length": 8,
                  "type": 0,
                  "weight": 52428
                }
              },
              {
                "OFPPGroupBucketPropWatch": {
                  "length": 8,
                  "type": 1,
                  "watch": 56797
                }
              },
              {
                "OFPPGroupBucketPropWatch": {
                  "length": 8,

```

(continues on next page)



(continued from previous page)

```

        "type": 2,
        "watch": 4008636142
    }
}
],
"command": 3,
"command_bucket_id": 3149642683,
"group_id": 2863311530,
"properties": [],
"type": 1
}
}
}
}

```

## Controller Status Message

```
class os_ken.ofproto.ofproto_v1_5_parser.OFPControllerStatus(datapath,
                                                             status=None)
```

Controller status message

The switch informs the controller about the status of the control channel it maintains with each controller.

Attribute	Description
status	OFPControllerStatusStats instance

Example:

```

@set_ev_cls(ofp_event.EventOFPControllerStatus, MAIN_DISPATCHER)
def table(self, ev):
    msg = ev.msg
    dp = msg.datapath
    ofp = dp.ofproto
    status = msg.status

    if status.role == ofp.OFPCR_ROLE_NOCHANGE:
        role = 'NOCHANGE'
    elif status.role == ofp.OFPCR_ROLE_EQUAL:
        role = 'EQUAL'
    elif status.role == ofp.OFPCR_ROLE_MASTER:
        role = 'MASTER'
    elif status.role == ofp.OFPCR_ROLE_SLAVE:
        role = 'SLAVE'
    else:

```

(continues on next page)

(continued from previous page)

```

        role = 'unknown'

    if status.reason == ofp.OFPCSR_REQUEST:
        reason = 'REQUEST'
    elif status.reason == ofp.OFPCSR_CHANNEL_STATUS:
        reason = 'CHANNEL_STATUS'
    elif status.reason == ofp.OFPCSR_ROLE:
        reason = 'ROLE'
    elif status.reason == ofp.OFPCSR_CONTROLLER_ADDED:
        reason = 'CONTROLLER_ADDED'
    elif status.reason == ofp.OFPCSR_CONTROLLER_REMOVED:
        reason = 'CONTROLLER_REMOVED'
    elif status.reason == ofp.OFPCSR_SHORT_ID:
        reason = 'SHORT_ID'
    elif status.reason == ofp.OFPCSR_EXPERIMENTER:
        reason = 'EXPERIMENTER'
    else:
        reason = 'unknown'

    if status.channel_status == OFPCT_STATUS_UP:
        channel_status = 'UP'
    if status.channel_status == OFPCT_STATUS_DOWN:
        channel_status = 'DOWN'
    else:
        channel_status = 'unknown'

    self.logger.debug('OFPControllerStatus received: short_id=%d'
                      'role=%s reason=%s channel_status=%s '
                      'properties=%s',
                      status.short_id, role, reason, channel_status,
                      repr(status.properties))

```

JSON Example:

```

{
  "OFPControllerStatus": {
    "status": {
      "OFPControllerStatusStats": {
        "channel_status": 1,
        "length": 48,
        "properties": [
          {
            "OFPControllerStatusPropUri": {
              "length": 26,
              "type": 0,
              "uri": "tls:192.168.34.23:6653"
            }
          }
        ]
      },
      "reason": 1,

```

(continues on next page)

(continued from previous page)

```

        "role": 1,
        "short_id": 65535
    }
}
}
}

```

## Symmetric Messages

### Hello

**class** `os_ken.ofproto.ofproto_v1_5_parser.OFPHello`(*datapath*, *elements=None*)

Hello message

When connection is started, the hello message is exchanged between a switch and a controller.

This message is handled by the OSKen framework, so the OSKen application do not need to process this typically.

Attribute	Description
<code>elements</code>	list of <code>OFPHelloElemVersionBitmap</code> instance

JSON Example:

```

{
  "OFPHello": {
    "elements": [
      {
        "OFPHelloElemVersionBitmap": {
          "length": 8,
          "type": 1,
          "versions": [
            6
          ]
        }
      }
    ]
  }
}

```

**class** `os_ken.ofproto.ofproto_v1_5_parser.OFPHelloElemVersionBitmap`(*versions*,  
*type\_=None*,  
*length=None*)

Version bitmap Hello Element

Attribute	Description
<code>versions</code>	list of versions of OpenFlow protocol a device supports

## Echo Request

**class** `os_ken.ofproto.ofproto_v1_5_parser.OFPEchoRequest` (*datapath*, *data=None*)

Echo request message

This message is handled by the OSKen framework, so the OSKen application do not need to process this typically.

Attribute	Description
<code>data</code>	An arbitrary length data

Example:

```
def send_echo_request(self, datapath, data):
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPEchoRequest(datapath, data)
    datapath.send_msg(req)

    @set_ev_cls(ofp_event.EventOFPEchoRequest,
                [HANDSHAKE_DISPATCHER, CONFIG_DISPATCHER, MAIN_DISPATCHER])
def echo_request_handler(self, ev):
    self.logger.debug('OFPEchoRequest received: data=%s',
                      utils.hex_array(ev.msg.data))
```

JSON Example:

```
{
  "OFPEchoRequest": {
    "data": ""
  }
}
```

## Echo Reply

**class** `os_ken.ofproto.ofproto_v1_5_parser.OFPEchoReply` (*datapath*, *data=None*)

Echo reply message

This message is handled by the OSKen framework, so the OSKen application do not need to process this typically.

Attribute	Description
<code>data</code>	An arbitrary length data

Example:

```
def send_echo_reply(self, datapath, data):
    ofp_parser = datapath.ofproto_parser
```

(continues on next page)

(continued from previous page)

```

reply = ofp_parser.OFPEchoReply(datapath, data)
datapath.send_msg(reply)

@set_ev_cls(ofp_event.EventOFPEchoReply,
            [HANDSHAKE_DISPATCHER, CONFIG_DISPATCHER, MAIN_DISPATCHER])
def echo_reply_handler(self, ev):
    self.logger.debug('OFPEchoReply received: data=%s',
                      utils.hex_array(ev.msg.data))

```

JSON Example:

```

{
  "OFPEchoReply": {
    "data": ""
  }
}

```

## Error Message

```

class os_ken.ofproto.ofproto_v1_5_parser.OFPErrormsg(datapath, type_=None,
                                                    code=None, data=None,
                                                    **kwargs)

```

Error message

The switch notifies controller of problems by this message.

Attribute	Description
type	High level type of error
code	Details depending on the type
data	Variable length data depending on the type and code

type attribute corresponds to type\_ parameter of \_\_init\_\_.

Types and codes are defined in os\_ken.ofproto.ofproto.

Type	Code
OFPET_HELLO_FAILED	OFPHFC_*
OFPET_BAD_REQUEST	OFPBRC_*
OFPET_BAD_ACTION	OFPBAC_*
OFPET_BAD_INSTRUCTION	OFPBIC_*
OFPET_BAD_MATCH	OFPBMC_*
OFPET_FLOW_MOD_FAILED	OFPFMFC_*
OFPET_GROUP_MOD_FAILED	OFPGMFC_*
OFPET_PORT_MOD_FAILED	OFPPMFC_*
OFPET_TABLE_MOD_FAILED	OFPTMFC_*
OFPET_QUEUE_OP_FAILED	OFPQOFC_*
OFPET_SWITCH_CONFIG_FAILED	OFPSCFC_*
OFPET_ROLE_REQUEST_FAILED	OFPRRFC_*
OFPET_METER_MOD_FAILED	OFPMFC_*
OFPET_TABLE_FEATURES_FAILED	OFPTFFC_*
OFPET_EXPERIMENTER	N/A

If `type == OFPET_EXPERIMENTER`, this message has also the following attributes.

Attribute	Description
<code>exp_type</code>	Experimenter defined type
<code>experimenter</code>	Experimenter ID

Example:

```
@set_ev_cls(ofp_event.EventOFPErrormsg,
            [HANDSHAKE_DISPATCHER, CONFIG_DISPATCHER, MAIN_DISPATCHER])
def error_msg_handler(self, ev):
    msg = ev.msg

    self.logger.debug('OFPErrormsg received: type=0x%02x code=0x%02x '
                      'message=%s',
                      msg.type, msg.code, utils.hex_array(msg.data))
```

JSON Example:

```
{
  "OFPErrormsg": {
    "code": 6,
    "data": "Bg4ACAAAAA=",
    "type": 4
  }
}
```

## Experimenter

```
class os_ken.ofproto.ofproto_v1_5_parser.OFPExperimenter(datapath,
                                                         experimenter=None,
                                                         exp_type=None,
                                                         data=None)
```

Experimenter extension message

Attribute	Description
experimenter	Experimenter ID
exp_type	Experimenter defined
data	Experimenter defined arbitrary additional data

JSON Example:

```
{
  "OFPErrormsg": {
    "code": null,
    "data": "amlra2VuIGRhGE=",
    "exp_type": 60000,
    "experimenter": 999999,
    "type": 65535
  }
}
```

## Port Structures

```
class os_ken.ofproto.ofproto_v1_5_parser.OFPPort(port_no=None, length=None,
                                                  hw_addr=None, name=None,
                                                  config=None, state=None,
                                                  properties=None)
```

Description of a port

Attribute	Description
port_no	Port number and it uniquely identifies a port within a switch.
length	Length of ofp_port (excluding padding).
hw_addr	MAC address for the port.
name	Null-terminated string containing a human-readable name for the interface.
config	Bitmap of port configuration flags.  OFPPC_PORT_DOWN OFPPC_NO_RECV OFPPC_NO_FWD OFPPC_NO_PACKET_IN
state	Bitmap of port state flags.  OFPPS_LINK_DOWN OFPPS_BLOCKED OFPPS_LIVE
properties	List of OFPPortDescProp subclass instance

## Flow Match Structure

```
class os_ken.ofproto.ofproto_v1_5_parser.OFPMatch(type_=None, length=None,
                                                _ordered_fields=None, **kwargs)
```

Flow Match Structure

This class is implementation of the flow match structure having compose/query API.

You can define the flow match by the keyword arguments. The following arguments are available.

Argument	Value	Description
in_port	Integer 32bit	Switch input port
in_phy_port	Integer 32bit	Switch physical input port
metadata	Integer 64bit	Metadata passed between tables
eth_dst	MAC address	Ethernet destination address
eth_src	MAC address	Ethernet source address
eth_type	Integer 16bit	Ethernet frame type
vlan_vid	Integer 16bit	VLAN id
vlan_pcp	Integer 8bit	VLAN priority
ip_dscp	Integer 8bit	IP DSCP (6 bits in ToS field)
ip_ecn	Integer 8bit	IP ECN (2 bits in ToS field)
ip_proto	Integer 8bit	IP protocol
ipv4_src	IPv4 address	IPv4 source address
ipv4_dst	IPv4 address	IPv4 destination address
tcp_src	Integer 16bit	TCP source port
tcp_dst	Integer 16bit	TCP destination port

continues on next page



Table 4 – continued from previous page

Argument	Value	Description
udp_src	Integer 16bit	UDP source port
udp_dst	Integer 16bit	UDP destination port
sctp_src	Integer 16bit	SCTP source port
sctp_dst	Integer 16bit	SCTP destination port
icmpv4_type	Integer 8bit	ICMP type
icmpv4_code	Integer 8bit	ICMP code
arp_op	Integer 16bit	ARP opcode
arp_spa	IPv4 address	ARP source IPv4 address
arp_tpa	IPv4 address	ARP target IPv4 address
arp_sha	MAC address	ARP source hardware address
arp_tha	MAC address	ARP target hardware address
ipv6_src	IPv6 address	IPv6 source address
ipv6_dst	IPv6 address	IPv6 destination address
ipv6_flabel	Integer 32bit	IPv6 Flow Label
icmpv6_type	Integer 8bit	ICMPv6 type
icmpv6_code	Integer 8bit	ICMPv6 code
ipv6_nd_target	IPv6 address	Target address for ND
ipv6_nd_sll	MAC address	Source link-layer for ND
ipv6_nd_tll	MAC address	Target link-layer for ND
mpls_label	Integer 32bit	MPLS label
mpls_tc	Integer 8bit	MPLS TC
mpls_bos	Integer 8bit	MPLS BoS bit
pbb_isid	Integer 24bit	PBB I-SID
tunnel_id	Integer 64bit	Logical Port Metadata
ipv6_exthdr	Integer 16bit	IPv6 Extension Header pseudo-field
pbb_uca	Integer 8bit	PBB UCA header field
tcp_flags	Integer 16bit	TCP flags
actset_output	Integer 32bit	Output port from action set metadata
packet_type	Integer 32bit	Packet type value

Example:

```
>>> # compose
>>> match = parser.OFPMatch(
...     in_port=1,
...     eth_type=0x86dd,
...     ipv6_src=('2001:db8:bd05:1d2:288a:1fc0:1:10ee',
...              'ffff:ffff:ffff:ffff:'),
...     ipv6_dst='2001:db8:bd05:1d2:288a:1fc0:1:10ee')
>>> # query
>>> if 'ipv6_src' in match:
...     print match['ipv6_src']
...
('2001:db8:bd05:1d2:288a:1fc0:1:10ee', 'ffff:ffff:ffff:ffff:')
```

**Note**

For the list of the supported Nicira experimenter matches, please refer to [os\\_ken.ofproto.nx\\_match](#).

## Note

For VLAN id match field, special values are defined in OpenFlow Spec.

### 1) Packets with and without a VLAN tag

- Example:

```
match = parser.OFPMatch()
```

- Packet Matching

non-VLAN-tagged	MATCH
VLAN-tagged(vlan_id=3)	MATCH
VLAN-tagged(vlan_id=5)	MATCH

### 2) Only packets without a VLAN tag

- Example:

```
match = parser.OFPMatch(vlan_vid=0x0000)
```

- Packet Matching

non-VLAN-tagged	MATCH
VLAN-tagged(vlan_id=3)	x
VLAN-tagged(vlan_id=5)	x

### 3) Only packets with a VLAN tag regardless of its value

- Example:

```
match = parser.OFPMatch(vlan_vid=(0x1000, 0x1000))
```

- Packet Matching

non-VLAN-tagged	x
VLAN-tagged(vlan_id=3)	MATCH
VLAN-tagged(vlan_id=5)	MATCH

### 4) Only packets with VLAN tag and VID equal

- Example:

```
match = parser.OFPMatch(vlan_vid=(0x1000 | 3))
```

- Packet Matching

non-VLAN-tagged	x
VLAN-tagged(vlan_id=3)	MATCH
VLAN-tagged(vlan_id=5)	x

## Flow Stats Structures

**class** `os_ken.ofproto.ofproto_v1_5_parser.OFPStats`(*length=None*, *\_ordered\_fields=None*, *\*\*kwargs*)

Flow Stats Structure

This class is implementation of the flow stats structure having compose/query API.

You can define the flow stats by the keyword arguments. The following arguments are available.

Argument	Value	Description
duration	Integer 32bit*2	Time flow entry has been alive. This field is a tuple of two Integer 32bit. The first value is duration_sec and the second is duration_nsec.
idle_time	Integer 32bit*2	Time flow entry has been idle.
flow_count	Integer 32bit	Number of aggregated flow entries.
packet_count	Integer 64bit	Number of packets matched by a flow entry.
byte_count	Integer 64bit	Number of bytes matched by a flow entry.

Example:

```
>>> # compose
>>> stats = parser.OFPStats(
...     packet_count=100,
...     duration=(100, 200)
>>> # query
>>> if 'duration' in stats:
...     print stats['duration']
...
(100, 200)
```

## Flow Instruction Structures

**class** `os_ken.ofproto.ofproto_v1_5_parser.OFPInstructionGotoTable`(*table\_id*, *type\_=None*, *len\_=None*)

Goto table instruction

This instruction indicates the next table in the processing pipeline.

Attribute	Description
table_id	Next table

```
class os_ken.ofproto.ofproto_v1_5_parser.OFPInstructionWriteMetadata(metadata,
                                                                    meta-
                                                                    data_mask,
                                                                    type_=None,
                                                                    len_=None)
```

Write metadata instruction

This instruction writes the masked metadata value into the metadata field.

Attribute	Description
metadata	Metadata value to write
metadata_mask	Metadata write bitmask

```
class os_ken.ofproto.ofproto_v1_5_parser.OFPInstructionActions(type_,
                                                                actions=None,
                                                                len_=None)
```

Actions instruction

This instruction writes/applies/clears the actions.

Attribute	Description
type	One of following values.  OFPIT_WRITE_ACTIONS OFPIT_APPLY_ACTIONS OFPIT_CLEAR_ACTIONS
actions	list of OpenFlow action class

type attribute corresponds to type\_ parameter of \_\_init\_\_.

```
class os_ken.ofproto.ofproto_v1_5_parser.OFPInstructionStatTrigger(flags,
                                                                    thresholds,
                                                                    type_=None,
                                                                    len_=None)
```

Statistics triggers instruction

This instruction defines a set of statistics thresholds using OXS.

Attribute	Description
flags	Bitmap of the following flags.  OFPSTF_PERIODIC OFPSTF_ONLY_FIRST
thresholds	Instance of OFPStats

## Action Structures

```
class os_ken.ofproto.ofproto_v1_5_parser.OFPActionOutput(port, max_len=65509,
                                                    type_=None, len_=None)
```

Output action

This action indicates output a packet to the switch port.

Attribute	Description
port	Output port
max_len	Max length to send to controller

```
class os_ken.ofproto.ofproto_v1_5_parser.OFPActionCopyTtlOut(type_=None,
                                                            len_=None)
```

Copy TTL Out action

This action copies the TTL from the next-to-outermost header with TTL to the outermost header with TTL.

```
class os_ken.ofproto.ofproto_v1_5_parser.OFPActionCopyTtlIn(type_=None,
                                                            len_=None)
```

Copy TTL In action

This action copies the TTL from the outermost header with TTL to the next-to-outermost header with TTL.

```
class os_ken.ofproto.ofproto_v1_5_parser.OFPActionSetMplsTtl(mpls_ttl, type_=None,
                                                            len_=None)
```

Set MPLS TTL action

This action sets the MPLS TTL.

Attribute	Description
mpls_ttl	MPLS TTL

```
class os_ken.ofproto.ofproto_v1_5_parser.OFPActionDecMplsTtl(type_=None,
                                                            len_=None)
```

Decrement MPLS TTL action

This action decrements the MPLS TTL.

```
class os_ken.ofproto.ofproto_v1_5_parser.OFPActionPushVlan(ethertype=33024,
                                                            type_=None,
                                                            len_=None)
```

Push VLAN action

This action pushes a new VLAN tag to the packet.

Attribute	Description
ethertype	Ether type. The default is 802.1Q. (0x8100)

**class** `os_ken.ofproto.ofproto_v1_5_parser.OFPActionPopVlan`(*type\_=None, len\_=None*)

Pop VLAN action

This action pops the outermost VLAN tag from the packet.

**class** `os_ken.ofproto.ofproto_v1_5_parser.OFPActionPushMpls`(*ethertype=34887, type\_=None, len\_=None*)

Push MPLS action

This action pushes a new MPLS header to the packet.

Attribute	Description
<code>ethertype</code>	Ether type

**class** `os_ken.ofproto.ofproto_v1_5_parser.OFPActionPopMpls`(*ethertype=2048, type\_=None, len\_=None*)

Pop MPLS action

This action pops the MPLS header from the packet.

**class** `os_ken.ofproto.ofproto_v1_5_parser.OFPActionSetQueue`(*queue\_id, type\_=None, len\_=None*)

Set queue action

This action sets the queue id that will be used to map a flow to an already-configured queue on a port.

Attribute	Description
<code>queue_id</code>	Queue ID for the packets

**class** `os_ken.ofproto.ofproto_v1_5_parser.OFPActionGroup`(*group\_id=0, type\_=None, len\_=None*)

Group action

This action indicates the group used to process the packet.

Attribute	Description
<code>group_id</code>	Group identifier

**class** `os_ken.ofproto.ofproto_v1_5_parser.OFPActionSetNwTtl`(*nw\_ttl, type\_=None, len\_=None*)

Set IP TTL action

This action sets the IP TTL.

Attribute	Description
<code>nw_ttl</code>	IP TTL

```
class os_ken.ofproto.ofproto_v1_5_parser.OFPActionDecNwTtl(type_=None,
                                                         len_=None)
```

Decrement IP TTL action

This action decrements the IP TTL.

```
class os_ken.ofproto.ofproto_v1_5_parser.OFPActionSetField(field=None, **kwargs)
```

Set field action

This action modifies a header field in the packet.

The set of keywords available for this is same as OFPMatch which including with/without mask.

Example:

```
set_field = OFPActionSetField(eth_src="00:00:00:00:00:00")
set_field = OFPActionSetField(ipv4_src=("192.168.100.0",
                                         "255.255.255.0"))
```

```
class os_ken.ofproto.ofproto_v1_5_parser.OFPActionPushPbb(ethertype, type_=None,
                                                           len_=None)
```

Push PBB action

This action pushes a new PBB header to the packet.

Attribute	Description
ethertype	Ether type

```
class os_ken.ofproto.ofproto_v1_5_parser.OFPActionPopPbb(type_=None, len_=None)
```

Pop PBB action

This action pops the outermost PBB service instance header from the packet.

```
class os_ken.ofproto.ofproto_v1_5_parser.OFPActionCopyField(n_bits=0, src_offset=0,
                                                             dst_offset=0,
                                                             oxm_ids=None,
                                                             type_=None,
                                                             len_=None)
```

Copy Field action

This action copy value between header and register.

Attribute	Description
n_bit	Number of bits to copy.
src_c	Starting bit offset in source.
dst_c	Starting bit offset in destination.
oxm_	List of OFPOxmId instances. The first element of this list, src_oxm_id, identifies the field where the value is copied from. The second element of this list, dst_oxm_id, identifies the field where the value is copied to. The default is [].

```
class os_ken.ofproto.ofproto_v1_5_parser.OFPActionMeter(meter_id, type_=None,
                                                    len_=None)
```

Meter action

This action applies meter (rate limiter)

Attribute	Description
<code>meter_id</code>	Meter instance

```
class os_ken.ofproto.ofproto_v1_5_parser.OFPActionExperimenter(experimenter)
```

Experimenter action

This action is an extensible action for the experimenter.

Attribute	Description
<code>experimenter</code>	Experimenter ID

#### Note

For the list of the supported Nicira experimenter actions, please refer to *os\_ken.ofproto.nx\_actions*.

## Controller Status Structure

```
class os_ken.ofproto.ofproto_v1_5_parser.OFPControllerStatusStats(short_id=None,
                                                                    role=None,
                                                                    reason=None,
                                                                    chan-
                                                                    nel_status=None,
                                                                    proper-
                                                                    ties=None,
                                                                    length=None)
```

Controller status structure



Attribute	Description
length	Length of this entry.
short_id	ID number which identifies the controller.
role	Bitmap of controller's role flags.  OFPCR_ROLE_NOCHANGE OFPCR_ROLE_EQUAL OFPCR_ROLE_MASTER OFPCR_ROLE_SLAVE
reason	Bitmap of controller status reason flags.  OFPCSR_REQUEST OFPCSR_CHANNEL_STATUS OFPCSR_ROLE OFPCSR_CONTROLLER_ADDED OFPCSR_CONTROLLER_REMOVED OFPCSR_SHORT_ID OFPCSR_EXPERIMENTER
channel_status	Bitmap of control channel status flags.  OFPCT_STATUS_UP OFPCT_STATUS_DOWN
properties	List of <code>OFPControllerStatusProp</code> sub-class instance

## 8.1.6 Nicira Extension Structures

### Nicira Extension Actions Structures

The followings shows the supported `NXAction` classes only in `OpenFlow1.0`

```
class os_ken.ofproto.ofproto_v1_0_parser.NXActionSetQueue(queue_id, type_=None,
                                                         len_=None, vendor=None,
                                                         subtype=None)
```

Set queue action

This action sets the queue that should be used to queue when packets are output.

And equivalent to the followings action of `ovs-ofctl` command.

```
set_queue:queue
```

Attribute	Description
queue_id	Queue ID for the packets

**Note**

This actions is supported by `OFPActionSetQueue` in OpenFlow1.2 or later.

Example:

```
actions += [parser.NXActionSetQueue(queue_id=10)]
```

```
class os_ken.ofproto.ofproto_v1_0_parser.NXActionDecTtl(type_=None, len_=None,
                                                       vendor=None,
                                                       subtype=None)
```

Decrement IP TTL action

This action decrements TTL of IPv4 packet or hop limit of IPv6 packet.

And equivalent to the followings action of ovs-ofctl command.

```
dec_ttl
```

**Note**

This actions is supported by `OFPActionDecNwTtl` in OpenFlow1.2 or later.

Example:

```
actions += [parser.NXActionDecTtl()]
```

```
class os_ken.ofproto.ofproto_v1_0_parser.NXActionPushMpls(ethertype, type_=None,
                                                         len_=None, vendor=None,
                                                         subtype=None)
```

Push MPLS action

This action pushes a new MPLS header to the packet.

And equivalent to the followings action of ovs-ofctl command.

```
push_mpls:ethertype
```

Attribute	Description
ethertype	Ether type(The value must be either 0x8847 or 0x8848)

**Note**

This actions is supported by `OFPActionPushMpls` in OpenFlow1.2 or later.

Example:

```
match = parser.OFPMatch(dl_type=0x0800)
actions += [parser.NXActionPushMpls(ethertype=0x8847)]
```

**class** `os_ken.ofproto.ofproto_v1_0_parser.NXActionPopMpls`(*ethertype*, *type\_=None*, *len\_=None*, *vendor=None*, *subtype=None*)

Pop MPLS action

This action pops the MPLS header from the packet.

And equivalent to the followings action of ovs-ofctl command.

```
pop_mpls:ethertype
```

Attribute	Description
ethertype	Ether type

#### Note

This actions is supported by `OFPACTIONPopMpls` in OpenFlow1.2 or later.

Example:

```
match = parser.OFPMatch(dl_type=0x8847)
actions += [parser.NXActionPushMpls(ethertype=0x0800)]
```

**class** `os_ken.ofproto.ofproto_v1_0_parser.NXActionSetMplsTtl`(*ttl*, *type\_=None*, *len\_=None*, *vendor=None*, *subtype=None*)

Set MPLS TTL action

This action sets the MPLS TTL.

And equivalent to the followings action of ovs-ofctl command.

```
set_mpls_ttl:tll
```

Attribute	Description
tll	MPLS TTL

#### Note

This actions is supported by `OFPACTIONSetMplsTtl` in OpenFlow1.2 or later.

Example:

```
actions += [parser.NXActionSetMplsTtl(ttl=128)]
```

```
class os_ken.ofproto.ofproto_v1_0_parser.NXActionDecMplsTtl(type_=None,
                                                         len_=None,
                                                         vendor=None,
                                                         subtype=None)
```

Decrement MPLS TTL action

This action decrements the MPLS TTL.

And equivalent to the followings action of ovs-ofctl command.

```
dec_mpls_ttl
```

#### Note

This actions is supported by OFPActionDecMplsTtl in OpenFlow1.2 or later.

Example:

```
actions += [parser.NXActionDecMplsTtl()]
```

```
class os_ken.ofproto.ofproto_v1_0_parser.NXActionSetMplsLabel(label, type_=None,
                                                             len_=None,
                                                             vendor=None,
                                                             subtype=None)
```

Set MPLS Level action

This action sets the MPLS Label.

And equivalent to the followings action of ovs-ofctl command.

```
set_mpls_label:label
```

Attribute	Description
label	MPLS Label

#### Note

This actions is supported by OFPActionSetField(mpls\_label=label) in OpenFlow1.2 or later.

Example:

```
actions += [parser.NXActionSetMplsLabel(label=0x10)]
```

```
class os_ken.ofproto.ofproto_v1_0_parser.NXActionSetMplsTc(tc, type_=None,
                                                         len_=None,
                                                         vendor=None,
                                                         subtype=None)
```

Set MPLS Tc action

This action sets the MPLS Tc.

And equivalent to the followings action of ovs-ofctl command.

```
set_mpls_tc:tc
```

Attribute	Description
tc	MPLS Tc

#### Note

This actions is supported by `OFFActionSetField(mpls_label=tc)` in OpenFlow1.2 or later.

Example:

```
actions += [parser.NXActionSetMplsLabel(tc=0x10)]
```

The followings shows the supported `NXAction` classes in OpenFlow1.0 or later

```
class os_ken.ofproto.ofproto_v1_3_parser.NXActionPopQueue(type_=None, len_=None,
                                                           experimenter=None,
                                                           subtype=None)
```

Pop queue action

This action restores the queue to the value it was before any `set_queue` actions were applied.

And equivalent to the followings action of ovs-ofctl command.

```
pop_queue
```

Example:

```
actions += [parser.NXActionPopQueue()]
```

```
class os_ken.ofproto.ofproto_v1_3_parser.NXActionRegLoad(ofs_nbits, dst, value,
                                                          type_=None, len_=None,
                                                          experimenter=None,
                                                          subtype=None)
```

Load literal value action

This action loads a literal value into a field or part of a field.

And equivalent to the followings action of ovs-ofctl command.

**load:value->dst[start..end]**

At-tribute	Description
ofs_nbits	Start and End for the OXM/NXM field. Setting method refer to the nicira_ext. ofs_nbits
dst	OXM/NXM header for destination field
value	OXM/NXM value to be loaded

Example:

```
actions += [parser.NXActionRegLoad(
    ofs_nbits=nicira_ext.ofs_nbits(4, 31),
    dst="eth_dst",
    value=0x112233)]
```

```
class os_ken.ofproto.ofproto_v1_3_parser.NXActionRegLoad2(dst, value, mask=None,
    type_=None, len_=None,
    experimenter=None,
    subtype=None)
```

Load literal value action

This action loads a literal value into a field or part of a field.

And equivalent to the followings action of ovs-ofctl command.

**set\_field:value[/mask]->dst**

Attribute	Description
value	OXM/NXM value to be loaded
mask	Mask for destination field
dst	OXM/NXM header for destination field

Example:

```
actions += [parser.NXActionRegLoad2(dst="tun_ipv4_src",
    value="192.168.10.0",
    mask="255.255.255.0")]
```

```
class os_ken.ofproto.ofproto_v1_3_parser.NXActionNote(note, type_=None, len_=None,
    experimenter=None,
    subtype=None)
```

Note action

This action does nothing at all.

And equivalent to the followings action of ovs-ofctl command.

**note:***[hh]*..

Attribute	Description
note	A list of integer type values

Example:

```
actions += [parser.NXActionNote(note=[0xaa, 0xbb, 0xcc, 0xdd])]
```

```
class os_ken.ofproto.ofproto_v1_3_parser.NXActionSetTunnel(tun_id, type_=None,
                                                         len_=None,
                                                         experimenter=None,
                                                         subtype=None)
```

Set Tunnel action

This action sets the identifier (such as GRE) to the specified id.

And equivalent to the followings action of ovs-ofctl command.

#### Note

This actions is supported by OFPActionSetField in OpenFlow1.2 or later.

**set\_tunnel:***id*

Attribute	Description
tun_id	Tunnel ID(32bits)

Example:

```
actions += [parser.NXActionSetTunnel(tun_id=0xa)]
```

```
class os_ken.ofproto.ofproto_v1_3_parser.NXActionSetTunnel64(tun_id, type_=None,
                                                             len_=None,
                                                             experimenter=None,
                                                             subtype=None)
```

Set Tunnel action

This action outputs to a port that encapsulates the packet in a tunnel.

And equivalent to the followings action of ovs-ofctl command.

**Note**

This actions is supported by `OFPActionSetField` in OpenFlow1.2 or later.

**set\_tunnel64:id**

Attribute	Description
tun_id	Tunnel ID(64bits)

Example:

```
actions += [parser.NXActionSetTunnel64(tun_id=0xa)]
```

```
class os_ken.ofproto.ofproto_v1_3_parser.NXActionRegMove(src_field, dst_field, n_bits,
                                                         src_ofs=0, dst_ofs=0,
                                                         type_=None, len_=None,
                                                         experimenter=None,
                                                         subtype=None)
```

Move register action

This action copies the src to dst.

And equivalent to the followings action of ovs-ofctl command.

```
move:src[start..end]->dst[start..end ]
```

Attribute	Description
src_field	OXM/NXM header for source field
dst_field	OXM/NXM header for destination field
n_bits	Number of bits
src_ofs	Starting bit offset in source
dst_ofs	Starting bit offset in destination

**Caution**

**src\_start and src\_end difference and dst\_start and dst\_end difference must be the same.**

Example:

```
actions += [parser.NXActionRegMove(src_field="reg0",
                                   dst_field="reg1",
                                   n_bits=5,
```

(continues on next page)



(continued from previous page)

```
src_ofs=0
dst_ofs=10)]
```

```
class os_ken.ofproto.ofproto_v1_3_parser.NXActionResubmit(in_port=65528,
                                                         type_=None, len_=None,
                                                         experimenter=None,
                                                         subtype=None)
```

Resubmit action

This action searches one of the switch's flow tables.

And equivalent to the followings action of ovs-ofctl command.

```
resubmit:port
```

Attribute	Description
<i>in_port</i>	New <i>in_port</i> for checking flow table

Example:

```
actions += [parser.NXActionResubmit(in_port=8080)]
```

```
class os_ken.ofproto.ofproto_v1_3_parser.NXActionResubmitTable(in_port=65528,
                                                                table_id=255,
                                                                type_=None,
                                                                len_=None,
                                                                experimenter=None,
                                                                subtype=None)
```

Resubmit action

This action searches one of the switch's flow tables.

And equivalent to the followings action of ovs-ofctl command.

```
resubmit([port],[table])
```

Attribute	Description
<i>in_port</i>	New <i>in_port</i> for checking flow table
<i>table_id</i>	Checking flow tables

Example:

```
actions += [parser.NXActionResubmitTable(in_port=8080,
                                         table_id=10)]
```

```
class os_ken.ofproto.ofproto_v1_3_parser.NXActionOutputReg(ofs_nbits, src, max_len,
                                                         type_=None, len_=None,
                                                         experimenter=None,
                                                         subtype=None)
```

Add output action

This action outputs the packet to the OpenFlow port number read from *src*.

And equivalent to the followings action of ovs-ofctl command.

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

At-tribute	Description
<i>ofs_nbits</i>	Start and End for the OXM/NXM field. Setting method refer to the <i>nicira_ext.ofs_nbits</i>
<i>src</i>	OXM/NXM header for source field
<i>max_len</i>	Max length to send to controller

Example:

```
actions += [parser.NXActionOutputReg(
             ofs_nbits=nicira_ext.ofs_nbits(4, 31),
             src="reg0",
             max_len=1024)]
```

```
class os_ken.ofproto.ofproto_v1_3_parser.NXActionOutputReg2(ofs_nbits, src, max_len,
                                                            type_=None,
                                                            len_=None,
                                                            experimenter=None,
                                                            subtype=None)
```

Add output action

This action outputs the packet to the OpenFlow port number read from *src*.

And equivalent to the followings action of ovs-ofctl command.

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

#### Note

Like the `NXActionOutputReg` but organized so that there is room for a 64-bit experimenter OXM as 'src'.

At-tribute	Description
ofs_nbits	Start and End for the OXM/NXM field. Setting method refer to the <code>nicira_ext.ofs_nbits</code>
src	OXM/NXM header for source field
max_len	Max length to send to controller

Example:

```
actions += [parser.NXActionOutputReg2(
    ofs_nbits=nicira_ext.ofs_nbits(4, 31),
    src="reg0",
    max_len=1024)]
```

```
class os_ken.ofproto.ofproto_v1_3_parser.NXActionLearn(table_id, specs,
    idle_timeout=0,
    hard_timeout=0,
    priority=32768, cookie=0,
    flags=0, fin_idle_timeout=0,
    fin_hard_timeout=0,
    type_=None, len_=None,
    experimenter=None,
    subtype=None)
```

Adds or modifies flow action

This action adds or modifies a flow in OpenFlow table.

And equivalent to the followings action of `ovs-ofctl` command.

```
learn(argument[,argument]...)
```

Attribute	Description
table_id	The table in which the new flow should be inserted
specs	<p>Adds a match criterion to the new flow Please use the NXFlowSpecMatch in order to set the following format</p> <p><i>field=value</i> <i>field[start..end] =src[start..end]</i> <i>field[start..end]</i></p> <p>Please use the NXFlowSpecLoad in order to set the following format</p> <p><b>load:value-&gt;dst[start..end]</b> <b>load:src[start..end] -&gt;dst[start..end]</b></p> <p>Please use the NXFlowSpecOutput in order to set the following format</p> <p><b>output:field[start..end]</b></p>
idle_timeout	Idle time before discarding(seconds)
hard_timeout	Max time before discarding(seconds)
priority	Priority level of flow entry
cookie	Cookie for new flow
flags	send_flow_rem
fin_idle_timeout	Idle timeout after FIN(seconds)
fin_hard_timeout	Hard timeout after FIN(seconds)

**Caution**

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.

Example:

```
actions += [
    parser.NXActionLearn(able_id=10,
        specs=[parser.NXFlowSpecMatch(src=0x800,
            dst=('eth_type_nxm', 0),
            n_bits=16),
            parser.NXFlowSpecMatch(src=('reg1', 1),
            dst=('reg2', 3),
            n_bits=5),
```

(continues on next page)

(continued from previous page)

```

parser.NXFlowSpecMatch(src=('reg3', 1),
                        dst=('reg3', 1),
                        n_bits=5),
parser.NXFlowSpecLoad(src=0,
                       dst=('reg4', 3),
                       n_bits=5),
parser.NXFlowSpecLoad(src=('reg5', 1),
                       dst=('reg6', 3),
                       n_bits=5),
parser.NXFlowSpecOutput(src=('reg7', 1),
                         dst="",
                         n_bits=5)],

idle_timeout=180,
hard_timeout=300,
priority=1,
cookie=0x64,
flags=ofproto.OFPFF_SEND_FLOW_REM,
fin_idle_timeout=180,
fin_hard_timeout=300)]

```

**class** `os_ken.ofproto.ofproto_v1_3_parser.NXActionExit`(*type\_=None, len\_=None, experimenter=None, subtype=None*)

Halt action

This action causes OpenvSwitch to immediately halt execution of further actions.

And equivalent to the followings action of ovs-ofctl command.

**exit**

Example:

```
actions += [parser.NXActionExit()]
```

**class** `os_ken.ofproto.ofproto_v1_3_parser.NXActionController`(*max\_len, controller\_id, reason, type\_=None, len\_=None, experimenter=None, subtype=None*)

Send packet in message action

This action sends the packet to the OpenFlow controller as a packet in message.

And equivalent to the followings action of ovs-ofctl command.

**controller(key=value...)**

Attribute	Description
max_len	Max length to send to controller
controller_id	Controller ID to send packet-in
reason	Reason for sending the message

Example:

```
actions += [
    parser.NXActionController(max_len=1024,
                             controller_id=1,
                             reason=ofproto.OFPR_INVALID_TTL)]
```

```
class os_ken.ofproto.ofproto_v1_3_parser.NXActionController2(type_=None,
                                                            len_=None,
                                                            vendor=None,
                                                            subtype=None,
                                                            **kwargs)
```

Send packet in message action

This action sends the packet to the OpenFlow controller as a packet in message.

And equivalent to the followings action of ovs-ofctl command.

**controller**(key=value...)

Attribute	Description
max_len	Max length to send to controller
controller_id	Controller ID to send packet-in
reason	Reason for sending the message
userdata	Additional data to the controller in the packet-in message
pause	Flag to pause pipeline to resume later

Example:

```
actions += [
    parser.NXActionController(max_len=1024,
                             controller_id=1,
                             reason=ofproto.OFPR_INVALID_TTL,
                             userdata=[0xa, 0xb, 0xc],
                             pause=True)]
```

```
class os_ken.ofproto.ofproto_v1_3_parser.NXActionDecTtlCntIds(cnt_ids, type_=None,
                                                             len_=None,
                                                             experimenter=None,
                                                             subtype=None)
```

Decrement TTL action

This action decrements TTL of IPv4 packet or hop limits of IPv6 packet.

And equivalent to the followings action of ovs-ofctl command.

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

Attribute	Description
cnt_ids	Controller ids

Example:

```
actions += [parser.NXActionDecTtlCntIds(cnt_ids=[1,2,3])]
```

#### Note

If you want to set the following ovs-ofctl command. Please use `OFPAActionDecNwTtl`.

```
dec_ttl
```

```
class os_ken.ofproto.ofproto_v1_3_parser.NXActionStackPush(field, start, end,
                                                            type_=None, len_=None,
                                                            experimenter=None,
                                                            subtype=None)
```

Push field action

This action pushes field to top of the stack.

And equivalent to the followings action of ovs-ofctl command.

```
pop:dst[start...end]
```

Attribute	Description
field	OXM/NXM header for source field
start	Start bit for source field
end	End bit for source field

Example:

```
actions += [parser.NXActionStackPush(field="reg2",
                                       start=0,
                                       end=5)]
```

```
class os_ken.ofproto.ofproto_v1_3_parser.NXActionStackPop(field, start, end,
                                                            type_=None, len_=None,
                                                            experimenter=None,
                                                            subtype=None)
```

Pop field action

This action pops field from top of the stack.

And equivalent to the followings action of ovs-ofctl command.

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

Attribute	Description
field	OXM/NXM header for destination field
start	Start bit for destination field
end	End bit for destination field

Example:

```
actions += [parser.NXActionStackPop(field="reg2",
                                     start=0,
                                     end=5)]
```

```
class os_ken.ofproto.ofproto_v1_3_parser.NXActionSample(probability,
                                                         collector_set_id=0,
                                                         obs_domain_id=0,
                                                         obs_point_id=0,
                                                         type_=None, len_=None,
                                                         experimenter=None,
                                                         subtype=None)
```

Sample packets action

This action samples packets and sends one sample for every sampled packet.

And equivalent to the followings action of ovs-ofctl command.

**sample(argument[,argument]...)**

Attribute	Description
probability	The number of sampled packets
collector_set_id	The unsigned 32-bit integer identifier of the set of sample collectors to send sampled packets to
obs_domain_id	The Unsigned 32-bit integer Observation Domain ID
obs_point_id	The unsigned 32-bit integer Observation Point ID

Example:

```
actions += [parser.NXActionSample(probability=3,
                                   collector_set_id=1,
                                   obs_domain_id=2,
                                   obs_point_id=3,)]
```



```
class os_ken.ofproto.ofproto_v1_3_parser.NXActionSample2(probability,
                                                         collector_set_id=0,
                                                         obs_domain_id=0,
                                                         obs_point_id=0,
                                                         sampling_port=0,
                                                         type_=None, len_=None,
                                                         experimenter=None,
                                                         subtype=None)
```

Sample packets action

This action samples packets and sends one sample for every sampled packet. 'sampling\_port' can be equal to ingress port or one of egress ports.

And equivalent to the followings action of ovs-ofctl command.

```
sample(argument[,argument]...)
```

Attribute	Description
probability	The number of sampled packets
collector_set_id	The unsigned 32-bit integer identifier of the set of sample collectors to send sampled packets to
obs_domain_id	The Unsigned 32-bit integer Observation Domain ID
obs_point_id	The unsigned 32-bit integer Observation Point ID
sampling_port	Sampling port number

Example:

```
actions += [parser.NXActionSample2(probability=3,
                                   collector_set_id=1,
                                   obs_domain_id=2,
                                   obs_point_id=3,
                                   sampling_port=8080)]
```

```
class os_ken.ofproto.ofproto_v1_3_parser.NXActionFinTimeout(fin_idle_timeout,
                                                            fin_hard_timeout,
                                                            type_=None,
                                                            len_=None,
                                                            experimenter=None,
                                                            subtype=None)
```

Change TCP timeout action

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.

And equivalent to the followings action of ovs-ofctl command.

```
fin_timeout(argument[,argument]...)
```

Attribute	Description
<code>fin_idle_timeout</code>	Causes the flow to expire after the given number of seconds of inactivity
<code>fin_idle_timeout</code>	Causes the flow to expire after the given number of second, regardless of activity

Example:

```
match = parser.OFPMatch(ip_proto=6, eth_type=0x0800)
actions += [parser.NXActionFinTimeout(fin_idle_timeout=30,
                                       fin_hard_timeout=60)]
```

```
class os_ken.ofproto.ofproto_v1_3_parser.NXActionConjunction(clause, n_clauses, id_,
                                                            type_=None,
                                                            len_=None,
                                                            experimenter=None,
                                                            subtype=None)
```

Conjunctive matches action

This action ties groups of individual OpenFlow flows into higher-level conjunctive flows. Please refer to the ovs-ofctl command manual for details.

And equivalent to the followings action of ovs-ofctl command.

**conjunction(*id,k/n*)**

Attribute	Description
<code>clause</code>	Number assigned to the flow's dimension
<code>n_clauses</code>	Specify the conjunctive flow's match condition
<code>id_</code>	Conjunction ID

Example:

```
actions += [parser.NXActionConjunction(clause=1,
                                       n_clauses=2,
                                       id_=10)]
```

```
class os_ken.ofproto.ofproto_v1_3_parser.NXActionMultipath(fields, basis, algorithm,
                                                            max_link, arg, ofs_nbits,
                                                            dst, type_=None,
                                                            len_=None,
                                                            experimenter=None,
                                                            subtype=None)
```

Select multipath link action

This action selects multipath link based on the specified parameters. Please refer to the ovs-ofctl command manual for details.

And equivalent to the followings action of ovs-ofctl command.

**multipath**(*fields, basis, algorithm, n\_links, arg, dst[start..end]*)

Attribute	Description
fields	One of NX_HASH_FIELDS_*
basis	Universal hash parameter
algorithm	One of NX_MP_ALG_*
max_link	Number of output links
arg	Algorithm-specific argument
ofs_nbits	Start and End for the OXM/NXM field. Setting method refer to the nicira_ext. ofs_nbits
dst	OXM/NXM header for source field

Example:

```
actions += [parser.NXActionMultipath(
    fields=nicira_ext.NX_HASH_FIELDS_SYMMETRIC_L4,
    basis=1024,
    algorithm=nicira_ext.NX_MP_ALG_HRW,
    max_link=5,
    arg=0,
    ofs_nbits=nicira_ext.ofs_nbits(4, 31),
    dst="reg2")]
```

**class** os\_ken.ofproto.ofproto\_v1\_3\_parser.NXActionBundle(*algorithm, fields, basis, slave\_type, n\_slaves, ofs\_nbits, dst, slaves*)

Select bundle link action

This action selects bundle link based on the specified parameters. Please refer to the ovs-ofctl command manual for details.

And equivalent to the followings action of ovs-ofctl command.

**bundle**(*fields, basis, algorithm, slave\_type, slaves:[ s1, s2,...]*)

Attribute	Description
algorithm	One of NX_MP_ALG_*
fields	One of NX_HASH_FIELDS_*
basis	Universal hash parameter
slave_type	Type of slaves(must be NXM_OF_IN_PORT)
n_slaves	Number of slaves
ofs_nbits	Start and End for the OXM/NXM field. (must be zero)
dst	OXM/NXM header for source field(must be zero)
slaves	List of slaves

Example:

```
actions += [parser.NXActionBundle(
    algorithm=nicira_ext.NX_MP_ALG_HRW,
    fields=nicira_ext.NX_HASH_FIELDS_ETH_SRC,
    basis=0,
    slave_type=nicira_ext.NXM_OF_IN_PORT,
    n_slaves=2,
    ofs_nbits=0,
    dst=0,
    slaves=[2, 3])]
```

**class** os\_ken.ofproto.ofproto\_v1\_3\_parser.NXActionBundleLoad(*algorithm, fields, basis, slave\_type, n\_slaves, ofs\_nbits, dst, slaves*)

Select bundle link action

This action has the same behavior as the bundle action, with one exception. Please refer to the ovs-ofctl command manual for details.

And equivalent to the followings action of ovs-ofctl command.

```
bundle_load(fields, basis, algorithm, slave_type, dst[start... *emd*], slaves:[ s1, s2,...]) |
```

Attribute	Description
algo-rithm	One of NX_MP_ALG_*
fields	One of NX_HASH_FIELDS_*
basis	Universal hash parameter
slave_type	Type of slaves(must be NXM_OF_IN_PORT)
n_slaves	Number of slaves
ofs_nbits	Start and End for the OXM/NXM field. Setting method refer to the nicira_ext.ofs_nbits
dst	OXM/NXM header for source field
slaves	List of slaves

Example:

```
actions += [parser.NXActionBundleLoad(
    algorithm=nicira_ext.NX_MP_ALG_HRW,
    fields=nicira_ext.NX_HASH_FIELDS_ETH_SRC,
    basis=0,
    slave_type=nicira_ext.NXM_OF_IN_PORT,
    n_slaves=2,
    ofs_nbits=nicira_ext.ofs_nbits(4, 31),
    dst="reg0",
    slaves=[2, 3])]
```

```
class os_ken.ofproto.ofproto_v1_3_parser.NXActionCT(flags, zone_src, zone_ofs_nbits,
                                                    recirc_table, alg, actions,
                                                    type_=None, len_=None,
                                                    experimenter=None,
                                                    subtype=None)
```

Pass traffic to the connection tracker action

This action sends the packet through the connection tracker.

And equivalent to the followings action of ovs-ofctl command.

```
ct(argument[,argument]...)
```

Attribute	Description
flags	Zero or more(Unspecified flag bits must be zero.)
zone_sr	OXM/NXM header for source field
zone_of	Start and End for the OXM/NXM field. Setting method refer to the <code>nicira_ext.ofs_nbits</code> . If you need set the Immediate value for zone, <code>zone_src</code> must be set to None or empty character string.
re-circ_tab	Recirculate to a specific table
alg	Well-known port number for the protocol
actions	Zero or more actions may immediately follow this action

#### Note

If you set number to `zone_src`, Traceback occurs when you run the `to_jsondict`.

Example:

```
match = parser.OFPMatch(eth_type=0x0800, ct_state=(0, 32))
actions += [parser.NXActionCT(
    flags = 1,
    zone_src = "reg0",
    zone_ofs_nbits = nicira_ext.ofs_nbits(4, 31),
    recirc_table = 4,
    alg = 0,
    actions = [])]
```

```
class os_ken.ofproto.ofproto_v1_3_parser.NXActionNAT(flags, range_ipv4_min="",
                                                    range_ipv4_max="",
                                                    range_ipv6_min="",
                                                    range_ipv6_max="",
                                                    range_proto_min=None,
                                                    range_proto_max=None,
                                                    type_=None, len_=None,
                                                    experimenter=None,
                                                    subtype=None)
```

Network address translation action

This action sends the packet through the connection tracker.

And equivalent to the followings action of ovs-ofctl command.

**Note**

The following command image does not exist in ovs-ofctl command manual and has been created from the command response.

```
nat(src=ip_min-ip_max : proto_min-proto-max)
```

Attribute	Description
flags	Zero or more(Unspecified flag bits must be zero.)
range_ipv4_min	Range ipv4 address minimum
range_ipv4_max	Range ipv4 address maximum
range_ipv6_min	Range ipv6 address minimum
range_ipv6_max	Range ipv6 address maximum
range_proto_min	Range protocol minimum
range_proto_max	Range protocol maximum

**Caution**

NXActionNAT must be defined in the actions in the NXActionCT.

Example:

```
match = parser.OFPMatch(eth_type=0x0800)
actions += [
    parser.NXActionCT(
        flags = 1,
        zone_src = "reg0",
        zone_ofs_nbits = nicira_ext.ofs_nbits(4, 31),
        recirc_table = 255,
        alg = 0,
        actions = [
```

(continues on next page)

(continued from previous page)

```

        parser.NXActionNAT(
            flags = 1,
            range_ipv4_min = "10.1.12.0",
            range_ipv4_max = "10.1.13.255",
            range_ipv6_min = "",
            range_ipv6_max = "",
            range_proto_min = 1,
            range_proto_max = 1023
        )
    ]
)
]

```

```

class os_ken.ofproto.ofproto_v1_3_parser.NXActionOutputTrunc(port, max_len,
                                                             type_=None,
                                                             len_=None,
                                                             experimenter=None,
                                                             subtype=None)

```

Truncate output action

This action truncate a packet into the specified size and outputs it.

And equivalent to the followings action of ovs-ofctl command.

```
output(port=port,max_len=max_len)
```

Attribute	Description
port	Output port
max_len	Max bytes to send

Example:

```

actions += [parser.NXActionOutputTrunc(port=8080,
                                       max_len=1024)]

```

```

class os_ken.ofproto.ofproto_v1_3_parser.NXActionDecNshTtl(type_=None, len_=None,
                                                            vendor=None,
                                                            subtype=None)

```

Decrement NSH TTL action

This action decrements the TTL in the Network Service Header(NSH).

This action was added in OVS v2.9.

And equivalent to the followings action of ovs-ofctl command.

```
dec_nsh_ttl
```

Example:

```
actions += [parser.NXActionDecNshTtl()]
```

**class** `os_ken.ofproto.ofproto_v1_3_parser.NXFlowSpecMatch`(*src, dst, n\_bits*)

Specification for adding match criterion

This class is used by `NXActionLearn`.

For the usage of this class, please refer to `NXActionLearn`.

Attribute	Description
<code>src</code>	OXM/NXM header and Start bit for source field
<code>dst</code>	OXM/NXM header and Start bit for destination field
<code>n_bits</code>	The number of bits from the start bit

**class** `os_ken.ofproto.ofproto_v1_3_parser.NXFlowSpecLoad`(*src, dst, n\_bits*)

Add `NXAST_REG_LOAD` actions

This class is used by `NXActionLearn`.

For the usage of this class, please refer to `NXActionLearn`.

Attribute	Description
<code>src</code>	OXM/NXM header and Start bit for source field
<code>dst</code>	OXM/NXM header and Start bit for destination field
<code>n_bits</code>	The number of bits from the start bit

**class** `os_ken.ofproto.ofproto_v1_3_parser.NXFlowSpecOutput`(*src, n\_bits, dst=""*)

Add an `OPPAT_OUTPUT` action

This class is used by `NXActionLearn`.

For the usage of this class, please refer to `NXActionLearn`.

Attribute	Description
<code>src</code>	OXM/NXM header and Start bit for source field
<code>dst</code>	Must be ""
<code>n_bits</code>	The number of bits from the start bit

`os_ken.ofproto.nicira_ext.ofs_nbits`(*start, end*)

The utility method for `ofs_nbits`

This method is used in the class to set the `ofs_nbits`.

This method converts start/end bits into `ofs_nbits` required to specify the bit range of OXM/NXM fields.

`ofs_nbits` can be calculated as following:

```
ofs_nbits = (start << 6) + (end - start)
```



The parameter start/end means the OXM/NXM field of ovs-ofctl command.

`field[start..end]`

Attribute	Description
start	Start bit for OXM/NXM field
end	End bit for OXM/NXM field

## Nicira Extended Match Structures

The API of this class is the same as `OFPMatch`.

You can define the flow match by the keyword arguments. The following arguments are available.

Argument	Value	Description
<code>in_port_nxm</code>	Integer 16bit	OpenFlow port number.
<code>eth_dst_nxm</code>	MAC address	Ethernet destination address.
<code>eth_src_nxm</code>	MAC address	Ethernet source address.
<code>eth_type_nxm</code>	Integer 16bit	Ethernet type. Needed to support Nicira extensions that require the eth_type to be
<code>vlan_tci</code>	Integer 16bit	VLAN TCI. Basically same as <code>vlan_vid</code> plus <code>vlan_pcp</code> .
<code>nw_tos</code>	Integer 8bit	IP ToS or IPv6 traffic class field dscp. Requires setting fields: <code>eth_type_nxm = [</code>
<code>ip_proto_nxm</code>	Integer 8bit	IP protocol. Needed to support Nicira extensions that require the ip_proto to be
<code>ipv4_src_nxm</code>	IPv4 address	IPv4 source address. Requires setting fields: <code>eth_type_nxm = 0x0800 (IPv4)</code>
<code>ipv4_dst_nxm</code>	IPv4 address	IPv4 destination address. Requires setting fields: <code>eth_type_nxm = 0x0800 (IPv4)</code>
<code>tcp_src_nxm</code>	Integer 16bit	TCP source port. Requires setting fields: <code>eth_type_nxm = [0x0800 (IPv4)]0x86</code>
<code>tcp_dst_nxm</code>	Integer 16bit	TCP destination port. Requires setting fields: <code>eth_type_nxm = [0x0800 (IPv4)]0</code>
<code>udp_src_nxm</code>	Integer 16bit	UDP source port. Requires setting fields: <code>eth_type_nxm = [0x0800 (IPv4)]0x86</code>
<code>udp_dst_nxm</code>	Integer 16bit	UDP destination port. <code>eth_type_nxm = [0x0800 (IPv4)]0x86dd (IPv6)]</code> and <code>ip_p</code>
<code>icmpv4_type_nxm</code>	Integer 8bit	Type matches the ICMP type and code matches the ICMP code. Requires setting
<code>icmpv4_code_nxm</code>	Integer 8bit	Type matches the ICMP type and code matches the ICMP code. Requires setting
<code>arp_op_nxm</code>	Integer 16bit	Only ARP opcodes between 1 and 255 should be specified for matching. Requir
<code>arp_spa_nxm</code>	IPv4 address	An address may be specified as an IP address or host name. Requires setting fiel
<code>arp_tpa_nxm</code>	IPv4 address	An address may be specified as an IP address or host name. Requires setting fiel
<code>tunnel_id_nxm</code>	Integer 64bit	Tunnel identifier.
<code>arp_sha_nxm</code>	MAC address	An address is specified as 6 pairs of hexadecimal digits delimited by colons. Re
<code>arp_tha_nxm</code>	MAC address	An address is specified as 6 pairs of hexadecimal digits delimited by colons. Re
<code>ipv6_src_nxm</code>	IPv6 address	IPv6 source address. Requires setting fields: <code>eth_type_nxm = 0x86dd (IPv6)</code>
<code>ipv6_dst_nxm</code>	IPv6 address	IPv6 destination address. Requires setting fields: <code>eth_type_nxm = 0x86dd (IPv6)</code>
<code>icmpv6_type_nxm</code>	Integer 8bit	Type matches the ICMP type and code matches the ICMP code. Requires setting
<code>icmpv6_code_nxm</code>	Integer 8bit	Type matches the ICMP type and code matches the ICMP code. Requires setting
<code>nd_target</code>	IPv6 address	The target address ipv6. Requires setting fields: <code>eth_type_nxm = 0x86dd (IPv6)</code>
<code>nd_sll</code>	MAC address	The source link-layer address option. Requires setting fields: <code>eth_type_nxm = 0</code>
<code>nd_tll</code>	MAC address	The target link-layer address option. Requires setting fields: <code>eth_type_nxm = 0x</code>
<code>ip_frag</code>	Integer 8bit	<code>frag_type</code> specifies what kind of IP fragments or non-fragments to match. Requi
<code>ipv6_label</code>	Integer 32bit	Matches IPv6 flow label. Requires setting fields: <code>eth_type_nxm = 0x86dd (IPv6)</code>
<code>ip_ecn_nxm</code>	Integer 8bit	Matches ecn bits in IP ToS or IPv6 traffic class fields. Requires setting fields: <code>eth</code>
<code>nw_ttl</code>	Integer 8bit	IP TTL or IPv6 hop limit value <code>ttl</code> . Requires setting fields: <code>eth_type_nxm = [0x</code>

Table 5 – continued from

Argument	Value	Description
mpls_ttl	Integer 8bit	The TTL of the outer MPLS label stack entry of a packet. Requires setting fields:
tun_ipv4_src	IPv4 address	Tunnel IPv4 source address. Requires setting fields: eth_type_nxm = 0x0800 (IP)
tun_ipv4_dst	IPv4 address	Tunnel IPv4 destination address. Requires setting fields: eth_type_nxm = 0x0800 (IP)
pkt_mark	Integer 32bit	Packet metadata mark.
tcp_flags_nxm	Integer 16bit	TCP Flags. Requires setting fields: eth_type_nxm = [0x0800 (IP)]0x86dd (IPv6)
conj_id	Integer 32bit	Conjunction ID used only with the conjunction action
tun_gbp_id	Integer 16bit	The group policy identifier in the VXLAN header.
tun_gbp_flags	Integer 8bit	The group policy flags in the VXLAN header.
tun_flags	Integer 16bit	Flags indicating various aspects of the tunnel encapsulation.
ct_state	Integer 32bit	Conntrack state.
ct_zone	Integer 16bit	Conntrack zone.
ct_mark	Integer 32bit	Conntrack mark.
ct_label	Integer 128bit	Conntrack label.
tun_ipv6_src	IPv6 address	Tunnel IPv6 source address. Requires setting fields: eth_type_nxm = 0x86dd (IPv6)
tun_ipv6_dst	IPv6 address	Tunnel IPv6 destination address. Requires setting fields: eth_type_nxm = 0x86dd (IPv6)
_recirc_id	Integer 32bit	ID for recirculation.
_dp_hash	Integer 32bit	Flow hash computed in Datapath.
nsh_flags	Integer 8bit	Flags field in NSH Base Header. Requires eth_type_nxm = 0x894f (NSH). Since
nsh_mdtype	Integer 8bit	Metadata Type in NSH Base Header. Requires eth_type_nxm = 0x894f (NSH). Since
nsh_np	Integer 8bit	Next Protocol type in NSH Base Header. Requires eth_type_nxm = 0x894f (NSH). Since
nsh_spi	Integer 32bit	Service Path Identifier in NSH Service Path Header. Requires eth_type_nxm = 0x894f (NSH). Since
nsh_si	Integer 8bit	Service Index in NSH Service Path Header. Requires eth_type_nxm = 0x894f (NSH). Since
nsh_c<N>	Integer 32bit	Context fields in NSH Context Header. <N> is a number of 1-4. Requires eth_type_nxm = 0x894f (NSH). Since
nsh_ttl	Integer 8bit	TTL field in NSH Base Header. Requires eth_type_nxm = 0x894f (NSH). Since
reg<idx>	Integer 32bit	Packet register. <idx> is register number 0-15.
xxreg<idx>	Integer 128bit	Packet extended-extended register. <idx> is register number 0-3.

**Note**

Setting the TCP flags via the nicira extensions. This is required when using OVS version < 2.4. When using the nxm fields, you need to use any nxm prereq fields as well or you will receive a OFPBMC\_BAD\_PREREQ error

Example:

```
# WILL NOT work
flag = tcp.TCP_ACK
match = parser.OFPMatch(
    tcp_flags_nxm=(flag, flag),
    ip_proto=inet.IPPROTO_TCP,
    eth_type=eth_type)

# Works
flag = tcp.TCP_ACK
match = parser.OFPMatch(
    tcp_flags_nxm=(flag, flag),
    ip_proto_nxm=inet.IPPROTO_TCP,
    eth_type_nxm=eth_type)
```

### 8.1.7 OS-Ken API Reference

**class** `os_ken.base.app_manager.OSKenApp(*_args, **_kwargs)`

The base class for OSKen applications.

OSKenApp subclasses are instantiated after osken-manager loaded all requested OSKen application modules. `__init__` should call `OSKenApp.__init__` with the same arguments. It's illegal to send any events in `__init__`.

The instance attribute 'name' is the name of the class used for message routing among OSKen applications. (Cf. `send_event`) It's set to `__class__.__name__` by `OSKenApp.__init__`. It's discouraged for subclasses to override this.

**OFP\_VERSIONS = None**

A list of supported OpenFlow versions for this OSKenApp. The default is all versions supported by the framework.

Examples:

```
OFP_VERSIONS = [ofproto_v1_0.OFP_VERSION,
                ofproto_v1_2.OFP_VERSION]
```

If multiple OSKen applications are loaded in the system, the intersection of their `OFP_VERSIONS` is used.

**\_CONTEXTS = {}**

A dictionary to specify contexts which this OSKen application wants to use. Its key is a name of context and its value is an ordinary class which implements the context. The class is instantiated by `app_manager` and the instance is shared among OSKenApp subclasses which has `_CONTEXTS` member with the same key. A OSKenApp subclass can obtain a reference to the instance via its `__init__`'s kwargs as the following.

Example:

```
_CONTEXTS = {
    'network': network.Network
}

def __init__(self, *_args, *_kwargs):
    self.network = kwargs['network']
```

**\_EVENTS = []**

A list of event classes which this OSKenApp subclass would generate. This should be specified if and only if event classes are defined in a different python module from the OSKenApp subclass is.

**close()**

teardown method. The method name, `close`, is chosen for python context manager

**classmethod context\_iteritems()**

Return iterator over the (key, contxt class) of application context

**reply\_to\_request(req, rep)**

Send a reply for a synchronous request sent by `send_request`. The first argument should be an instance of `EventRequestBase`. The second argument should be an instance of `EventReplyBase`.

**send\_event**(*name, ev, state=None*)

Send the specified event to the OSKenApp instance specified by name.

**send\_event\_to\_observers**(*ev, state=None*)

Send the specified event to all observers of this OSKenApp.

**send\_request**(*req*)

Make a synchronous request. Set `req.sync` to `True`, send it to a OSKen application specified by `req.dst`, and block until receiving a reply. Returns the received reply. The argument should be an instance of `EventRequestBase`.

**start**()

Hook that is called after startup initialization is done.

**class** `os_ken.controller.dpset.DPSet(*args, **kwargs)`

DPSet application manages a set of switches (datapaths) connected to this controller.

Usage Example:

```
# ... (snip) ...
from os_ken.controller import dpset

class MyApp(app_manager.OSKenApp):
    _CONTEXTS = {
        'dpset': dpset.DPSet,
    }

    def __init__(self, *args, **kwargs):
        super(MyApp, self).__init__(*args, **kwargs)
        # Stores DPSet instance to call its API in this app
        self.dpset = kwargs['dpset']

    def _my_handler(self):
        # Get the datapath object which has the given dpid
        dpid = 1
        dp = self.dpset.get(dpid)
        if dp is None:
            self.logger.info('No such datapath: dpid=%d', dpid)
```

**get**(*dp\_id*)

This method returns the `os_ken.controller.controller.Datapath` instance for the given Datapath ID.

**get\_all**()

This method returns a list of tuples which represents instances for switches connected to this controller. The tuple consists of a Datapath ID and an instance of `os_ken.controller.controller.Datapath`.

A return value looks like the following:

```
[ (dpid_A, Datapath_A), (dpid_B, Datapath_B), ... ]
```

**get\_port**(*dpid*, *port\_no*)

This method returns the `os_ken.controller.dpset.PortState` instance for the given Datapath ID and the port number. Raises `os_ken_exc.PortNotFound` if no such a datapath connected to this controller or no such a port exists.

**get\_ports**(*dpid*)

This method returns a list of `os_ken.controller.dpset.PortState` instances for the given Datapath ID. Raises `KeyError` if no such a datapath connected to this controller.

## 8.2 Configuration

### 8.2.1 Setup TLS Connection

If you want to use secure channel to connect OpenFlow switches, you need to use TLS connection. This document describes how to setup OS-Ken to connect to the Open vSwitch over TLS.

#### Configuring a Public Key Infrastructure

If you don't have a PKI, the `ovs-pki` script included with Open vSwitch can help you. This section is based on the `INSTALL.SSL` in the Open vSwitch source code.

NOTE: How to install Open vSwitch isn't described in this document. Please refer to the Open vSwitch documents.

Create a PKI by using `ovs-pki` script:

```
% ovs-pki init
(Default directory is /usr/local/var/lib/openvswitch/pki)
```

The `pki` directory consists of `controllerca` and `switchca` subdirectories. Each directory contains CA files.

Create a controller private key and certificate:

```
% ovs-pki req+sign ctl controller
```

`ctl-privkey.pem` and `ctl-cert.pem` are generated in the current directory.

Create a switch private key and certificate:

```
% ovs-pki req+sign sc switch
```

`sc-privkey.pem` and `sc-cert.pem` are generated in the current directory.

#### Testing TLS Connection

Configuring `ovs-vsctld` to use CA files using the `ovs-vsctl "set-ssl"` command, e.g.:

```
% ovs-vsctl set-ssl /etc/openvswitch/sc-privkey.pem \
  /etc/openvswitch/sc-cert.pem \
  /usr/local/var/lib/openvswitch/pki/controllerca/cacert.pem
% ovs-vsctl add-br br0
% ovs-vsctl set-controller br0 ssl:127.0.0.1:6633
```

Substitute the correct file names, if they differ from the ones used above. You should use absolute file names.

Run OS-Ken with CA files:

```
% osken-manager --ctl-privkey ctl-privkey.pem \
                --ctl-cert ctl-cert.pem \
                --ca-certs /usr/local/var/lib/openswitch/pki/switchca/cacert.
↪pem \
                --verbose
```

You can see something like:

```
loading app os_ken.controller.ofp_handler
instantiating app os_ken.controller.ofp_handler
BRICK ofp_event
  CONSUMES EventOFPSwitchFeatures
  CONSUMES EventOFPErrormsg
  CONSUMES EventOFPHello
  CONSUMES EventOFPEchoRequest
connected socket:<SSLSocket fileno=4 sock=127.0.0.1:6633 peer=127.0.0.1:61302>
↪ a
address:('127.0.0.1', 61302)
hello ev <os_ken.controller.ofp_event.EventOFPHello object at 0x1047806d0>
move onto config mode
switch features ev version: 0x1 msg_type 0x6 xid 0xb0bb34e5 port_
↪OFPPhyPort(port
  _no=65534, hw_addr='\x16\xdc\xa2\xe2}K', name='br0\x00\x00\x00\x00\x00\x00\
↪x00\x
00\x00\x00\x00\x00\x00', config=0, state=0, curr=0, advertised=0, supported=0,
↪ p
eer=0)
move onto main mode
```

## 8.3 Tests

### 8.3.1 Testing VRRP Module

This page describes how to test OS-Ken VRRP service

#### Running integrated tests

Some testing scripts are available.

- os\_ken/tests/integrated/test\_vrrp\_linux\_multi.py
- os\_ken/tests/integrated/test\_vrrp\_multi.py

Each files include how to run in the comment. Please refer to it.

#### Running multiple OS-Ken VRRP in network namespace

The following command lines set up necessary bridges and interfaces.

And then run OSKen-VRRP:

```
# ip netns add gateway1
# ip netns add gateway2

# ip link add dev vrrp-br0 type bridge
# ip link add dev vrrp-br1 type bridge

# ip link add veth0 type veth peer name veth0-br0
# ip link add veth1 type veth peer name veth1-br0
# ip link add veth2 type veth peer name veth2-br0
# ip link add veth3 type veth peer name veth3-br1
# ip link add veth4 type veth peer name veth4-br1
# ip link add veth5 type veth peer name veth5-br1

# ip link set dev veth0-br0 master vrrp-br0
# ip link set dev veth1-br0 master vrrp-br0
# ip link set dev veth2-br0 master vrrp-br0
# ip link set dev veth3-br0 master vrrp-br1
# ip link set dev veth4-br0 master vrrp-br1
# ip link set dev veth5-br0 master vrrp-br1

# ip link set vrrp-br0 up
# ip link set vrrp-br1 up

# ip link set veth0 up
# ip link set veth0-br0 up
# ip link set veth1-br0 up
# ip link set veth2-br0 up
# ip link set veth3-br1 up
# ip link set veth4-br1 up
# ip link set veth5 up
# ip link set veth5-br1 up

# ip link set veth1 netns gateway1
# ip link set veth2 netns gateway2
# ip link set veth3 netns gateway1
# ip link set veth4 netns gateway2

# ip netns exec gateway1 ip link set veth1 up
# ip netns exec gateway2 ip link set veth2 up
# ip netns exec gateway1 ip link set veth3 up
# ip netns exec gateway2 ip link set veth4 up

# ip netns exec gateway1 .os_ken-vrrp veth1 '10.0.0.2' 254
# ip netns exec gateway2 .os_ken-vrrp veth2 '10.0.0.3' 100
```

### Caveats

Please make sure that all interfaces and bridges are UP. Don't forget interfaces in netns gateway1/gateway2.

```

          ^ veth5
          |
          V veth5-br1
-----
|Linux Brirge vrrp-br1|
-----
veth3-br1^          ^ veth4-br1
  |              |
  veth3V          V veth4
-----          -----
|netns      |      |netns      |
|gateway1   |      |gateway2   |
|os_ken-vrrp|      |os_ken-vrrp|
-----          -----
veth1^          ^ veth2
  |              |
veth1-br0V      V veth2-br0
-----
|Linux Brirge vrrp-br0|
-----
          ^ veth0-br0
          |
          V veth0

```

Here's the helper executable, `os_ken-vrrp`:

```

#!/usr/bin/env python3
#
# Copyright (C) 2013 Nippon Telegraph and Telephone Corporation.
# Copyright (C) 2013 Isaku Yamahata <yamahata at valinux co jp>
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#   http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
# implied.
# See the License for the specific language governing permissions and
# limitations under the License.

from os_ken.lib import hub
hub.patch()

# TODO:
#   Right now, we have our own patched copy of ovs python bindings
#   Once our modification is upstreamed and widely deployed,

```

(continues on next page)



(continued from previous page)

```
# use it
#
# NOTE: this modifies sys.path and thus affects the following imports.
# eg. oslo.config.cfg.
import os_ken.contrib

from oslo.config import cfg
import logging
import netaddr
import sys
import time

from os_ken import log
log.early_init_log(logging.DEBUG)

from os_ken import flags
from os_ken import version
from os_ken.base import app_manager
from os_ken.controller import controller
from os_ken.lib import mac as lib_mac
from os_ken.lib.packet import vrrp
from os_ken.services.protocols.vrrp import api as vrrp_api
from os_ken.services.protocols.vrrp import event as vrrp_event

CONF = cfg.CONF

_VRID = 7
_IP_ADDRESS = '10.0.0.1'
_PRIORITY = 100

class VRRPTestRouter(app_manager.OSKenApp):
    def __init__(self, *args, **kwargs):
        super(VRRPTestRouter, self).__init__(*args, **kwargs)
        print args
        self.logger.debug('vrrp_config %s', args)
        self._ifname = args[0]
        self._primary_ip_address = args[1]
        self._priority = int(args[2])

    def start(self):
        print 'start'
        hub.spawn(self._main)

    def _main(self):
        print self
        interface = vrrp_event.VRRPInterfaceNetworkDevice(
            lib_mac.DONTCARE, self._primary_ip_address, None, self._ifname)
```

(continues on next page)

(continued from previous page)

```

self.logger.debug('%s', interface)

ip_addresses = [_IP_ADDRESS]
config = vrrp_event.VRRPConfig(
    version=vrrp.VRRP_VERSION_V3, vrid=_VRID, priority=self._priority,
    ip_addresses=ip_addresses)
self.logger.debug('%s', config)

rep = vrrp_api.vrrp_config(self, interface, config)
self.logger.debug('%s', rep)

def main():
    vrrp_config = sys.argv[-3:]
    sys.argv = sys.argv[:-3]
    CONF(project='os_ken', version='os_ken-vrrp %s' % version)

    log.init_log()
    # always enable ofp for now.
    app_lists = ['os_ken.services.protocols.vrrp.manager',
                'os_ken.services.protocols.vrrp.dumper',
                'os_ken.services.protocols.vrrp.sample_manager']

    app_mgr = app_manager.AppManager.get_instance()
    app_mgr.load_apps(app_lists)
    contexts = app_mgr.create_contexts()
    app_mgr.instantiate_apps(**contexts)
    vrrp_router = app_mgr.instantiate(VRRPTestRouter, *vrrp_config,
    ↪ **contexts)
    vrrp_router.start()

    while True:
        time.sleep(999999)

    app_mgr.close()

if __name__ == "__main__":
    main()

```

### 8.3.2 Testing OF-config support with LINC

This page describes how to setup LINC and test OS-Ken OF-config with it.

The procedure is as follows. Although all the procedure is written for reader's convenience, please refer to LINC document for latest informations of LINC.

<https://github.com/FlowForwarding/LINC-Switch>

The test procedure

- install Erlang environment

- build LINC
- configure LINC switch
- setup for LINC
- run LINC switch
- run OS-Ken test\_of\_config app

For getting/installing OS-Ken itself, please refer to [http://osrg.github.io/os\\_ken/](http://osrg.github.io/os_ken/)

## Install Erlang environment

Since LINC is written in Erlang, you need to install Erlang execution environment. Required version is R15B+.

The easiest way is to use binary package from <https://www.erlang-solutions.com/downloads/download-erlang-otp>

The distribution may also provide Erlang package.

## build LINC

### install necessary packages for build

### install necessary build tools

On Ubuntu:

```
# apt-get install git-core bridge-utils libpcap0.8 libpcap-dev libcap2-bin_
↪uml-utilities
```

On RedHat/CentOS:

```
# yum install git sudo bridge-utils libpcap libpcap-devel libcap tunc
```

Note that on RedHat/CentOS 5.x you need a newer version of libpcap:

```
# yum erase libpcap libpcap-devel
# yum install flex byacc
# wget http://www.tcpdump.org/release/libpcap-1.2.1.tar.gz
# tar xzf libpcap-1.2.1.tar.gz
# cd libpcap-1.2.1
# ./configure
# make && make install
```

## get LINC repo and built

Clone LINC repo:

```
% git clone git://github.com/FlowForwarding/LINC-Switch.git
```

Then compile everything:

```
% cd LINC-Switch
% make
```

### Note

At the time of this writing, `test_of_config` fails due to a bug of LINC. You can try this test with LINC which is built by the following methods.

```
% cd LINC-Switch
% make
% cd deps/of_config
% git reset --hard f772af4b765984381ad024ca8e5b5b8c54362638
% cd ../../
% make offline
```

## Setup LINC

edit LINC switch configuration file. `rel/linc/releases/0.1/sys.config` Here is the sample `sys.config` for `test_of_config` to run.

```
[{linc,
  [{of_config,enabled},
   {capable_switch_ports,
    [{port,1,[{interface,"linc-port"}]},
     {port,2,[{interface,"linc-port2"}]},
     {port,3,[{interface,"linc-port3"}]},
     {port,4,[{interface,"linc-port4"}]}]}],
   {capable_switch_queues,
    [
      {queue,991,[{min_rate,10},{max_rate,120}]},
      {queue,992,[{min_rate,10},{max_rate,130}]},
      {queue,993,[{min_rate,200},{max_rate,300}]},
      {queue,994,[{min_rate,400},{max_rate,900}]}
    ]},
   {logical_switches,
    [{switch,0,
      [{backend,linc_us4},
       {controllers,[{"Switch0-Default-Controller","127.0.0.1",6633,
→tcp}]},
       {controllers_listener,{"127.0.0.1",9998,tcp}},
       {queues_status,enabled},
       {ports,[{port,1,{queues,[]}},{port,2,{queues,[991,992]}]}]}]
     },
     {switch,7,
      [{backend,linc_us3},
       {controllers,[{"Switch7-Controller","127.0.0.1",6633,tcp}]},
       {controllers_listener,disabled},
       {queues_status,enabled},
       {ports,[{port,4,{queues,[]}},{port,3,{queues,[993,994]}]}]}]
     }
    ]}
  ]}
```

(continues on next page)

(continued from previous page)

```

    ]]]},
{enetconf,
  [{capabilities,
    [{base,{1,0}},
     {base,{1,1}},
     {startup,{1,0}},
     {'writable-running',{1,0}}]},
   {callback_module,linc_ofconfig},
   {sshd_ip,{127,0,0,1}},
   {sshd_port,1830},
   {sshd_user_passwords,[{"linc","linc"}]}]},
{lager,
  [{handlers,
    [{lager_console_backend,debug},
     {lager_file_backend,
      [{"log/error.log",error,10485760,"$D0",5},
       {"log/console.log",info,10485760,"$D0",5}]}]}]},
{sasl,
  [{sasl_error_logger,{file,"log/sasl-error.log"}},
   {errlog_type,error},
   {error_logger_mf_dir,"log/sasl"},
   {error_logger_mf_maxbytes,10485760},
   {error_logger_mf_maxfiles,5}]},
{sync,[{excluded_modules,[procket]}]}].

```

### setup for LINC

As the above sys.config requires some network interface, create them:

```

# ip link add linc-port type veth peer name linc-port-peer
# ip link set linc-port up
# ip link add linc-port2 type veth peer name linc-port-peer2
# ip link set linc-port2 up
# ip link add linc-port3 type veth peer name linc-port-peer3
# ip link set linc-port3 up
# ip link add linc-port4 type veth peer name linc-port-peer4
# ip link set linc-port4 up

```

After stopping LINC, those created interfaces can be deleted:

```

# ip link delete linc-port
# ip link delete linc-port2
# ip link delete linc-port3
# ip link delete linc-port4

```

## Starting LINC OpenFlow switch

Then run LINC:

```
# rel/linc/bin/linc console
```

## 8.4 Snort Intergration

This document describes how to integrate OS-Ken with Snort.

### 8.4.1 Overview

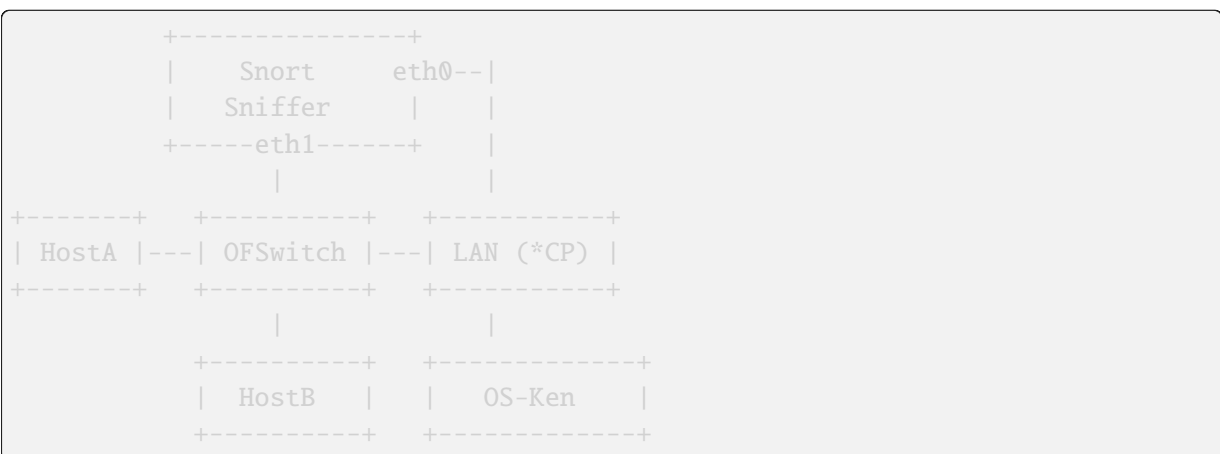
There are two options can send alert to OS-Ken controller. The Option 1 is easier if you just want to demonstrate or test. Since Snort need very large computation power for analyzing packets you can choose Option 2 to separate them.

#### [Option 1] OS-Ken and Snort are on the same machine



The above depicts OS-Ken and Snort architecture. OS-Ken receives Snort alert packet via **Unix Domain Socket** . To monitor packets between HostA and HostB, installing a flow that mirrors packets to Snort.

#### [Option 2] OS-Ken and Snort are on the different machines



#### \*CP: Control Plane

The above depicts OS-Ken and Snort architecture. OS-Ken receives Snort alert packet via **Network Socket** . To monitor packets between HostA and HostB, installing a flow that mirrors packets to Snort.

## 8.4.2 Installation Snort

Snort is an open source network intrusion prevention and detection system developed by Sourcefire. If you are not familiar with installing/setting up Snort, please refer to snort setup guides.

<http://www.snort.org/documents>

## 8.4.3 Configure Snort

The configuration example is below:

- Add a snort rules file into `/etc/snort/rules` named `Myrules.rules`

```
alert icmp any any -> any any (msg:"Pinging...";sid:1000004;)
alert tcp any any -> any 80 (msg:"Port 80 is accessing"; sid:1000003;)
```

- Add the custom rules in `/etc/snort/snort.conf`

```
include $RULE_PATH/Myrules.rules
```

Configure NIC as a promiscuous mode.

```
$ sudo ifconfig eth1 promisc
```

## 8.4.4 Usage

### [Option 1]

1. Modify the `simple_switch_snort.py`:

```
socket_config = {'unixsock': True}
# True: Unix Domain Socket Server [Option1]
# False: Network Socket Server [Option2]
```

2. Run OS-Ken with sample application:

```
$ sudo osken-manager os_ken/app/simple_switch_snort.py
```

The incoming packets will all mirror to **port 3** which should be connect to Snort network interface. You can modify the mirror port by assign a new value in the `self.snort_port = 3` of `simple_switch_snort.py`

3. Run Snort:

```
$ sudo -i
$ snort -i eth1 -A unsock -l /tmp -c /etc/snort/snort.conf
```

4. Send an ICMP packet from HostA (192.168.8.40) to HostB (192.168.8.50):

```
$ ping 192.168.8.50
```

5. You can see the result under next section.

### [Option 2]

1. Modify the `simple_switch_snort.py`:

```
socket_config = {'unixsock': False}
# True: Unix Domain Socket Server [Option1]
# False: Network Socket Server [Option2]
```

2. Run OS-Ken with sample application (On the Controller):

```
$ osken-manager os_ken/app/simple_switch_snort.py
```

3. Run Snort (On the Snort machine):

```
$ sudo -i
$ snort -i eth1 -A unsock -l /tmp -c /etc/snort/snort.conf
```

4. Run pigrelay.py (On the Snort machine):

```
$ sudo python pigrelay.py
```

This program listening snort alert messages from unix domain socket and sending it to OS-Ken using network socket.

You can clone the source code from this repo. <https://github.com/John-Lin/pigrelay>

5. Send an ICMP packet from HostA (192.168.8.40) to HostB (192.168.8.50):

```
$ ping 192.168.8.50
```

6. You can see the alert message below:

```
alertmsg: Pinging...
icmp(code=0,csum=19725,data=echo(data=array('B', [97, 98, 99, 100, 101,
↪102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115,
↪116, 117, 118, 119, 97, 98, 99, 100, 101, 102, 103, 104, 105])),id=1,
↪seq=78),type=8)

ipv4(csum=42562,dst='192.168.8.50',flags=0,header_length=5,
↪identification=724,offset=0,option=None,proto=1,src='192.168.8.40',
↪tos=0,total_length=60,ttl=128,version=4)

ethernet(dst='00:23:54:5a:05:14',ethertype=2048,src='00:23:54:6c:1d:17')

alertmsg: Pinging...
icmp(code=0,csum=21773,data=echo(data=array('B', [97, 98, 99, 100, 101,
↪102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115,
↪116, 117, 118, 119, 97, 98, 99, 100, 101, 102, 103, 104, 105])),id=1,
↪seq=78),type=0)

ipv4(csum=52095,dst='192.168.8.40',flags=0,header_length=5,
↪identification=7575,offset=0,option=None,proto=1,src='192.168.8.50',
↪tos=0,total_length=60,ttl=64,version=4)
```



## 8.5 Built-in OS-Ken applications

OS-Ken has some built-in OS-Ken applications. Some of them are examples. Others provide some functionalities to other OS-Ken applications.

### 8.5.1 os\_ken.app.ofctl

os\_ken.app.ofctl provides a convenient way to use OpenFlow messages synchronously.

OfctlService os\_ken application is automatically loaded if your OS-Ken application imports ofctl.api module.

Example:

```
import os_ken.app.ofctl.api
```

OfctlService application internally uses OpenFlow barrier messages to ensure message boundaries. As OpenFlow messages are asynchronous and some of messages does not have any replies on success, barriers are necessary for correct error handling.

#### api module

os\_ken.app.ofctl.api.get\_datapath(*app*, *dpid=None*)

Get datapath object by dpid.

##### Parameters

- **app** -- Client OSKenApp instance
- **dpid** -- Datapath ID (int type) or None to get all datapath objects

Returns a object of datapath, a list of datapath objects when no dpid given or None when error.

Raises an exception if any of the given values is invalid.

Example:

```
# ... (snip) ...
import os_ken.app.ofctl.api as ofctl_api

class MyApp(app_manager.OSKenApp):

    def _my_handler(self, ev):
        # Get all datapath objects
        result = ofctl_api.get_datapath(self)

        # Get the datapath object which has the given dpid
        result = ofctl_api.get_datapath(self, dpid=1)
```

os\_ken.app.ofctl.api.send\_msg(*app*, *msg*, *reply\_cls=None*, *reply\_multi=False*)

Send an OpenFlow message and wait for reply messages.

##### Parameters

- **app** -- Client OSKenApp instance
- **msg** -- An OpenFlow controller-to-switch message to send

- **reply\_cls** -- OpenFlow message class for expected replies. None means no replies are expected. The default is None.
- **reply\_multi** -- True if multipart replies are expected. The default is False.

If no replies, returns None. If `reply_multi=False`, returns OpenFlow switch-to-controller message. If `reply_multi=True`, returns a list of OpenFlow switch-to-controller messages.

Raise an exception on error.

Example:

```
# ... (snip) ...
import os_ken.app.ofctl.api as ofctl_api

class MyApp(app_manager.OSKenApp):

    def _my_handler(self, ev):
        # ... (snip) ...
        msg = parser.OFPPortDescStatsRequest(datapath=datapath)
        result = ofctl_api.send_msg(
            self, msg,
            reply_cls=parser.OFPPortDescStatsReply,
            reply_multi=True)
```

## exceptions

**exception** `os_ken.app.ofctl.exception.InvalidDatapath(result)`

Datapath is invalid.

This can happen when the bridge disconnects.

**exception** `os_ken.app.ofctl.exception.OFError(result)`

OFErrorMsg is received.

**exception** `os_ken.app.ofctl.exception.UnexpectedMultiReply(result)`

Two or more replies are received for `reply_multi=False` request.

## PYTHON MODULE INDEX

### O

- os\_ken.app.ofctl.api, 583
- os\_ken.app.ofctl.exception, 584
- os\_ken.base.app\_manager, 10
- os\_ken.controller.controller, 10
- os\_ken.controller.dpset, 10
- os\_ken.controller.ofp\_event, 10
- os\_ken.controller.ofp\_handler, 10
- os\_ken.lib.netconf, 12
- os\_ken.lib.of\_config, 12
- os\_ken.lib.ovs, 12
- os\_ken.lib.ovs.bridge, 156
- os\_ken.lib.ovs.vsctl, 155
- os\_ken.lib.packet, 12
- os\_ken.lib.packet.arp, 23
- os\_ken.lib.packet.bfd, 24
- os\_ken.lib.packet.bgp, 30
- os\_ken.lib.packet.bmp, 50
- os\_ken.lib.packet.bpdu, 56
- os\_ken.lib.packet.cfm, 61
- os\_ken.lib.packet.dhcp, 67
- os\_ken.lib.packet.dhcp6, 69
- os\_ken.lib.packet.ethernet, 71
- os\_ken.lib.packet.geneve, 72
- os\_ken.lib.packet.gre, 73
- os\_ken.lib.packet.icmp, 75
- os\_ken.lib.packet.icmpv6, 76
- os\_ken.lib.packet.igmp, 81
- os\_ken.lib.packet.ipv4, 86
- os\_ken.lib.packet.ipv6, 87
- os\_ken.lib.packet.llc, 91
- os\_ken.lib.packet.lldp, 94
- os\_ken.lib.packet.mpls, 97
- os\_ken.lib.packet.openflow, 98
- os\_ken.lib.packet.ospf, 99
- os\_ken.lib.packet.packet, 21
- os\_ken.lib.packet.packet\_base, 22
- os\_ken.lib.packet.pbb, 101
- os\_ken.lib.packet.sctp, 102
- os\_ken.lib.packet.slow, 115
- os\_ken.lib.packet.stream\_parser, 22
- os\_ken.lib.packet.tcp, 120
- os\_ken.lib.packet.udp, 121
- os\_ken.lib.packet.vlan, 121
- os\_ken.lib.packet.vrrp, 122
- os\_ken.lib.packet.vxlan, 126
- os\_ken.lib.packet.zebra, 127
- os\_ken.lib.xflow, 12
- os\_ken.ofproto.nicira\_ext, 567
- os\_ken.ofproto.ofproto\_v1\_0, 10
- os\_ken.ofproto.ofproto\_v1\_0\_parser, 11
- os\_ken.ofproto.ofproto\_v1\_2, 11
- os\_ken.ofproto.ofproto\_v1\_2\_parser, 11
- os\_ken.ofproto.ofproto\_v1\_3, 11
- os\_ken.ofproto.ofproto\_v1\_3\_parser, 11
- os\_ken.ofproto.ofproto\_v1\_4, 11
- os\_ken.ofproto.ofproto\_v1\_4\_parser, 11
- os\_ken.ofproto.ofproto\_v1\_5, 12
- os\_ken.ofproto.ofproto\_v1\_5\_parser, 12
- os\_ken.topology, 12

## Symbols

- `_CONTEXTS` (*os\_ken.base.app\_manager.OSKenApp* attribute), 569
  - `_EVENTS` (*os\_ken.base.app\_manager.OSKenApp* attribute), 569
  - `_TYPE` (*os\_ken.ofproto.ofproto\_parser.MsgBase* attribute), 160
- ### A
- `add_bond()` (*os\_ken.lib.ovs.bridge.OVSBridge* method), 156
  - `add_db_attribute()` (*os\_ken.lib.ovs.bridge.OVSBridge* method), 157
  - `add_gre_port()` (*os\_ken.lib.ovs.bridge.OVSBridge* method), 157
  - `add_protocol()` (*os\_ken.lib.packet.packet.Packet* method), 21
  - `add_tunnel_port()` (*os\_ken.lib.ovs.bridge.OVSBridge* method), 157
  - `add_vxlan_port()` (*os\_ken.lib.ovs.bridge.OVSBridge* method), 157
  - `AdminReset`, 30
  - `AdminShutdown`, 30
  - `arp` (class in *os\_ken.lib.packet.arp*), 23
  - `arp_ip()` (in module *os\_ken.lib.packet.arp*), 24
  - `ASPathFilter` (class in *os\_ken.services.protocols.bgp.info\_base.base*), 149
  - `AttrFlagError`, 30
  - `attribute_map_get()` (*os\_ken.services.protocols.bgp.bgpspeaker.BGPSpeaker* method), 139
  - `attribute_map_set()` (*os\_ken.services.protocols.bgp.bgpspeaker.BGPSpeaker* method), 139
  - `AttributeMap` (class in *os\_ken.services.protocols.bgp.info\_base.base*), 150
  - `AttrLenError`, 31
  - `auth` (class in *os\_ken.lib.packet.ipv6*), 87
  - `authenticate()` (*os\_ken.lib.packet.bfd.bfd* method), 29
  - `authenticate()` (*os\_ken.lib.packet.bfd.KeyedMD5* method), 26
  - `authenticate()` (*os\_ken.lib.packet.bfd.KeyedSHA1* method), 27
  - `authenticate()` (*os\_ken.lib.packet.bfd.SimplePassword* method), 28
  - `AuthFailure`, 31
- ### B
- `BadBgpId`, 36
  - `BadLen`, 36
  - `BadMsg`, 37
  - `BadNotification`, 37
  - `BadPeerAs`, 37
  - `bfd` (class in *os\_ken.lib.packet.bfd*), 28
  - `BFDAuth` (class in *os\_ken.lib.packet.bfd*), 25
  - `BGPevpnEsiLabelExtendedCommunity` (class in *os\_ken.lib.packet.bgp*), 31
  - `BGPevpnEsImportRTEExtendedCommunity` (class in *os\_ken.lib.packet.bgp*), 31
  - `BGPevpnMacMobilityExtendedCommunity` (class in *os\_ken.lib.packet.bgp*), 31
  - `BgpExc`, 37
  - `BGPFlowSpecRedirectCommunity` (class in *os\_ken.lib.packet.bgp*), 31
  - `BGPFlowSpecTPIDActionCommunity` (class in *os\_ken.lib.packet.bgp*), 31
  - `BGPFlowSpecTrafficActionCommunity` (class in *os\_ken.lib.packet.bgp*), 31
  - `BGPFlowSpecTrafficMarkingCommunity` (class in *os\_ken.lib.packet.bgp*), 32
  - `BGPFlowSpecTrafficRateCommunity` (class in *os\_ken.lib.packet.bgp*), 32

- os\_ken.lib.packet.bgp*), 32
- BGPFlowSpecVlanActionCommunity (class in *os\_ken.lib.packet.bgp*), 32
- BGPKeepAlive (class in *os\_ken.lib.packet.bgp*), 32
- BGPMessage (class in *os\_ken.lib.packet.bgp*), 33
- BGPNotification (class in *os\_ken.lib.packet.bgp*), 33
- BGPOpen (class in *os\_ken.lib.packet.bgp*), 34
- BGPPathAttributePmsiTunnel (class in *os\_ken.lib.packet.bgp*), 35
- BGPRouteRefresh (class in *os\_ken.lib.packet.bgp*), 35
- BGPSpeaker (class in *os\_ken.services.protocols.bgp.bgpspeaker*), 138
- BGPUpdate (class in *os\_ken.lib.packet.bgp*), 35
- bmp\_server\_add() (method), 140
- bmp\_server\_del() (method), 140
- BMPInitiation (class in *os\_ken.lib.packet.bmp*), 50
- BMPMessage (class in *os\_ken.lib.packet.bmp*), 51
- BMPPeerDownNotification (class in *os\_ken.lib.packet.bmp*), 51
- BMPPeerMessage (class in *os\_ken.lib.packet.bmp*), 52
- BMPPeerUpNotification (class in *os\_ken.lib.packet.bmp*), 53
- BMPRouteMonitoring (class in *os\_ken.lib.packet.bmp*), 53
- BMPStatisticsReport (class in *os\_ken.lib.packet.bmp*), 54
- BMPTermination (class in *os\_ken.lib.packet.bmp*), 55
- bpdu (class in *os\_ken.lib.packet.bpdu*), 60
- ## C
- cause\_cookie\_while\_shutdown (class in *os\_ken.lib.packet.sctp*), 102
- cause\_invalid\_param (class in *os\_ken.lib.packet.sctp*), 102
- cause\_invalid\_stream\_id (class in *os\_ken.lib.packet.sctp*), 102
- cause\_missing\_param (class in *os\_ken.lib.packet.sctp*), 103
- cause\_no\_userdata (class in *os\_ken.lib.packet.sctp*), 103
- cause\_out\_of\_resource (class in *os\_ken.lib.packet.sctp*), 103
- cause\_protocol\_violation (class in *os\_ken.lib.packet.sctp*), 104
- cause\_restart\_with\_new\_addr (class in *os\_ken.lib.packet.sctp*), 104
- cause\_stale\_cookie (class in *os\_ken.lib.packet.sctp*), 104
- cause\_unrecognized\_chunk (class in *os\_ken.lib.packet.sctp*), 105
- cause\_unrecognized\_param (class in *os\_ken.lib.packet.sctp*), 105
- cause\_unresolvable\_addr (class in *os\_ken.lib.packet.sctp*), 105
- cause\_user\_initiated\_abort (class in *os\_ken.lib.packet.sctp*), 106
- cc\_message (class in *os\_ken.lib.packet.cfm*), 61
- cfm (class in *os\_ken.lib.packet.cfm*), 61
- CGMP (class in *os\_ken.lib.packet.lldp*), 94
- chunk\_abort (class in *os\_ken.lib.packet.sctp*), 106
- chunk\_cookie\_ack (class in *os\_ken.lib.packet.sctp*), 106
- chunk\_cookie\_echo (class in *os\_ken.lib.packet.sctp*), 107
- chunk\_cwr (class in *os\_ken.lib.packet.sctp*), 107
- chunk\_data (class in *os\_ken.lib.packet.sctp*), 107
- chunk\_ecn\_echo (class in *os\_ken.lib.packet.sctp*), 108
- chunk\_error (class in *os\_ken.lib.packet.sctp*), 108
- chunk\_heartbeat (class in *os\_ken.lib.packet.sctp*), 109
- chunk\_heartbeat\_ack (class in *os\_ken.lib.packet.sctp*), 109
- chunk\_init (class in *os\_ken.lib.packet.sctp*), 109
- chunk\_init\_ack (class in *os\_ken.lib.packet.sctp*), 110
- chunk\_sack (class in *os\_ken.lib.packet.sctp*), 110
- chunk\_shutdown (class in *os\_ken.lib.packet.sctp*), 111
- chunk\_shutdown\_ack (class in *os\_ken.lib.packet.sctp*), 111
- chunk\_shutdown\_complete (class in *os\_ken.lib.packet.sctp*), 111
- clear\_db\_attribute() (method), 157
- clone() (method), 150
- clone() (method), 150

- method), 151
- `clone()` (*os\_ken.services.protocols.bgp.info\_base.base.PrefixAttribute*), 49
- method), 149
- `close()` (*os\_ken.base.app\_manager.OSKenApp* method), 569
- CODE (*os\_ken.lib.packet.bgp.AdminReset* attribute), 30
- CODE (*os\_ken.lib.packet.bgp.AdminShutdown* attribute), 30
- CODE (*os\_ken.lib.packet.bgp.AttrFlagError* attribute), 30
- CODE (*os\_ken.lib.packet.bgp.AttrLenError* attribute), 31
- CODE (*os\_ken.lib.packet.bgp.AuthFailure* attribute), 31
- CODE (*os\_ken.lib.packet.bgp.BadBgpId* attribute), 36
- CODE (*os\_ken.lib.packet.bgp.BadLen* attribute), 36
- CODE (*os\_ken.lib.packet.bgp.BadMsg* attribute), 37
- CODE (*os\_ken.lib.packet.bgp.BadPeerAs* attribute), 37
- CODE (*os\_ken.lib.packet.bgp.BgpExc* attribute), 37
- CODE (*os\_ken.lib.packet.bgp.CollisionResolution* attribute), 37
- CODE (*os\_ken.lib.packet.bgp.ConnRejected* attribute), 38
- CODE (*os\_ken.lib.packet.bgp.FiniteStateMachineError* attribute), 40
- CODE (*os\_ken.lib.packet.bgp.HoldTimerExpired* attribute), 46
- CODE (*os\_ken.lib.packet.bgp.InvalidNetworkField* attribute), 46
- CODE (*os\_ken.lib.packet.bgp.InvalidNextHop* attribute), 47
- CODE (*os\_ken.lib.packet.bgp.InvalidOriginError* attribute), 47
- CODE (*os\_ken.lib.packet.bgp.MalformedAsPath* attribute), 47
- CODE (*os\_ken.lib.packet.bgp.MalformedAttrList* attribute), 47
- CODE (*os\_ken.lib.packet.bgp.MalformedOptionalParameter* attribute), 47
- CODE (*os\_ken.lib.packet.bgp.MaxPrefixReached* attribute), 48
- CODE (*os\_ken.lib.packet.bgp.MissingWellKnown* attribute), 48
- CODE (*os\_ken.lib.packet.bgp.NotSync* attribute), 48
- CODE (*os\_ken.lib.packet.bgp.OptAttrError* attribute), 48
- CODE (*os\_ken.lib.packet.bgp.OtherConfChange* attribute), 48
- CODE (*os\_ken.lib.packet.bgp.OutOfResource* attribute), 49
- CODE (*os\_ken.lib.packet.bgp.PeerDeConfig* attribute), 49
- CODE (*os\_ken.lib.packet.bgp.RoutingLoop* attribute), 49
- CODE (*os\_ken.lib.packet.bgp.UnacceptableHoldTime* attribute), 50
- CODE (*os\_ken.lib.packet.bgp.UnRegWellKnowAttr* attribute), 49
- CODE (*os\_ken.lib.packet.bgp.UnsupportedOptParam* attribute), 50
- CODE (*os\_ken.lib.packet.bgp.UnsupportedVersion* attribute), 50
- CollisionResolution, 37
- ConfigurationBPDUs (class in *os\_ken.lib.packet.bpdu*), 57
- ConnRejected, 37
- `context_iteritems()` (*os\_ken.base.app\_manager.OSKenApp* class method), 569
- ControlFormatI (class in *os\_ken.lib.packet.llc*), 92
- ControlFormatS (class in *os\_ken.lib.packet.llc*), 92
- ControlFormatU (class in *os\_ken.lib.packet.llc*), 93
- `create()` (*os\_ken.lib.packet.vrrp.vrrpv2* static method), 125
- `create()` (*os\_ken.lib.packet.vrrp.vrrpv3* static method), 125
- `create_packet()` (*os\_ken.lib.packet.vrrp.vrrp* method), 124
- ## D
- `data_tlv` (class in *os\_ken.lib.packet.cfm*), 62
- Datapath (class in *os\_ken.controller.controller*), 15
- `db_get_map()` (*os\_ken.lib.ovs.bridge.OVSBridge* method), 157
- `db_get_val()` (*os\_ken.lib.ovs.bridge.OVSBridge* method), 157
- `del_controller()` (*os\_ken.lib.ovs.bridge.OVSBridge* method), 158
- `del_port()` (*os\_ken.lib.ovs.bridge.OVSBridge* method), 158
- `del_qos()` (*os\_ken.lib.ovs.bridge.OVSBridge* method), 158
- `delete_port()` (*os\_ken.lib.ovs.bridge.OVSBridge* method), 158



- `dest_unreach` (class in `os_ken.lib.packet.icmp`), 75
- `dhcp` (class in `os_ken.lib.packet.dhcp`), 67
- `dhcp6` (class in `os_ken.lib.packet.dhcp6`), 70
- `DPSet` (class in `os_ken.controller.dpset`), 570
- `dst_opts` (class in `os_ken.lib.packet.ipv6`), 88
- ## E
- `echo` (class in `os_ken.lib.packet.icmp`), 75
- `echo` (class in `os_ken.lib.packet.icmpv6`), 76
- `End` (class in `os_ken.lib.packet.lldp`), 94
- `ethernet` (class in `os_ken.lib.packet.ethernet`), 71
- `evaluate()` (`os_ken.services.protocols.bgp.info_base.base.ASPacketFilter` method), 150
- `evaluate()` (`os_ken.services.protocols.bgp.info_base.base.AttributeMap` method), 151
- `evaluate()` (`os_ken.services.protocols.bgp.info_base.base.PrefixFilter` method), 149
- `EventBase` (class in `os_ken.controller.event`), 15
- `EventDP` (class in `os_ken.controller.dpset`), 16
- `EventMacAddress` (class in `os_ken.controller.network`), 18
- `EventNetworkDel` (class in `os_ken.controller.network`), 18
- `EventNetworkPort` (class in `os_ken.controller.network`), 17
- `EventOFPMsgBase` (class in `os_ken.controller.ofp_event`), 14
- `EventOFPPortStateChange` (class in `os_ken.controller.ofp_event`), 16
- `EventOFPPortStateChange` (class in `os_ken.controller.ofp_event`), 16
- `EventPortAdd` (class in `os_ken.controller.dpset`), 16
- `EventPortDelete` (class in `os_ken.controller.dpset`), 17
- `EventPortModify` (class in `os_ken.controller.dpset`), 17
- `EventPrefix` (class in `os_ken.services.protocols.bgp.bgpspeaker`), 148
- `EventReplyBase` (class in `os_ken.controller.event`), 16
- `EventRequestBase` (class in `os_ken.controller.event`), 15
- `EventTunnelKeyAdd` (class in `os_ken.controller.tunnels`), 18
- `EventTunnelKeyDel` (class in `os_ken.controller.tunnels`), 18
- `EventTunnelPort` (class in `os_ken.controller.tunnels`), 19
- `evpn_prefix_add()` (`os_ken.services.protocols.bgp.bgpspeaker.BGPspeaker` method), 140
- `evpn_prefix_del()` (`os_ken.services.protocols.bgp.bgpspeaker.BGPspeaker` method), 141
- `EvpnArbitraryEsi` (class in `os_ken.lib.packet.bgp`), 38
- `EvpnASBasedEsi` (class in `os_ken.lib.packet.bgp`), 38
- `EvpnEsi` (class in `os_ken.lib.packet.bgp`), 38
- `EvpnEthernetAutoDiscoveryNLRI` (class in `os_ken.lib.packet.bgp`), 38
- `EvpnEthernetSegmentNLRI` (class in `os_ken.lib.packet.bgp`), 38
- `EvpnInclusiveMulticastEthernetTagNLRI` (class in `os_ken.lib.packet.bgp`), 38
- `EvpnIpPrefixNLRI` (class in `os_ken.lib.packet.bgp`), 39
- `EvpnL2BridgeEsi` (class in `os_ken.lib.packet.bgp`), 39
- `EvpnLACPesi` (class in `os_ken.lib.packet.bgp`), 39
- `EvpnMacBasedEsi` (class in `os_ken.lib.packet.bgp`), 39
- `EvpnMacIPAdvertisementNLRI` (class in `os_ken.lib.packet.bgp`), 39
- `EvpnNLRI` (class in `os_ken.lib.packet.bgp`), 39
- `EvpnRouterIDesi` (class in `os_ken.lib.packet.bgp`), 39
- `EvpnUnknownEsi` (class in `os_ken.lib.packet.bgp`), 39
- `EvpnUnknownNLRI` (class in `os_ken.lib.packet.bgp`), 39
- ## F
- `find_db_attributes()` (`os_ken.lib.ovs.bridge.OVSBridge` method), 158
- `FiniteStateMachineError`, 39
- `flowspec_prefix_add()` (`os_ken.services.protocols.bgp.bgpspeaker.BGPspeaker` method), 141
- `flowspec_prefix_del()` (`os_ken.services.protocols.bgp.bgpspeaker.BGPspeaker` method), 143
- `FlowSpecComponentUnknown` (class in `os_ken.lib.packet.bgp`), 40
- `FlowSpecDestinationMac` (class in `os_ken.lib.packet.bgp`), 40
- `FlowSpecDestPort` (class in `os_ken.lib.packet.bgp`), 40

FlowSpecDestPrefix	(class in <i>os_ken.lib.packet.bgp</i> ), 40	in	<i>os_ken.lib.packet.bgp</i> ), 45
FlowSpecDSCP	(class in <i>os_ken.lib.packet.bgp</i> ), 40		FlowSpecTCPFlags (class in <i>os_ken.lib.packet.bgp</i> ), 45
FlowSpecEtherType	(class in <i>os_ken.lib.packet.bgp</i> ), 40	in	FlowSpecVLANCoS (class in <i>os_ken.lib.packet.bgp</i> ), 45
FlowSpecFragment	(class in <i>os_ken.lib.packet.bgp</i> ), 40	in	FlowSpecVLANID (class in <i>os_ken.lib.packet.bgp</i> ), 45
FlowSpecIcmpCode	(class in <i>os_ken.lib.packet.bgp</i> ), 43	in	FlowSpecVPNv4NLRI (class in <i>os_ken.lib.packet.bgp</i> ), 45
FlowSpecIcmpType	(class in <i>os_ken.lib.packet.bgp</i> ), 44	in	FlowSpecVPNv6NLRI (class in <i>os_ken.lib.packet.bgp</i> ), 46
FlowSpecInnerVLANCoS	(class in <i>os_ken.lib.packet.bgp</i> ), 44	in	fragment (class in <i>os_ken.lib.packet.ipv6</i> ), 88
FlowSpecInnerVLANID	(class in <i>os_ken.lib.packet.bgp</i> ), 44	in	from_jsondict() (in <i>os_ken.lib.packet.bgp.BGPPathAttributePmsiTunnel</i> class method), 35
FlowSpecIPProtocol	(class in <i>os_ken.lib.packet.bgp</i> ), 40	in	from_jsondict() (in <i>os_ken.lib.packet.packet.Packet</i> class method), 21
FlowSpecIPv4NLRI	(class in <i>os_ken.lib.packet.bgp</i> ), 40	in	from_jsondict() (in <i>os_ken.ofproto.ofproto_parser.MsgBase</i> class method), 161
FlowSpecIPv6DestPrefix	(class in <i>os_ken.lib.packet.bgp</i> ), 42	in	from_user() ( <i>os_ken.lib.packet.bgp.FlowSpecIPv4NLRI</i> class method), 41
FlowSpecIPv6FlowLabel	(class in <i>os_ken.lib.packet.bgp</i> ), 42	in	from_user() ( <i>os_ken.lib.packet.bgp.FlowSpecIPv6NLRI</i> class method), 43
FlowSpecIPv6Fragment	(class in <i>os_ken.lib.packet.bgp</i> ), 42	in	from_user() ( <i>os_ken.lib.packet.bgp.FlowSpecL2VPNNLRI</i> class method), 44
FlowSpecIPv6NLRI	(class in <i>os_ken.lib.packet.bgp</i> ), 42	in	from_user() ( <i>os_ken.lib.packet.bgp.FlowSpecVPNv4NLRI</i> class method), 45
FlowSpecIPv6SrcPrefix	(class in <i>os_ken.lib.packet.bgp</i> ), 43	in	from_user() ( <i>os_ken.lib.packet.bgp.FlowSpecVPNv6NLRI</i> class method), 46
FlowSpecL2VPNNLRI	(class in <i>os_ken.lib.packet.bgp</i> ), 44	in	
FlowSpecLLCControl	(class in <i>os_ken.lib.packet.bgp</i> ), 44	in	<b>G</b>
FlowSpecLLCDSAP	(class in <i>os_ken.lib.packet.bgp</i> ), 44	in	geneve (class in <i>os_ken.lib.packet.geneve</i> ), 72
FlowSpecLLCSSAP	(class in <i>os_ken.lib.packet.bgp</i> ), 44	in	get() ( <i>os_ken.controller.dpset.DPSet</i> method), 570
FlowSpecNextHeader	(class in <i>os_ken.lib.packet.bgp</i> ), 45	in	get_all() ( <i>os_ken.controller.dpset.DPSet</i> method), 570
FlowSpecPacketLen	(class in <i>os_ken.lib.packet.bgp</i> ), 45	in	get_controller() (in <i>os_ken.lib.ovs.bridge.OVSBridge</i> method), 158
FlowSpecPort	(class in <i>os_ken.lib.packet.bgp</i> ), 45		get_datapath() (in <i>os_ken.app.ofctl.api</i> ), 583
FlowSpecSNAP	(class in <i>os_ken.lib.packet.bgp</i> ), 45		get_datapath_id() (in <i>os_ken.lib.ovs.bridge.OVSBridge</i> method), 158
FlowSpecSourceMac	(class in <i>os_ken.lib.packet.bgp</i> ), 45	in	get_db_attribute() (in <i>os_ken.lib.ovs.bridge.OVSBridge</i> method), 159
FlowSpecSrcPort	(class in <i>os_ken.lib.packet.bgp</i> ), 45	in	get_ofport() ( <i>os_ken.lib.ovs.bridge.OVSBridge</i> method), 159
FlowSpecSrcPrefix	(class in <i>os_ken.lib.packet.bgp</i> ), 45	in	



- `get_packet_type()` (`os_ken.lib.packet.ethernet.ethernet` class method), 72
- `get_packet_type()` (`os_ken.lib.packet.packet_base.PacketBase` class method), 22
- `get_packet_type()` (`os_ken.lib.packet.udp.udp` static method), 121
- `get_packet_type()` (`os_ken.lib.packet.vlan.svlan` class method), 122
- `get_packet_type()` (`os_ken.lib.packet.vlan.vlan` class method), 122
- `get_port()` (`os_ken.controller.dpset.DPSet` method), 570
- `get_port_name_list()` (`os_ken.lib.ovs.bridge.OVSBridge` method), 159
- `get_ports()` (`os_ken.controller.dpset.DPSet` method), 571
- `get_protocol()` (`os_ken.lib.packet.packet.Packet` method), 21
- `get_protocols()` (`os_ken.lib.packet.packet.Packet` method), 21
- `get_vif_ports()` (`os_ken.lib.ovs.bridge.OVSBridge` method), 159
- `gre` (class in `os_ken.lib.packet.gre`), 73
- ## H
- `has_flags()` (`os_ken.lib.packet.tcp.tcp` method), 120
- `header` (class in `os_ken.lib.packet.ipv6`), 88
- `HoldTimerExpired`, 46
- `hop_opts` (class in `os_ken.lib.packet.ipv6`), 88
- ## I
- `icmp` (class in `os_ken.lib.packet.icmp`), 75
- `icmpv6` (class in `os_ken.lib.packet.icmpv6`), 76
- `igmp` (class in `os_ken.lib.packet.igmp`), 83
- `igmpv3_query` (class in `os_ken.lib.packet.igmp`), 84
- `igmpv3_report` (class in `os_ken.lib.packet.igmp`), 85
- `igmpv3_report_group` (class in `os_ken.lib.packet.igmp`), 85
- `in_filter_get()` (`os_ken.services.protocols.bgp.bgpspeaker.BGPspeaker` method), 143
- `in_filter_set()` (`os_ken.services.protocols.bgp.bgpspeaker.BGPspeaker` method), 143
- `init()` (`os_ken.lib.ovs.bridge.OVSBridge` method), 159
- `interface_status_tlv` (class in `os_ken.lib.packet.cfm`), 62
- `InterfaceLinkParams` (class in `os_ken.lib.packet.zebra`), 127
- `InvalidChecksum`, 99
- `InvalidDatapath`, 584
- `InvalidNetworkField`, 46
- `InvalidNextHop`, 47
- `InvalidOriginError`, 47
- `ipv4` (class in `os_ken.lib.packet.ipv4`), 86
- `ipv6` (class in `os_ken.lib.packet.ipv6`), 89
- `itag` (class in `os_ken.lib.packet.pbb`), 101
- ## K
- `KeyedMD5` (class in `os_ken.lib.packet.bfd`), 26
- `KeyedSHA1` (class in `os_ken.lib.packet.bfd`), 26
- ## L
- `label_from_bin()` (in `os_ken.lib.packet.mpls` module), 97
- `label_to_bin()` (in `os_ken.lib.packet.mpls` module), 97
- `lacp` (class in `os_ken.lib.packet.slow`), 115
- `link_trace_message` (class in `os_ken.lib.packet.cfm`), 63
- `link_trace_reply` (class in `os_ken.lib.packet.cfm`), 63
- `list_db_attributes()` (`os_ken.lib.ovs.bridge.OVSBridge` method), 159
- `llc` (class in `os_ken.lib.packet.llc`), 93
- `lldp` (class in `os_ken.lib.packet.lldp`), 96
- `loopback_message` (class in `os_ken.lib.packet.cfm`), 63
- `loopback_reply` (class in `os_ken.lib.packet.cfm`), 64
- `ltm_egress_identifier_tlv` (class in `os_ken.lib.packet.cfm`), 64
- `ltr_egress_identifier_tlv` (class in `os_ken.lib.packet.cfm`), 64
- ## M
- `MalformedAsPath`, 47
- `MalformedAttrList`, 47
- `MalformedOptionalParam`, 47
- `ManagementAddress` (class in `os_ken.lib.packet.lldp`), 94

- MaxPrefixReached, 48  
 MeticulousKeyedMD5 (class in *os\_ken.lib.packet.bfd*), 27  
 MeticulousKeyedSHA1 (class in *os\_ken.lib.packet.bfd*), 27  
 MissingWellKnown, 48  
 mld (class in *os\_ken.lib.packet.icmpv6*), 77  
 mldv2\_query (class in *os\_ken.lib.packet.icmpv6*), 78  
 mldv2\_report (class in *os\_ken.lib.packet.icmpv6*), 78  
 mldv2\_report\_group (class in *os\_ken.lib.packet.icmpv6*), 78  
 module  
   os\_ken.app.ofctl.api, 583  
   os\_ken.app.ofctl.exception, 584  
   os\_ken.base.app\_manager, 10  
   os\_ken.controller.controller, 10  
   os\_ken.controller.dpset, 10  
   os\_ken.controller.ofp\_event, 10  
   os\_ken.controller.ofp\_handler, 10  
   os\_ken.lib.netconf, 12  
   os\_ken.lib.of\_config, 12  
   os\_ken.lib.ovs, 12  
   os\_ken.lib.ovs.bridge, 156  
   os\_ken.lib.ovs.vsctl, 155  
   os\_ken.lib.packet, 12  
   os\_ken.lib.packet.arp, 23  
   os\_ken.lib.packet.bfd, 24  
   os\_ken.lib.packet.bgp, 30  
   os\_ken.lib.packet.bmp, 50  
   os\_ken.lib.packet.bpdu, 56  
   os\_ken.lib.packet.cfm, 61  
   os\_ken.lib.packet.dhcp, 67  
   os\_ken.lib.packet.dhcp6, 69  
   os\_ken.lib.packet.ethernet, 71  
   os\_ken.lib.packet.geneve, 72  
   os\_ken.lib.packet.gre, 73  
   os\_ken.lib.packet.icmp, 75  
   os\_ken.lib.packet.icmpv6, 76  
   os\_ken.lib.packet.igmp, 81  
   os\_ken.lib.packet.ipv4, 86  
   os\_ken.lib.packet.ipv6, 87  
   os\_ken.lib.packet.llc, 91  
   os\_ken.lib.packet.lldp, 94  
   os\_ken.lib.packet.mpls, 97  
   os\_ken.lib.packet.openflow, 98  
   os\_ken.lib.packet.ospf, 99  
   os\_ken.lib.packet.packet, 21  
   os\_ken.lib.packet.packet\_base, 22  
   os\_ken.lib.packet.pbb, 101  
   os\_ken.lib.packet.sctp, 102  
   os\_ken.lib.packet.slow, 115  
   os\_ken.lib.packet.stream\_parser, 22  
   os\_ken.lib.packet.tcp, 120  
   os\_ken.lib.packet.udp, 121  
   os\_ken.lib.packet.vlan, 121  
   os\_ken.lib.packet.vrrp, 122  
   os\_ken.lib.packet.vxlan, 126  
   os\_ken.lib.packet.zebra, 127  
   os\_ken.lib.xflow, 12  
   os\_ken.ofproto.nicira\_ext, 567  
   os\_ken.ofproto.ofproto\_v1\_0, 10  
   os\_ken.ofproto.ofproto\_v1\_0\_parser, 11  
   os\_ken.ofproto.ofproto\_v1\_2, 11  
   os\_ken.ofproto.ofproto\_v1\_2\_parser, 11  
   os\_ken.ofproto.ofproto\_v1\_3, 11  
   os\_ken.ofproto.ofproto\_v1\_3\_parser, 11  
   os\_ken.ofproto.ofproto\_v1\_4, 11  
   os\_ken.ofproto.ofproto\_v1\_4\_parser, 11  
   os\_ken.ofproto.ofproto\_v1\_5, 12  
   os\_ken.ofproto.ofproto\_v1\_5\_parser, 12  
   os\_ken.topology, 12  
   mpls (class in *os\_ken.lib.packet.mpls*), 97  
   MsgBase (class in *os\_ken.ofproto.ofproto\_parser*), 160
- ## N
- nd\_neighbor (class in *os\_ken.lib.packet.icmpv6*), 79  
 nd\_option\_mtu (class in *os\_ken.lib.packet.icmpv6*), 79  
 nd\_option\_pi (class in *os\_ken.lib.packet.icmpv6*), 79  
 nd\_option\_sla (class in *os\_ken.lib.packet.icmpv6*), 80  
 nd\_option\_tla (class in *os\_ken.lib.packet.icmpv6*), 80  
 nd\_router\_advert (class in *os\_ken.lib.packet.icmpv6*), 80  
 nd\_router\_solicit (class in *os\_ken.lib.packet.icmpv6*), 81  
 neighbor\_add()  
   (*os\_ken.services.protocols.bgp.bgpspeaker.BGPSpeaker* method), 144  
 neighbor\_del()  
   (*os\_ken.services.protocols.bgp.bgpspeaker.BGPSpeaker*

<i>method</i> ), 145			NXActionCT	(class	in
neighbor_get()	( <i>os_ken.services.protocols.bgp.bgpspeaker.BGPSpeaker</i> ),	145		<i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	
<i>method</i> ), 145			NXActionDecMplsTtl	(class	in
neighbor_reset()	( <i>os_ken.services.protocols.bgp.bgpspeaker.BGPSpeaker</i> ),	145		<i>os_ken.ofproto.ofproto_v1_0_parser</i> ),	
<i>method</i> ), 145			NXActionDecNshTtl	(class	in
neighbor_state_get()	( <i>os_ken.services.protocols.bgp.bgpspeaker.BGPSpeaker</i> ),	145		<i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	
<i>method</i> ), 145			NXActionDecTtl	(class	in
neighbor_update()	( <i>os_ken.services.protocols.bgp.bgpspeaker.BGPSpeaker</i> ),	145		<i>os_ken.ofproto.ofproto_v1_0_parser</i> ),	
<i>method</i> ), 145			NXActionDecTtlCntIds	(class	in
neighbors_get()	( <i>os_ken.services.protocols.bgp.bgpspeaker.BGPSpeaker</i> ),	146		<i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	
<i>method</i> ), 146			NXActionExit	(class	in
NextHopBlackhole	(class	in	<i>os_ken.lib.packet.zebra</i> ),	127	
	<i>os_ken.lib.packet.zebra</i> ),	127		555	
NextHopIFIndex	(class	in	NXActionFinTimeout	(class	in
	<i>os_ken.lib.packet.zebra</i> ),	127		<i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	
NextHopIFName	(class	in		559	
	<i>os_ken.lib.packet.zebra</i> ),	127		NXActionLearn	(class
NextHopIPv4	(class in <i>os_ken.lib.packet.zebra</i> ),	127		<i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	in
				553	
NextHopIPv4IFIndex	(class	in	NXActionMultipath	(class	in
	<i>os_ken.lib.packet.zebra</i> ),	127		<i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	
NextHopIPv4IFName	(class	in		560	
	<i>os_ken.lib.packet.zebra</i> ),	127		NXActionNAT	(class
NextHopIPv6	(class in <i>os_ken.lib.packet.zebra</i> ),	127		<i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	in
				563	
NextHopIPv6IFIndex	(class	in	NXActionNote	(class	in
	<i>os_ken.lib.packet.zebra</i> ),	127		<i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	
NextHopIPv6IFName	(class	in		548	
	<i>os_ken.lib.packet.zebra</i> ),	127		NXActionOutputReg	(class
NotSync	48			<i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	in
nvgre()	(in module <i>os_ken.lib.packet.gre</i> ),	74		551	
NXActionBundle	(class	in	NXActionOutputReg2	(class	in
	<i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	561		<i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	
NXActionBundleLoad	(class	in	NXActionOutputTrunc	(class	in
	<i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	562		<i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	
NXActionConjunction	(class	in	NXActionPopMpls	(class	in
	<i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	560		<i>os_ken.ofproto.ofproto_v1_0_parser</i> ),	
NXActionController	(class	in		545	
	<i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	555		NXActionPopQueue	(class
				<i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	in
				547	
NXActionController2	(class	in	NXActionPushMpls	(class	in
	<i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	556		<i>os_ken.ofproto.ofproto_v1_0_parser</i> ),	
				544	

NXActionRegLoad	(class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	547	NXFlowSpecOutput	(class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	566
NXActionRegLoad2	(class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	548	<b>O</b>		
NXActionRegMove	(class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	550	OFError,	584	
NXActionResubmit	(class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	551	ofp_msg_from_jsondict()	(in module <i>os_ken.ofproto.ofproto_parser</i> ),	161
NXActionResubmitTable	(class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	551	OFP_VERSIONS	( <i>os_ken.base.app_manager.OSKenApp</i> attribute),	569
NXActionSample	(class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	558	OFPAction	(class in <i>os_ken.ofproto.ofproto_v1_0_parser</i> ),	186
NXActionSample2	(class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	558	OFPActionCopyField	(class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),	541
NXActionSetMplsLabel	(class in <i>os_ken.ofproto.ofproto_v1_0_parser</i> ),	546	OFPActionCopyTtlIn	(class in <i>os_ken.ofproto.ofproto_v1_2_parser</i> ),	242
NXActionSetMplsTc	(class in <i>os_ken.ofproto.ofproto_v1_0_parser</i> ),	547	OFPActionCopyTtlIn	(class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	316
NXActionSetMplsTtl	(class in <i>os_ken.ofproto.ofproto_v1_0_parser</i> ),	545	OFPActionCopyTtlIn	(class in <i>os_ken.ofproto.ofproto_v1_4_parser</i> ),	428
NXActionSetQueue	(class in <i>os_ken.ofproto.ofproto_v1_0_parser</i> ),	543	OFPActionCopyTtlIn	(class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),	539
NXActionSetTunnel	(class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	549	OFPActionCopyTtlOut	(class in <i>os_ken.ofproto.ofproto_v1_2_parser</i> ),	242
NXActionSetTunnel64	(class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	549	OFPActionCopyTtlOut	(class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	315
NXActionStackPop	(class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	557	OFPActionCopyTtlOut	(class in <i>os_ken.ofproto.ofproto_v1_4_parser</i> ),	428
NXActionStackPush	(class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	557	OFPActionCopyTtlOut	(class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),	539
NXFlowSpecLoad	(class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	566	OFPActionDecMplsTtl	(class in <i>os_ken.ofproto.ofproto_v1_2_parser</i> ),	241
NXFlowSpecMatch	(class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	566	OFPActionDecMplsTtl	(class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	315
			OFPActionDecMplsTtl	(class in <i>os_ken.ofproto.ofproto_v1_4_parser</i> ),	428
			OFPActionDecMplsTtl	(class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),	

539			187		
OFPACTIONDecNwTtl	(class	in	OFPACTIONOutput	(class	in
	<i>os_ken.ofproto.ofproto_v1_2_parser</i> ),			<i>os_ken.ofproto.ofproto_v1_0_parser</i> ),	
242			186		
OFPACTIONDecNwTtl	(class	in	OFPACTIONOutput	(class	in
	<i>os_ken.ofproto.ofproto_v1_3_parser</i> ),			<i>os_ken.ofproto.ofproto_v1_2_parser</i> ),	
315			241		
OFPACTIONDecNwTtl	(class	in	OFPACTIONOutput	(class	in
	<i>os_ken.ofproto.ofproto_v1_4_parser</i> ),			<i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	
430			314		
OFPACTIONDecNwTtl	(class	in	OFPACTIONOutput	(class	in
	<i>os_ken.ofproto.ofproto_v1_5_parser</i> ),			<i>os_ken.ofproto.ofproto_v1_4_parser</i> ),	
540			428		
OFPACTIONDlAddr	(class	in	OFPACTIONOutput	(class	in
	<i>os_ken.ofproto.ofproto_v1_0_parser</i> ),			<i>os_ken.ofproto.ofproto_v1_5_parser</i> ),	
187			539		
OFPACTIONEnqueue	(class	in	OFPACTIONPopMpls	(class	in
	<i>os_ken.ofproto.ofproto_v1_0_parser</i> ),			<i>os_ken.ofproto.ofproto_v1_2_parser</i> ),	
188			243		
OFPACTIONExperimenter	(class	in	OFPACTIONPopMpls	(class	in
	<i>os_ken.ofproto.ofproto_v1_2_parser</i> ),			<i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	
243			316		
OFPACTIONExperimenter	(class	in	OFPACTIONPopMpls	(class	in
	<i>os_ken.ofproto.ofproto_v1_3_parser</i> ),			<i>os_ken.ofproto.ofproto_v1_4_parser</i> ),	
317			429		
OFPACTIONExperimenter	(class	in	OFPACTIONPopMpls	(class	in
	<i>os_ken.ofproto.ofproto_v1_4_parser</i> ),			<i>os_ken.ofproto.ofproto_v1_5_parser</i> ),	
430			540		
OFPACTIONExperimenter	(class	in	OFPACTIONPopPbb	(class	in
	<i>os_ken.ofproto.ofproto_v1_5_parser</i> ),			<i>os_ken.ofproto.ofproto_v1_4_parser</i> ),	
542			430		
OFPACTIONGroup	(class	in	OFPACTIONPopPbb	(class	in
	<i>os_ken.ofproto.ofproto_v1_2_parser</i> ),			<i>os_ken.ofproto.ofproto_v1_5_parser</i> ),	
241			541		
OFPACTIONGroup	(class	in	OFPACTIONPopVlan	(class	in
	<i>os_ken.ofproto.ofproto_v1_3_parser</i> ),			<i>os_ken.ofproto.ofproto_v1_2_parser</i> ),	
314			242		
OFPACTIONGroup	(class	in	OFPACTIONPopVlan	(class	in
	<i>os_ken.ofproto.ofproto_v1_4_parser</i> ),			<i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	
429			316		
OFPACTIONGroup	(class	in	OFPACTIONPopVlan	(class	in
	<i>os_ken.ofproto.ofproto_v1_5_parser</i> ),			<i>os_ken.ofproto.ofproto_v1_4_parser</i> ),	
540			429		
OFPACTIONHeader	(class	in	OFPACTIONPopVlan	(class	in
	<i>os_ken.ofproto.ofproto_v1_0_parser</i> ),			<i>os_ken.ofproto.ofproto_v1_5_parser</i> ),	
186			539		
OFPACTIONMeter	(class	in	OFPACTIONPushMpls	(class	in
	<i>os_ken.ofproto.ofproto_v1_5_parser</i> ),			<i>os_ken.ofproto.ofproto_v1_2_parser</i> ),	
541			242		
OFPACTIONNwAddr	(class	in	OFPACTIONPushMpls	(class	in
	<i>os_ken.ofproto.ofproto_v1_0_parser</i> ),			<i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	

316			428		
OFPACTIONPushMpls	(class	in	OFPACTIONSetMplsTtl	(class	in
	<i>os_ken.ofproto.ofproto_v1_4_parser</i> ),			<i>os_ken.ofproto.ofproto_v1_5_parser</i> ),	
429			539		
OFPACTIONPushMpls	(class	in	OFPACTIONSetNwDst	(class	in
	<i>os_ken.ofproto.ofproto_v1_5_parser</i> ),			<i>os_ken.ofproto.ofproto_v1_0_parser</i> ),	
540			188		
OFPACTIONPushPbb	(class	in	OFPACTIONSetNwSrc	(class	in
	<i>os_ken.ofproto.ofproto_v1_4_parser</i> ),			<i>os_ken.ofproto.ofproto_v1_0_parser</i> ),	
430			187		
OFPACTIONPushPbb	(class	in	OFPACTIONSetNwTos	(class	in
	<i>os_ken.ofproto.ofproto_v1_5_parser</i> ),			<i>os_ken.ofproto.ofproto_v1_0_parser</i> ),	
541			188		
OFPACTIONPushVlan	(class	in	OFPACTIONSetNwTtl	(class	in
	<i>os_ken.ofproto.ofproto_v1_2_parser</i> ),			<i>os_ken.ofproto.ofproto_v1_2_parser</i> ),	
242			241		
OFPACTIONPushVlan	(class	in	OFPACTIONSetNwTtl	(class	in
	<i>os_ken.ofproto.ofproto_v1_3_parser</i> ),			<i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	
316			315		
OFPACTIONPushVlan	(class	in	OFPACTIONSetNwTtl	(class	in
	<i>os_ken.ofproto.ofproto_v1_4_parser</i> ),			<i>os_ken.ofproto.ofproto_v1_4_parser</i> ),	
428			430		
OFPACTIONPushVlan	(class	in	OFPACTIONSetNwTtl	(class	in
	<i>os_ken.ofproto.ofproto_v1_5_parser</i> ),			<i>os_ken.ofproto.ofproto_v1_5_parser</i> ),	
539			540		
OFPACTIONSetDlDst	(class	in	OFPACTIONSetQueue	(class	in
	<i>os_ken.ofproto.ofproto_v1_0_parser</i> ),			<i>os_ken.ofproto.ofproto_v1_2_parser</i> ),	
187			241		
OFPACTIONSetDlSrc	(class	in	OFPACTIONSetQueue	(class	in
	<i>os_ken.ofproto.ofproto_v1_0_parser</i> ),			<i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	
187			315		
OFPACTIONSetField	(class	in	OFPACTIONSetQueue	(class	in
	<i>os_ken.ofproto.ofproto_v1_2_parser</i> ),			<i>os_ken.ofproto.ofproto_v1_4_parser</i> ),	
243			429		
OFPACTIONSetField	(class	in	OFPACTIONSetQueue	(class	in
	<i>os_ken.ofproto.ofproto_v1_3_parser</i> ),			<i>os_ken.ofproto.ofproto_v1_5_parser</i> ),	
316			540		
OFPACTIONSetField	(class	in	OFPACTIONSetTpDst	(class	in
	<i>os_ken.ofproto.ofproto_v1_4_parser</i> ),			<i>os_ken.ofproto.ofproto_v1_0_parser</i> ),	
430			188		
OFPACTIONSetField	(class	in	OFPACTIONSetTpSrc	(class	in
	<i>os_ken.ofproto.ofproto_v1_5_parser</i> ),			<i>os_ken.ofproto.ofproto_v1_0_parser</i> ),	
541			188		
OFPACTIONSetMplsTtl	(class	in	OFPACTIONStripVlan	(class	in
	<i>os_ken.ofproto.ofproto_v1_2_parser</i> ),			<i>os_ken.ofproto.ofproto_v1_0_parser</i> ),	
241			187		
OFPACTIONSetMplsTtl	(class	in	OFPACTIONTpPort	(class	in
	<i>os_ken.ofproto.ofproto_v1_3_parser</i> ),			<i>os_ken.ofproto.ofproto_v1_0_parser</i> ),	
315			188		
OFPACTIONSetMplsTtl	(class	in	OFPACTIONVendor	(class	in
	<i>os_ken.ofproto.ofproto_v1_4_parser</i> ),			<i>os_ken.ofproto.ofproto_v1_0_parser</i> ),	



188			500
OFPACTIONVLANPCP	(class in os_ken.ofproto.ofproto_v1_0_parser), 187	in	OFPBarrierRequest (class in os_ken.ofproto.ofproto_v1_0_parser), 179
OFPACTIONVLANVID	(class in os_ken.ofproto.ofproto_v1_0_parser), 186	in	OFPBarrierRequest (class in os_ken.ofproto.ofproto_v1_2_parser), 224
OFPAggregateStatsReply	(class in os_ken.ofproto.ofproto_v1_0_parser), 173	in	OFPBarrierRequest (class in os_ken.ofproto.ofproto_v1_3_parser), 293
OFPAggregateStatsReply	(class in os_ken.ofproto.ofproto_v1_2_parser), 209	in	OFPBarrierRequest (class in os_ken.ofproto.ofproto_v1_4_parser), 387
OFPAggregateStatsReply	(class in os_ken.ofproto.ofproto_v1_3_parser), 271	in	OFPBarrierRequest (class in os_ken.ofproto.ofproto_v1_5_parser), 499
OFPAggregateStatsReply	(class in os_ken.ofproto.ofproto_v1_4_parser), 353	in	OFPBUNDLEADDMSG (class in os_ken.ofproto.ofproto_v1_4_parser), 392
OFPAggregateStatsReply	(class in os_ken.ofproto.ofproto_v1_5_parser), 459	in	OFPBUNDLEADDMSG (class in os_ken.ofproto.ofproto_v1_5_parser), 504
OFPAggregateStatsRequest	(class in os_ken.ofproto.ofproto_v1_0_parser), 173	in	OFPBUNDLECTRLMSG (class in os_ken.ofproto.ofproto_v1_4_parser), 390
OFPAggregateStatsRequest	(class in os_ken.ofproto.ofproto_v1_2_parser), 208	in	OFPBUNDLECTRLMSG (class in os_ken.ofproto.ofproto_v1_5_parser), 503
OFPAggregateStatsRequest	(class in os_ken.ofproto.ofproto_v1_3_parser), 270	in	OFPBUNDLEFEATURESSTATSREPLY (class in os_ken.ofproto.ofproto_v1_5_parser), 496
OFPAggregateStatsRequest	(class in os_ken.ofproto.ofproto_v1_4_parser), 352	in	OFPBUNDLEFEATURESSTATSREQUEST (class in os_ken.ofproto.ofproto_v1_5_parser), 494
OFPAggregateStatsRequest	(class in os_ken.ofproto.ofproto_v1_5_parser), 458	in	OFPCONTROLLERSTATUS (class in os_ken.ofproto.ofproto_v1_5_parser), 527
OFPBarrierReply	(class in os_ken.ofproto.ofproto_v1_0_parser), 179	in	OFPCONTROLLERSTATUSSTATS (class in os_ken.ofproto.ofproto_v1_5_parser), 542
OFPBarrierReply	(class in os_ken.ofproto.ofproto_v1_2_parser), 225	in	OFPCONTROLLERSTATUSSTATSREPLY (class in os_ken.ofproto.ofproto_v1_5_parser), 483
OFPBarrierReply	(class in os_ken.ofproto.ofproto_v1_3_parser), 294	in	OFPCONTROLLERSTATUSSTATSREQUEST (class in os_ken.ofproto.ofproto_v1_5_parser), 482
OFPBarrierReply	(class in os_ken.ofproto.ofproto_v1_4_parser), 388	in	OFPDDESCSTATS (class in os_ken.ofproto.ofproto_v1_2_parser), 201
OFPBarrierReply	(class in os_ken.ofproto.ofproto_v1_5_parser),	in	OFPDDESCSTATSREPLY (class in os_ken.ofproto.ofproto_v1_0_parser),

171			420		
OFPDescStatsReply	(class	in	OFPEchoRequest	(class	in
	<i>os_ken.ofproto.ofproto_v1_3_parser</i> ),			<i>os_ken.ofproto.ofproto_v1_5_parser</i> ),	
260			530		
OFPDescStatsReply	(class	in	OFPErrorMsg	(class	in
	<i>os_ken.ofproto.ofproto_v1_4_parser</i> ),			<i>os_ken.ofproto.ofproto_v1_0_parser</i> ),	
342			182		
OFPDescStatsReply	(class	in	OFPErrorMsg	(class	in
	<i>os_ken.ofproto.ofproto_v1_5_parser</i> ),			<i>os_ken.ofproto.ofproto_v1_2_parser</i> ),	
451			233		
OFPDescStatsRequest	(class	in	OFPErrorMsg	(class	in
	<i>os_ken.ofproto.ofproto_v1_0_parser</i> ),			<i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	
171			306		
OFPDescStatsRequest	(class	in	OFPErrorMsg	(class	in
	<i>os_ken.ofproto.ofproto_v1_2_parser</i> ),			<i>os_ken.ofproto.ofproto_v1_4_parser</i> ),	
201			421		
OFPDescStatsRequest	(class	in	OFPErrorMsg	(class	in
	<i>os_ken.ofproto.ofproto_v1_3_parser</i> ),			<i>os_ken.ofproto.ofproto_v1_5_parser</i> ),	
260			531		
OFPDescStatsRequest	(class	in	OFPExperimenter	(class	in
	<i>os_ken.ofproto.ofproto_v1_4_parser</i> ),			<i>os_ken.ofproto.ofproto_v1_2_parser</i> ),	
342			236		
OFPDescStatsRequest	(class	in	OFPExperimenter	(class	in
	<i>os_ken.ofproto.ofproto_v1_5_parser</i> ),			<i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	
451			309		
OFPEchoReply	(class	in	OFPExperimenter	(class	in
	<i>os_ken.ofproto.ofproto_v1_0_parser</i> ),			<i>os_ken.ofproto.ofproto_v1_4_parser</i> ),	
184			423		
OFPEchoReply	(class	in	OFPExperimenter	(class	in
	<i>os_ken.ofproto.ofproto_v1_2_parser</i> ),			<i>os_ken.ofproto.ofproto_v1_5_parser</i> ),	
235			533		
OFPEchoReply	(class	in	OFPExperimenterStatsReply	(class	in
	<i>os_ken.ofproto.ofproto_v1_3_parser</i> ),			<i>os_ken.ofproto.ofproto_v1_4_parser</i> ),	
309			386		
OFPEchoReply	(class	in	OFPExperimenterStatsReply	(class	in
	<i>os_ken.ofproto.ofproto_v1_4_parser</i> ),			<i>os_ken.ofproto.ofproto_v1_5_parser</i> ),	
420			498		
OFPEchoReply	(class	in	OFPExperimenterStatsRequest	(class	in
	<i>os_ken.ofproto.ofproto_v1_5_parser</i> ),			<i>os_ken.ofproto.ofproto_v1_4_parser</i> ),	
530			385		
OFPEchoRequest	(class	in	OFPExperimenterStatsRequest	(class	in
	<i>os_ken.ofproto.ofproto_v1_0_parser</i> ),			<i>os_ken.ofproto.ofproto_v1_5_parser</i> ),	
183			497		
OFPEchoRequest	(class	in	OFPFFeaturesRequest	(class	in
	<i>os_ken.ofproto.ofproto_v1_2_parser</i> ),			<i>os_ken.ofproto.ofproto_v1_0_parser</i> ),	
235			162		
OFPEchoRequest	(class	in	OFPFFeaturesRequest	(class	in
	<i>os_ken.ofproto.ofproto_v1_3_parser</i> ),			<i>os_ken.ofproto.ofproto_v1_2_parser</i> ),	
308			189		
OFPEchoRequest	(class	in	OFPFFeaturesRequest	(class	in
	<i>os_ken.ofproto.ofproto_v1_4_parser</i> ),			<i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	



243			409		
OFPFFeaturesRequest	(class in <i>os_ken.ofproto.ofproto_v1_4_parser</i> ),	317	OFPFFlowRemoved	(class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),	517
OFPFFeaturesRequest	(class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),	431	OFPFFlowStats	(class in <i>os_ken.ofproto.ofproto_v1_2_parser</i> ),	203
OFPFFlowDescStatsReply	(class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),	453	OFPFFlowStatsReply	(class in <i>os_ken.ofproto.ofproto_v1_0_parser</i> ),	172
OFPFFlowDescStatsRequest	(class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),	452	OFPFFlowStatsReply	(class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	262
OFPFFlowMod	(class in <i>os_ken.ofproto.ofproto_v1_0_parser</i> ),	166	OFPFFlowStatsReply	(class in <i>os_ken.ofproto.ofproto_v1_4_parser</i> ),	344
OFPFFlowMod	(class in <i>os_ken.ofproto.ofproto_v1_2_parser</i> ),	193	OFPFFlowStatsReply	(class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),	456
OFPFFlowMod	(class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	247	OFPFFlowStatsRequest	(class in <i>os_ken.ofproto.ofproto_v1_0_parser</i> ),	171
OFPFFlowMod	(class in <i>os_ken.ofproto.ofproto_v1_4_parser</i> ),	322	OFPFFlowStatsRequest	(class in <i>os_ken.ofproto.ofproto_v1_2_parser</i> ),	202
OFPFFlowMod	(class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),	435	OFPFFlowStatsRequest	(class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	261
OFPFFlowMonitorReply	(class in <i>os_ken.ofproto.ofproto_v1_4_parser</i> ),	383	OFPFFlowStatsRequest	(class in <i>os_ken.ofproto.ofproto_v1_4_parser</i> ),	343
OFPFFlowMonitorReply	(class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),	492	OFPFFlowStatsRequest	(class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),	455
OFPFFlowMonitorRequest	(class in <i>os_ken.ofproto.ofproto_v1_4_parser</i> ),	381	OFPFGetAsyncReply	(class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	298
OFPFFlowMonitorRequest	(class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),	490	OFPFGetAsyncReply	(class in <i>os_ken.ofproto.ofproto_v1_4_parser</i> ),	396
OFPFFlowRemoved	(class in <i>os_ken.ofproto.ofproto_v1_0_parser</i> ),	181	OFPFGetAsyncReply	(class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),	512
OFPFFlowRemoved	(class in <i>os_ken.ofproto.ofproto_v1_2_parser</i> ),	230	OFPFGetAsyncRequest	(class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	298
OFPFFlowRemoved	(class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	303	OFPFGetAsyncRequest	(class in <i>os_ken.ofproto.ofproto_v1_4_parser</i> ),	396
OFPFFlowRemoved	(class in <i>os_ken.ofproto.ofproto_v1_4_parser</i> ),		OFPFGetAsyncRequest	(class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),	

511			373
OFPGetConfigReply	(class in <i>os_ken.ofproto.ofproto_v1_0_parser</i> ),	165	OFPGroupDescStatsRequest (class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),
OFPGetConfigReply	(class in <i>os_ken.ofproto.ofproto_v1_2_parser</i> ),	191	OFPGroupFeaturesStats (class in <i>os_ken.ofproto.ofproto_v1_2_parser</i> ),
OFPGetConfigReply	(class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	246	OFPGroupFeaturesStatsReply (class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),
OFPGetConfigReply	(class in <i>os_ken.ofproto.ofproto_v1_4_parser</i> ),	319	OFPGroupFeaturesStatsReply (class in <i>os_ken.ofproto.ofproto_v1_4_parser</i> ),
OFPGetConfigReply	(class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),	433	OFPGroupFeaturesStatsReply (class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),
OFPGetConfigRequest	(class in <i>os_ken.ofproto.ofproto_v1_0_parser</i> ),	165	OFPGroupFeaturesStatsRequest (class in <i>os_ken.ofproto.ofproto_v1_2_parser</i> ),
OFPGetConfigRequest	(class in <i>os_ken.ofproto.ofproto_v1_2_parser</i> ),	191	OFPGroupFeaturesStatsRequest (class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),
OFPGetConfigRequest	(class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	245	OFPGroupFeaturesStatsRequest (class in <i>os_ken.ofproto.ofproto_v1_4_parser</i> ),
OFPGetConfigRequest	(class in <i>os_ken.ofproto.ofproto_v1_4_parser</i> ),	319	OFPGroupFeaturesStatsRequest (class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),
OFPGetConfigRequest	(class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),	433	OFPGroupMod (class in <i>os_ken.ofproto.ofproto_v1_2_parser</i> ),
OFPGroupDescStats	(class in <i>os_ken.ofproto.ofproto_v1_2_parser</i> ),	218	OFPGroupMod (class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),
OFPGroupDescStatsReply	(class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	281	OFPGroupMod (class in <i>os_ken.ofproto.ofproto_v1_4_parser</i> ),
OFPGroupDescStatsReply	(class in <i>os_ken.ofproto.ofproto_v1_4_parser</i> ),	373	OFPGroupMod (class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),
OFPGroupDescStatsReply	(class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),	474	OFPGroupStats (class in <i>os_ken.ofproto.ofproto_v1_2_parser</i> ),
OFPGroupDescStatsRequest	(class in <i>os_ken.ofproto.ofproto_v1_2_parser</i> ),	218	OFPGroupStatsReply (class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),
OFPGroupDescStatsRequest	(class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	280	OFPGroupStatsReply (class in <i>os_ken.ofproto.ofproto_v1_4_parser</i> ),
OFPGroupDescStatsRequest	(class in <i>os_ken.ofproto.ofproto_v1_4_parser</i> ),		OFPGroupStatsReply (class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),

473			240
OFPGroupStatsRequest	(class in <i>os_ken.ofproto.ofproto_v1_2_parser</i> ),	216	OFPInstructionGotoTable (class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),
OFPGroupStatsRequest	(class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	279	OFPInstructionGotoTable (class in <i>os_ken.ofproto.ofproto_v1_4_parser</i> ),
OFPGroupStatsRequest	(class in <i>os_ken.ofproto.ofproto_v1_4_parser</i> ),	371	OFPInstructionGotoTable (class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),
OFPGroupStatsRequest	(class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),	472	OFPInstructionMeter (class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),
OFPHello	(class in <i>os_ken.ofproto.ofproto_v1_0_parser</i> ),	183	OFPInstructionMeter (class in <i>os_ken.ofproto.ofproto_v1_4_parser</i> ),
OFPHello	(class in <i>os_ken.ofproto.ofproto_v1_2_parser</i> ),	234	OFPInstructionStatTrigger (class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),
OFPHello	(class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	307	OFPInstructionWriteMetadata (class in <i>os_ken.ofproto.ofproto_v1_2_parser</i> ),
OFPHello	(class in <i>os_ken.ofproto.ofproto_v1_4_parser</i> ),	419	OFPInstructionWriteMetadata (class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),
OFPHello	(class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),	529	OFPInstructionWriteMetadata (class in <i>os_ken.ofproto.ofproto_v1_4_parser</i> ),
OFPHelloElemVersionBitmap	(class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	308	OFPInstructionWriteMetadata (class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),
OFPHelloElemVersionBitmap	(class in <i>os_ken.ofproto.ofproto_v1_4_parser</i> ),	419	OFPMatch (class in <i>os_ken.ofproto.ofproto_v1_0_parser</i> ),
OFPHelloElemVersionBitmap	(class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),	529	OFPMatch (class in <i>os_ken.ofproto.ofproto_v1_2_parser</i> ),
OFPInstructionActions	(class in <i>os_ken.ofproto.ofproto_v1_2_parser</i> ),	240	OFPMatch (class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),
OFPInstructionActions	(class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	314	OFPMatch (class in <i>os_ken.ofproto.ofproto_v1_4_parser</i> ),
OFPInstructionActions	(class in <i>os_ken.ofproto.ofproto_v1_4_parser</i> ),	427	OFPMatch (class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),
OFPInstructionActions	(class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),	538	OFPMeterConfigStatsReply (class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),
OFPInstructionGotoTable	(class in <i>os_ken.ofproto.ofproto_v1_2_parser</i> ),		OFPMeterConfigStatsReply (class in <i>os_ken.ofproto.ofproto_v1_4_parser</i> ),

379		284	
OFPMeterConfigStatsRequest	(class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	OFPMeterStatsRequest	(class in <i>os_ken.ofproto.ofproto_v1_4_parser</i> ),
285		376	
OFPMeterConfigStatsRequest	(class in <i>os_ken.ofproto.ofproto_v1_4_parser</i> ),	OFPMeterStatsRequest	(class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),
378		477	
OFPMeterDescStatsReply	(class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),	OFPPacketIn	(class in <i>os_ken.ofproto.ofproto_v1_0_parser</i> ),
480		179	
OFPMeterDescStatsRequest	(class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),	OFPPacketIn	(class in <i>os_ken.ofproto.ofproto_v1_2_parser</i> ),
479		227	
OFPMeterFeaturesStatsReply	(class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	OFPPacketIn	(class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),
288		300	
OFPMeterFeaturesStatsReply	(class in <i>os_ken.ofproto.ofproto_v1_4_parser</i> ),	OFPPacketIn	(class in <i>os_ken.ofproto.ofproto_v1_4_parser</i> ),
380		400	
OFPMeterFeaturesStatsReply	(class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),	OFPPacketIn	(class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),
481		515	
OFPMeterFeaturesStatsRequest	(class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	OFPPacketOut	(class in <i>os_ken.ofproto.ofproto_v1_0_parser</i> ),
287		178	
OFPMeterFeaturesStatsRequest	(class in <i>os_ken.ofproto.ofproto_v1_4_parser</i> ),	OFPPacketOut	(class in <i>os_ken.ofproto.ofproto_v1_2_parser</i> ),
380		223	
OFPMeterFeaturesStatsRequest	(class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),	OFPPacketOut	(class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),
481		292	
OFPMeterMod	(class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	OFPPacketOut	(class in <i>os_ken.ofproto.ofproto_v1_4_parser</i> ),
258		386	
OFPMeterMod	(class in <i>os_ken.ofproto.ofproto_v1_4_parser</i> ),	OFPPacketOut	(class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),
340		498	
OFPMeterMod	(class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),	OFPPhyPort	(class in <i>os_ken.ofproto.ofproto_v1_0_parser</i> ),
449		184	
OFPMeterStatsReply	(class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	OFPPort	(class in <i>os_ken.ofproto.ofproto_v1_2_parser</i> ),
284		236	
OFPMeterStatsReply	(class in <i>os_ken.ofproto.ofproto_v1_4_parser</i> ),	OFPPort	(class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),
377		310	
OFPMeterStatsReply	(class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),	OFPPort	(class in <i>os_ken.ofproto.ofproto_v1_4_parser</i> ),
478		423	
OFPMeterStatsRequest	(class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	OFPPort	(class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),

533			175		
OFPPortDescStatsReply	(class in	OFPPortStatsRequest	(class in		
<i>os_ken.ofproto.ofproto_v1_3_parser</i> ),		<i>os_ken.ofproto.ofproto_v1_2_parser</i> ),			
276		212			
OFPPortDescStatsReply	(class in	OFPPortStatsRequest	(class in		
<i>os_ken.ofproto.ofproto_v1_4_parser</i> ),		<i>os_ken.ofproto.ofproto_v1_3_parser</i> ),			
363		273			
OFPPortDescStatsReply	(class in	OFPPortStatsRequest	(class in		
<i>os_ken.ofproto.ofproto_v1_5_parser</i> ),		<i>os_ken.ofproto.ofproto_v1_4_parser</i> ),			
464		359			
OFPPortDescStatsRequest	(class in	OFPPortStatsRequest	(class in		
<i>os_ken.ofproto.ofproto_v1_3_parser</i> ),		<i>os_ken.ofproto.ofproto_v1_5_parser</i> ),			
275		460			
OFPPortDescStatsRequest	(class in	OFPPortStatus	(class in		
<i>os_ken.ofproto.ofproto_v1_4_parser</i> ),		<i>os_ken.ofproto.ofproto_v1_0_parser</i> ),			
362		182			
OFPPortDescStatsRequest	(class in	OFPPortStatus	(class in		
<i>os_ken.ofproto.ofproto_v1_5_parser</i> ),		<i>os_ken.ofproto.ofproto_v1_2_parser</i> ),			
463		232			
OFPPortMod	(class in	OFPPortStatus	(class in		
<i>os_ken.ofproto.ofproto_v1_0_parser</i> ),		<i>os_ken.ofproto.ofproto_v1_3_parser</i> ),			
168		305			
OFPPortMod	(class in	OFPPortStatus	(class in		
<i>os_ken.ofproto.ofproto_v1_2_parser</i> ),		<i>os_ken.ofproto.ofproto_v1_4_parser</i> ),			
199		411			
OFPPortMod	(class in	OFPPortStatus	(class in		
<i>os_ken.ofproto.ofproto_v1_3_parser</i> ),		<i>os_ken.ofproto.ofproto_v1_5_parser</i> ),			
256		519			
OFPPortMod	(class in	OFPQueueDescStatsReply	(class in		
<i>os_ken.ofproto.ofproto_v1_4_parser</i> ),		<i>os_ken.ofproto.ofproto_v1_4_parser</i> ),			
338		369			
OFPPortMod	(class in	OFPQueueDescStatsReply	(class in		
<i>os_ken.ofproto.ofproto_v1_5_parser</i> ),		<i>os_ken.ofproto.ofproto_v1_5_parser</i> ),			
447		470			
OFPPortStats	(class in	OFPQueueDescStatsRequest	(class in		
<i>os_ken.ofproto.ofproto_v1_2_parser</i> ),		<i>os_ken.ofproto.ofproto_v1_4_parser</i> ),			
212		368			
OFPPortStatsReply	(class in	OFPQueueDescStatsRequest	(class in		
<i>os_ken.ofproto.ofproto_v1_0_parser</i> ),		<i>os_ken.ofproto.ofproto_v1_5_parser</i> ),			
175		469			
OFPPortStatsReply	(class in	OFPQueueGetConfigReply	(class in		
<i>os_ken.ofproto.ofproto_v1_3_parser</i> ),		<i>os_ken.ofproto.ofproto_v1_0_parser</i> ),			
274		170			
OFPPortStatsReply	(class in	OFPQueueGetConfigReply	(class in		
<i>os_ken.ofproto.ofproto_v1_4_parser</i> ),		<i>os_ken.ofproto.ofproto_v1_2_parser</i> ),			
359		222			
OFPPortStatsReply	(class in	OFPQueueGetConfigReply	(class in		
<i>os_ken.ofproto.ofproto_v1_5_parser</i> ),		<i>os_ken.ofproto.ofproto_v1_3_parser</i> ),			
460		290			
OFPPortStatsRequest	(class in	OFPQueueGetConfigRequest	(class in		
<i>os_ken.ofproto.ofproto_v1_0_parser</i> ),		<i>os_ken.ofproto.ofproto_v1_0_parser</i> ),			

170			389
OFPQueueGetConfigRequest	(class in <i>os_ken.ofproto.ofproto_v1_2_parser</i> ),	OFPRoleReply	(class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),
221		501	
OFPQueueGetConfigRequest	(class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	OFPRoleRequest	(class in <i>os_ken.ofproto.ofproto_v1_2_parser</i> ),
289		225	
OFPQueueStats	(class in <i>os_ken.ofproto.ofproto_v1_2_parser</i> ),	OFPRoleRequest	(class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),
215		294	
OFPQueueStatsReply	(class in <i>os_ken.ofproto.ofproto_v1_0_parser</i> ),	OFPRoleRequest	(class in <i>os_ken.ofproto.ofproto_v1_4_parser</i> ),
177		388	
OFPQueueStatsReply	(class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	OFPRoleRequest	(class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),
278		500	
OFPQueueStatsReply	(class in <i>os_ken.ofproto.ofproto_v1_4_parser</i> ),	OFPRoleStatus	(class in <i>os_ken.ofproto.ofproto_v1_4_parser</i> ),
366		413	
OFPQueueStatsReply	(class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),	OFPRoleStatus	(class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),
467		521	
OFPQueueStatsRequest	(class in <i>os_ken.ofproto.ofproto_v1_0_parser</i> ),	OFPSetAsync	(class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),
176		296	
OFPQueueStatsRequest	(class in <i>os_ken.ofproto.ofproto_v1_2_parser</i> ),	OFPSetAsync	(class in <i>os_ken.ofproto.ofproto_v1_4_parser</i> ),
214		393	
OFPQueueStatsRequest	(class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	OFPSetAsync	(class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),
277		508	
OFPQueueStatsRequest	(class in <i>os_ken.ofproto.ofproto_v1_4_parser</i> ),	OFPSetConfig	(class in <i>os_ken.ofproto.ofproto_v1_0_parser</i> ),
365		164	
OFPQueueStatsRequest	(class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),	OFPSetConfig	(class in <i>os_ken.ofproto.ofproto_v1_2_parser</i> ),
466		190	
OFPRequestForward	(class in <i>os_ken.ofproto.ofproto_v1_4_parser</i> ),	OFPSetConfig	(class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),
417		245	
OFPRequestForward	(class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),	OFPSetConfig	(class in <i>os_ken.ofproto.ofproto_v1_4_parser</i> ),
525		318	
OFPRoleReply	(class in <i>os_ken.ofproto.ofproto_v1_2_parser</i> ),	OFPSetConfig	(class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),
226		432	
OFPRoleReply	(class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	OFPStats	(class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),
295		537	
OFPRoleReply	(class in <i>os_ken.ofproto.ofproto_v1_4_parser</i> ),	OFPSwitchFeatures	(class in <i>os_ken.ofproto.ofproto_v1_0_parser</i> ),

162			320		
OFPSwitchFeatures	(class in <i>os_ken.ofproto.ofproto_v1_2_parser</i> ),	189	OFPTableMod	(class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),	434
OFPSwitchFeatures	(class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	244	OFPTableStats	(class in <i>os_ken.ofproto.ofproto_v1_2_parser</i> ),	210
OFPSwitchFeatures	(class in <i>os_ken.ofproto.ofproto_v1_4_parser</i> ),	317	OFPTableStatsReply	(class in <i>os_ken.ofproto.ofproto_v1_0_parser</i> ),	174
OFPSwitchFeatures	(class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),	431	OFPTableStatsReply	(class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	272
OFPTableDescStatsReply	(class in <i>os_ken.ofproto.ofproto_v1_4_parser</i> ),	356	OFPTableStatsReply	(class in <i>os_ken.ofproto.ofproto_v1_4_parser</i> ),	354
OFPTableDescStatsReply	(class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),	486	OFPTableStatsReply	(class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),	484
OFPTableDescStatsRequest	(class in <i>os_ken.ofproto.ofproto_v1_4_parser</i> ),	355	OFPTableStatsRequest	(class in <i>os_ken.ofproto.ofproto_v1_0_parser</i> ),	174
OFPTableDescStatsRequest	(class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),	485	OFPTableStatsRequest	(class in <i>os_ken.ofproto.ofproto_v1_2_parser</i> ),	210
OFPTableFeaturesStatsReply	(class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	289	OFPTableStatsRequest	(class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	272
OFPTableFeaturesStatsReply	(class in <i>os_ken.ofproto.ofproto_v1_4_parser</i> ),	358	OFPTableStatsRequest	(class in <i>os_ken.ofproto.ofproto_v1_4_parser</i> ),	354
OFPTableFeaturesStatsReply	(class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),	489	OFPTableStatsRequest	(class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),	484
OFPTableFeaturesStatsRequest	(class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	288	OFPTableStatus	(class in <i>os_ken.ofproto.ofproto_v1_4_parser</i> ),	415
OFPTableFeaturesStatsRequest	(class in <i>os_ken.ofproto.ofproto_v1_4_parser</i> ),	358	OFPTableStatus	(class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),	523
OFPTableFeaturesStatsRequest	(class in <i>os_ken.ofproto.ofproto_v1_5_parser</i> ),	488	OFPUncparsableMsg	(class in <i>os_ken.lib.packet.openflow</i> ),	98
OFPTableMod	(class in <i>os_ken.ofproto.ofproto_v1_2_parser</i> ),	193	OFPPVendor	(class in <i>os_ken.ofproto.ofproto_v1_0_parser</i> ),	184
OFPTableMod	(class in <i>os_ken.ofproto.ofproto_v1_3_parser</i> ),	247	OFPPVendorStatsReply	(class in <i>os_ken.ofproto.ofproto_v1_0_parser</i> ),	178
OFPTableMod	(class in <i>os_ken.ofproto.ofproto_v1_4_parser</i> ),		OFPPVendorStatsRequest	(class in <i>os_ken.ofproto.ofproto_v1_0_parser</i> ),	177



ofs\_nbits() (in module module, 50  
*os\_ken.ofproto.nicira\_ext*), 566  
 openflow (class in *os\_ken.lib.packet.openflow*),  
 98  
 opt\_header (class in *os\_ken.lib.packet.ipv6*), 89  
 OptAttrError, 48  
 option (class in *os\_ken.lib.packet.dhcp*), 68  
 option (class in *os\_ken.lib.packet.dhcp6*), 71  
 Option (class in *os\_ken.lib.packet.geneve*), 72  
 option (class in *os\_ken.lib.packet.ipv6*), 90  
 OptionDataUnknown (class in  
*os\_ken.lib.packet.geneve*), 72  
 options (class in *os\_ken.lib.packet.dhcp*), 69  
 options (class in *os\_ken.lib.packet.dhcp6*), 71  
 organization\_specific\_tlv (class in  
*os\_ken.lib.packet.cfm*), 65  
 OrganizationallySpecific (class in  
*os\_ken.lib.packet.lldp*), 95  
 os\_ken.app.ofctl.api  
 module, 583  
 os\_ken.app.ofctl.exception  
 module, 584  
 os\_ken.base.app\_manager  
 module, 10  
 os\_ken.controller.controller  
 module, 10  
 os\_ken.controller.dpset  
 module, 10  
 os\_ken.controller.ofp\_event  
 module, 10  
 os\_ken.controller.ofp\_handler  
 module, 10  
 os\_ken.lib.netconf  
 module, 12  
 os\_ken.lib.of\_config  
 module, 12  
 os\_ken.lib.ovs  
 module, 12  
 os\_ken.lib.ovs.bridge  
 module, 156  
 os\_ken.lib.ovs.vsctl  
 module, 155  
 os\_ken.lib.packet  
 module, 12  
 os\_ken.lib.packet.arp  
 module, 23  
 os\_ken.lib.packet.bfd  
 module, 24  
 os\_ken.lib.packet.bgp  
 module, 30  
 os\_ken.lib.packet.bmp  
 module, 50  
 os\_ken.lib.packet.bpdu  
 module, 56  
 os\_ken.lib.packet.cfm  
 module, 61  
 os\_ken.lib.packet.dhcp  
 module, 67  
 os\_ken.lib.packet.dhcp6  
 module, 69  
 os\_ken.lib.packet.ethernet  
 module, 71  
 os\_ken.lib.packet.geneve  
 module, 72  
 os\_ken.lib.packet.gre  
 module, 73  
 os\_ken.lib.packet.icmp  
 module, 75  
 os\_ken.lib.packet.icmpv6  
 module, 76  
 os\_ken.lib.packet.igmp  
 module, 81  
 os\_ken.lib.packet.ipv4  
 module, 86  
 os\_ken.lib.packet.ipv6  
 module, 87  
 os\_ken.lib.packet.llc  
 module, 91  
 os\_ken.lib.packet.lldp  
 module, 94  
 os\_ken.lib.packet.mpls  
 module, 97  
 os\_ken.lib.packet.openflow  
 module, 98  
 os\_ken.lib.packet.ospf  
 module, 99  
 os\_ken.lib.packet.packet  
 module, 21  
 os\_ken.lib.packet.packet\_base  
 module, 22  
 os\_ken.lib.packet.pbb  
 module, 101  
 os\_ken.lib.packet.sctp  
 module, 102  
 os\_ken.lib.packet.slow  
 module, 115  
 os\_ken.lib.packet.stream\_parser  
 module, 22  
 os\_ken.lib.packet.tcp  
 module, 120  
 os\_ken.lib.packet.udp  
 module, 121



- os\_ken.lib.packet.vlan
    - module, 121
  - os\_ken.lib.packet.vrrp
    - module, 122
  - os\_ken.lib.packet.vxlan
    - module, 126
  - os\_ken.lib.packet.zebra
    - module, 127
  - os\_ken.lib.xflow
    - module, 12
  - os\_ken.ofproto.nicira\_ext
    - module, 567
  - os\_ken.ofproto.ofproto\_v1\_0
    - module, 10
  - os\_ken.ofproto.ofproto\_v1\_0\_parser
    - module, 11
  - os\_ken.ofproto.ofproto\_v1\_2
    - module, 11
  - os\_ken.ofproto.ofproto\_v1\_2\_parser
    - module, 11
  - os\_ken.ofproto.ofproto\_v1\_3
    - module, 11
  - os\_ken.ofproto.ofproto\_v1\_3\_parser
    - module, 11
  - os\_ken.ofproto.ofproto\_v1\_4
    - module, 11
  - os\_ken.ofproto.ofproto\_v1\_4\_parser
    - module, 11
  - os\_ken.ofproto.ofproto\_v1\_5
    - module, 12
  - os\_ken.ofproto.ofproto\_v1\_5\_parser
    - module, 12
  - os\_ken.topology
    - module, 12
  - OSKenApp (class in os\_ken.base.app\_manager), 569
  - ospf (in module os\_ken.lib.packet.ospf), 101
  - OSPFDBDesc (class in os\_ken.lib.packet.ospf), 99
  - OSPFHello (class in os\_ken.lib.packet.ospf), 99
  - OSPFLSack (class in os\_ken.lib.packet.ospf), 99
  - OSPFLSReq (class in os\_ken.lib.packet.ospf), 100
  - OSPFLSUpd (class in os\_ken.lib.packet.ospf), 100
  - OSPFMessage (class in os\_ken.lib.packet.ospf), 100
  - OtherConfChange, 48
  - out\_filter\_get()
    - (os\_ken.services.protocols.bgp.bgpspeaker.BGPSpeaker class method), 146
  - out\_filter\_set()
    - (os\_ken.services.protocols.bgp.bgpspeaker.BGPSpeaker class method), 146
  - OutOfResource, 48
  - OVSBridge (class in os\_ken.lib.ovs.bridge), 156
  - OVSBridgeNotFound, 160
- ## P
- pack() (os\_ken.lib.packet.bfd.bfd method), 29
  - Packet (class in os\_ken.lib.packet.packet), 21
  - PacketBase (class in os\_ken.lib.packet.packet\_base), 22
  - param\_cookie\_preserve (class in os\_ken.lib.packet.sctp), 112
  - param\_ecn (class in os\_ken.lib.packet.sctp), 112
  - param\_heartbeat (class in os\_ken.lib.packet.sctp), 112
  - param\_host\_addr (class in os\_ken.lib.packet.sctp), 113
  - param\_ipv4 (class in os\_ken.lib.packet.sctp), 113
  - param\_ipv6 (class in os\_ken.lib.packet.sctp), 113
  - param\_state\_cookie (class in os\_ken.lib.packet.sctp), 114
  - param\_supported\_addr (class in os\_ken.lib.packet.sctp), 114
  - param\_unrecognized\_param (class in os\_ken.lib.packet.sctp), 114
  - parse() (os\_ken.lib.packet.stream\_parser.StreamParser method), 22
  - parser() (os\_ken.lib.packet.arp.arp class method), 23
  - parser() (os\_ken.lib.packet.bfd.bfd class method), 29
  - parser() (os\_ken.lib.packet.bgp.BGPKeepAlive class method), 32
  - parser() (os\_ken.lib.packet.bgp.BGPMessage class method), 33
  - parser() (os\_ken.lib.packet.bgp.BGPNotification class method), 34
  - parser() (os\_ken.lib.packet.bgp.BGPOpen class method), 34
  - parser() (os\_ken.lib.packet.bgp.BGPRouteRefresh class method), 35
  - parser() (os\_ken.lib.packet.bgp.BGPUUpdate class method), 36
  - parser() (os\_ken.lib.packet.bmp.BMPInitiation class method), 50
  - parser() (os\_ken.lib.packet.bmp.BMPMessage class method), 51
  - parser() (os\_ken.lib.packet.bmp.BMPPeerDownNotification class method), 52
  - parser() (os\_ken.lib.packet.bmp.BMPPeerMessage class method), 52
  - parser() (os\_ken.lib.packet.bmp.BMPPeerUpNotification class method), 52

- class method*), 53
- `parser()` (*os\_ken.lib.packet.bmp.BMPRouteMonitoring class method*), 54
- `parser()` (*os\_ken.lib.packet.bmp.BMPStatisticsReport class method*), 55
- `parser()` (*os\_ken.lib.packet.bmp.BMPTermination class method*), 55
- `parser()` (*os\_ken.lib.packet.bpdu.bpdu class method*), 60
- `parser()` (*os\_ken.lib.packet.bpdu.ConfigurationBPDUs class method*), 58
- `parser()` (*os\_ken.lib.packet.bpdu.RstBPDUs class method*), 59
- `parser()` (*os\_ken.lib.packet.bpdu.TopologyChangeNotificationBPDUs class method*), 60
- `parser()` (*os\_ken.lib.packet.cfm.cfm class method*), 62
- `parser()` (*os\_ken.lib.packet.dhcp.dhcp class method*), 68
- `parser()` (*os\_ken.lib.packet.dhcp6.dhcp6 class method*), 70
- `parser()` (*os\_ken.lib.packet.ethernet.ethernet class method*), 72
- `parser()` (*os\_ken.lib.packet.geneve.geneve class method*), 73
- `parser()` (*os\_ken.lib.packet.gre.gre class method*), 74
- `parser()` (*os\_ken.lib.packet.icmp.icmp class method*), 76
- `parser()` (*os\_ken.lib.packet.icmpv6.icmpv6 class method*), 77
- `parser()` (*os\_ken.lib.packet.igmp.igmp class method*), 83
- `parser()` (*os\_ken.lib.packet.igmp.igmpv3\_query class method*), 84
- `parser()` (*os\_ken.lib.packet.igmp.igmpv3\_report class method*), 85
- `parser()` (*os\_ken.lib.packet.ipv4.ipv4 class method*), 87
- `parser()` (*os\_ken.lib.packet.ipv6.ipv6 class method*), 89
- `parser()` (*os\_ken.lib.packet.llc.llc class method*), 93
- `parser()` (*os\_ken.lib.packet.lldp.lldp class method*), 96
- `parser()` (*os\_ken.lib.packet.mpls.mpls class method*), 97
- `parser()` (*os\_ken.lib.packet.openflow.openflow class method*), 98
- `parser()` (*os\_ken.lib.packet.ospf.OSPFDBDesc class method*), 99
- `parser()` (*os\_ken.lib.packet.ospf.OSPFHello class method*), 99
- `parser()` (*os\_ken.lib.packet.ospf.OSPFLSAck class method*), 100
- `parser()` (*os\_ken.lib.packet.ospf.OSPFLSReq class method*), 100
- `parser()` (*os\_ken.lib.packet.ospf.OSPFLSUdp class method*), 100
- `parser()` (*os\_ken.lib.packet.ospf.OSPFMessage class method*), 100
- `parser()` (*os\_ken.lib.packet.packet\_base.PacketBase class method*), 22
- `parser()` (*os\_ken.lib.packet.pbb.itag class method*), 101
- `parser()` (*os\_ken.lib.packet.sctp.sctp class method*), 115
- `parser()` (*os\_ken.lib.packet.slow.lacp class method*), 119
- `parser()` (*os\_ken.lib.packet.slow.slow class method*), 119
- `parser()` (*os\_ken.lib.packet.tcp.tcp class method*), 120
- `parser()` (*os\_ken.lib.packet.udp.udp class method*), 121
- `parser()` (*os\_ken.lib.packet.vrrp.vrrp class method*), 124
- `parser()` (*os\_ken.lib.packet.vrrp.vrrpv2 class method*), 125
- `parser()` (*os\_ken.lib.packet.vrrp.vrrpv3 class method*), 125
- `parser()` (*os\_ken.lib.packet.vxlan.vxlan class method*), 126
- `parser()` (*os\_ken.lib.packet.zebra.ZebraMessage class method*), 132
- `parser_hdr()` (*os\_ken.lib.packet.bfd.BFDAuth class method*), 25
- PeerDeConfig, 49
- PmsiTunnelIdUnknown (*class in os\_ken.lib.packet.bgp*), 49
- port\_status\_tlv (*class in os\_ken.lib.packet.cfm*), 65
- PortDescription (*class in os\_ken.lib.packet.lldp*), 95
- PortID (*class in os\_ken.lib.packet.lldp*), 95
- prefix\_add() (*os\_ken.services.protocols.bgp.bgpspeaker.BGSP class method*), 146
- prefix\_del() (*os\_ken.services.protocols.bgp.bgpspeaker.BGSP class method*), 147
- PrefixFilter (*class in os\_ken.services.protocols.bgp.info\_base.base*), 148

## R

- `Reader` (class in `os_ken.lib.mrtlib`), 151
- `Reader` (class in `os_ken.lib.pcaplib`), 135
- `register_packet_type()` (`os_ken.lib.packet.packet_base.PacketBase` class method), 23
- `RegisteredNexthop` (class in `os_ken.lib.packet.zebra`), 127
- `remove_db_attribute()` (`os_ken.lib.ovs.bridge.OVSBridge` method), 159
- `reply_egress_tlv` (class in `os_ken.lib.packet.cfm`), 65
- `reply_ingress_tlv` (class in `os_ken.lib.packet.cfm`), 66
- `reply_to_request()` (`os_ken.base.app_manager.OSKenApp` method), 569
- `rib_get()` (`os_ken.services.protocols.bgp.bgpspeaker.BGPspeaker` method), 147
- `RouteTargetMembershipNLRI` (class in `os_ken.lib.packet.bgp`), 49
- `routing` (class in `os_ken.lib.packet.ipv6`), 90
- `routing_type3` (class in `os_ken.lib.packet.ipv6`), 90
- `RoutingLoop`, 49
- `RstBPDUs` (class in `os_ken.lib.packet.bpdu`), 58
- `run_command()` (`os_ken.lib.ovs.bridge.OVSBridge` method), 159
- `run_command()` (`os_ken.lib.ovs.vsctl.VSctl` method), 155
- S**
- `sctp` (class in `os_ken.lib.packet.sctp`), 115
- `SEND_ERROR` (`os_ken.lib.packet.bgp.BadNotification` attribute), 37
- `SEND_ERROR` (`os_ken.lib.packet.bgp.BgpExc` attribute), 37
- `send_event()` (`os_ken.base.app_manager.OSKenApp` method), 570
- `send_event_to_observers()` (`os_ken.base.app_manager.OSKenApp` method), 570
- `send_msg()` (in module `os_ken.app.ofctl.api`), 583
- `send_request()` (`os_ken.base.app_manager.OSKenApp` method), 570
- `sender_id_tlv` (class in `os_ken.lib.packet.cfm`), 66
- `serialize()` (`os_ken.lib.packet.arp.arp` method), 24
- `serialize()` (`os_ken.lib.packet.bfd.bfd` method), 30
- `serialize()` (`os_ken.lib.packet.bfd.KeyedMD5` method), 26
- `serialize()` (`os_ken.lib.packet.bfd.KeyedSHA1` method), 27
- `serialize()` (`os_ken.lib.packet.bfd.SimplePassword` method), 28
- `serialize()` (`os_ken.lib.packet.bgp.BGPMessage` method), 33
- `serialize()` (`os_ken.lib.packet.bmp.BMPMessage` method), 51
- `serialize()` (`os_ken.lib.packet.bpdu.bpdu` method), 60
- `serialize()` (`os_ken.lib.packet.bpdu.ConfigurationBPDUs` method), 58
- `serialize()` (`os_ken.lib.packet.bpdu.RstBPDUs` method), 60
- `serialize()` (`os_ken.lib.packet.cfm.cfm` method), 62
- `serialize()` (`os_ken.lib.packet.dhcp.dhcp` method), 68
- `serialize()` (`os_ken.lib.packet.dhcp6.dhcp6` method), 70
- `serialize()` (`os_ken.lib.packet.ethernet.ethernet` method), 72
- `serialize()` (`os_ken.lib.packet.geneve.geneve` method), 73
- `serialize()` (`os_ken.lib.packet.gre.gre` method), 74
- `serialize()` (`os_ken.lib.packet.icmp.icmp` method), 76
- `serialize()` (`os_ken.lib.packet.icmpv6.icmpv6` method), 77
- `serialize()` (`os_ken.lib.packet.igmp.igmp` method), 84
- `serialize()` (`os_ken.lib.packet.igmp.igmpv3_query` method), 85
- `serialize()` (`os_ken.lib.packet.igmp.igmpv3_report` method), 85
- `serialize()` (`os_ken.lib.packet.ipv4.ipv4` method), 87
- `serialize()` (`os_ken.lib.packet.ipv6.ipv6` method), 89
- `serialize()` (`os_ken.lib.packet.llc.llc` method), 93
- `serialize()` (`os_ken.lib.packet.lldp.lldp` method), 96
- `serialize()` (`os_ken.lib.packet.mpls.mpls` method), 97
- `serialize()` (`os_ken.lib.packet.openflow.openflow` method), 97

method), 99

serialize() (*os\_ken.lib.packet.ospf.OSPFMessage* SUB\_CODE (*os\_ken.lib.packet.bgp.AttrFlagError* attribute), 30  
method), 101

serialize() (*os\_ken.lib.packet.packet.Packet* SUB\_CODE (*os\_ken.lib.packet.bgp.AttrLenError* attribute), 31  
method), 22

serialize() (*os\_ken.lib.packet.packet\_base.PacketBase* SUB\_CODE (*os\_ken.lib.packet.bgp.AuthFailure* attribute), 31  
method), 23

serialize() (*os\_ken.lib.packet.pbb.itag* SUB\_CODE (*os\_ken.lib.packet.bgp.BadBgpId* attribute), 36  
method), 101

serialize() (*os\_ken.lib.packet.sctp.sctp* SUB\_CODE (*os\_ken.lib.packet.bgp.BadLen* attribute), 36  
method), 115

serialize() (*os\_ken.lib.packet.slow.lacp* SUB\_CODE (*os\_ken.lib.packet.bgp.BadMsg* attribute), 37  
method), 119

serialize() (*os\_ken.lib.packet.tcp.tcp* method), SUB\_CODE (*os\_ken.lib.packet.bgp.BadPeerAs* attribute), 37  
120

serialize() (*os\_ken.lib.packet.udp.udp* SUB\_CODE (*os\_ken.lib.packet.bgp.BgpExc* attribute), 37  
method), 121

serialize() (*os\_ken.lib.packet.vrrp.vrrp* SUB\_CODE (*os\_ken.lib.packet.bgp.CollisionResolution* attribute), 37  
method), 124

serialize() (*os\_ken.lib.packet.vxlan.vxlan* SUB\_CODE (*os\_ken.lib.packet.bgp.ConnRejected* attribute), 38  
method), 126

serialize() (*os\_ken.lib.packet.zebra.ZebraMessage* SUB\_CODE (*os\_ken.lib.packet.bgp.FiniteStateMachineError* attribute), 40  
method), 133

serialize\_hdr() SUB\_CODE (*os\_ken.lib.packet.bgp.HoldTimerExpired* attribute), 46  
(*os\_ken.lib.packet.bfd.BFDAuth* method),  
25

set\_controller() SUB\_CODE (*os\_ken.lib.packet.bgp.InvalidNetworkField* attribute), 46  
(*os\_ken.lib.ovs.bridge.OVSBridge* method), 159

set\_db\_attribute() SUB\_CODE (*os\_ken.lib.packet.bgp.InvalidNextHop* attribute), 47  
(*os\_ken.lib.ovs.bridge.OVSBridge* method), 160

set\_ev\_cls() (in module SUB\_CODE (*os\_ken.lib.packet.bgp.InvalidOriginError* attribute), 47  
*os\_ken.controller.handler*), 14

set\_qos() (*os\_ken.lib.ovs.bridge.OVSBridge* SUB\_CODE (*os\_ken.lib.packet.bgp.MalformedAsPath* attribute), 47  
method), 160

shutdown() (*os\_ken.services.protocols.bgp.bgpspeaker.BGPspeaker* SUB\_CODE (*os\_ken.lib.packet.bgp.MalformedAttrList* attribute), 47  
method), 147

SimplePassword (class in *os\_ken.lib.packet.bfd*), SUB\_CODE (*os\_ken.lib.packet.bgp.MalformedOptionalParam* attribute), 48  
28

slow (class in *os\_ken.lib.packet.slow*), 119

start() (*os\_ken.base.app\_manager.OSKenApp* SUB\_CODE (*os\_ken.lib.packet.bgp.MaxPrefixReached* attribute), 48  
method), 570

StreamParser (class in *os\_ken.lib.packet.bgp*), SUB\_CODE (*os\_ken.lib.packet.bgp.MissingWellKnown* attribute), 48  
49

StreamParser (class in *os\_ken.lib.packet.stream\_parser*), 22

StreamParser.TooSmallException, 22

SUB\_CODE (*os\_ken.lib.packet.bgp.AdminReset* attribute), 30

SUB\_CODE (*os\_ken.lib.packet.bgp.AdminShutdown* attribute), 49



SUB\_CODE (*os\_ken.lib.packet.bgp.RoutingLoop* attribute), 49

SUB\_CODE (*os\_ken.lib.packet.bgp.UnacceptableHoldTime* attribute), 50

SUB\_CODE (*os\_ken.lib.packet.bgp.UnRegWellKnowAttr* attribute), 49

SUB\_CODE (*os\_ken.lib.packet.bgp.UnsupportedOptParam* attribute), 50

SUB\_CODE (*os\_ken.lib.packet.bgp.UnsupportedVersion* attribute), 50

svlan (class in *os\_ken.lib.packet.vlan*), 121

SystemCapabilities (class in *os\_ken.lib.packet.lldp*), 95

SystemDescription (class in *os\_ken.lib.packet.lldp*), 95

SystemName (class in *os\_ken.lib.packet.lldp*), 96

**T**

tcp (class in *os\_ken.lib.packet.tcp*), 120

TimeExceeded (class in *os\_ken.lib.packet.icmp*), 75

to\_jsondict() (*os\_ken.ofproto.ofproto\_parser.Message* method), 161

TopologyChangeNotificationBPDUs (class in *os\_ken.lib.packet.bpdu*), 60

try\_parse() (*os\_ken.lib.packet.bgp.StreamParser* method), 49

try\_parse() (*os\_ken.lib.packet.stream\_parser.StreamParser* method), 22

TTL (class in *os\_ken.lib.packet.lldp*), 96

**U**

udp (class in *os\_ken.lib.packet.udp*), 121

UnacceptableHoldTime, 49

UnexpectedMultiReply, 584

UnRegWellKnowAttr, 49

UnsupportedOptParam, 50

UnsupportedVersion, 50

**V**

valid\_ovsdb\_addr() (in module *os\_ken.lib.ovs.vscctl*), 156

vlan (class in *os\_ken.lib.packet.vlan*), 122

vni\_from\_bin() (in module *os\_ken.lib.packet.vxlan*), 126

vni\_to\_bin() (in module *os\_ken.lib.packet.vxlan*), 126

vrf\_add() (*os\_ken.services.protocols.bgp.bgpspeaker.BGPSpeaker* method), 147

vrf\_del() (*os\_ken.services.protocols.bgp.bgpspeaker.BGPSpeaker* method), 147

vrfs\_get() (*os\_ken.services.protocols.bgp.bgpspeaker.BGPSpeaker* method), 147

Vrrp (class in *os\_ken.lib.packet.vrrp*), 123

vrrpv2 (class in *os\_ken.lib.packet.vrrp*), 124

vrrpv3 (class in *os\_ken.lib.packet.vrrp*), 125

VSCtl (class in *os\_ken.lib.ovs.vscctl*), 155

VSCtlCommand (class in *os\_ken.lib.ovs.vscctl*), 155

vxlan (class in *os\_ken.lib.packet.vxlan*), 126

**W**

Writer (class in *os\_ken.lib.mrtlib*), 152

Writer (class in *os\_ken.lib.pcaplib*), 135

**Z**

zebra (in module *os\_ken.lib.packet.zebra*), 135

ZebraBfdClientRegister (class in *os\_ken.lib.packet.zebra*), 127

ZebraBfdDestinationDeregister (class in *os\_ken.lib.packet.zebra*), 128

ZebraBfdDestinationRegister (class in *os\_ken.lib.packet.zebra*), 128

ZebraBfdDestinationReply (class in *os\_ken.lib.packet.zebra*), 128

ZebraBfdDestinationUpdate (class in *os\_ken.lib.packet.zebra*), 128

ZebraHello (class in *os\_ken.lib.packet.zebra*), 128

ZebraImportCheckUpdate (class in *os\_ken.lib.packet.zebra*), 130

ZebraImportRouteRegister (class in *os\_ken.lib.packet.zebra*), 130

ZebraImportRouteUnregister (class in *os\_ken.lib.packet.zebra*), 130

ZebraInterfaceAdd (class in *os\_ken.lib.packet.zebra*), 130

ZebraInterfaceAddressAdd (class in *os\_ken.lib.packet.zebra*), 130

ZebraInterfaceAddressDelete (class in *os\_ken.lib.packet.zebra*), 131

ZebraInterfaceBfdDestinationUpdate (class in *os\_ken.lib.packet.zebra*), 131

ZebraInterfaceDelete (class in *os\_ken.lib.packet.zebra*), 131

ZebraInterfaceDisableRadv (class in *os\_ken.lib.packet.zebra*), 131

ZebraInterfaceDown (class in *os\_ken.lib.packet.zebra*), 131

ZebraInterfaceEnableRadv (class in *os\_ken.lib.packet.zebra*), 131

ZebraInterfaceLinkParams (class in *os\_ken.lib.packet.zebra*), 131

ZebraInterfaceNbrAddressAdd	(class in <i>os_ken.lib.packet.zebra</i> ), 131	<i>os_ken.lib.packet.zebra</i> ), 133
ZebraInterfaceNbrAddressDelete	(class in <i>os_ken.lib.packet.zebra</i> ), 131	ZebraRedistributeDefaultDelete (class in <i>os_ken.lib.packet.zebra</i> ), 133
ZebraInterfaceUp	(class in <i>os_ken.lib.packet.zebra</i> ), 131	ZebraRedistributeDelete (class in <i>os_ken.lib.packet.zebra</i> ), 133
ZebraInterfaceVrfUpdate	(class in <i>os_ken.lib.packet.zebra</i> ), 132	ZebraRedistributeIPv4Add (class in <i>os_ken.lib.packet.zebra</i> ), 133
ZebraIPv4ImportLookup	(class in <i>os_ken.lib.packet.zebra</i> ), 128	ZebraRedistributeIPv4Delete (class in <i>os_ken.lib.packet.zebra</i> ), 134
ZebraIPv4NexthopAdd	(class in <i>os_ken.lib.packet.zebra</i> ), 128	ZebraRedistributeIPv6Add (class in <i>os_ken.lib.packet.zebra</i> ), 134
ZebraIPv4NexthopDelete	(class in <i>os_ken.lib.packet.zebra</i> ), 128	ZebraRedistributeIPv6Delete (class in <i>os_ken.lib.packet.zebra</i> ), 134
ZebraIPv4NexthopLookup	(class in <i>os_ken.lib.packet.zebra</i> ), 129	ZebraRouterIDAdd (class in <i>os_ken.lib.packet.zebra</i> ), 134
ZebraIPv4NexthopLookupMRib	(class in <i>os_ken.lib.packet.zebra</i> ), 129	ZebraRouterIDDelete (class in <i>os_ken.lib.packet.zebra</i> ), 134
ZebraIPv4RouteAdd	(class in <i>os_ken.lib.packet.zebra</i> ), 129	ZebraRouterIDUpdate (class in <i>os_ken.lib.packet.zebra</i> ), 134
ZebraIPv4RouteDelete	(class in <i>os_ken.lib.packet.zebra</i> ), 129	ZebraUnknownMessage (class in <i>os_ken.lib.packet.zebra</i> ), 134
ZebraIPv4RouteIPv6NexthopAdd	(class in <i>os_ken.lib.packet.zebra</i> ), 129	ZebraVrfAdd (class in <i>os_ken.lib.packet.zebra</i> ), 135
ZebraIPv6ImportLookup	(class in <i>os_ken.lib.packet.zebra</i> ), 129	ZebraVrfDelete (class in <i>os_ken.lib.packet.zebra</i> ), 135
ZebraIPv6NexthopAdd	(class in <i>os_ken.lib.packet.zebra</i> ), 129	ZebraVrfUnregister (class in <i>os_ken.lib.packet.zebra</i> ), 135
ZebraIPv6NexthopDelete	(class in <i>os_ken.lib.packet.zebra</i> ), 130	
ZebraIPv6NexthopLookup	(class in <i>os_ken.lib.packet.zebra</i> ), 130	
ZebraIPv6RouteAdd	(class in <i>os_ken.lib.packet.zebra</i> ), 130	
ZebraIPv6RouteDelete	(class in <i>os_ken.lib.packet.zebra</i> ), 130	
ZebraMessage	(class in <i>os_ken.lib.packet.zebra</i> ), 132	
ZebraMplsLabelsAdd	(class in <i>os_ken.lib.packet.zebra</i> ), 133	
ZebraMplsLabelsDelete	(class in <i>os_ken.lib.packet.zebra</i> ), 133	
ZebraNexthopRegister	(class in <i>os_ken.lib.packet.zebra</i> ), 133	
ZebraNexthopUnregister	(class in <i>os_ken.lib.packet.zebra</i> ), 133	
ZebraNexthopUpdate	(class in <i>os_ken.lib.packet.zebra</i> ), 133	
ZebraRedistributeAdd	(class in <i>os_ken.lib.packet.zebra</i> ), 133	
ZebraRedistributeDefaultAdd	(class in <i>os_ken.lib.packet.zebra</i> ), 133	