Chapter 1

Getting Started

1.1 What’s Ryu

Ryu is a component-based software defined networking framework.

Ryu provides software components with well defined API’s that make it easy for developers to create new network management and control applications. Ryu supports various protocols for managing network devices, such as OpenFlow, Netconf, OF-config, etc. About OpenFlow, Ryu supports fully 1.0, 1.2, 1.3, 1.4, 1.5 and Nicira Extensions.

All of the code is freely available under the Apache 2.0 license. Ryu is fully written in Python.

1.2 Quick Start

Installing Ryu is quite easy:

```bash
% pip install ryu
```

If you prefer to install Ryu from the source code:

```bash
% git clone https://github.com/faucetsdn/ryu.git
% cd ryu; pip install .
```

If you want to write your Ryu application, have a look at Writing Ryu application document. After writing your application, just type:

```bash
% ryu-manager yourapp.py
```

1.3 Optional Requirements

Some functions of Ryu require extra packages:
• OF-Config requires lxml and ncclient
• NETCONF requires paramiko
• BGP speaker (SSH console) requires paramiko
• Zebra protocol service (database) requires SQLAlchemy

If you want to use these functions, please install the requirements:

```
% pip install -r tools/optional-requires
```

Please refer to tools/optional-requires for details.

## 1.4 Prerequisites

If you got some error messages at the installation stage, please confirm dependencies for building the required Python packages.

On Ubuntu (16.04 LTS or later):

```
% apt install gcc python-dev libffi-dev libssl-dev libxml2-dev libxslt1-dev zlib1g-dev
```

## 1.5 Support

Ryu Official site is [https://ryu-sdn.org/](https://ryu-sdn.org/).

If you have any questions, suggestions, and patches, the mailing list is available at [ryu-devel ML](https://ryu-devel.ml). The ML archive at [Gmane](http://ryu-devel.mrlabs.org) is also available.
2.1 The First Application

2.1.1 Whetting Your Appetite

If you want to manage network gear (switches, routers, etc) your own way, you just need to write your own Ryu application. Your application tells Ryu how you want to manage the gear. Then Ryu configures the gear by using OpenFlow protocol, etc.

Writing Ryu applications is easy. They’re just Python scripts.

2.1.2 Start Writing

Here we show a Ryu application that makes an OpenFlow switch work as a dumb layer 2 switch.

Open a text editor and create a new file with the following content:

```python
from ryu.base import app_manager

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

Ryu applications are just Python scripts so you can save the file with any name, any extension, and any place you want. Let’s name the file ‘l2.py’ in your home directory.

This application does nothing useful yet, however it’s a complete Ryu application. In fact, you can run this Ryu application:

```bash
% ryu-manager ~/l2.py
loading app /Users/fujita/l2.py
instantiating app /Users/fujita/l2.py
```
All you have to do is define a new subclass of RyuApp to run your Python script as a Ryu application.

Next let’s add some functionality that sends a received packet to all the ports.

```python
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_0

class L2Switch(app_manager.RyuApp):
    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)]

        data = None
        if msg.buffer_id == ofp.OFP_NO BUFFER:
            data = msg.data

        out = ofp_parser.OFPPacketOut(
            datapath=dp, buffer_id=msg.buffer_id, in_port=msg.in_port,
            actions=actions, data=data)
        dp.send_msg(out)
```

A new method ‘packet_in_handler’ is added to the L2Switch class. This is called when Ryu receives an OpenFlow packet_in message. The trick is the ‘set_ev_cls’ decorator. This decorator tells Ryu when the decorated function should be called.

The first argument of the decorator indicates which type of event this function should be called for. As you might expect, every time Ryu gets a packet_in message, this function is called.

The second argument indicates the state of the switch. You probably want to ignore packet_in messages before the negotiation between Ryu and the switch is finished. 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 Ryu 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 uses the OFPP_FLOOD flag to indicate that the packet should be sent out on all ports.
- 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, Ryu builds and sends the on-wire data format to the switch.

There, you finished implementing your first Ryu application. You are ready to run a Ryu application that does something useful.

Is a dumb L2 switch is too dumb? You want to implement a learning L2 switch? Move to the next step. You can learn from the existing Ryu applications at ryu/app directory and integrated tests directory.

2.2 Components of Ryu

2.2.1 Executables

bin/ryu-manager

The main executable.

2.2.2 Base components

ryu.base.app_manager

The central management of Ryu applications.

• Load Ryu applications
• Provide contexts to Ryu applications
• Route messages among Ryu applications

2.2.3 OpenFlow controller

ryu.controller.controller

The main component of OpenFlow controller.

• Handle connections from switches
• Generate and route events to appropriate entities like Ryu applications

ryu.controller.dpset

Manage switches.
Planned to be replaced by ryu/topology.

ryu.controller.ofp_event

OpenFlow event definitions.

ryu.controller.ofp_handler

Basic OpenFlow handling including negotiation.
2.2.4 OpenFlow wire protocol encoder and decoder

`ryu.ofproto.ofproto_v1_0`

OpenFlow 1.0 definitions.

`ryu.ofproto.ofproto_v1_0_parser`

Decoder/Encoder implementations of OpenFlow 1.0.

`ryu.ofproto.ofproto_v1_2`

OpenFlow 1.2 definitions.

`ryu.ofproto.ofproto_v1_2_parser`

Decoder/Encoder implementations of OpenFlow 1.2.

`ryu.ofproto.ofproto_v1_3`

OpenFlow 1.3 definitions.

`ryu.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
ryu.ofproto.ofproto_v1_4

OpenFlow 1.4 definitions.

ryu.ofproto.ofproto_v1_4_parser

Decoder/Encoder implementations of OpenFlow 1.4.

ryu.ofproto.ofproto_v1_5

OpenFlow 1.5 definitions.

ryu.ofproto.ofproto_v1_5_parser

Decoder/Encoder implementations of OpenFlow 1.5.

2.2.5 Ryu applications

ryu.app.cbench

A dumb OpenFlow 1.0 responder for benchmarking the controller framework. Intended to be used with oflops cbench.

ryu.app.simple_switch

An OpenFlow 1.0 L2 learning switch implementation.

ryu.topology

Switch and link discovery module. Planned to replace ryu/controller/dpset.

2.2.6 Libraries

ryu.lib.packet

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

ryu.lib.ovs

ovsdb interaction library.

ryu.lib.of_config

OF-Config implementation.

ryu.lib.netconf

NETCONF definitions used by ryu/lib/of_config.

2.2. Components of Ryu
ryu Documentation, Release 4.34

ryn.lib.xflow
An implementation of sFlow and NetFlow.

2.2.7 Third party libraries

ryn.contrib.ovs
Open vSwitch python binding. Used by ryn.lib.ovs.

ryn.contrib.oslo.config
Oslo configuration library. Used for ryn-manager’s command-line options and configuration files.

ryn.contrib.ncclient
Python library for NETCONF client. Used by ryn.lib.of_config.

2.3 Ryu application API

2.3.1 Ryu application programming model

Threads, events, and event queues
Ryu applications are single-threaded entities which implement various functionalities in Ryu. Events are messages between them.

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

Each Ryu application has a receive queue for events. The queue is FIFO and preserves the order of events. Each Ryu application has a thread for event processing. The thread keeps draining the receive queue by dequeueuing 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 Ryu application will be processed.

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

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

Contexts
Contexts are ordinary python objects shared among Ryu applications. The use of contexts are discouraged for new code.
2.3.2 Create a Ryu application

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

2.3.3 Observe events

A Ryu application can register itself to listen for specific events using ryu.controller.handler.set_ev_cls decorator.

2.3.4 Generate events

A Ryu application can raise events by calling appropriate ryu.base.app_manager.RyuApp’s methods like send_event or send_event_to_observers.

2.3.5 Event classes

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

OpenFlow event classes

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

class ryu.controller.ofp_event.EventOFPMsgBase(msg)
The base class of OpenFlow event class.

OpenFlow event classes have at least the following attributes.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>msg</td>
<td>An object which describes the corresponding OpenFlow message.</td>
</tr>
<tr>
<td>msg.datapath</td>
<td>A ryu.controller.controller.Datapath instance which describes an OpenFlow switch from which we received this OpenFlow message.</td>
</tr>
<tr>
<td>timestamp</td>
<td>Timestamp when Datapath instance generated this event.</td>
</tr>
</tbody>
</table>

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.

2.3.6 ryu.base.app_manager.RyuApp

See Ryu API Reference.
2.3.7 ryu.controller.handler.set_ev_cls

`ryu.controller.handler.set_ev_cls(ev_cls, dispatchers=None)`

A decorator for Ryu application to declare an event handler.

Decorated method will become an event handler. `ev_cls` is an event class whose instances this RyuApp 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.

<table>
<thead>
<tr>
<th>Negotiation phase</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ryu.controller.handler.HANDSHAKE_DISPATCHER</td>
<td>Sending and waiting for hello message</td>
</tr>
<tr>
<td>ryu.controller.handler.CONFIG_DISPATCHER</td>
<td>Version negotiated and sent features-request message</td>
</tr>
<tr>
<td>ryu.controller.handler.MAIN_DISPATCHER</td>
<td>Switch-features message received and sent set-config message</td>
</tr>
<tr>
<td>ryu.controller.handler.DEAD_DISPATCHER</td>
<td>Disconnect from the peer. Or disconnecting due to some unrecoverable errors.</td>
</tr>
</tbody>
</table>

2.3.8 ryu.controller.controller.Datapath

class `ryu.controller.controller.Datapath(socket, address)`

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

An instance has the following attributes.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>64-bit OpenFlow Datapath ID. Only available for ryu.controller.handler.MAIN_DISPATCHER phase.</td>
</tr>
<tr>
<td>ofproto</td>
<td>A module which exports OpenFlow definitions, mainly constants appeared in the specification, for the negotiated OpenFlow version. For example, ryu.ofproto.ofproto_v1_0 for OpenFlow 1.0.</td>
</tr>
<tr>
<td>ofproto_parser</td>
<td>A module which exports OpenFlow wire message encoder and decoder for the negotiated OpenFlow version. For example, ryu.ofproto.ofproto_v1_0_parser for OpenFlow 1.0.</td>
</tr>
<tr>
<td>ofproto_parser.OFPxxxx(datapath,...)</td>
<td>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. Argument depends on the message.</td>
</tr>
<tr>
<td>set_xid(self, msg)</td>
<td>Generate an OpenFlow XID and put it in msg.xid.</td>
</tr>
<tr>
<td>send_msg(self, msg)</td>
<td>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.</td>
</tr>
<tr>
<td>send_packet_out</td>
<td>deprecated</td>
</tr>
<tr>
<td>send_flow_mod</td>
<td>deprecated</td>
</tr>
<tr>
<td>send_flow_del</td>
<td>deprecated</td>
</tr>
<tr>
<td>send_delete_all_flows</td>
<td>deprecated</td>
</tr>
<tr>
<td>send_barrier</td>
<td>Queue an OpenFlow barrier message to send to the switch.</td>
</tr>
<tr>
<td>send_nxt_set_flow_format</td>
<td>deprecated</td>
</tr>
<tr>
<td>is_reserved_port</td>
<td>deprecated</td>
</tr>
</tbody>
</table>
2.3.9 ryu.controller.event.EventBase

class ryu.controller.event.EventBase
    The base of all event classes.

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

2.3.10 ryu.controller.event.EventRequestBase

class ryu.controller.event.EventRequestBase
    The base class for synchronous request for RyuApp.send_request.

2.3.11 ryu.controller.event.EventReplyBase

class ryu.controller.event.EventReplyBase(dst)
    The base class for synchronous request reply for RyuApp.send_reply.

2.3.12 ryu.controller.ofp_event.EventOFPStateChange

class ryu.controller.ofp_event.EventOFPStateChange(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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>datapath</td>
<td>ryu.controller.controller.Datapath instance of the switch</td>
</tr>
</tbody>
</table>

2.3.13 ryu.controller.ofp_event.EventOFPPortStateChange

class ryu.controller.ofp_event.EventOFPPortStateChange(dp, reason, port_no)
    An event class to notify the port state changes of Ddatapath instance.

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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>datapath</td>
<td>ryu.controller.controller.Datapath instance of the switch</td>
</tr>
<tr>
<td>reason</td>
<td>one of OFPPR_*</td>
</tr>
<tr>
<td>port_no</td>
<td>Port number which state was changed</td>
</tr>
</tbody>
</table>

2.3.14 ryu.controller.dpset.EventDP

class ryu.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 ryu.controller.ofp_event.EventOFPStateChange. An instance has at least the following attributes.
### 2.3.15 `ryu.controller.dpset.EventPortAdd`

**class** `ryu.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 `ryu.controller.ofp_event.EventOFPPortStatus`. An instance has at least the following attributes.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dp</td>
<td>A <code>ryu.controller.controller.Datapath</code> instance of the switch</td>
</tr>
<tr>
<td>port</td>
<td>port number</td>
</tr>
</tbody>
</table>

### 2.3.16 `ryu.controller.dpset.EventPortDelete`

**class** `ryu.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 `ryu.controller.ofp_event.EventOFPPortStatus`. An instance has at least the following attributes.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dp</td>
<td>A <code>ryu.controller.controller.Datapath</code> instance of the switch</td>
</tr>
<tr>
<td>port</td>
<td>port number</td>
</tr>
</tbody>
</table>

### 2.3.17 `ryu.controller.dpset.EventPortModify`

**class** `ryu.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 `ryu.controller.ofp_event.EventOFPPortStatus`. An instance has at least the following attributes.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dp</td>
<td>A <code>ryu.controller.controller.Datapath</code> instance of the switch</td>
</tr>
<tr>
<td>port</td>
<td>port number</td>
</tr>
</tbody>
</table>

### 2.3.18 `ryu.controller.network.EventNetworkPort`

**class** `ryu.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.

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

2.3.19 ryu.controller.network.EventNetworkDel

class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>network_id</td>
<td>Network ID</td>
</tr>
</tbody>
</table>

2.3.20 ryu.controller.network.EventMacAddress

class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>network_id</td>
<td>Network ID</td>
</tr>
<tr>
<td>dpid</td>
<td>OpenFlow Datapath ID of the switch to which the port belongs.</td>
</tr>
<tr>
<td>port_no</td>
<td>OpenFlow port number of the port</td>
</tr>
<tr>
<td>mac_address</td>
<td>The old MAC address of the port if add_del is False. Otherwise the new MAC address.</td>
</tr>
<tr>
<td>add_del</td>
<td>False if this event is a result of a port removal. Otherwise True.</td>
</tr>
</tbody>
</table>

2.3.21 ryu.controller.tunnels.EventTunnelKeyAdd

class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>network_id</td>
<td>Network ID</td>
</tr>
<tr>
<td>tunnel_key</td>
<td>Tunnel Key</td>
</tr>
</tbody>
</table>
2.3.22  ryu.controller.tunnels.EventTunnelKeyDel

class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>network_id</td>
<td>Network ID</td>
</tr>
<tr>
<td>tunnel_key</td>
<td>Tunnel Key</td>
</tr>
</tbody>
</table>

2.3.23  ryu.controller.tunnels.EventTunnelPort

class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dpid</td>
<td>OpenFlow Datapath ID</td>
</tr>
<tr>
<td>port_no</td>
<td>OpenFlow port number</td>
</tr>
<tr>
<td>remote_dpid</td>
<td>OpenFlow port number of the tunnel peer</td>
</tr>
<tr>
<td>add_del</td>
<td>True for adding a tunnel. False for removal.</td>
</tr>
</tbody>
</table>

2.4  Library

Ryu provides some useful library for your network applications.

2.4.1  Packet library

Introduction

Ryu 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:4e7’, ‘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 ryu.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:

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

If vlan is not used, you see something like:

<ryu.lib.packet.ethernet.ethernet object at 0x107a5d790>
<ryu.lib.packet.ipv4.ipv4 object at 0x107a5d810>
<ryu.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 ryu.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 <ryu.lib.packet.ethernet.ethernet object at 0x107a5d790>
vlan <ryu.lib.packet.vlan.vlan object at 0x107a5d7d0>
vid = 10
ipv4 <ryu.lib.packet.ipv4.ipv4 object at 0x107a5d810>
tcp <ryu.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 ryu.ofproto import ether
from ryu.lib.packet import ethernet, arp, packet

e = ethernet.ethernet(dst='ff:ff:ff:ff:ff:ff',
                      src='08:60:6e:7f:74:7e',
                      ...)

(continues on next page)
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',
    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

2.4.2 Packet library API Reference

Packet class

class ryu.lib.packet.packet.Packet (data=None, protocols=None, parse_cls=<class
    'ryu.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.

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

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 ryu.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.

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:

• ryu.lib.packet.bgp.StreamParser

Protocol Header classes

Packet Base Class

class ryu.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.

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 **ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>hwtype</td>
<td>Hardware address.</td>
<td>'ff:ff:ff:ff:ff:ff'</td>
</tr>
<tr>
<td>proto</td>
<td>Protocol address.</td>
<td>'08:60:6e:7f:74:e7'</td>
</tr>
<tr>
<td>hlen</td>
<td>byte length of each hardware address.</td>
<td>6</td>
</tr>
<tr>
<td>plen</td>
<td>byte length of each protocol address.</td>
<td>4</td>
</tr>
<tr>
<td>opcode</td>
<td>operation codes.</td>
<td>1</td>
</tr>
<tr>
<td>src_mac</td>
<td>Hardware address of sender.</td>
<td>'ff:ff:ff:ff:ff:ff'</td>
</tr>
<tr>
<td>src_ip</td>
<td>Protocol address of sender.</td>
<td>'0.0.0.0'</td>
</tr>
<tr>
<td>dst_mac</td>
<td>Hardware address of target.</td>
<td>'ff:ff:ff:ff:ff:ff'</td>
</tr>
<tr>
<td>dst_ip</td>
<td>Protocol address of target.</td>
<td>'192.0.2.1'</td>
</tr>
</tbody>
</table>

**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.

```python
ryu.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.

```python
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:

```
| Vers | Diag | Sta|P|F|C|A|D|M| Detect Mult | Length |
|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|
| My Discriminator |
| Your Discriminator |
| Desired Min TX Interval |
| Required Min RX Interval |
| Required Min Echo RX Interval |
```

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

1. Format of Simple Password Authentication Section:

```
<table>
<thead>
<tr>
<th>Auth Type</th>
<th>Auth Len</th>
<th>Auth Key ID</th>
<th>Password...</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

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

```
<table>
<thead>
<tr>
<th>Auth Type</th>
<th>Auth Len</th>
<th>Auth Key ID</th>
<th>Reserved</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence Number</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Auth Key/Digest...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

(continues on next page)
3. Format of Keyed SHA1 and Meticulous Keyed SHA1 Authentication Section:

```
| Auth Type | Auth Len | Auth Key ID | Reserved |
+------------+----------+-------------+----------+
| Sequence Number |
+----------------+
| Auth Key/Hash... |
+----------------+
| ... |
```

**class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>auth_type</td>
<td>The authentication type in use.</td>
</tr>
<tr>
<td>auth_len</td>
<td>The length, in bytes, of the authentication section, including the auth_type and auth_len fields.</td>
</tr>
</tbody>
</table>

**classmethod parser_hdr**(buf)

Parser for common part of authentication section.

**serialize_hdr**()

Serialization function for common part of authentication section.

**class ryu.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.

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

**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 ryu.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.

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

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

```
class ryu.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.

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

```
class ryu.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.

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

```
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 ryu.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 looks 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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ver</td>
<td>The version number of the protocol. This class implements protocol version 1.</td>
</tr>
<tr>
<td>diag</td>
<td>A diagnostic code specifying the local system’s reason for the last change in session state.</td>
</tr>
<tr>
<td>state</td>
<td>The current BFD session state as seen by the transmitting system.</td>
</tr>
<tr>
<td>flags</td>
<td>Bitmap of the following flags.</td>
</tr>
<tr>
<td></td>
<td>BFD_FLAG_POLL</td>
</tr>
<tr>
<td></td>
<td>BFD_FLAG_FINAL</td>
</tr>
<tr>
<td></td>
<td>BFD_FLAG_CTRL_PLANE_INDEP</td>
</tr>
<tr>
<td></td>
<td>BFD_FLAG_AUTH_PRESENT</td>
</tr>
<tr>
<td></td>
<td>BFD_FLAG_DEMAND</td>
</tr>
<tr>
<td></td>
<td>BFD_FLAG_MULTIPOINT</td>
</tr>
<tr>
<td>detect_mult</td>
<td>Detection time multiplier.</td>
</tr>
<tr>
<td>my_discr</td>
<td>My Discriminator.</td>
</tr>
<tr>
<td>your_discr</td>
<td>Your Discriminator.</td>
</tr>
<tr>
<td>desired_min_tx_interval</td>
<td>Desired Min TX Interval. (in microseconds)</td>
</tr>
<tr>
<td>required_min_rx_interval</td>
<td>Required Min RX Interval. (in microseconds)</td>
</tr>
<tr>
<td>required_min_echo_rx_interval</td>
<td>Required Min Echo RX Interval. (in microseconds)</td>
</tr>
<tr>
<td>auth_cls</td>
<td>(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: SimplePassword, KeyedMD5, MeticulousKeyedMD5, KeyedSHA1, and MeticulousKeyedSHA1.</td>
</tr>
<tr>
<td>length</td>
<td>(Optional) Length of the BFD Control packet, in bytes.</td>
</tr>
</tbody>
</table>

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.
Decoding 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 ryu.lib.packet.bgp.AdminReset(data="")

exception ryu.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'.

exception ryu.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).

exception ryu.lib.packet.bgp.AttrLenError(data="")
exception ryu.lib.packet.bgp.AuthFailure(data="")

class ryu.lib.packet.bgp.BGPEvpnEsImportRTExtendedCommunity(**kwargs)
ES-Import Route Target Extended Community

class ryu.lib.packet.bgp.BGPEvpnEsiLabelExtendedCommunity
ESI Label Extended Community

(label=None, mpls_label=None, vni=None, **kwargs)
class ryu.lib.packet.bgp.BGPEvpnMacMobilityExtendedCommunity(**kwargs)
    MAC Mobility Extended Community

class ryu.lib.packet.bgp.BGPEFlowSpecRedirectCommunity(**kwargs)

    Attribute       Description
    as_number       Autonomous System number.
    local_administrator  Local Administrator.

class ryu.lib.packet.bgp.BGPEFlowSpecTPIDActionCommunity(**kwargs)
    Flow Specification TPID Actions.

    Attribute       Description
    actions         Bit representation of actions. Supported actions are TI(inner TPID action) and TO(outer TPID action).
    tpid_1          TPID used by TI.
    tpid_2          TPID used by TO.

class ryu.lib.packet.bgp.BGPEFlowSpecTrafficActionCommunity(**kwargs)

    Attribute       Description
    action          Apply action. The supported action are SAMPLE and TERMINAL.

class ryu.lib.packet.bgp.BGPEFlowSpecTrafficMarkingCommunity(**kwargs)

    Attribute       Description
    dscp            Differentiated Services Code Point.

class ryu.lib.packet.bgp.BGPEFlowSpecTrafficRateCommunity(**kwargs)

    Attribute       Description
    as_number       Autonomous System number.
    rate_info       rate information.

class ryu.lib.packet.bgp.BGPEFlowSpecVlanActionCommunity(**kwargs)
    Flow Specification Vlan Actions.

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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>marker</td>
<td>Marker field. Ignored when encoding.</td>
</tr>
<tr>
<td>len</td>
<td>Length field. Ignored when encoding.</td>
</tr>
<tr>
<td>type</td>
<td>Type field.</td>
</tr>
</tbody>
</table>

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 ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>marker</td>
<td>Marker field. Ignored when encoding.</td>
</tr>
<tr>
<td>len</td>
<td>Length field. Ignored when encoding.</td>
</tr>
<tr>
<td>type</td>
<td>Type field. one of BGP_MSG_* constants.</td>
</tr>
</tbody>
</table>

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 ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>marker</td>
<td>Marker field. Ignored when encoding.</td>
</tr>
<tr>
<td>len</td>
<td>Length field. Ignored when encoding.</td>
</tr>
<tr>
<td>type</td>
<td>Type field.</td>
</tr>
<tr>
<td>error_code</td>
<td>Error code field.</td>
</tr>
<tr>
<td>error_subcode</td>
<td>Error subcode field.</td>
</tr>
<tr>
<td>data</td>
<td>Data field.</td>
</tr>
</tbody>
</table>

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 ryu.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.

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

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.
class ryu.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.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dict_</td>
<td>A dictionary which describes the parameters. For example, {&quot;Param1&quot;: 100, &quot;Param2&quot;: 200}</td>
</tr>
<tr>
<td>decode_string</td>
<td>(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 _TYPE class attribute.</td>
</tr>
<tr>
<td>additional_args</td>
<td>(Optional) Additional kwargs for constructor.</td>
</tr>
</tbody>
</table>

class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>marker</td>
<td>Marker field. Ignored when encoding.</td>
</tr>
<tr>
<td>len</td>
<td>Length field. Ignored when encoding.</td>
</tr>
<tr>
<td>type</td>
<td>Type field.</td>
</tr>
<tr>
<td>afi</td>
<td>Address Family Identifier</td>
</tr>
<tr>
<td>safi</td>
<td>Subsequent Address Family Identifier</td>
</tr>
</tbody>
</table>

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 ryu.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.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>marker</td>
<td>Marker field. Ignored when encoding.</td>
</tr>
<tr>
<td>len</td>
<td>Length field. Ignored when encoding.</td>
</tr>
<tr>
<td>type</td>
<td>Type field.</td>
</tr>
<tr>
<td>withdrawn_routes_len</td>
<td>Withdrawn Routes Length field. Ignored when encoding.</td>
</tr>
<tr>
<td>withdrawn_routes</td>
<td>Withdrawn Routes field. A list of BGPWithdrawnRoute instances. The default is</td>
</tr>
<tr>
<td></td>
<td>[].</td>
</tr>
<tr>
<td>total_path_attribute_len</td>
<td>Total Path Attribute Length field. Ignored when encoding.</td>
</tr>
<tr>
<td>path_attributes</td>
<td>Path Attributes field. A list of BGPPacketAttribute instances. The default is</td>
</tr>
<tr>
<td></td>
<td>[].</td>
</tr>
<tr>
<td>nlri</td>
<td>Network Layer Reachability Information field. A list of BGPNLRI instances.</td>
</tr>
<tr>
<td></td>
<td>The default is [].</td>
</tr>
</tbody>
</table>

```python

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.
```

```python
exception ryu.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.
```

```python
exception ryu.lib.packet.bgp.BadLen(msg_type_code, message_length)
```

```python
exception ryu.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.
```

```python
exception ryu.lib.packet.bgp.BadNotification(data='')
```

```python
exception ryu.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.
```

```python
exception ryu.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 ryu.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".

exception ryu.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".

class ryu.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 ryu.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 ryu.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 ryu.lib.packet.bgp.EvpnEthernetAutoDiscoveryNLRI(route_dist, esi, ethernet_tag_id, mpls_label=None, vni=None, label=None, type_=None, length=None)
   Ethernet A-D route type specific EVPN NLRI

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

BGP Network Layer Reachability Information (NLRI) for EVPN

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

ESI value for unknown type

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

Unknown route type specific EVPN NLRI

exception ryu.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.

class ryu.lib.packet.bgp.FlowSpecComponentUnknown (buf, type_=None)

Unknown component type for Flow Specification NLRI component

class ryu.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 ryu.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 ryu.lib.packet.bgp.FlowSpecDestPrefix (length, addr, type_=None)
    Destination Prefix for Flow Specification NLRI component

class ryu.lib.packet.bgp.FlowSpecDestinationMac (length, addr, type_=None)
    Destination Mac Address.
    Set the Mac Address at value.

class ryu.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 ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LF</td>
<td>Last fragment</td>
</tr>
<tr>
<td>FF</td>
<td>First fragment</td>
</tr>
<tr>
<td>ISF</td>
<td>Is a fragment</td>
</tr>
<tr>
<td>DF</td>
<td>Don’t fragment</td>
</tr>
</tbody>
</table>

class ryu.lib.packet.bgp.FlowSpecIPProtocol (operator, value, type_=None)
    IP Protocol for Flow Specification NLRI component
    Set the IP protocol number at value.

class ryu.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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Value</th>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dst_prefix</td>
<td>IPv4 Prefix</td>
<td>Nothing</td>
<td>Destination Prefix.</td>
</tr>
<tr>
<td>src_prefix</td>
<td>IPv4 Prefix</td>
<td>Nothing</td>
<td>Source Prefix.</td>
</tr>
<tr>
<td>ip_proto</td>
<td>Integer</td>
<td>Numeric</td>
<td>IP Protocol.</td>
</tr>
<tr>
<td>port</td>
<td>Integer</td>
<td>Numeric</td>
<td>Port number.</td>
</tr>
<tr>
<td>dst_port</td>
<td>Integer</td>
<td>Numeric</td>
<td>Destination port number.</td>
</tr>
<tr>
<td>src_port</td>
<td>Integer</td>
<td>Numeric</td>
<td>Source port number.</td>
</tr>
<tr>
<td>icmp_type</td>
<td>Integer</td>
<td>Numeric</td>
<td>ICMP type.</td>
</tr>
<tr>
<td>icmp_code</td>
<td>Integer</td>
<td>Numeric</td>
<td>ICMP code.</td>
</tr>
<tr>
<td>tcp_flags</td>
<td>Fixed string</td>
<td>Bit-mask</td>
<td>TCP flags. Supported values are CWR, ECN, URGENT, ACK, PUSH, RST, SYN and FIN.</td>
</tr>
<tr>
<td>packet_len</td>
<td>Integer</td>
<td>Numeric</td>
<td>Packet length.</td>
</tr>
<tr>
<td>dscp</td>
<td>Integer</td>
<td>Numeric</td>
<td>Differentiated Services Code Point.</td>
</tr>
<tr>
<td>fragment</td>
<td>Fixed string</td>
<td>Bit-mask</td>
<td>Fragment. Supported values are DF (Don’t fragment), ISF (Is a fragment), FF (First fragment) and LF (Last fragment)</td>
</tr>
</tbody>
</table>

Example:

```python
>>> 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',
    ...     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.
The following keywords can be used when the operator type is Bitmask.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>!=</td>
<td>Not equal operation.</td>
</tr>
<tr>
<td>==</td>
<td>Exact match operation if specified. Otherwise partial match operation.</td>
</tr>
<tr>
<td>+</td>
<td>Used for the summation of bitmask values. (e.g., SYN+ACK)</td>
</tr>
</tbody>
</table>

You can combine the multiple conditions with the following operators.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Logical OR operation</td>
</tr>
<tr>
<td>&amp;</td>
<td>Logical AND operation</td>
</tr>
</tbody>
</table>

Returns A instance of FlowSpecVPNv4NLRI.

class ryu.lib.packet.bgp.FlowSpecIPv6DestPrefix (length, addr, offset=0, type_=None)  
IPv6 destination Prefix for Flow Specification NLRI component

class ryu.lib.packet.bgp.FlowSpecIPv6FlowLabel (operator, value, type_=None)  
Fragment for Flow Specification for IPv6 NLRI component

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

**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.

```python
class ryu.lib.packet.bgp.FlowSpecIPv6SrcPrefix(length, addr, offset=0, type_=None)
   IPv6 source Prefix for Flow Specification NLRI component

class ryu.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 ryu.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 ryu.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 ryu.lib.packet.bgp.FlowSpecInnerVLANID(operator, value, type_=None)
   Inner VLAN ID.
   Set VLAN ID at value.
```
class ryu.lib.packet.bgp.FlowSpecL2VPNNLRI(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.

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

class ryu.lib.packet.bgp.FlowSpecLLControl(operator, value, type_=None)
Control field in LLC header in an Ethernet frame.

Set the Contorol field at value.

class ryu.lib.packet.bgp.FlowSpecLLCSAP(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 ryu.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 ryu.lib.packet.bgp.FlowSpecNextHeader(operator, value, type_=None)
Next Header value in IPv6 packets

Set the IP protocol number at value

class ryu.lib.packet.bgp.FlowSpecPacketLen(operator, value, type_=None)
Packet length for Flow Specification NLRI component

Set the total IP packet length at value.

class ryu.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 ryu.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 ryu.lib.packet.bgp.FlowSpecSourceMac(length, addr, type_=None)
Source Mac Address.

Set the Mac Address at value.
class ryu.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 ryu.lib.packet.bgp.FlowSpecSrcPrefix (length, addr, type_=None)
Source Prefix for Flow Specification NLRI component

class ryu.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 ryu.lib.packet.bgp.FlowSpecVLANCoS (operator, value, type_=None)
VLAN CoS Fields in an Ethernet frame.
Set the 3 bit CoS field at value.

class ryu.lib.packet.bgp.FlowSpecVLANID (operator, value, type_=None)
VLAN ID.
Set VLAN ID at value.

class ryu.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 ryu.lib.packet.bgp.FlowSpecIPv4NLRI

Example:

```python
>>> 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')
```
- **route_dist** – Route Distinguisher.
- **kwargs** – See `ryu.lib.packet.bgp.FlowSpecIPv6NLRI`

**exception** `ryu.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.

**exception** `ryu.lib.packet.bgp.InvalidNetworkField(data="")`

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

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

Error indicates undefined Origin attribute value.

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

**exception** `ryu.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.

**exception** `ryu.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.

**exception** `ryu.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).

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

**exception** `ryu.lib.packet.bgp.MissingWellKnown(pattr_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.

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

**exception** `ryu.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).

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

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

**exception** `ryu.lib.packet.bgp.PeerDeConfig(data="")`
class ryu.lib.packet.bgp.PmsiTunnelIdUnknown(value)

Unknown route type specific _PmsiTunnelId

class ryu.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 ryu.lib.packet.bgp.RoutingLoop(data="")

class ryu.lib.packet.bgp.StreamParser

Streaming parser for BGP-4 messages.

This is a subclass of ryu.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 ryu.lib.packet.bgp.UnRegWellKnowAttr(data="")

exception ryu.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.

exception ryu.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.

exception ryu.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.

BMP

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

class ryu.lib.packet.bmp.BMPIInitiation(info, type_=4, len_=None, version=3)

BMP Initiation Message
### Class Method `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.

```python
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>Version. this packet lib defines BMP ver. 3</td>
</tr>
<tr>
<td>len</td>
<td>Length field. Ignored when encoding.</td>
</tr>
<tr>
<td>type</td>
<td>Type field. one of BMP_MSG_ constants.</td>
</tr>
</tbody>
</table>

### Class Method `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.
### 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.

1. An object to describe the decoded header.
2. 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.
3. The rest of packet.

### BMP Message with Per Peer Header

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

- BMP_MSG_TYPE_ROUTE_MONITORING
- 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 bytestring `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 the packet.

```python
class ryu.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

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

```python
classmethod parser(buf):
    Decode a protocol header.
    This method is used only when decoding a packet.
    Decode a protocol header at offset 0 in bytestring `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 the packet.
```

```python
class ryu.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

44 Chapter 2. Writing Your Ryu Application
### Attribute Description

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

#### 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.

```python
class ryu.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

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

#### 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.

```python
class Ryu.lib.packet.bmp.BMPTermination(info, type_=5, len_=None, version=3)
```

BMP Termination Message

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>Version. this packet lib defines BMP ver. 3</td>
</tr>
<tr>
<td>len</td>
<td>Length field. Ignored when encoding.</td>
</tr>
<tr>
<td>type</td>
<td>Type field. one of BMP_MSG_ constants.</td>
</tr>
<tr>
<td>info</td>
<td>One or more piece of information encoded as a TLV</td>
</tr>
</tbody>
</table>

```python
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**


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

Topology Change Notification BPDUs format

<table>
<thead>
<tr>
<th>Structure</th>
<th>Octet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protocol Identifier = 0000 0000 0000 0000</td>
<td>1 - 2</td>
</tr>
<tr>
<td>Protocol Version Identifier = 0000 0000</td>
<td>3</td>
</tr>
<tr>
<td>BPDU Type = 1000 0000</td>
<td>4</td>
</tr>
</tbody>
</table>

Rapid Spanning Tree BPDUs (RST BPDUs) format
<table>
<thead>
<tr>
<th>Structure</th>
<th>Octet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protocol Identifier = 0000 0000 0000 0000</td>
<td>1 - 2</td>
</tr>
<tr>
<td>Protocol Version Identifier = 0000 0010</td>
<td>3</td>
</tr>
<tr>
<td>BPDU Type = 0000 0010</td>
<td>4</td>
</tr>
<tr>
<td>Flags</td>
<td>5</td>
</tr>
<tr>
<td><strong>Root Identifier</strong></td>
<td></td>
</tr>
<tr>
<td>include - priority</td>
<td></td>
</tr>
<tr>
<td>system ID extension</td>
<td></td>
</tr>
<tr>
<td>MAC address</td>
<td>6 - 13</td>
</tr>
<tr>
<td><strong>Root Path Cost</strong></td>
<td></td>
</tr>
<tr>
<td>14 - 17</td>
<td></td>
</tr>
<tr>
<td><strong>Bridge Identifier</strong></td>
<td></td>
</tr>
<tr>
<td>include - priority</td>
<td></td>
</tr>
<tr>
<td>system ID extension</td>
<td></td>
</tr>
<tr>
<td>MAC address</td>
<td>18 - 25</td>
</tr>
<tr>
<td><strong>Port Identifier</strong></td>
<td></td>
</tr>
<tr>
<td>include - priority</td>
<td></td>
</tr>
<tr>
<td>port number</td>
<td>26 - 27</td>
</tr>
<tr>
<td><strong>Message Age</strong></td>
<td></td>
</tr>
<tr>
<td>28 - 29</td>
<td></td>
</tr>
<tr>
<td><strong>Max Age</strong></td>
<td></td>
</tr>
<tr>
<td>30 - 31</td>
<td></td>
</tr>
<tr>
<td><strong>Hello Time</strong></td>
<td></td>
</tr>
<tr>
<td>32 - 33</td>
<td></td>
</tr>
<tr>
<td><strong>Forward Delay</strong></td>
<td></td>
</tr>
<tr>
<td>34 - 35</td>
<td></td>
</tr>
<tr>
<td>Version 1 Length = 0000 0000</td>
<td>36</td>
</tr>
</tbody>
</table>

**class** ryu.lib.packet.bpdu.ConfigurationBPDU

( 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 BPDU(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.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Bit 1: Topology Change flag&lt;br&gt;Bits 2 through 7: unused and take the value 0&lt;br&gt;Bit 8: Topology Change Acknowledgment flag</td>
</tr>
<tr>
<td>root_priority</td>
<td>Root Identifier priority set 0-61440 in steps of 4096</td>
</tr>
<tr>
<td>root_system_id_extension</td>
<td>Root Identifier system ID extension</td>
</tr>
<tr>
<td>root_mac_address</td>
<td>Root Identifier MAC address</td>
</tr>
<tr>
<td>root_path_cost</td>
<td>Root Path Cost</td>
</tr>
<tr>
<td>bridge_priority</td>
<td>Bridge Identifier priority set 0-61440 in steps of 4096</td>
</tr>
<tr>
<td>bridge_system_id_extension</td>
<td>Bridge Identifier system ID extension</td>
</tr>
<tr>
<td>bridge_mac_address</td>
<td>Bridge Identifier MAC address</td>
</tr>
<tr>
<td>port_priority</td>
<td>Port Identifier priority set 0-240 in steps of 16</td>
</tr>
<tr>
<td>port_number</td>
<td>Port Identifier number</td>
</tr>
<tr>
<td>message_age</td>
<td>Message Age timer value</td>
</tr>
<tr>
<td>max_age</td>
<td>Max Age timer value</td>
</tr>
<tr>
<td>hello_time</td>
<td>Hello Time timer value</td>
</tr>
<tr>
<td>forward_delay</td>
<td>Forward Delay timer value</td>
</tr>
</tbody>
</table>

**classmethod parser** (`buf`)  
Decide 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** `ryu.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

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

**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 ryu.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.

```python
class ryu.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**

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

```python
class ryu.lib.packet.cfm.cfm(op=None)

CFM (Connectivity Fault Management) Protocol header class.


OpCode Field range assignments

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

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.

```python
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>op</td>
<td>CFM PDU</td>
</tr>
</tbody>
</table>
```

```python
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 ryu.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 `ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>Length of Value field. (0 means automatically-calculate when encoding)</td>
</tr>
<tr>
<td>data_value</td>
<td>Bit pattern of any of n octets. (n = length)</td>
</tr>
</tbody>
</table>

**class ryu.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 `ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>Length of Value field. (0 means automatically-calculate when encoding.)</td>
</tr>
<tr>
<td>interface_status</td>
<td>Interface Status. The default is 1 (isUp)</td>
</tr>
</tbody>
</table>

**class ryu.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 `ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>md_lv</td>
<td>Maintenance Domain Level.</td>
</tr>
<tr>
<td>version</td>
<td>The protocol version number.</td>
</tr>
<tr>
<td>use_fdb_only</td>
<td>UseFDBonly bit.</td>
</tr>
<tr>
<td>transaction_id</td>
<td>LTM Transaction Identifier.</td>
</tr>
<tr>
<td>ttl</td>
<td>LTM TTL.</td>
</tr>
<tr>
<td>ltm_orig_addr</td>
<td>Original MAC Address.</td>
</tr>
<tr>
<td>ltm_targ_addr</td>
<td>Target MAC Address.</td>
</tr>
<tr>
<td>tlvs</td>
<td>TLVs.</td>
</tr>
</tbody>
</table>
class ryu.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 ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>The protocol version number.</td>
</tr>
<tr>
<td>use_fdb_only</td>
<td>UseFDBonly bit.</td>
</tr>
<tr>
<td>fwd_yes</td>
<td>FwdYes bit.</td>
</tr>
<tr>
<td>terminal_mep</td>
<td>TerminalMep bit.</td>
</tr>
<tr>
<td>transaction_id</td>
<td>LTR Transaction Identifier.</td>
</tr>
<tr>
<td>ttl</td>
<td>Reply TTL.</td>
</tr>
<tr>
<td>relay_action</td>
<td>Relay Action. The default is 1 (RlyHit)</td>
</tr>
<tr>
<td>tlvs</td>
<td>TLVs.</td>
</tr>
</tbody>
</table>

class ryu.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 ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>md_lv</td>
<td>Maintenance Domain Level.</td>
</tr>
<tr>
<td>version</td>
<td>The protocol version number.</td>
</tr>
<tr>
<td>transaction_id</td>
<td>Loopback Transaction Identifier.</td>
</tr>
<tr>
<td>tlvs</td>
<td>TLVs.</td>
</tr>
</tbody>
</table>

class ryu.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 ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>md_lv</td>
<td>Maintenance Domain Level.</td>
</tr>
<tr>
<td>version</td>
<td>The protocol version number.</td>
</tr>
<tr>
<td>transaction_id</td>
<td>Loopback Transaction Identifier.</td>
</tr>
<tr>
<td>tlvs</td>
<td>TLVs.</td>
</tr>
</tbody>
</table>

class ryu.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 ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>Length of Value field. (0 means automatically-calculate when encoding.)</td>
</tr>
<tr>
<td>egress_id_ui</td>
<td>Egress Identifier of Unique ID.</td>
</tr>
<tr>
<td>egress_id_mac</td>
<td>Egress Identifier of MAC address.</td>
</tr>
</tbody>
</table>

```python
class ryu.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 ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>Length of Value field. (0 means automatically-calculate when encoding.)</td>
</tr>
<tr>
<td>last_egress_id_ui</td>
<td>Last Egress Identifier of Unique ID.</td>
</tr>
<tr>
<td>last_egress_id_mac</td>
<td>Last Egress Identifier of MAC address.</td>
</tr>
<tr>
<td>next_egress_id_ui</td>
<td>Next Egress Identifier of Unique ID.</td>
</tr>
<tr>
<td>next_egress_id_mac</td>
<td>Next Egress Identifier of MAC address.</td>
</tr>
</tbody>
</table>

```python
class ryu.lib.packet.cfm.organization_specific_tlv(length=0, oui=b'x00x00x00',
    subtype=0, value=b")
```

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

This is used with ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>Length of Value field. (0 means automatically-calculate when encoding.)</td>
</tr>
<tr>
<td>oui</td>
<td>Organizationally Unique Identifier.</td>
</tr>
<tr>
<td>subtype</td>
<td>Subtype.</td>
</tr>
<tr>
<td>value</td>
<td>Value.(optional)</td>
</tr>
</tbody>
</table>

```python
class ryu.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 ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>Length of Value field. (0 means automatically-calculate when encoding.)</td>
</tr>
<tr>
<td>port_status</td>
<td>Port Status. The default is 1 (psUp)</td>
</tr>
</tbody>
</table>
class ryu.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 ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>Length of Value field. (0 means automatically-calculate when encoding.)</td>
</tr>
<tr>
<td>action</td>
<td>Egress Action. The default is 1 (EgrOK)</td>
</tr>
<tr>
<td>mac_address</td>
<td>Egress MAC Address.</td>
</tr>
<tr>
<td>port_id_length</td>
<td>Egress PortID Length. (0 means automatically-calculate when encoding.)</td>
</tr>
<tr>
<td>port_id_subtype</td>
<td>Egress PortID Subtype.</td>
</tr>
<tr>
<td>port_id</td>
<td>Egress PortID.</td>
</tr>
</tbody>
</table>

class ryu.lib.packet.cfm.reply_ingress_tlv(length=0, action=1, 
    mac_address='00:00:00:00:00:00:00', 
    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 ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>Length of Value field. (0 means automatically-calculate when encoding.)</td>
</tr>
<tr>
<td>action</td>
<td>Ingress Action. The default is 1 (IngOK)</td>
</tr>
<tr>
<td>mac_address</td>
<td>Ingress MAC Address.</td>
</tr>
<tr>
<td>port_id_length</td>
<td>Ingress PortID Length. (0 means automatically-calculate when encoding.)</td>
</tr>
<tr>
<td>port_id_subtype</td>
<td>Ingress PortID Subtype.</td>
</tr>
<tr>
<td>port_id</td>
<td>Ingress PortID.</td>
</tr>
</tbody>
</table>

class ryu.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 ryu.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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>Length of Value field. (0 means automatically-calculate when encoding.)</td>
</tr>
<tr>
<td>chassis_id_length</td>
<td>Chassis ID Length. (0 means automatically-calculate when encoding.)</td>
</tr>
<tr>
<td>chassis_id_subtype</td>
<td>Chassis ID Subtype. The default is 4 (Mac Address)</td>
</tr>
<tr>
<td>chassis_id</td>
<td>Chassis ID.</td>
</tr>
<tr>
<td>ma_domain_length</td>
<td>Management Address Domain Length. (0 means automatically-calculate when encoding.)</td>
</tr>
<tr>
<td>ma_domain</td>
<td>Management Address Domain.</td>
</tr>
<tr>
<td>ma_length</td>
<td>Management Address Length. (0 means automatically-calculate when encoding.)</td>
</tr>
<tr>
<td>ma</td>
<td>Management Address.</td>
</tr>
</tbody>
</table>

### DHCP

DHCP packet parser/serializer

```python
class ryu.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 looks 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

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

**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.

```python
class ryu.lib.packet.dhcp.option(tag, value, length=0)
```

DHCP (RFC 2132) options encoder/decoder class.

This is used with ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tag</td>
<td>Option type. (except for the 'magic cookie', 'pad option' and 'end option'.)</td>
</tr>
<tr>
<td>value</td>
<td>Option’s value. (set the value that has been converted to hexadecimal.)</td>
</tr>
<tr>
<td>length</td>
<td>Option’s value length. (calculated automatically from the length of value.)</td>
</tr>
</tbody>
</table>

```python
class ryu.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 ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>option_list</td>
<td>‘end option’ and ‘pad option’ are added automatically after the option class is stored in array.</td>
</tr>
<tr>
<td>options_len</td>
<td>Option’s byte length. (‘magic cookie’, ‘end option’ and ‘pad option’ length including.)</td>
</tr>
<tr>
<td>magic_cookie</td>
<td>The first four octets contain the decimal values 99, 130, 83 and 99.</td>
</tr>
</tbody>
</table>

**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:

```
+-----------------------------+
| msg_type | hop_count | |
+-----------------------------+
|                     |             |
+-----------------------------+
| link_address |
+-----------------------------+
|                     |             |
+-----------------------------+
| peer_address |
+-----------------------------+
|                     |             |
+-----------------------------+
| options (variable number and length) .... | |
+-----------------------------+
```

```
class ryu.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 looks 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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>msg_type</td>
<td>Identifies the DHCP message type</td>
</tr>
<tr>
<td>transaction_id</td>
<td>For unrelayed messages only: the transaction ID for this message exchange.</td>
</tr>
<tr>
<td>hop_count</td>
<td>For relayed messages only: number of relay agents that have relayed this message.</td>
</tr>
<tr>
<td>link_address</td>
<td>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.</td>
</tr>
<tr>
<td>peer_address</td>
<td>For relayed messages only: the address of the client or relay agent from which the message to be relayed was received.</td>
</tr>
<tr>
<td>options</td>
<td>Options carried in this message</td>
</tr>
</tbody>
</table>

```
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.

```python
serialize(payload=None, prev=None)
```

Encode a protocol header.

This is used only when encoding a packet.

```python
encode(protocol_header)
```

This method is used only when encoding a packet.

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:

```
+-------------------------------+-------------------------------+-------------------------------+
| option-code | option-len | option-data | (option-len octets) |
+-------------------------------+-------------------------------+-------------------------------+
```

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

```python
class ryu.lib.packet.dhcp6.options(option_list=None, options_len=0)
```

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

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.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>dst</td>
<td>destination address</td>
<td>‘ff:ff:ff:ff:ff’</td>
</tr>
<tr>
<td>src</td>
<td>source address</td>
<td>‘08:60:6e:7f:74:e7’</td>
</tr>
<tr>
<td>ethertype</td>
<td>ether type</td>
<td>0x0800</td>
</tr>
</tbody>
</table>

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` `ryu.lib.packet.geneve.Option`(option_class=None, type_=None, length=0)  
Tunnel Options  
`class` `ryu.lib.packet.geneve.OptionDataUnknown`(buf, option_class=None, type_=None, length=0)  
Unknown Option Class and Type specific Option  
`class` `ryu.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.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>Version.</td>
</tr>
<tr>
<td>opt_len</td>
<td>The length of the options fields.</td>
</tr>
<tr>
<td>flags</td>
<td>Flag field for OAM packet and Critical options present.</td>
</tr>
<tr>
<td>protocol</td>
<td>Protocol Type field. The Protocol Type is defined as &quot;ETHER TYPES&quot;.</td>
</tr>
<tr>
<td>vni</td>
<td>Identifier for unique element of virtual network.</td>
</tr>
<tr>
<td>options</td>
<td>List of Option* instance.</td>
</tr>
</tbody>
</table>

**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 ryu.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.

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

**ryu.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** *ryu.lib.packet.icmp.TimeExceeded*(*data_len=0, data=None*)

ICMP sub encoder/decoder class for Time Exceeded Message.

This is used with ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data_len</td>
<td>data length</td>
</tr>
<tr>
<td>data</td>
<td>Internet Header + leading octets of original datagram</td>
</tr>
</tbody>
</table>

**class** *ryu.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 ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data_len</td>
<td>data length</td>
</tr>
<tr>
<td>mtu</td>
<td>Next-Hop MTU</td>
</tr>
</tbody>
</table>

NOTE: This field is required when icmp code is 4
code 4 = fragmentation needed and DF set

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>Internet Header + leading octets of original datagram</td>
</tr>
</tbody>
</table>

```python
class ryu.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 ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Identifier</td>
</tr>
<tr>
<td>seq</td>
<td>Sequence Number</td>
</tr>
<tr>
<td>data</td>
<td>Internet Header + 64 bits of Original Data Datagram</td>
</tr>
</tbody>
</table>

```python
class ryu.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.

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

```python
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.

```python
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 ryu.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 ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Identifier</td>
</tr>
<tr>
<td>seq</td>
<td>Sequence Number</td>
</tr>
<tr>
<td>data</td>
<td>Data</td>
</tr>
</tbody>
</table>

class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type_</td>
<td>Type</td>
</tr>
<tr>
<td>code</td>
<td>Code</td>
</tr>
<tr>
<td>csum</td>
<td>CheckSum (0 means automatically-calculate when encoding)</td>
</tr>
</tbody>
</table>

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.

```python
class ryu.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 ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>maxresp</td>
<td>max response time in millisecond. it is meaningful only in Query Message.</td>
</tr>
<tr>
<td>address</td>
<td>a group address value.</td>
</tr>
</tbody>
</table>

```python
class ryu.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 ryu.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.

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

```python
class ryu.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 ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>record_num</td>
<td>a number of the group records.</td>
</tr>
<tr>
<td>records</td>
<td>a list of ryu.lib.packet.icmpv6.mldv2_report_group. None if no records.</td>
</tr>
</tbody>
</table>

```python
class ryu.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 ryu.lib.packet.icmpv6.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type_</td>
<td>a group record type for v3.</td>
</tr>
<tr>
<td>aux_len</td>
<td>the length of the auxiliary data in 32-bit words.</td>
</tr>
<tr>
<td>num</td>
<td>a number of the multicast servers.</td>
</tr>
<tr>
<td>address</td>
<td>a group address value.</td>
</tr>
<tr>
<td>srcs</td>
<td>a list of IPv6 addresses of the multicast servers.</td>
</tr>
<tr>
<td>aux</td>
<td>the auxiliary data.</td>
</tr>
</tbody>
</table>

**class** ryu.lib.packet.icmpv6.nd_neighbor

<table>
<thead>
<tr>
<th>res=0, dst='::', option=None</th>
</tr>
</thead>
</table>

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

This is used with ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>R,S,O Flags for Neighbor Advertisement. The 3 MSBs of &quot;Reserved&quot; field for Neighbor Solicitation.</td>
</tr>
<tr>
<td>dst</td>
<td>Target Address</td>
</tr>
<tr>
<td>option</td>
<td>a derived object of ryu.lib.packet.icmpv6.nd_option or a bytearray. None if no options.</td>
</tr>
</tbody>
</table>

**class** ryu.lib.packet.icmpv6.nd_option_mtu

<table>
<thead>
<tr>
<th>mtu=1500</th>
</tr>
</thead>
</table>

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

This is used with ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mtu</td>
<td>MTU.</td>
</tr>
</tbody>
</table>

**class** ryu.lib.packet.icmpv6.nd_option_pi

<table>
<thead>
<tr>
<th>length=0, pl=0, res1=0, val_l=0, pre_l=0, res2=0, prefix='::'</th>
</tr>
</thead>
</table>

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

This is used with ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>length of the option. (0 means automatically-calculate when encoding)</td>
</tr>
<tr>
<td>pl</td>
<td>Prefix Length.</td>
</tr>
<tr>
<td>res1</td>
<td>L,A,R* Flags for Prefix Information.</td>
</tr>
<tr>
<td>val_l</td>
<td>Valid Lifetime.</td>
</tr>
<tr>
<td>pre_l</td>
<td>Preferred Lifetime.</td>
</tr>
<tr>
<td>res2</td>
<td>This field is unused. It MUST be initialized to zero.</td>
</tr>
<tr>
<td>prefix</td>
<td>An IP address or a prefix of an IP address.</td>
</tr>
</tbody>
</table>
*R flag is defined in (RFC 3775)

```python
class ryu.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 `ryu.lib.packet.icmpv6.nd_neighbor`, `ryu.lib.packet.icmpv6.nd_router_solicit` or `ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>length of the option. (0 means automatically-calculate when encoding)</td>
</tr>
<tr>
<td>hw_src</td>
<td>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.</td>
</tr>
<tr>
<td>data</td>
<td>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.</td>
</tr>
</tbody>
</table>

```python
class ryu.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 `ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>length of the option. (0 means automatically-calculate when encoding)</td>
</tr>
<tr>
<td>hw_src</td>
<td>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.</td>
</tr>
<tr>
<td>data</td>
<td>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.</td>
</tr>
</tbody>
</table>

```python
class ryu.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 `ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ch_l</td>
<td>Cur Hop Limit.</td>
</tr>
<tr>
<td>res</td>
<td>M,O Flags for Router Advertisement.</td>
</tr>
<tr>
<td>rou_l</td>
<td>Router Lifetime.</td>
</tr>
<tr>
<td>rea_t</td>
<td>Reachable Time.</td>
</tr>
<tr>
<td>ret_t</td>
<td>Retrans Timer.</td>
</tr>
<tr>
<td>options</td>
<td>List of a derived object of <code>ryu.lib.packet.icmpv6.nd_option</code> or a bytearray. None if no options.</td>
</tr>
</tbody>
</table>
**class** ryu.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 ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>This field is unused. It MUST be initialized to zero.</td>
</tr>
<tr>
<td>option</td>
<td>a derived object of ryu.lib.packet.icmpv6.nd_option or a bytearray. None if no options.</td>
</tr>
</tbody>
</table>

**IGMP**

Internet Group Management Protocol (IGMP) packet parser/serializer

[RFC 1112] IGMP v1 format:

```
+---------------------------------------------+  +---------------------------------------------+
<table>
<thead>
<tr>
<th>Version</th>
<th>Type</th>
<th>Unused</th>
<th>Checksum</th>
</tr>
</thead>
</table>
+---------------------------------------------+  +---------------------------------------------+
| Group Address |
+---------------------------------------------+
```

[RFC 2236] IGMP v2 format:

```
+---------------------------------------------+  +---------------------------------------------+
<table>
<thead>
<tr>
<th>Type</th>
<th>Max Resp Time</th>
<th>Checksum</th>
</tr>
</thead>
</table>
+---------------------------------------------+  +---------------------------------------------+
| Group Address |
+---------------------------------------------+
```

[RFC 3376] IGMP v3 Membership Query format:

```
+---------------------------------------------+  +---------------------------------------------+
<table>
<thead>
<tr>
<th>Type = 0x11</th>
<th>Max Resp Code</th>
<th>Checksum</th>
</tr>
</thead>
</table>
+---------------------------------------------+  +---------------------------------------------+
| Group Address |
+---------------------------------------------+

<table>
<thead>
<tr>
<th>Resv</th>
<th>S</th>
<th>QRV</th>
<th>QQIC</th>
<th>Number of Sources (N)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source Address [1]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>Source Address [2]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>Source Address [3]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>Source Address [N]</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
+---------------------------------------------+  +---------------------------------------------+
IGMP v3 Membership Report format:

```
+-----------------+-----------------------+-----------------+
| Type = 0x22     | Reserved              | Checksum        |
+-----------------+-----------------------+-----------------+
| Reserved        | Number of Group Records (M) |
+-----------------+-----------------------+-----------------+
```

Where each Group Record has the following internal format:

```
+-----------------+-----------------+-----------------+
| Record Type     | Aux Data Len    | Number of Sources (N) |
+-----------------+-----------------+-----------------+
| Multicast Address |
+-----------------+-----------------+-----------------+
```

```
+-----------------+-----------------+-----------------+
| Source Address [1] |
+-----------------+-----------------+-----------------+
| Source Address [2] |
+-----------------+-----------------+-----------------+
```

```
+-----------------+-----------------+-----------------+
| Source Address [N] |
+-----------------+-----------------+-----------------+
```

```
+-----------------+-----------------+
| Auxiliary Data  |
+-----------------+-----------------+
```

```python
class ryu.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.
```
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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>msgtype</td>
<td>a message type for v2, or a combination of version and a message type for v1.</td>
</tr>
<tr>
<td>maxresp</td>
<td>max response time in unit of 1/10 second. It is meaningful only in Query Message.</td>
</tr>
<tr>
<td>csum</td>
<td>a check sum value. 0 means automatically-calculate when encoding.</td>
</tr>
<tr>
<td>address</td>
<td>a group address value.</td>
</tr>
</tbody>
</table>

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 ryu.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.

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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>msgtype</td>
<td>a message type for v3.</td>
</tr>
<tr>
<td>maxresp</td>
<td>max response time in unit of 1/10 second.</td>
</tr>
<tr>
<td>csum</td>
<td>a check sum value. 0 means automatically-calculate when encoding.</td>
</tr>
<tr>
<td>address</td>
<td>a group address value.</td>
</tr>
<tr>
<td>s_flg</td>
<td>when set to 1, routers suppress the timer process.</td>
</tr>
<tr>
<td>qrv</td>
<td>robustness variable for a querier.</td>
</tr>
<tr>
<td>qqic</td>
<td>an interval time for a querier in unit of seconds.</td>
</tr>
<tr>
<td>num</td>
<td>a number of the multicast servers.</td>
</tr>
<tr>
<td>srcs</td>
<td>a list of IPv4 addresses of the multicast servers.</td>
</tr>
</tbody>
</table>
class method 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 ryu.lib.packet.igmp.igmpv3_report (msgtype=34, csum=0, record_num=0, records=None)


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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>msgtype</td>
<td>a message type for v3.</td>
</tr>
<tr>
<td>csum</td>
<td>a check sum value. 0 means automatically-calculate when encoding.</td>
</tr>
<tr>
<td>record_num</td>
<td>a number of the group records.</td>
</tr>
<tr>
<td>records</td>
<td>a list of ryu.lib.packet.igmp.igmpv3_report_group. None if no records.</td>
</tr>
</tbody>
</table>

class method 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.

```python
class ryu.lib.packet.igmp.igmpv3_report_group (type_=0, aux_len=0, num=0, address='0.0.0.0', srcs=None, aux=None)
```


http://www.ietf.org/rfc/rfc3376.txt

This is used with ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>type_</td>
<td>a group record type for v3.</td>
<td></td>
</tr>
<tr>
<td>aux_len</td>
<td>the length of the auxiliary data.</td>
<td></td>
</tr>
<tr>
<td>num</td>
<td>a number of the multicast servers.</td>
<td></td>
</tr>
<tr>
<td>address</td>
<td>a group address value.</td>
<td></td>
</tr>
<tr>
<td>srcs</td>
<td>a list of IPv4 addresses of the multicast servers.</td>
<td></td>
</tr>
<tr>
<td>aux</td>
<td>the auxiliary data.</td>
<td></td>
</tr>
</tbody>
</table>

IPv4

```python
class ryu.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’s 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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>Version</td>
<td></td>
</tr>
<tr>
<td>header_length</td>
<td>IHL</td>
<td></td>
</tr>
<tr>
<td>tos</td>
<td>Type of Service</td>
<td></td>
</tr>
<tr>
<td>total_length</td>
<td>Total Length (0 means automatically-calculate when encoding)</td>
<td></td>
</tr>
<tr>
<td>identification</td>
<td>Identification</td>
<td></td>
</tr>
<tr>
<td>flags</td>
<td>Flags</td>
<td></td>
</tr>
<tr>
<td>offset</td>
<td>Fragment Offset</td>
<td></td>
</tr>
<tr>
<td>ttl</td>
<td>Time to Live</td>
<td></td>
</tr>
<tr>
<td>proto</td>
<td>Protocol</td>
<td></td>
</tr>
<tr>
<td>csum</td>
<td>Header Checksum (Ignored and automatically-calculated when encoding)</td>
<td></td>
</tr>
<tr>
<td>src</td>
<td>Source Address</td>
<td>’192.0.2.1’</td>
</tr>
<tr>
<td>dst</td>
<td>Destination Address</td>
<td>’192.0.2.2’</td>
</tr>
<tr>
<td>option</td>
<td>A bytearray which contains the entire Options, or None for no Options</td>
<td></td>
</tr>
</tbody>
</table>

```python
classmethod parser (buf)
```

Decide a protocol header.
This method is used only when decoding a packet.

Decode a protocol header at offset 0 in bytearray \( \text{buf} \). 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.

```python
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** `ryu.lib.packet.ipv6.auth (nxt=6, size=2, spi=0, seq=0, data=b'x00x00x00x00')`

IP Authentication header (RFC 2402) encoder/decoder class.

This is used with ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nxt</td>
<td>Next Header</td>
</tr>
<tr>
<td>size</td>
<td>the length of the Authentication Header in 64-bit words, subtracting 1.</td>
</tr>
<tr>
<td>spi</td>
<td>security parameters index.</td>
</tr>
<tr>
<td>seq</td>
<td>sequence number.</td>
</tr>
<tr>
<td>data</td>
<td>authentication data.</td>
</tr>
</tbody>
</table>

**class** `ryu.lib.packet.ipv6.dst_opt (nxt=6, size=0, data=None)`

IPv6 (RFC 2460) destination header encoder/decoder class.

This is used with ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nxt</td>
<td>Next Header</td>
</tr>
<tr>
<td>size</td>
<td>the length of the destination header, not include the first 8 octet.</td>
</tr>
<tr>
<td>data</td>
<td>IPv6 options.</td>
</tr>
</tbody>
</table>

**class** `ryu.lib.packet.ipv6.fragment (nxt=6, offset=0, more=0, id_=0)`

IPv6 (RFC 2460) fragment header encoder/decoder class.

This is used with ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nxt</td>
<td>Next Header</td>
</tr>
<tr>
<td>offset</td>
<td>offset, in 8-octet units, relative to the start of the fragmentable part of the original packet.</td>
</tr>
<tr>
<td>more</td>
<td>1 means more fragments follow; 0 means last fragment.</td>
</tr>
<tr>
<td>id_</td>
<td>packet identification value.</td>
</tr>
</tbody>
</table>

```python
class ryu.lib.packet.ipv6.header(nxt)
```
extension header abstract class.

```python
class ryu.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 ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nxt</td>
<td>Next Header</td>
</tr>
<tr>
<td>size</td>
<td>the length of the Hop-by-Hop Options header, not include the first 8 octet.</td>
</tr>
<tr>
<td>data</td>
<td>IPv6 options.</td>
</tr>
</tbody>
</table>

```python
class ryu.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_hdr=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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>Version</td>
<td></td>
</tr>
<tr>
<td>traffic_class</td>
<td>Traffic Class</td>
<td></td>
</tr>
<tr>
<td>flow_label</td>
<td>When decoding, Flow Label. When encoding, the most significant 8 bits of Flow Label.</td>
<td></td>
</tr>
<tr>
<td>payload_length</td>
<td>Payload Length</td>
<td></td>
</tr>
<tr>
<td>nxt</td>
<td>Next Header</td>
<td></td>
</tr>
<tr>
<td>hop_limit</td>
<td>Hop Limit</td>
<td></td>
</tr>
<tr>
<td>src</td>
<td>Source Address</td>
<td>'ff02::1'</td>
</tr>
<tr>
<td>dst</td>
<td>Destination Address</td>
<td>'::'</td>
</tr>
<tr>
<td>ext_hdr</td>
<td>Extension Headers</td>
<td></td>
</tr>
</tbody>
</table>

```python
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.

```python
def 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 ryu.lib.packet.ipv6.opt_header(nxt, size, data)
    # an abstract class for Hop-by-Hop Options header and destination header.

class ryu.lib.packet.ipv6.option(type_=0, len_-1, data=None)
    # IPv6 (RFC 2460) Options header encoder/decoder class.
    # This is used with `ryu.lib.packet.ipv6.hop_opts` or `ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type_</td>
<td>option type.</td>
</tr>
<tr>
<td>len_-</td>
<td>the length of data. -1 if type_ is 0.</td>
</tr>
<tr>
<td>data</td>
<td>an option value. None if len_ is 0 or -1.</td>
</tr>
</tbody>
</table>

class ryu.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

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Source Route (DEPRECATED)</td>
<td>[IPv6][RFC5095]</td>
</tr>
<tr>
<td>1</td>
<td>Nimrod (DEPRECATED 2009-05-06)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Type 2 Routing Header</td>
<td>[RFC6275]</td>
</tr>
<tr>
<td>3</td>
<td>RPL Source Route Header</td>
<td>[RFC6554]</td>
</tr>
<tr>
<td>4 - 252</td>
<td>Unassigned</td>
<td></td>
</tr>
<tr>
<td>253</td>
<td>RFC3692-style Experiment 1 [2]</td>
<td>[RFC4727]</td>
</tr>
<tr>
<td>254</td>
<td>RFC3692-style Experiment 2 [2]</td>
<td>[RFC4727]</td>
</tr>
<tr>
<td>255</td>
<td>Reserved</td>
<td></td>
</tr>
</tbody>
</table>

class ryu.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 `ryu.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.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nxt</td>
<td>Next Header</td>
</tr>
<tr>
<td>size</td>
<td>The length of the Routing header, not include the first 8 octet. (0 means automatically-calculate when encoding)</td>
</tr>
<tr>
<td>type</td>
<td>Identifies the particular Routing header variant.</td>
</tr>
<tr>
<td>seg</td>
<td>Number of route segments remaining.</td>
</tr>
<tr>
<td>cmpi</td>
<td>Number of prefix octets from segments 1 through n-1.</td>
</tr>
<tr>
<td>cmpe</td>
<td>Number of prefix octets from segment n.</td>
</tr>
<tr>
<td>pad</td>
<td>Number of octets that are used for padding after Address[n] at the end of the SRH.</td>
</tr>
<tr>
<td>adrs</td>
<td>Vector of addresses, numbered 1 to n.</td>
</tr>
</tbody>
</table>

**LLC**


LLC format:

```
+-----------------+--------------+
| DSAP address   | 8 bits      |
+-----------------+--------------+
| SSAP address   | 8 bits      |
+-----------------+--------------+
| Control        | 8 or 16 bits|
+-----------------+--------------+
```

DSAP address field:

```
+-----+---+---+---+---+---+---+---+
| 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:

```
+-----+---+---+---+---+---+---+---+
| 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):

```
+---+---+---+---+---+---+---+---+-----+------+
| 0 | N(S) | P/F | N(R) |
+---+---+---+---+---+---+---+---+-----+------+
```

Supervisory commands/responses (S-format PDUs):

**2.4. Library**
Unnumbered commands/responses (U-format PDUs):

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>( M_1 )</th>
<th>( M_1 )</th>
<th>( P/F )</th>
<th>( M_2 )</th>
<th>( M_2 )</th>
<th>( M_2 )</th>
</tr>
</thead>
</table>

N(S) : sender send sequence number (Bit 2=lower-order-bit)
N(R) : sender receive sequence number (Bit 10=lower-order-bit)
S : supervisory function bit
M1/M2: modifier function bit
P/F : poll bit - command LLC PDUs
      final bit - response LLC PDUs

**class** `ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>send_sequence_number</td>
<td>sender send sequence number</td>
</tr>
<tr>
<td>pf_bit</td>
<td>poll/final bit</td>
</tr>
<tr>
<td>receive_sequence_number</td>
<td>sender receive sequence number</td>
</tr>
</tbody>
</table>

**class** `ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>supervisory_function</td>
<td>supervisory function bit</td>
</tr>
<tr>
<td>pf_bit</td>
<td>poll/final bit</td>
</tr>
<tr>
<td>receive_sequence_number</td>
<td>sender receive sequence number</td>
</tr>
</tbody>
</table>

**class** `ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>modifier_function1</td>
<td>modifier function bit</td>
</tr>
<tr>
<td>pf_bit</td>
<td>poll/final bit</td>
</tr>
<tr>
<td>modifier_function2</td>
<td>modifier function bit</td>
</tr>
</tbody>
</table>
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dsap_addr</td>
<td>Destination service access point address field includes I/G bit at least significant bit.</td>
</tr>
<tr>
<td>ssap_addr</td>
<td>Source service access point address field includes C/R bit at least significant bit.</td>
</tr>
<tr>
<td>control</td>
<td>Control field [16 bits for formats that include sequence numbering, and 8 bits for formats that do not]. Either ryu.lib.packet.llc.ControlFormatI or ryu.lib.packet.llc.ControlFormatS or ryu.lib.packet.llc.ControlFormatU object.</td>
</tr>
</tbody>
</table>

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


basic TLV format:

<table>
<thead>
<tr>
<th>octets</th>
<th>1</th>
<th>2</th>
<th>3 ...</th>
<th>n + 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>TLV type</td>
<td>TLV information string</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(7bits)</td>
<td>string length</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(9bits)</td>
</tr>
</tbody>
</table>

Organizationally specific TLV format:

<table>
<thead>
<tr>
<th>octets</th>
<th>1</th>
<th>2</th>
<th>3 ...</th>
<th>5</th>
<th>6</th>
<th>7 ...</th>
<th>n + 6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>TLV type</td>
<td>Length</td>
<td>OUI</td>
<td>Subtype</td>
<td>Information</td>
</tr>
</tbody>
</table>
LLDPDU format:

<table>
<thead>
<tr>
<th>Chassis ID</th>
<th>Port ID</th>
<th>TTL</th>
<th>optional TLV</th>
<th>...</th>
<th>optional TLV</th>
<th>End</th>
</tr>
</thead>
</table>

Chasis ID, Port ID, TTL, End are mandatory optional TLV may be inserted in any order

class ryu.lib.packet.lldp.ChassisID(buf=None, *args, **kwargs)
Chassis ID TLV encoder/decoder class

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>buf</td>
<td>Binary data to parse.</td>
</tr>
<tr>
<td>subtype</td>
<td>Subtype.</td>
</tr>
<tr>
<td>chassis_id</td>
<td>Chassis id corresponding to subtype.</td>
</tr>
</tbody>
</table>

class ryu.lib.packet.lldp.End(buf=None, *args, **kwargs)
End TLV encoder/decoder class

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>buf</td>
<td>Binary data to parse.</td>
</tr>
</tbody>
</table>

class ryu.lib.packet.lldp.ManagementAddress(buf=None, *args, **kwargs)
Management Address TLV encoder/decoder class

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>buf</td>
<td>Binary data to parse.</td>
</tr>
<tr>
<td>addr_subtype</td>
<td>Address type.</td>
</tr>
<tr>
<td>addr</td>
<td>Device address.</td>
</tr>
<tr>
<td>intf_subtype</td>
<td>Interface type.</td>
</tr>
<tr>
<td>intf_num</td>
<td>Interface number.</td>
</tr>
<tr>
<td>oid</td>
<td>Object ID.</td>
</tr>
</tbody>
</table>

class ryu.lib.packet.lldp.OrganizationallySpecific(buf=None, *args, **kwargs)
Organizationally Specific TLV encoder/decoder class

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>buf</td>
<td>Binary data to parse.</td>
</tr>
<tr>
<td>oui</td>
<td>Organizationally unique ID.</td>
</tr>
<tr>
<td>subtype</td>
<td>Organizationally defined subtype.</td>
</tr>
<tr>
<td>info</td>
<td>Organizationally defined information string.</td>
</tr>
</tbody>
</table>

class ryu.lib.packet.lldp.PortDescription(buf=None, *args, **kwargs)
Port description TLV encoder/decoder class

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>buf</td>
<td>Binary data to parse.</td>
</tr>
<tr>
<td>port_description</td>
<td>Port description.</td>
</tr>
</tbody>
</table>
class ryu.lib.packet.lldp.PortID(buf=None, *args, **kwargs)
    Port ID TLV encoder/decoder class

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>buf</td>
<td>Binary data to parse.</td>
</tr>
<tr>
<td>subtype</td>
<td>Subtype.</td>
</tr>
<tr>
<td>port_id</td>
<td>Port ID corresponding to subtype.</td>
</tr>
</tbody>
</table>

class ryu.lib.packet.lldp.SystemCapabilities(buf=None, *args, **kwargs)
    System Capabilities TLV encoder/decoder class

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>buf</td>
<td>Binary data to parse.</td>
</tr>
<tr>
<td>system_cap</td>
<td>System Capabilities.</td>
</tr>
<tr>
<td>enabled_cap</td>
<td>Enabled Capabilities.</td>
</tr>
</tbody>
</table>

class ryu.lib.packet.lldp.SystemDescription(buf=None, *args, **kwargs)
    System description TLV encoder/decoder class

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>buf</td>
<td>Binary data to parse.</td>
</tr>
<tr>
<td>system_description</td>
<td>System description.</td>
</tr>
</tbody>
</table>

class ryu.lib.packet.lldp.SystemName(buf=None, *args, **kwargs)
    System name TLV encoder/decoder class

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>buf</td>
<td>Binary data to parse.</td>
</tr>
<tr>
<td>system_name</td>
<td>System name.</td>
</tr>
</tbody>
</table>

class ryu.lib.packet.lldp.TTL(buf=None, *args, **kwargs)
    Time To Live TLV encoder/decoder class

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>buf</td>
<td>Binary data to parse.</td>
</tr>
<tr>
<td>ttl</td>
<td>Time To Live.</td>
</tr>
</tbody>
</table>

class ryu.lib.packet.lldp.lldp(tlvs)
    LLDPDU encoder/decoder class.

An instance has the following attributes at least.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tlvs</td>
<td>List of TLV instance.</td>
</tr>
</tbody>
</table>

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**

ryu.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.

ryu.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 ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>label</td>
<td>Label Value</td>
</tr>
<tr>
<td>exp</td>
<td>Experimental Use</td>
</tr>
<tr>
<td>bsb</td>
<td>Bottom of Stack</td>
</tr>
<tr>
<td>ttl</td>
<td>Time To Live</td>
</tr>
</tbody>
</table>

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 `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** `ryu.lib.packet.openflow.OffUnparseableMsg`(*datapath, version, msg_type, msg_len, xid, body*)

Unparseable OpenFlow message encoder class.

An instance has the following attributes at least.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>datapath</td>
<td>A <code>ryu.ofproto.ofproto_protocol.ProtocolDesc</code> instance for this message or <code>None</code> if OpenFlow protocol version is unsupported version.</td>
</tr>
<tr>
<td>version</td>
<td>OpenFlow protocol version</td>
</tr>
<tr>
<td>msg_type</td>
<td>Type of OpenFlow message</td>
</tr>
<tr>
<td>msg_len</td>
<td>Length of the message</td>
</tr>
<tr>
<td>xid</td>
<td>Transaction id</td>
</tr>
<tr>
<td>body</td>
<td>OpenFlow body data</td>
</tr>
</tbody>
</table>

**Note:** "datapath" attribute is different from `ryu.controller.controller.Datapath`. So you cannot use "datapath" attribute to send OpenFlow messages. For example, "datapath" attribute does not have `send_msg` method.

**class** `ryu.lib.packet.openflow.openflow`(*msg*)

OpenFlow message encoder/decoder class.

An instance has the following attributes at least.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>msg</td>
<td>An instance of OpenFlow message (see OpenFlow protocol API Reference) or an instance of <code>OffUnparseableMsg</code> if failed to parse packet as OpenFlow message.</td>
</tr>
</tbody>
</table>

**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.

```python
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

```python
exception ryu.lib.packet.ospf.InvalidChecksum
class ryu.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.

```python
class ryu.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 ryu.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)
Decide 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 ryu.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)
Decide 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 ryu.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)
Decide 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 ryu.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)
Decide 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.

```
ryu.lib.packet.ospf.ospf
alias of ryu.lib.packet.ospf.OSPMessage
```

PBB

```
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pcp</td>
<td>Priority Code Point</td>
</tr>
<tr>
<td>dei</td>
<td>Drop Eligible Indication</td>
</tr>
<tr>
<td>uca</td>
<td>Use Customer Address</td>
</tr>
<tr>
<td>sid</td>
<td>Service Instance ID</td>
</tr>
</tbody>
</table>

```
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

```python
class ryu.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.
    • ryu.lib.packet.sctp.chunk_abort
    • ryu.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                      |
    +----------------+----------------------------------+
    | length          | length of this cause containing this header. (0 means automatically-calculate when encoding) |
```

```python
class ryu.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.
    • ryu.lib.packet.sctp.chunkAbort
    • ryu.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                      |
    +----------------+----------------------------------+
    | length          | length of this cause containing this header. (0 means automatically-calculate when encoding) |
```

```python
class ryu.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.
    • ryu.lib.packet.sctp.chunk_abort
    • ryu.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           | stream id.                       |
    | length          | length of this cause containing this header. (0 means automatically-calculate when encoding) |
```

```python
class ryu.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.
    • ryu.lib.packet.sctp.chunk_abort
```

2.4. Library 87
• ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>types</td>
<td>a list of missing params.</td>
</tr>
<tr>
<td>num</td>
<td>Number of missing params. (0 means automatically-calculate when encoding)</td>
</tr>
<tr>
<td>length</td>
<td>length of this cause containing this header. (0 means automatically-calculate when encoding)</td>
</tr>
</tbody>
</table>

```python
class ryu.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.

• ryu.lib.packet.sctp.chunk_abort

• ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>the TSN of the DATA chunk received with no user data field.</td>
</tr>
<tr>
<td>length</td>
<td>length of this cause containing this header. (0 means automatically-calculate when encoding)</td>
</tr>
</tbody>
</table>

```python
class ryu.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.

• ryu.lib.packet.sctp.chunk_abort

• ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>length of this cause containing this header. (0 means automatically-calculate when encoding)</td>
</tr>
</tbody>
</table>

```python
class ryu.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.

• ryu.lib.packet.sctp.chunk_abort

• ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Additional Information.</td>
</tr>
<tr>
<td>length</td>
<td>length of this cause containing this header. (0 means automatically-calculate when encoding)</td>
</tr>
</tbody>
</table>
class ryu.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.
- ryu.lib.packet.sctp.chunk_abort
- ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>New Address TLVs.</td>
</tr>
<tr>
<td>length</td>
<td>length of this cause containing this header. (0 means automatically-calculate when encoding)</td>
</tr>
</tbody>
</table>

class ryu.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.
- ryu.lib.packet.sctp.chunk_abort
- ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Measure of Staleness.</td>
</tr>
<tr>
<td>length</td>
<td>length of this cause containing this header. (0 means automatically-calculate when encoding)</td>
</tr>
</tbody>
</table>

class ryu.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.
- ryu.lib.packet.sctp.chunk_abort
- ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Unrecognized Chunk.</td>
</tr>
<tr>
<td>length</td>
<td>length of this cause containing this header. (0 means automatically-calculate when encoding)</td>
</tr>
</tbody>
</table>

class ryu.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.
- ryu.lib.packet.sctp.chunk_abort
- ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Unrecognized Parameter.</td>
</tr>
<tr>
<td>length</td>
<td>length of this cause containing this header. (0 means automatically-calculate when encoding)</td>
</tr>
</tbody>
</table>

```python
class ryu.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.
- ryu.lib.packet.sctp.chunk_abort
- ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Unresolvable Address. one of follows: ryu.lib.packet.sctp.param_host_addr, ryu.lib.packet.sctp.param_ipv4, or ryu.lib.packet.sctp.param_ipv6.</td>
</tr>
<tr>
<td>length</td>
<td>length of this cause containing this header. (0 means automatically-calculate when encoding)</td>
</tr>
</tbody>
</table>

```python
class ryu.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.
- ryu.lib.packet.sctp.chunk_abort
- ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Upper Layer Abort Reason.</td>
</tr>
<tr>
<td>length</td>
<td>length of this cause containing this header. (0 means automatically-calculate when encoding)</td>
</tr>
</tbody>
</table>

```python
class ryu.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.
- ryu.lib.packet.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tflag</td>
<td>'0' means the Verification tag is normal. '1' means the Verification tag is copy of the sender.</td>
</tr>
<tr>
<td>length</td>
<td>length of this chunk containing this header. (0 means automatically-calculate when encoding)</td>
</tr>
<tr>
<td>causes</td>
<td>a list of derived classes of ryu.lib.packet.sctp.causes.</td>
</tr>
</tbody>
</table>
class ryu.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.

- ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>set to '0'. this field will be ignored.</td>
</tr>
<tr>
<td>length</td>
<td>length of this chunk containing this header. (0 means automatically-calculate when encoding)</td>
</tr>
</tbody>
</table>

class ryu.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.

- ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>set to '0'. this field will be ignored.</td>
</tr>
<tr>
<td>length</td>
<td>length of this chunk containing this header. (0 means automatically-calculate when encoding)</td>
</tr>
<tr>
<td>cookie</td>
<td>cookie data.</td>
</tr>
</tbody>
</table>

class ryu.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.

- ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>set to '0'. this field will be ignored.</td>
</tr>
<tr>
<td>length</td>
<td>length of this chunk containing this header. (0 means automatically-calculate when encoding)</td>
</tr>
<tr>
<td>low_tsn</td>
<td>the lowest TSN.</td>
</tr>
</tbody>
</table>

class ryu.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.

- ryu.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.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unordered</td>
<td>if set to '1', the receiver ignores the sequence number.</td>
</tr>
<tr>
<td>begin</td>
<td>if set to '1', this chunk is the first fragment.</td>
</tr>
<tr>
<td>end</td>
<td>if set to '1', this chunk is the last fragment.</td>
</tr>
<tr>
<td>length</td>
<td>length of this chunk containing this header. (0 means automatically-calculate when encoding)</td>
</tr>
<tr>
<td>tsn</td>
<td>Transmission Sequence Number.</td>
</tr>
<tr>
<td>sid</td>
<td>stream id.</td>
</tr>
<tr>
<td>seq</td>
<td>the sequence number.</td>
</tr>
<tr>
<td>payload_id</td>
<td>application specified protocol id. '0' means that no application id is identified.</td>
</tr>
<tr>
<td>payload_data</td>
<td>user data.</td>
</tr>
</tbody>
</table>

**class** `ryu.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.

- `ryu.lib.packet.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>set to '0'. this field will be ignored.</td>
</tr>
<tr>
<td>length</td>
<td>length of this chunk containing this header. (0 means automatically-calculate when encoding)</td>
</tr>
<tr>
<td>low_tsn</td>
<td>the lowest TSN.</td>
</tr>
</tbody>
</table>

**class** `ryu.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.

- `ryu.lib.packet.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>set to '0'. this field will be ignored.</td>
</tr>
<tr>
<td>length</td>
<td>length of this chunk containing this header. (0 means automatically-calculate when encoding)</td>
</tr>
<tr>
<td>causes</td>
<td>a list of derived classes of ryu.lib.packet.sctp.causes.</td>
</tr>
</tbody>
</table>

**class** `ryu.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.

- `ryu.lib.packet.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.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>set to '0'. this field will be ignored.</td>
</tr>
<tr>
<td>length</td>
<td>length of this chunk containing this header. (0 means automatically-calculate when encoding)</td>
</tr>
<tr>
<td>info</td>
<td>ryu.lib.packet.sctp.param_heartbeat.</td>
</tr>
</tbody>
</table>

**class ryu.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.

- `ryu.lib.packet.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>set to '0'. this field will be ignored.</td>
</tr>
<tr>
<td>length</td>
<td>length of this chunk containing this header. (0 means automatically-calculate when encoding)</td>
</tr>
<tr>
<td>info</td>
<td>ryu.lib.packet.sctp.param_heartbeat.</td>
</tr>
</tbody>
</table>

**class ryu.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.

- `ryu.lib.packet.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>set to '0'. this field will be ignored.</td>
</tr>
<tr>
<td>length</td>
<td>length of this chunk containing this header. (0 means automatically-calculate when encoding)</td>
</tr>
<tr>
<td>init_tag</td>
<td>the tag that be used as Verification Tag.</td>
</tr>
<tr>
<td>a_rwnd</td>
<td>Advertised Receiver Window Credit.</td>
</tr>
<tr>
<td>os</td>
<td>number of outbound streams.</td>
</tr>
<tr>
<td>mis</td>
<td>number of inbound streams.</td>
</tr>
<tr>
<td>i_tsn</td>
<td>Transmission Sequence Number that the sender will use.</td>
</tr>
<tr>
<td>params</td>
<td>Optional/Variable-Length Parameters. a list of derived classes of ryu.lib.packet.sctp.param.</td>
</tr>
</tbody>
</table>

**class ryu.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.

- `ryu.lib.packet.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>set to '0'. this field will be ignored.</td>
</tr>
<tr>
<td>length</td>
<td>length of this chunk containing this header. (0 means automatically-calculate when encoding)</td>
</tr>
<tr>
<td>init_tag</td>
<td>the tag that be used as Verification Tag.</td>
</tr>
<tr>
<td>a_rwnd</td>
<td>Advertised Receiver Window Credit.</td>
</tr>
<tr>
<td>os</td>
<td>number of outbound streams.</td>
</tr>
<tr>
<td>mis</td>
<td>number of inbound streams.</td>
</tr>
<tr>
<td>i_tsn</td>
<td>Transmission Sequence Number that the sender will use.</td>
</tr>
<tr>
<td>params</td>
<td>Optional/Variable-Length Parameters. a list of derived classes of ryu.lib.packet.sctp.param.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>flags</td>
<td>set to '0'. this field will be ignored.</td>
</tr>
<tr>
<td>length</td>
<td>length of this chunk containing this header. (0 means automatically-calculate when encoding)</td>
</tr>
<tr>
<td>init_tag</td>
<td>the tag that be used as Verification Tag.</td>
</tr>
<tr>
<td>a_rwnd</td>
<td>Advertised Receiver Window Credit.</td>
</tr>
<tr>
<td>os</td>
<td>number of outbound streams.</td>
</tr>
<tr>
<td>mis</td>
<td>number of inbound streams.</td>
</tr>
<tr>
<td>i_tsn</td>
<td>Transmission Sequence Number that the sender will use.</td>
</tr>
<tr>
<td>params</td>
<td>Optional/Variable-Length Parameters.</td>
</tr>
<tr>
<td></td>
<td>a list of derived classes of ryu.lib.packet.sctp.param.</td>
</tr>
</tbody>
</table>

```python
class ryu.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.

- ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>set to '0'. this field will be ignored.</td>
</tr>
<tr>
<td>length</td>
<td>length of this chunk containing this header. (0 means automatically-calculate when encoding)</td>
</tr>
<tr>
<td>tsn_ack</td>
<td>TSN of the last DATA chunk received in sequence before a gap.</td>
</tr>
<tr>
<td>a_rwnd</td>
<td>Advertised Receiver Window Credit.</td>
</tr>
<tr>
<td>gapack_num</td>
<td>number of Gap Ack blocks.</td>
</tr>
<tr>
<td>duptsn_num</td>
<td>number of duplicate TSNs.</td>
</tr>
<tr>
<td>gapacks</td>
<td>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]]</td>
</tr>
<tr>
<td>duptsns</td>
<td>a list of duplicate TSN.</td>
</tr>
</tbody>
</table>

```python
class ryu.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.

- ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>set to '0'. this field will be ignored.</td>
</tr>
<tr>
<td>length</td>
<td>length of this chunk containing this header. (0 means automatically-calculate when encoding)</td>
</tr>
<tr>
<td>tsn_ack</td>
<td>TSN of the last DATA chunk received in sequence before a gap.</td>
</tr>
</tbody>
</table>

```python
class ryu.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.

- `ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>set to '0'. this field will be ignored.</td>
</tr>
<tr>
<td>length</td>
<td>length of this chunk containing this header. (0 means automatically-calculate when encoding)</td>
</tr>
</tbody>
</table>

```python
class ryu.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.

- `ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tflag</td>
<td>'0' means the Verification tag is normal. '1' means the Verification tag is copy of the sender.</td>
</tr>
<tr>
<td>length</td>
<td>length of this chunk containing this header. (0 means automatically-calculate when encoding)</td>
</tr>
</tbody>
</table>

```python
class ryu.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.

- `ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Suggested Cookie Life-Span Increment (msec).</td>
</tr>
<tr>
<td>length</td>
<td>length of this param containing this header. (0 means automatically-calculate when encoding)</td>
</tr>
</tbody>
</table>

```python
class ryu.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.

- `ryu.lib.packet.sctp.chunk_init`
- `ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>set to None.</td>
</tr>
<tr>
<td>length</td>
<td>length of this param containing this header. (0 means automatically-calculate when encoding)</td>
</tr>
</tbody>
</table>
class ryu.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.

• ryu.lib.packet.sctp.chunk_heartbeat
• ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>the sender-specific heartbeat information.</td>
</tr>
<tr>
<td>length</td>
<td>length of this param containing this header. (0 means automatically-calculate when encoding)</td>
</tr>
</tbody>
</table>

class ryu.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.

• ryu.lib.packet.sctp.chunk_init
• ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>a host name that ends with null terminator.</td>
</tr>
<tr>
<td>length</td>
<td>length of this param containing this header. (0 means automatically-calculate when encoding)</td>
</tr>
</tbody>
</table>

class ryu.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.

• ryu.lib.packet.sctp.chunk_init
• ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>IPv4 address of the sending endpoint.</td>
</tr>
<tr>
<td>length</td>
<td>length of this param containing this header. (0 means automatically-calculate when encoding)</td>
</tr>
</tbody>
</table>

class ryu.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.

• ryu.lib.packet.sctp.chunk_init
• ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>IPv6 address of the sending endpoint.</td>
</tr>
<tr>
<td>length</td>
<td>length of this param containing this header. (0 means automatically-calculate when encoding)</td>
</tr>
</tbody>
</table>

```python
class ryu.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.

- `ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>the state cookie. see Section 5.1.3 in RFC 4960.</td>
</tr>
<tr>
<td>length</td>
<td>length of this param containing this header. (0 means automatically-calculate when encoding)</td>
</tr>
</tbody>
</table>

```python
class ryu.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.

- `ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>a list of parameter types. odd cases pad with 0x0000.</td>
</tr>
<tr>
<td>length</td>
<td>length of this param containing this header. (0 means automatically-calculate when encoding)</td>
</tr>
</tbody>
</table>

```python
class ryu.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.

- `ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>the unrecognized parameter in the INIT chunk.</td>
</tr>
<tr>
<td>length</td>
<td>length of this param containing this header. (0 means automatically-calculate when encoding)</td>
</tr>
</tbody>
</table>

```python
class ryu.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.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>src_port</td>
<td>Source Port</td>
</tr>
<tr>
<td>dst_port</td>
<td>Destination Port</td>
</tr>
<tr>
<td>vtag</td>
<td>Verification Tag</td>
</tr>
<tr>
<td>csum</td>
<td>Checksum (0 means automatically-calculate when encoding)</td>
</tr>
<tr>
<td>chunks</td>
<td>a list of derived classes of ryu.lib.packet.sctp.chunk.</td>
</tr>
</tbody>
</table>

**FromClass Method `parser`**

`parser(buf)`

Decodes 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.

**FromClass Method `serialize`**

`serialize(payload, prev)`

Encodes 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** `ryu.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.


LACPDU format
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:

<table>
<thead>
<tr>
<th>Actor_State</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXPR</td>
<td>DFLT</td>
<td>DIST</td>
<td>CLCT</td>
<td>SYNC</td>
<td>AGGR</td>
<td>TMO</td>
<td>ACT</td>
<td></td>
</tr>
</tbody>
</table>

**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.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>LACP version. This parameter must be set to LACP_VERSION_NUMBER (i.e. 1).</td>
</tr>
<tr>
<td>actor_system_priority</td>
<td>The priority assigned to this System.</td>
</tr>
<tr>
<td>actor_system</td>
<td>The Actor's System ID, encoded as a MAC address.</td>
</tr>
<tr>
<td>actor_key</td>
<td>The operational Key value assigned to the port by the Actor.</td>
</tr>
<tr>
<td>actor_port_priority</td>
<td>The priority assigned to this port.</td>
</tr>
<tr>
<td>actor_port</td>
<td>The port number assigned to the port by the Actor.</td>
</tr>
<tr>
<td>actor_state_activity</td>
<td>about the activity control value with regard to this link.</td>
</tr>
<tr>
<td></td>
<td>LACP_STATE_ACTIVE(1)</td>
</tr>
<tr>
<td></td>
<td>LACP_STATE_PASSIVE(0)</td>
</tr>
<tr>
<td>actor_state_timeout</td>
<td>about the timeout control value with regard to this link.</td>
</tr>
<tr>
<td></td>
<td>LACP_STATE_SHORT_TIMEOUT(1)</td>
</tr>
<tr>
<td></td>
<td>LACP_STATE_LONG_TIMEOUT(0)</td>
</tr>
<tr>
<td>actor_state_aggregation</td>
<td>about how the system regards this link from the point of view of the aggregation.</td>
</tr>
<tr>
<td></td>
<td>LACP_STATE_AGGREGATEABLE(1)</td>
</tr>
<tr>
<td></td>
<td>LACP_STATE_INDIVIDUAL(0)</td>
</tr>
<tr>
<td>actor_state_synchronization</td>
<td>about how the system regards this link from the point of view of the synchronization.</td>
</tr>
<tr>
<td></td>
<td>LACP_STATE_IN_SYNC(1)</td>
</tr>
<tr>
<td></td>
<td>LACP_STATE_OUT_OF_SYNC(0)</td>
</tr>
<tr>
<td>actor_state_collecting</td>
<td>about collecting of incoming frames.</td>
</tr>
<tr>
<td></td>
<td>LACP_STATE_COLLECTING_ENABLED(1)</td>
</tr>
<tr>
<td></td>
<td>LACP_STATE_COLLECTING_DISABLED(0)</td>
</tr>
<tr>
<td>actor_state_distributing</td>
<td>about distributing of outgoing frames.</td>
</tr>
<tr>
<td></td>
<td>LACP_STATE_DISTRIBUTING_ENABLED(1)</td>
</tr>
<tr>
<td></td>
<td>LACP_STATE_DISTRIBUTING_DISABLED(0)</td>
</tr>
<tr>
<td>actor_state_defaulted</td>
<td>about the Partner information which the the Actor use.</td>
</tr>
<tr>
<td></td>
<td>LACP_STATE_DEFAULTED_PARTNER(1)</td>
</tr>
<tr>
<td></td>
<td>LACP_STATEOPERATIONAL_PARTNER(0)</td>
</tr>
<tr>
<td>actor_state_expired</td>
<td>about the state of the Actor.</td>
</tr>
<tr>
<td></td>
<td>LACP_STATE_EXPIRED(1)</td>
</tr>
<tr>
<td></td>
<td>LACP_STATE_NOT_EXPIRED(0)</td>
</tr>
<tr>
<td>partner_system_priority</td>
<td>The priority assigned to the Partner System.</td>
</tr>
<tr>
<td>partner_system</td>
<td>The Partner's System ID, encoded as a MAC address.</td>
</tr>
<tr>
<td>partner_key</td>
<td>The operational Key value assigned to the port by the Partner.</td>
</tr>
<tr>
<td>partner_port_priority</td>
<td>The priority assigned to this port by the Partner.</td>
</tr>
<tr>
<td>partner_port</td>
<td>The port number assigned to the port by the Partner.</td>
</tr>
<tr>
<td>partner_state_activity</td>
<td>See actor_state_activity.</td>
</tr>
<tr>
<td>partner_state_timeout</td>
<td>See actor_state_timeout.</td>
</tr>
<tr>
<td>partner_state_aggregation</td>
<td>See actor_state_aggregation.</td>
</tr>
<tr>
<td>partner_state_synchronization</td>
<td>See actor_state_synchronization.</td>
</tr>
<tr>
<td>partner_state_collecting</td>
<td>See actor_state_collecting.</td>
</tr>
<tr>
<td>partner_state_distributing</td>
<td>See actor_state_distributing.</td>
</tr>
<tr>
<td>partner_state_defaulted</td>
<td>See actor_state_defaulted.</td>
</tr>
<tr>
<td>partner_state_expired</td>
<td>See actor_state_expired.</td>
</tr>
<tr>
<td>collector_max_delay</td>
<td>the maximum time that the Frame Collector may delay.</td>
</tr>
</tbody>
</table>

2.4. Library
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 ryu.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

Slow Protocols Subtypes

<table>
<thead>
<tr>
<th>Subtype Value</th>
<th>Protocol Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Unused - Illegal Value</td>
</tr>
<tr>
<td>1</td>
<td>Link Aggregation Control Protocol(LACP)</td>
</tr>
<tr>
<td>2</td>
<td>Link Aggregation - Marker Protocol</td>
</tr>
<tr>
<td>3</td>
<td>Operations, Administration, and Maintenance(OAM)</td>
</tr>
<tr>
<td>4 - 9</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>10</td>
<td>Organization Specific Slow Protocol(OSSP)</td>
</tr>
<tr>
<td>11 - 255</td>
<td>Unused - Illegal values</td>
</tr>
</tbody>
</table>

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 ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>src_port</td>
<td>Source Port</td>
</tr>
<tr>
<td>dst_port</td>
<td>Destination Port</td>
</tr>
<tr>
<td>seq</td>
<td>Sequence Number</td>
</tr>
<tr>
<td>ack</td>
<td>Acknowledgement Number</td>
</tr>
<tr>
<td>offset</td>
<td>Data Offset (0 means automatically-calculate when encoding)</td>
</tr>
<tr>
<td>bits</td>
<td>Control Bits</td>
</tr>
<tr>
<td>window_size</td>
<td>Window</td>
</tr>
<tr>
<td>csum</td>
<td>Checksum (0 means automatically-calculate when encoding)</td>
</tr>
<tr>
<td>urgent</td>
<td>Urgent Pointer</td>
</tr>
<tr>
<td>option</td>
<td>List of TCPOption sub-classes or an bytearray containing options. None if no options.</td>
</tr>
</tbody>
</table>

**has_flags (**flags**)

Check if flags are set on this packet.

returns boolean if all passed flags is set

Example:

```python
>>> 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** ryu.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.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>src_port</td>
<td>Source Port</td>
</tr>
<tr>
<td>dst_port</td>
<td>Destination Port</td>
</tr>
<tr>
<td>total_length</td>
<td>Length (0 means automatically-calculate when encoding)</td>
</tr>
<tr>
<td>csum</td>
<td>Checksum (0 means automatically-calculate when encoding)</td>
</tr>
</tbody>
</table>

**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** *ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pcp</td>
<td>Priority Code Point</td>
</tr>
<tr>
<td>cfi</td>
<td>Canonical Format Indicator. In a case to be used as B-TAG, this field means DEI(Drop Eligible Indication).</td>
</tr>
<tr>
<td>vid</td>
<td>VLAN Identifier</td>
</tr>
<tr>
<td>ethertype</td>
<td>EtherType</td>
</tr>
</tbody>
</table>

**classmethod get_packet_type**(*type_*)

Per-protocol dict-like get method.

Provided for convenience of protocol implementers. Internal use only.
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pcp</td>
<td>Priority Code Point</td>
</tr>
<tr>
<td>cfi</td>
<td>Canonical Format Indicator</td>
</tr>
<tr>
<td>vid</td>
<td>VLAN Identifier</td>
</tr>
<tr>
<td>ethertype</td>
<td>EtherType</td>
</tr>
</tbody>
</table>

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:

```
+----------------------------------------------------------+
| Version | Type  | Virtual Rtr ID| Priority | Count IP Addr|
+----------+--------+--------------+----------+--------------+
| Auth Type| Adver Int| Checksum     |
+----------+---------+--------------+
| IP Address (1) |
| IP Address (n) |
+-------------------+
| Authentication Data (1) |
| Authentication Data (2) |
+-------------------+
```

[RFC 5798] VRRP v3 packet format:

```
+----------------------------------------------------------+
| IPv4 Fields or IPv6 Fields |...|...|
+-------------------------+---+---+
| Version | Type  | Virtual Rtr ID| Priority | Count IPvX Addr|
| (rsvd)  | Max Adver Int | Checksum     |
+----------+---------+--------------+
```

(continues on next page)
class ryu.lib.packet.vrrp.vrrp(version, type_, vrid, priority, count_ip, max_adver_int, check-
sum, 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 ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>Version</td>
</tr>
<tr>
<td>type</td>
<td>Type</td>
</tr>
<tr>
<td>vrid</td>
<td>Virtual Rtr ID (VRID)</td>
</tr>
<tr>
<td>priority</td>
<td>Priority</td>
</tr>
<tr>
<td>count_ip</td>
<td>Count IPvX Addr. Calculated automatically when encoding.</td>
</tr>
<tr>
<td>max_adver_int</td>
<td>Maximum Advertisement Interval (Max Adver Int)</td>
</tr>
<tr>
<td>checksum</td>
<td>Checksum. Calculated automatically when encoding.</td>
</tr>
<tr>
<td>ip_addresses</td>
<td>IPvX Address(es). A python list of IP addresses.</td>
</tr>
<tr>
<td>auth_type</td>
<td>Authentication Type (only for VRRPv2)</td>
</tr>
<tr>
<td>auth_data</td>
<td>Authentication Data (only for VRRPv2)</td>
</tr>
</tbody>
</table>

create_packet (primary_ip_address, vlan_id=None)

Prepare a VRRP packet.

Returns a newly created ryu.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 looks like the ones
described in the following sections.

- RFC 3768 5.1. VRRP Packet Format
- RFC 5798 5.1. VRRP Packet Format

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>primary_ip_address</td>
<td>Source IP address</td>
</tr>
<tr>
<td>vlan_id</td>
<td>VLAN ID. None for no VLAN.</td>
</tr>
</tbody>
</table>

classmethod parser (buf)

Decode a protocol header.

This method is used only when decoding a packet.

Decode a protocol header at offset 0 in bytarray 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.

```python
class ryu.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 ryu.lib.packet.packet_base.PacketBase derived classes, `create` method should be used to instantiate an object of this class.

```python
static create(type_, vrid, priority, max_adver_int, ip_addresses)
```

Unlike other ryu.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 ryu.lib.packet.vrrp.vrrp object’s attributes of the same name. (except that `type_` corresponds to `type` attribute.)

```python
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.

```python
class ryu.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 ryu.lib.packet.packet_base.PacketBase derived classes, `create` method should be used to instantiate an object of this class.

```python
static create(type_, vrid, priority, max_adver_int, ip_addresses)
```

Unlike other ryu.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 ryu.lib.packet.vrrp.vrrp object’s attributes of the same name. (except that `type_` corresponds to `type` attribute.)

```python
classmethod parser(buf)
```

Decode a protocol header.

This method is used only when decoding a packet.
Decide 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**

```python
ryu.lib.packet.vxlan.vni_from_bin(buf)
```
Converts binary representation VNI to integer.

- **Parameters** `buf` – binary representation of VNI.
- **Returns** VNI integer.

```python
ryu.lib.packet.vxlan.vni_to_bin(vni)
```
Converts integer VNI to binary representation.

- **Parameters** `vni` – integer of VNI
- **Returns** binary representation of VNI.

```python
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vni</td>
<td>VXLAN Network Identifier</td>
</tr>
</tbody>
</table>

```python
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.

```python
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 ryu.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 ryu.lib.packet.zebra.NextHopBlackhole(ifindex=None, ifname=None, addr=None,
type_=None)

Nexthop class for ZEBRA_NEXTHOP_BLACKHOLE type.

class ryu.lib.packet.zebra.NextHopIFIndex(ifindex=None, ifname=None, addr=None,
type_=None)

Nexthop class for ZEBRA_NEXTHOP_IFINDEX type.

class ryu.lib.packet.zebra.NextHopIFName(ifindex=None, ifname=None, addr=None,
type_=None)

Nexthop class for ZEBRA_NEXTHOP_IFNAME type.

class ryu.lib.packet.zebra.NextHopIPv4(ifindex=None, ifname=None, addr=None,
type_=None)

Nexthop class for ZEBRA_NEXTHOP_IPV4 type.

class ryu.lib.packet.zebra.NextHopIPv4IFIndex(ifindex=None, ifname=None, addr=None,
type_=None)

Nexthop class for ZEBRA_NEXTHOP_IPV4_IFINDEX type.

class ryu.lib.packet.zebra.NextHopIPv4IFName(ifindex=None, ifname=None, addr=None,
type_=None)

Nexthop class for ZEBRA_NEXTHOP_IPV4_IFNAME type.

class ryu.lib.packet.zebra.NextHopIPv6(ifindex=None, ifname=None, addr=None,
type_=None)

Nexthop class for ZEBRA_NEXTHOP_IPV6 type.

class ryu.lib.packet.zebra.NextHopIPv6IFIndex(ifindex=None, ifname=None, addr=None,
type_=None)

Nexthop class for ZEBRA_NEXTHOP_IPV6_IFINDEX type.

class ryu.lib.packet.zebra.NextHopIPv6IFName(ifindex=None, ifname=None, addr=None,
type_=None)

Nexthop class for ZEBRA_NEXTHOP_IPV6_IFNAME type.

class ryu.lib.packet.zebra.RegisteredNexthop(connected, family, prefix)

Unit of ZEBRA_NEXTHOP_REGISTER message body.

class ryu.lib.packet.zebra.ZebraBfdClientRegister(pid)

Message body class for FRR_ZEBRA_BFD_CLIENT_REGISTER.

class ryu.lib.packet.zebra.ZebraBfdDestinationDeregister(pid, dst_family, dst_prefix,
src_family, src_prefix, multi_hop, multi_hop_count=None, ifname=None)

Message body class for FRR_ZEBRA_BFD_DEST_DEREGISTER.
class ryu.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 ryu.lib.packet.zebra.ZebraBfdDestinationReply
Message body class for FRR_ZEBRA_BFD_DEST_REPLAY.

class ryu.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 ryu.lib.packet.zebra.ZebraHello (route_type, instance=None)
Message body class for ZEBRA_HELLO.

class ryu.lib.packet.zebra.ZebraIPv4ImportLookup (prefix, metric=None, nexthops=None, from_zebra=False)
Message body class for ZEBRA_IPV4_IMPORT_LOOKUP.

class ryu.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 ryu.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 ryu.lib.packet.zebra.ZebraIPv4NexthopLookup (addr, metric=None, nexthops=None)
Message body class for ZEBRA_IPV4_NEXTHOP_LOOKUP.

class ryu.lib.packet.zebra.ZebraIPv4NexthopLookupMRib (addr, distance=None, metric=None, nexthops=None)
Message body class for ZEBRA_IPV4_NEXTHOP_LOOKUP_MRIB.

class ryu.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 ryu.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 ryu.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 ryu.lib.packet.zebra.ZebraIPv6ImportLookup (prefix, metric=None, nexthops=None, from_zebra=False)

Message body class for ZEBRA_IPV6_IMPORT_LOOKUP.

class ryu.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 ryu.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 ryu.lib.packet.zebra.ZebraIPv6NexthopLookup (addr, metric=None, nexthops=None)

Message body class for ZEBRA_IPV6_NEXTHOP_LOOKUP.

class ryu.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 ryu.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 ryu.lib.packet.zebra.ZebraImportCheckUpdate(family, prefix, distance=None, metric=None, nexthops=None)
Message body class for FRR_ZEBRA_IMPORT_CHECK_UPDATE.

class ryu.lib.packet.zebra.ZebraImportRouteRegister(nexthops)
Message body class for FRR_ZEBRA_IMPORT_ROUTE_REGISTER.

class ryu.lib.packet.zebra.ZebraImportRouteUnregister(nexthops)
Message body class for FRR_ZEBRA_IMPORT_ROUTE_UNREGISTER.

class ryu.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, il_type=None, hw_addr_len=0, hw_addr=None, link_params=None)
Message body class for ZEBRA_INTERFACE_ADD.

class ryu.lib.packet.zebra.ZebraInterfaceAddressAdd(ifindex, ifc_flags, family, prefix, dest)
Message body class for ZEBRA_INTERFACE_ADDRESS_ADD.

class ryu.lib.packet.zebra.ZebraInterfaceAddressDelete(ifindex, ifc_flags, family, prefix, dest)
Message body class for ZEBRA_INTERFACE_ADDRESS_DELETE.

class ryu.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 ryu.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, il_type=None, hw_addr_len=0, hw_addr=None, link_params=None)
Message body class for ZEBRA_INTERFACE_DELETE.

class ryu.lib.packet.zebra.ZebraInterfaceDisableRadv(ifindex, interval)
Message body class for FRR_ZEBRA_INTERFACE_DISABLE_RADV.

class ryu.lib.packet.zebra.ZebraInterfaceEnableRadv(ifindex, interval)
Message body class for FRR_ZEBRA_INTERFACE_ENABLE_RADV.

class ryu.lib.packet.zebra.ZebraInterfaceLinkParams(ifindex, link_params)
Message body class for ZEBRA_INTERFACE_LINK_PARAMS.
class ryu.lib.packet.zebra.ZebraInterfaceNbrAddressAdd(ifindex, family, prefix)
Message body class for FRR_ZEBRA_INTERFACE_NBR_ADDRESS_ADD.

class ryu.lib.packet.zebra.ZebraInterfaceNbrAddressDelete(ifindex, family, prefix)
Message body class for FRR_ZEBRA_INTERFACE_NBR_ADDRESS_DELETE.

class ryu.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, il_type=None, hw_addr_len=0, hw_addr=None, link_params=None)
Message body class for ZEBRA_INTERFACE_UP.

class ryu.lib.packet.zebra.ZebraInterfaceVrfUpdate(ifindex, vrf_id)
Message body class for FRR_ZEBRA_INTERFACE_VRF_UPDATE.

class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>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.</td>
</tr>
<tr>
<td>version</td>
<td>Version number of the Zebra protocol message. To instantiate messages with other than the default version, version must be specified.</td>
</tr>
<tr>
<td>vrf_id</td>
<td>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.</td>
</tr>
<tr>
<td>command</td>
<td>Zebra Protocol command, which denotes message type.</td>
</tr>
<tr>
<td>body</td>
<td>Messages body. An instance of subclass of _ZebraMessageBody named like &quot;Zebra + &lt;message name&gt;&quot; (e.g., ZebraHello). Or None if message does not contain any body.</td>
</tr>
</tbody>
</table>

**Note:** To instantiate Zebra messages, command can be omitted when the valid body is specified.

```python
>>> from ryu.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.

```python
>>> 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.

```python
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.

```python
class ryu.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.

```python
class ryu.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.

```python
class ryu.lib.packet.zebra.ZebraNexthopRegister(nexthops)
```

Message body class for ZEBRA_NEXTHOP_REGISTER.

```python
class ryu.lib.packet.zebra.ZebraNexthopUnregister(nexthops)
```

Message body class for ZEBRA_NEXTHOP_UNREGISTER.

```python
class ryu.lib.packet.zebra.ZebraNexthopUpdate(family, prefix, distance=None, metric=None, nexthops=None)
```

Message body class for ZEBRA_NEXTHOP_UPDATE.

```python
class ryu.lib.packet.zebra.ZebraRedistributeAdd(route_type, afi=None, instance=None)
```

Message body class for ZEBRA_REDISTIBUTE_ADD.

```python
class ryu.lib.packet.zebra.ZebraRedistributeDefaultAdd(route_type, afi=None, instance=None)
```

Message body class for ZEBRA_REDISTIBUTE_DEFAULT_ADD.

```python
class ryu.lib.packet.zebra.ZebraRedistributeDefaultDelete(route_type, afi=None, instance=None)
```

Message body class for ZEBRA_REDISTIBUTE_DEFAULT_DELETE.

```python
class ryu.lib.packet.zebra.ZebraRedistributeDelete(route_type, afi=None, instance=None)
```

Message body class for ZEBRA_REDISTIBUTE_DELETE.

```python
class ryu.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.
Message body class for FRR_ZEBRA_IPV4_ROUTE_DELETE.

```python
class ryu.lib.packet.zebra.ZebraRedistributeIPv4Delete:
    def __init__(self, route_type, flags, message, safi=None, prefix=None, src_prefix=None, nexthops=None, ifindices=None, distance=None, metric=None, mtu=None, tag=None, instance=None, from_zebra=False):
```

Message body class for FRR_ZEBRA_REDISTRIBUTE_IPV6_ADD.

```python
class ryu.lib.packet.zebra.ZebraRedistributeIPv6Add:
    def __init__(self, route_type, flags, message, safi=None, prefix=None, src_prefix=None, nexthops=None, ifindices=None, distance=None, metric=None, mtu=None, tag=None, instance=None, from_zebra=False):
```

Message body class for FRR_ZEBRA_REDISTRIBUTE_IPV6_DEL.

```python
class ryu.lib.packet.zebra.ZebraRedistributeIPv6Delete:
    def __init__(self, route_type, flags, message, safi=None, prefix=None, src_prefix=None, nexthops=None, ifindices=None, distance=None, metric=None, mtu=None, tag=None, instance=None, from_zebra=False):
```

Message body class for ZEBRA_ROUTER_ID_ADD.

```python
class ryu.lib.packet.zebra.ZebraRouterIDAdd:
    def __init__(self, family, prefix):
```

Message body class for ZEBRA_ROUTER_ID_DELETE.

```python
class ryu.lib.packet.zebra.ZebraRouterIDDelete:
    def __init__(self, family, prefix):
```

Message body class for ZEBRA_ROUTER_ID_UPDATE.

```python
class ryu.lib.packet.zebra.ZebraRouterIDUpdate:
    def __init__(self, family, prefix):
```

Message body class for Unknown command.

```python
class ryu.lib.packet.zebra.ZebraUnknownMessage:
    def __init__(self, buf):
```

Message body class for FRR_ZEBRA_VRF_ADD.

```python
class ryu.lib.packet.zebra.ZebraVrfAdd:
    def __init__(self, vrf_name):
```

Message body class for FRR_ZEBRA_VRF_DELETE.

```python
class ryu.lib.packet.zebra.ZebraVrfDelete:
    def __init__(self, vrf_name):
```

Message body class for ZEBRA_VRF_UNREGISTER.

```python
ryu.lib.packet.zebra.zebra:
    alias of ryu.lib.packet.zebra.ZebraMessage
```

### 2.4.3 PCAP file library
Introduction

Ryu 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.

```python
class ryu.lib.pcaplib.Reader(file_obj)

PCAP file reader
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file_obj</td>
<td>File object which reading PCAP file in binary mode</td>
</tr>
</tbody>
</table>

Example of usage:

```python
from ryu.lib import pcaplib
from ryu.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 RyuApp received, you can use pcaplib.Writer.

```python
class ryu.lib.pcaplib.Writer(file_obj, snaplen=65535, network=1)

PCAP file writer
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file_obj</td>
<td>File object which writing PCAP file in binary mode</td>
</tr>
<tr>
<td>snaplen</td>
<td>Max length of captured packets (in octets)</td>
</tr>
<tr>
<td>network</td>
<td>Data link type. (e.g. 1 for Ethernet, see tcpdump.org for details)</td>
</tr>
</tbody>
</table>

Example of usage:

```python...
from ryu.lib import pcaplib

class SimpleSwitch13(app_manager.RyuApp):
    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)
    ...
```

## 2.4.4 OF-Config support

Ryu has a library for OF-Config support.

### XML schema files for NETCONF 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

## 2.4.5 BGP speaker library

### Introduction

Ryu 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 "ryu.lib.hub"
import logging
import sys
log = logging.getLogger()
log.addHandler(logging.StreamHandler(sys.stderr))

from ryu.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) + '/24'
        print "add a new prefix", prefix
        speaker.prefix_add(prefix)
        count += 1
        if count == 4:
            speaker.shutdown()
            break

2.4.6 BGP speaker library API Reference
BGPSpeaker class

```python
class ryu.services.protocols.bgp.bgpspeaker.BGPSpeaker (as_number, router_id,
  bgp_server_host=('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),
  allow_local_as_in_count=0,
  cluster_id=None,
  local_pref=100)
```

Class to provide the APIs of Ryu 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:

```python
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\((route\_type, route\_dist, esi=0, ethernet\_tag\_id=None, mac\_addr=None, ip\_addr=None, ip\_prefix=None, gw\_ip\_addr=None, vni=None, next\_hop=None, tunnel\_type=None, pmsi\_tunnel\_type=None, redundancy\_mode=None)\)

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 ryu.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.
  - PM\_T\_NO\_T\_INFO = 0
  - PM\_T\_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'

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.
eternet_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.

- FLOWSPEC_FAMILY_IPV4 = 'ipv4fs'
- FLOWSPEC_FAMILY_IPV6 = 'ipv6fs'
- FLOWSPEC_FAMILY_VPNV4 = 'vpnv4fs'
- FLOWSPEC_FAMILY_VPNV6 = 'vpvn6fs'
- FLOWSPEC_FAMILY_L2VPN = 'l2vpnfs'

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.

- ryu.lib.packet.bgp.FlowSpecIPv4NLRI
- ryu.lib.packet.bgp.FlowSpecIPv6NLRI
- ryu.lib.packet.bgp.FlowSpecVPNv4NLRI
- ryu.lib.packet.bgp.FlowSpecVPNv6NLRI
- ryu.lib.packet.bgp.FlowSpecL2VPNNLRI

route_dist specifies a route distinguisher value. This parameter is required only if flowspec_family is one of the following address family.

- FLOWSPEC_FAMILY_VPNV4 = 'vpnv4fs'
- FLOWSPEC_FAMILY_VPNV6 = 'vpvn6fs'
- FLOWSPEC_FAMILY_L2VPN = 'l2vpnfs'

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.

<table>
<thead>
<tr>
<th>ACTION_NAME</th>
<th>Action Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>traffic_rate</td>
<td>ryu.lib.packet.bgp.BGPFlowSpecTrafficRateCommunity</td>
</tr>
<tr>
<td>traffic_action</td>
<td>ryu.lib.packet.bgp.BGPFlowSpecTrafficActionCommunity</td>
</tr>
<tr>
<td>redirect</td>
<td>ryu.lib.packet.bgp.BGPFlowSpecRedirectCommunity</td>
</tr>
<tr>
<td>traffic_marking</td>
<td>ryu.lib.packet.bgp.BGPFlowSpecTrafficMarkingCommunity</td>
</tr>
<tr>
<td>vlan_action</td>
<td>ryu.lib.packet.bgp.BGPFlowSpecVlanActionCommunity</td>
</tr>
<tr>
<td>tpid_action</td>
<td>ryu.lib.packet.bgp.BGPFlowSpecTPIDActionCommunity</td>
</tr>
</tbody>
</table>

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=FLOWSPEC_FAMILY_IPV4,
...    rules={
...        '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_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 ryu.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:

```python
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\(\text{route\_dist}, \text{import\_rts}, \text{export\_rts}, \text{site\_of\_origins=None, route\_family='ipv4', multi\_exit\_disc=None}\)  
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.  
routing_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 = 'l2vpnfs'  
multi_exit_disc specifies multi exit discriminator (MED) value. It must be an integer.

vrf_del\(\text{route\_dist}\)  
This method deletes the existing vrf.  
route_dist specifies a route distinguisher value.

vrfs_get\(\text{subcommand='routes', route\_dist=None, route\_family='all', format='json'}\)  
This method returns the existing vrf.

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.  
routing_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 ryu.services.protocols.bgp.bgpspeaker.EventPrefix\(\text{path, is\_withdraw}\)  
Used to pass an update on any best remote path to best_path_change_handler.

2.4. Library
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>remote_as</td>
<td>The AS number of a peer that caused this change</td>
<td></td>
</tr>
<tr>
<td>route_dist</td>
<td>None in the case of IPv4 or IPv6 family</td>
<td></td>
</tr>
<tr>
<td>prefix</td>
<td>A prefix was changed</td>
<td></td>
</tr>
<tr>
<td>nexthop</td>
<td>The nexthop of the changed prefix</td>
<td></td>
</tr>
<tr>
<td>label</td>
<td>MPLS label for VPNv4, VPNv6 or EVPN prefix</td>
<td></td>
</tr>
<tr>
<td>path</td>
<td>An instance of <code>info_base.base.Path</code> subclass</td>
<td></td>
</tr>
<tr>
<td>is_withdraw</td>
<td>True if this prefix has gone otherwise False</td>
<td></td>
</tr>
</tbody>
</table>

```python
class ryu.services.protocols.bgp.info_base.base.PrefixFilter(prefix, policy=None, ge=None, le=None)
```

Used to specify a prefix for filter.

We can create PrefixFilter object as follows:

```python
prefix_filter = PrefixFilter('10.5.111.0/24',
                          policy=PrefixFilter.POLICY_PERMIT)
```

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>prefix</td>
<td>A prefix used for this filter</td>
<td></td>
</tr>
<tr>
<td>policy</td>
<td>One of the following values.</td>
<td></td>
</tr>
<tr>
<td>ge</td>
<td>Prefix length that will be applied to this filter. ge means greater than or equal.</td>
<td></td>
</tr>
<tr>
<td>le</td>
<td>Prefix length that will be applied to this filter. le means less than or equal.</td>
<td></td>
</tr>
</tbody>
</table>

For example, when PrefixFilter object is created as follows:

```python
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:

```python
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** 

```python
ryu.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:

```python
as_path_filter = ASPathFilter(65000, policy=ASPathFilter.TOP)
```

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>as_number</td>
<td>A AS number used for this filter</td>
</tr>
<tr>
<td>policy</td>
<td>One of the following values.</td>
</tr>
<tr>
<td></td>
<td>ASPathFilter.POLICY_TOP</td>
</tr>
<tr>
<td></td>
<td>ASPathFilter.POLICY_END</td>
</tr>
<tr>
<td></td>
<td>ASPathFilter.POLICY_INCLUDE</td>
</tr>
<tr>
<td></td>
<td>ASPathFilter.POLICY_NOT_INCLUDE</td>
</tr>
</tbody>
</table>

Meaning of each policy is as follows:

<table>
<thead>
<tr>
<th>Policy</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>POLICY_TOP</td>
<td>Filter checks if the specified AS number is at the top of AS_PATH attribute.</td>
</tr>
<tr>
<td>POLICY_END</td>
<td>Filter checks if the specified AS number is at the last of AS_PATH attribute.</td>
</tr>
<tr>
<td>POLICY_INCLUDE</td>
<td>Filter checks if specified AS number exists in AS_PATH attribute.</td>
</tr>
<tr>
<td>POLICY_NOT_INCLUDE</td>
<td>Opposite to POLICY_INCLUDE.</td>
</tr>
</tbody>
</table>

**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** 

```python
ryu.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:

```python
pref_filter = PrefixFilter('192.168.103.0/30',
                          PrefixFilter.POLICY_PERMIT)

attribute_map = AttributeMap([pref_filter],
                             AttributeMap.ATTR_LOCAL_PREF, 250)
```

(continues on next page)
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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filters</td>
<td>A list of filter. Each object should be a Filter class or its sub-class</td>
</tr>
<tr>
<td>attr_type</td>
<td>A type of attribute to map on filters. Currently AttributeMap.ATTR_LOCAL_PREF is available.</td>
</tr>
<tr>
<td>attr_value</td>
<td>A attribute value</td>
</tr>
</tbody>
</table>

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.

2.4.7 MRT file library

Introduction

Ryu 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 ryu.lib.mrtlib.Reader()
MRT format file reader.

Argument          | Description  |
------------------|--------------|
f                | File object which reading MRT format file in binary mode. |
```

Example of Usage:

```
import bz2
from ryu.lib import mrtlib

count = 0
for record in mrtlib.Reader(bz2.BZ2File('rib.YYYYMMDD.hhmm.bz2', 'rb')):
```

(continues on next page)
Writing MRT file

For dumping the routing information which your RyuApp generated, you can use mrtlib.Writer.

```python
print("%d, %s" % (count, record))
count += 1
```

**class** `ryu.lib.mrtlib.Writer(f)`

MRT format file writer.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>f</code></td>
<td>File object which writing MRT format file in binary mode.</td>
</tr>
</tbody>
</table>

Example of usage:

```python
import bz2
import time
from ryu.lib import mrtlib
from ryu.lib.packet import bgp

mrt_writer = mrtlib.Writer(
    # Use a BZ2 file to compress the output.
    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)
```

### 2.4.8 OVSDB Manager library

**Path:** `ryu.services.protocols.ovsdb`

**Introduction**

Ryu 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 Ryu, 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.

```python
import uuid

from ryu.base import app_manager
from ryu.controller.handler import set_ev_cls
from ryu.services.protocols.ovsdb import api as ovsdb
from ryu.services.protocols.ovsdb import event as ovsdb_event

class MyApp(app_manager.RyuApp):
    @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)

        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)

    (continues on next page)
```
port.name = name
port.interfaces = [iface]
bridge.ports = bridge.ports + [port]

    return new_port_uuid, new_iface_uuid

req = ovsdb_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]

2.4.9 OVSDB library

Path: ryu.lib.ovs

Similar to the OVSDB Manager library, this library enables your application to speak the OVSDB protocol (RFC7047), but differ from the OVSDB 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 ryu.lib.ovs.vsctl.VSCtl.
2. Construct commands with ryu.lib.ovs.vsctl.VSCtlCommand. The syntax is almost the same as ovs-vsctl command.

Example

from ryu.lib.ovs import vsctl

OVSDB_ADDR = 'tcp:127.0.0.1:6640'
vsctl = vsctl.VSCtl(OVSDB_ADDR)

    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"
)

# Equivalent to
# $ ovs-vsctl list Port s1-eth1
command = vsctl.VSCtlCommand('list', ('Port', 's1-eth1'))
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

ryu.lib.ovs.vsctl

ovs-vsctl command like library to speak OVSDB protocol

class ryu.lib.ovs.vsctl.VSCtl(remote)
    A class to describe an Open vSwitch instance.
    remote specifies the address of the OVS instance. ryu.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 ryu.lib.ovs.vsctl.VSCtlCommand.

        If timeout_sec is specified, raises exception after the given timeout [sec]. Additionally, if exception
        is specified, this function will wraps exception using the given exception class.

        Retruns None but fills result attribute for each command instance.

class ryu.lib.ovs.vsctl.VSCtlCommand(command, args=None, options=None)
    Class to describe artgumens 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.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--may-exist</td>
<td>Does nothing when the given port already exists. The supported commands are add-port and add-bond.</td>
</tr>
<tr>
<td>--fake-iface</td>
<td>Creates a port as a fake interface. The supported command is add-bond.</td>
</tr>
<tr>
<td>--must-exist</td>
<td>Raises port if the given port does not exist. The supported command is del-port.</td>
</tr>
<tr>
<td>--with-iface</td>
<td>Takes effect to the interface which has the same name. The supported command is del-port.</td>
</tr>
<tr>
<td>--if-exists</td>
<td>Ignores exception when not found. The supported command is get.</td>
</tr>
</tbody>
</table>
**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.

**ryu.lib.ovs.bridge**

Wrapper utility library of ryu.lib.ovs.vsctl

**class ryu.lib.ovs.bridge.OVSBridge**(CONF, datapath_id, ovsdb_addr, timeout=None, exception=None, br_name=None)

Class to provide wrapper utilities of ryu.lib.ovs.vsctl.VSCtl

CONF is a instance of oslo_config.cfg.ConfigOpts. Mostly self.CONF is sufficient to instantiate this class from your Ryu 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 ryu.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 ryu.lib.ovs.

vsctl1.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

```python
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:
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 ryu.lib.ovs.bridge.OVSBridgeNotFoundException 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 *ryu.lib.ovs.vsctl.VSCtlCommand*.

The given timeout and exception when instantiation will be used to call *ryu.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 *ryu.lib.ovs.bridge.OVSBridgeNotFound* *(msg=None, **kwargs)*

2.5 OpenFlow protocol API Reference

2.5.1 OpenFlow version independent classes and functions

Base class for OpenFlow messages

class *ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>datapath</td>
<td>A <em>ryu.controller.controller.Datapath</em> instance for this message</td>
</tr>
<tr>
<td>version</td>
<td>OpenFlow protocol version</td>
</tr>
<tr>
<td>msg_type</td>
<td>Type of OpenFlow message</td>
</tr>
<tr>
<td>msg_len</td>
<td>Length of the message</td>
</tr>
<tr>
<td>xid</td>
<td>Transaction id</td>
</tr>
<tr>
<td>buf</td>
<td>Raw data</td>
</tr>
</tbody>
</table>
_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.

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ascii</td>
<td>US-ASCII</td>
</tr>
<tr>
<td>utf-8</td>
<td>UTF-8</td>
</tr>
</tbody>
</table>

Example:

```python
_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.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dict_</td>
<td>A dictionary which describes the parameters. For example, {&quot;Param1&quot;: 100, &quot;Param2&quot;: 200}</td>
</tr>
<tr>
<td>decode_string</td>
<td>(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 _TYPE class attribute.</td>
</tr>
<tr>
<td>additional_args</td>
<td>(Optional) Additional kwargs for constructor.</td>
</tr>
</tbody>
</table>

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:

```python
ClassName(Param1=100, Param2=200)
```

this method would produce:

```python
{ "ClassName": {"Param1": 100, "Param2": 200} }
```

This method takes the following arguments.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>encode_string</td>
<td>(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 _TYPE class attribute.</td>
</tr>
</tbody>
</table>
Functions

```
ryu.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.

```
jsonstr = '{ "OFPSetConfig": { "flags": 0, "miss_send_len": 128 } }'
jsondict = json.loads(jsonstr)
o = ofp_msg_from_jsondict(dp, jsondict)
```

```
o = dp.ofproto_parser.OFPSetConfig(flags=0, miss_send_len=128)
```

This function takes the following arguments.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dp</td>
<td>An instance of ryu.controller.Datapath.</td>
</tr>
<tr>
<td>jsondict</td>
<td>A JSON style dict.</td>
</tr>
</tbody>
</table>

2.5.2 OpenFlow v1.0 Messages and Structures

Controller-to-Switch Messages

Handshake

```
class ryu.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 Ryu framework, so the Ryu 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 ryu.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 Ryu framework, so the Ryu application do not need to process this typically.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>datapath_id</td>
<td>Datapath unique ID.</td>
</tr>
<tr>
<td>n_buffers</td>
<td>Max packets buffered at once.</td>
</tr>
<tr>
<td>n_tables</td>
<td>Number of tables supported by datapath.</td>
</tr>
<tr>
<td>capabilities</td>
<td>Bitmap of capabilities flag.</td>
</tr>
<tr>
<td></td>
<td>OFPC_FLOW_STATS</td>
</tr>
<tr>
<td></td>
<td>OFPC_TABLE_STATS</td>
</tr>
<tr>
<td></td>
<td>OFPC_PORT_STATS</td>
</tr>
<tr>
<td></td>
<td>OFPC_STP</td>
</tr>
<tr>
<td></td>
<td>OFPC_RESERVED</td>
</tr>
<tr>
<td></td>
<td>OFPC_IP_REASM</td>
</tr>
<tr>
<td></td>
<td>OFPC_QUEUE_STATS</td>
</tr>
<tr>
<td></td>
<td>OFPC_ARP_MATCH_IP</td>
</tr>
<tr>
<td>actions</td>
<td>Bitmap of supported OFPAT_*</td>
</tr>
<tr>
<td>ports</td>
<td>List of OFPPhyPort instances.</td>
</tr>
</tbody>
</table>

Example:

```python
@set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
def switch_features_handler(self, ev):
    msg = ev.msg

    self.logger.debug('OFPSwitchFeatures received: ')
    self.logger.debug(' 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:

```json
{
   "OFPSwitchFeatures": {
      "actions": 2115,
      "capabilities": 169,
      "datapath_id": 1095522080376,
      "n_buffers": 0,
      "n_tables": 255,
      "ports": {
         "6": {
            "OFPPhyPort": {
               "advertised": 640,
               "config": 0,
               "curr": 648,
               "hw_addr": "f2:0b:a4:7d:f8:ea",
               "name": "Port6",
               "peer": 648,
               "port_no": 6,
               "state": 2,
               "supported": 648
            }
         }
      }
   }
}```

(continues on next page)
Switch Configuration

class ryu.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 configuration parameters.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>One of the following configuration flags.</td>
</tr>
<tr>
<td></td>
<td>OFPC_FRAG_NORMAL</td>
</tr>
<tr>
<td></td>
<td>OFPC_FRAG_DROP</td>
</tr>
<tr>
<td></td>
<td>OFPC_FRAG_REASM</td>
</tr>
<tr>
<td></td>
<td>OFPC_FRAG_MASK</td>
</tr>
<tr>
<td>miss_send_len</td>
<td>Max bytes of new flow that datapath should send to the controller.</td>
</tr>
</tbody>
</table>

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 ryu.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:
```python
def send_get_config_request(self, datapath):
    ofp_parser = datapath.ofproto_parser
    req = ofp_parser.OFPGetConfigRequest(datapath)
    datapath.send_msg(req)
```

class ryu.ofproto.ofproto_v1_0_parser.OFPGetConfigReply(datapath)
Get config reply message

The switch responds to a configuration request with a get config reply message.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>One of the following configuration flags.</td>
</tr>
<tr>
<td></td>
<td>OFPC_FRAG_NORMAL</td>
</tr>
<tr>
<td></td>
<td>OFPC_FRAG_DROP</td>
</tr>
<tr>
<td></td>
<td>OFPC_FRAG_REASM</td>
</tr>
<tr>
<td></td>
<td>OFPC_FRAG_MASK</td>
</tr>
<tr>
<td>miss_send_len</td>
<td>Max bytes of new flow that datapath should send to the controller.</td>
</tr>
</tbody>
</table>

Example:

```python
@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'
    elif msg.flags == ofp.OFPC_FRAG_MASK:
        flags = 'MASK'
    else:
        flags = 'unknown'
    self.logger.debug('OFPGetConfigReply received: ' 'flags=%s miss_send_len=%d',
                      flags, msg.miss_send_len)
```

Modify State Messages

class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>match</td>
<td>Instance of OFPMatch.</td>
</tr>
<tr>
<td>cookie</td>
<td>Opaque controller-issued identifier.</td>
</tr>
<tr>
<td>command</td>
<td>One of the following values.</td>
</tr>
<tr>
<td></td>
<td>OFPFC_ADD</td>
</tr>
<tr>
<td></td>
<td>OFPFC_MODIFY</td>
</tr>
<tr>
<td></td>
<td>OFPFC_MODIFY_STRICT</td>
</tr>
<tr>
<td></td>
<td>OFPFC_DELETE</td>
</tr>
<tr>
<td></td>
<td>OFPFC_DELETE_STRICT</td>
</tr>
<tr>
<td>idle_timeout</td>
<td>Idle time before discarding (seconds).</td>
</tr>
<tr>
<td>hard_timeout</td>
<td>Max time before discarding (seconds).</td>
</tr>
<tr>
<td>priority</td>
<td>Priority level of flow entry.</td>
</tr>
<tr>
<td>buffer_id</td>
<td>Buffered packet to apply to (or 0xffffffff). Not meaningful for OFPFC_DELETE*.</td>
</tr>
<tr>
<td>out_port</td>
<td>For OFPFC_DELETE* commands, require matching entries to include this as an output port. A value of OFPP_NONE indicates no restriction.</td>
</tr>
<tr>
<td>flags</td>
<td>One of the following values.</td>
</tr>
<tr>
<td></td>
<td>OFPFF_SEND_FLOW_REM</td>
</tr>
<tr>
<td></td>
<td>OFPFF_CHECK_OVERLAP</td>
</tr>
<tr>
<td></td>
<td>OFPFF_EMERG</td>
</tr>
<tr>
<td>actions</td>
<td>List of OFPAction* instance.</td>
</tr>
</tbody>
</table>

Example:

```python
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:

```json
{
    "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,  
        "tp_src": 0,  
        "wildcards": 4194295  
    }  
},  
"out_port": 65532,  
"priority": 123  
}

```python
class ryu.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.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>port_no</td>
<td>Port number to modify.</td>
</tr>
<tr>
<td>hw_addr</td>
<td>The hardware address that must be the same as hw_addr of OFPPhyPort of OFPSwitchFeatures.</td>
</tr>
<tr>
<td>config</td>
<td>Bitmap of configuration flags.</td>
</tr>
<tr>
<td></td>
<td>OFPPC_PORT_DOWN</td>
</tr>
<tr>
<td></td>
<td>OFPPC_NO_STP</td>
</tr>
<tr>
<td></td>
<td>OFPPC_NO_RECV</td>
</tr>
<tr>
<td></td>
<td>OFPPC_NO_RECV_STP</td>
</tr>
<tr>
<td></td>
<td>OFPPC_NO_FLOOD</td>
</tr>
<tr>
<td></td>
<td>OFPPC_NO_FWD</td>
</tr>
<tr>
<td></td>
<td>OFPPC_NO_PACKET_IN</td>
</tr>
<tr>
<td>mask</td>
<td>Bitmap of configuration flags above to be changed</td>
</tr>
<tr>
<td>advertise</td>
<td>Bitmap of the following flags.</td>
</tr>
<tr>
<td></td>
<td>OFPPF_10MB_HD</td>
</tr>
<tr>
<td></td>
<td>OFPPF_10MB_FD</td>
</tr>
<tr>
<td></td>
<td>OFPPF_100MB_HD</td>
</tr>
<tr>
<td></td>
<td>OFPPF_100MB_FD</td>
</tr>
<tr>
<td></td>
<td>OFPPF_1GB_HD</td>
</tr>
<tr>
<td></td>
<td>OFPPF_1GB_FD</td>
</tr>
<tr>
<td></td>
<td>OFPPF_10GB_FD</td>
</tr>
<tr>
<td></td>
<td>OFPPF_COPPER</td>
</tr>
<tr>
<td></td>
<td>OFPPF_FIBER</td>
</tr>
<tr>
<td></td>
<td>OFPPF_AUTONEG</td>
</tr>
<tr>
<td></td>
<td>OFPPF_PAUSE</td>
</tr>
<tr>
<td></td>
<td>OFPPF_PAUSEASYM</td>
</tr>
</tbody>
</table>

Example:

```python
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.OFPC_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 |
                 ofp.OFPPF_AUTONEG | ofp.OFPPF_PAUSE |
                 ofp.OFPPF_PAUSEASYM)
    req = ofp_parser.OFPPortMod(datapath, port_no, hw_addr, config,
                                 mask, advertise)
    datapath.send_msg(req)
```
Queue Configuration Messages

class ryu.ofproto.ofproto_v1_0_parser.OFPQueueGetConfigRequest (datapath, port)
Queue configuration request message

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>port</td>
<td>Port to be queried. Should refer to a valid physical port (i.e. &lt; OFPP_MAX).</td>
</tr>
</tbody>
</table>

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 ryu.ofproto.ofproto_v1_0_parser.OFPQueueGetConfigReply (datapath)
Queue configuration reply message

The switch responds with this message to a queue configuration request.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>port</td>
<td>Port to be queried.</td>
</tr>
<tr>
<td>queues</td>
<td>List of OFPPacketQueue instance.</td>
</tr>
</tbody>
</table>

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 ryu.ofproto.ofproto_v1_0_parser.OFPDescStatsRequest (datapath, flags)
Description statistics request message

The controller uses this message to query description of the switch.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero (none yet defined in the spec).</td>
</tr>
</tbody>
</table>

Example:

def send_desc_stats_request(self, datapath):
    ofp_parser = datapath.ofproto_parser
    req = ofp_parser.OFPDescStatsRequest(datapath)
    datapath.send_msg(req)
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mfr_desc</td>
<td>Manufacturer description.</td>
</tr>
<tr>
<td>hw_desc</td>
<td>Hardware description.</td>
</tr>
<tr>
<td>sw_desc</td>
<td>Software description.</td>
</tr>
<tr>
<td>serial_num</td>
<td>Serial number.</td>
</tr>
<tr>
<td>dp_desc</td>
<td>Human readable description of datapath.</td>
</tr>
</tbody>
</table>

Example:

```python
@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 ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero (none yet defined in the spec).</td>
</tr>
<tr>
<td>match</td>
<td>Instance of OFPMacch.</td>
</tr>
<tr>
<td>table_id</td>
<td>ID of table to read (from ofp_table_stats), 0xff for all tables or 0xfe for emergency.</td>
</tr>
<tr>
<td>out_port</td>
<td>Require matching entries to include this as an output port. A value of OFPP_NONE indicates no restriction.</td>
</tr>
</tbody>
</table>

Example:

```python
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_NULL
    req = ofp_parser.OFPFlowStatsRequest(
        datapath, 0, match, table_id, out_port)
    datapath.send_msg(req)
```

class ryu.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.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_id</td>
<td>ID of table flow came from.</td>
</tr>
<tr>
<td>match</td>
<td>Instance of OFPMatch.</td>
</tr>
<tr>
<td>duration_sec</td>
<td>Time flow has been alive in seconds.</td>
</tr>
<tr>
<td>duration_nsec</td>
<td>Time flow has been alive in nanoseconds beyond duration_sec.</td>
</tr>
<tr>
<td>priority</td>
<td>Priority of the entry. Only meaningful when this is not an exact-match entry.</td>
</tr>
<tr>
<td>idle_timeout</td>
<td>Number of seconds idle before expiration.</td>
</tr>
<tr>
<td>hard_timeout</td>
<td>Number of seconds before expiration.</td>
</tr>
<tr>
<td>cookie</td>
<td>Opaque controller-issued identifier.</td>
</tr>
<tr>
<td>packet_count</td>
<td>Number of packets in flow.</td>
</tr>
<tr>
<td>byte_count</td>
<td>Number of bytes in flow.</td>
</tr>
<tr>
<td>actions</td>
<td>List of OFPAction* instance</td>
</tr>
</tbody>
</table>

Example:

```python
@set_ev_cls(ofp_event.EventOFPFlowStatsReply, MAIN_DISPATCHER)
def flow_stats_reply_handler(self, ev):
    msg = ev.msg
    ofp = msg.datapath.ofproto
    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 ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero (none yet defined in the spec).</td>
</tr>
<tr>
<td>match</td>
<td>Fields to match.</td>
</tr>
<tr>
<td>table_id</td>
<td>ID of table to read (from ofp_table_stats) 0xff for all tables or 0xfe for emergency.</td>
</tr>
<tr>
<td>out_port</td>
<td>Require matching entries to include this as an output port. A value of OFPP_NONE indicates no restriction.</td>
</tr>
</tbody>
</table>

Example:
```python
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 ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>packet_count</td>
<td>Number of packets in flows.</td>
</tr>
<tr>
<td>byte_count</td>
<td>Number of bytes in flows.</td>
</tr>
<tr>
<td>flow_count</td>
<td>Number of flows.</td>
</tr>
</tbody>
</table>

Example:

```python
@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 ryu.ofproto.ofproto_v1_0_parser.OFPTableStatsRequest (datapath, flags)

Table statistics request message

The controller uses this message to query flow table statistics.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero (none yet defined in the spec).</td>
</tr>
</tbody>
</table>

Example:

```python
def send_table_stats_request(self, datapath):
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPTableStatsRequest(datapath)
    datapath.send_msg(req)
```

### class ryu.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.

---

150 Chapter 2. Writing Your Ryu Application
### Attribute Table

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_id</td>
<td>ID of table.</td>
</tr>
<tr>
<td>name</td>
<td>Table name.</td>
</tr>
<tr>
<td>wildcards</td>
<td>Bitmap of OFPFW_* wildcards that are supported by the table.</td>
</tr>
<tr>
<td>max_entries</td>
<td>Max number of entries supported.</td>
</tr>
<tr>
<td>active_count</td>
<td>Number of active entries</td>
</tr>
<tr>
<td>lookup_count</td>
<td>Number of packets looked up in table</td>
</tr>
<tr>
<td>matched_count</td>
<td>Number of packets that hit table</td>
</tr>
</tbody>
</table>

### Example:

```python
@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)
```

### ryu.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.

### Example:

```python
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)
```

### ryu.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.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>port_no</td>
<td>Port number.</td>
</tr>
<tr>
<td>rx_packets</td>
<td>Number of received packets.</td>
</tr>
<tr>
<td>tx_packets</td>
<td>Number of transmitted packets.</td>
</tr>
<tr>
<td>rx_bytes</td>
<td>Number of received bytes.</td>
</tr>
<tr>
<td>tx_bytes</td>
<td>Number of transmitted bytes.</td>
</tr>
<tr>
<td>rx_dropped</td>
<td>Number of packets dropped by RX.</td>
</tr>
<tr>
<td>tx_dropped</td>
<td>Number of packets dropped by TX.</td>
</tr>
<tr>
<td>rx_errors</td>
<td>Number of receive errors.</td>
</tr>
<tr>
<td>tx_errors</td>
<td>Number of transmit errors.</td>
</tr>
<tr>
<td>rx_frame_err</td>
<td>Number of frame alignment errors.</td>
</tr>
<tr>
<td>rx_over_err</td>
<td>Number of packet with RX overrun.</td>
</tr>
<tr>
<td>rx_crc_err</td>
<td>Number of CRC errors.</td>
</tr>
<tr>
<td>collisions</td>
<td>Number of collisions.</td>
</tr>
</tbody>
</table>

Example:

```python
@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 ' %
                      (stat.port_no,)
                      + '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)
```

```python
class ryu.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.

```python
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero (none yet defined in the spec)</td>
</tr>
<tr>
<td>port_no</td>
<td>Port number to read (All ports if OFPT_ALL)</td>
</tr>
<tr>
<td>queue_id</td>
<td>ID of queue to read (All queues if OFPQ_ALL)</td>
</tr>
</tbody>
</table>
```

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 ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>port_no</td>
<td>Port number.</td>
</tr>
<tr>
<td>queue_id</td>
<td>ID of queue.</td>
</tr>
<tr>
<td>tx_bytes</td>
<td>Number of transmitted bytes.</td>
</tr>
<tr>
<td>tx_packets</td>
<td>Number of transmitted packets.</td>
</tr>
<tr>
<td>tx_errors</td>
<td>Number of packets dropped due to overrun.</td>
</tr>
</tbody>
</table>

Example:

```python
@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 ' %
                       (stat.port_no, stat.queue_id,)
                       'tx_bytes=%d tx_packets=%d tx_errors=%d ' %
                       (stat.tx_bytes, stat.tx_packets, stat.tx_errors))

    self.logger.debug('QueueStats: %s', queues)
```

class ryu.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 ryu.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 ryu.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.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>buffer_id</td>
<td>ID assigned by datapath (0xffffffff if none).</td>
</tr>
<tr>
<td>in_port</td>
<td>Packet’s input port (OFPP_NONE if none).</td>
</tr>
<tr>
<td>actions</td>
<td>ist of OFPAction* instance.</td>
</tr>
<tr>
<td>data</td>
<td>Packet data of a binary type value or an instances of packet.Packet.</td>
</tr>
</tbody>
</table>

Example:

```python
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:

```json
{
    "OFPPacketOut": {
        "actions": [
            {
                "OFPActionOutput": {
                    "max_len": 65535,
                    "port": 65532
                }
            }
        ],
        "buffer_id": 4294967295,
        "data": "8guk0D9w8gukffjqCABFAABU+BoAAP8Br4sKAAABCgAAAgAAlj3YAAAMdYCAAAAAACrjS0xAAAAABAPEmUFRYXGBkaGxwdHh8gISIjJCUmJygpKissLS4vAAAAAAAAAAA=",
        "in_port": 65533
    }
}
```

**Barrier Message**

class ryu.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:

```python
def send_barrier_request(self, datapath):
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPBarrierRequest(datapath)
    datapath.send_msg(req)
```
**class** ryu.ofproto.ofproto_v1_0_parser.OFPBarrierReply(*datapath*)

Barrier reply message

The switch responds with this message to a barrier request.

Example:

```python
@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** ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>buffer_id</td>
<td>ID assigned by datapath.</td>
</tr>
<tr>
<td>total_len</td>
<td>Full length of frame.</td>
</tr>
<tr>
<td>in_port</td>
<td>Port on which frame was received.</td>
</tr>
<tr>
<td>reason</td>
<td>Reason packet is being sent.</td>
</tr>
<tr>
<td></td>
<td>OFPR_NO_MATCH</td>
</tr>
<tr>
<td></td>
<td>OFPR_ACTION</td>
</tr>
<tr>
<td></td>
<td>OFPR_INVALID_TTL</td>
</tr>
</tbody>
</table>

| data | Ethernet frame. |

Example:

```python
@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',
```

(continues on next page)
msg.buffer_id, msg.total_len, msg.in_port,
reason, utils.hex_array(msg.data))

JSON Example:

```json
{
  "OFPPacketIn": {
    "buffer_id": 2,
    "data": "////////8gukffjqCAYAAQgABgQAAfILpH346goAAEESAAAEEEKAAAAAD",
    "in_port": 99,
    "reason": 1,
    "total_len": 42
  }
}
```

**Flow Removed Message**

class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>match</td>
<td>Instance of OFPMatch.</td>
</tr>
<tr>
<td>cookie</td>
<td>Opaque controller-issued identifier.</td>
</tr>
<tr>
<td>priority</td>
<td>Priority level of flow entry.</td>
</tr>
<tr>
<td>reason</td>
<td>One of the following values.</td>
</tr>
<tr>
<td></td>
<td>OFPRR_IDLE_TIMEOUT</td>
</tr>
<tr>
<td></td>
<td>OFPRR_HARD_TIMEOUT</td>
</tr>
<tr>
<td></td>
<td>OFPRR_DELETE</td>
</tr>
<tr>
<td>duration_sec</td>
<td>Time flow was alive in seconds.</td>
</tr>
<tr>
<td>duration_nsec</td>
<td>Time flow was alive in nanoseconds beyond duration_sec.</td>
</tr>
<tr>
<td>idle_timeout</td>
<td>Idle timeout from original flow mod.</td>
</tr>
<tr>
<td>packet_count</td>
<td>Number of packets that was associated with the flow.</td>
</tr>
<tr>
<td>byte_count</td>
<td>Number of bytes that was associated with the flow.</td>
</tr>
</tbody>
</table>

Example:

```python
@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'
```

(continues on next page)
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)

Port Status Message

class ryu.ofproto.ofproto_v1_0_parser.OFPPortStatus(datapath, reason=None, desc=None)

Port status message

The switch notifies controller of change of ports.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reason</td>
<td>One of the following values.</td>
</tr>
<tr>
<td></td>
<td>OFPPR_ADD</td>
</tr>
<tr>
<td></td>
<td>OFPPR_DELETE</td>
</tr>
<tr>
<td></td>
<td>OFPPR_MODIFY</td>
</tr>
<tr>
<td>desc</td>
<td>instance of OFPPhyPort</td>
</tr>
</tbody>
</table>

Example:

```python
@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

```python
class ryu.ofproto.ofproto_v1_0_parser.OFPErrorMsg(datapath, type_=None, code=None, data=None)
```

Error message

The switch notifies controller of problems by this message.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>High level type of error</td>
</tr>
<tr>
<td>code</td>
<td>Details depending on the type</td>
</tr>
<tr>
<td>data</td>
<td>Variable length data depending on the type and code</td>
</tr>
</tbody>
</table>

`type` attribute corresponds to `type_` parameter of `__init__`.

Types and codes are defined in `ryu.ofproto.ofproto`.

```
<table>
<thead>
<tr>
<th>Type</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>OFPET_HELLO_FAILED</td>
<td>OFPHFC_*</td>
</tr>
<tr>
<td>OFPET_BAD_REQUEST</td>
<td>OFPBRC_*</td>
</tr>
<tr>
<td>OFPET_BAD_ACTION</td>
<td>OFPBAC_*</td>
</tr>
<tr>
<td>OFPET_FLOW_MOD_FAILED</td>
<td>OFPFMFC_*</td>
</tr>
<tr>
<td>OFPET_PORT_MOD_FAILED</td>
<td>OFPPMFC_*</td>
</tr>
<tr>
<td>OFPET_QUEUE_OP_FAILED</td>
<td>OFPQOFC_*</td>
</tr>
</tbody>
</table>
```

Example:

```python
@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

```python
class ryu.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 Ryu framework, so the Ryu application do not need to process this typically.

Echo Request

```python
class ryu.ofproto.ofproto_v1_0_parser.OFPEchoRequest(datapath, data=None)
```

Echo request message

This message is handled by the Ryu framework, so the Ryu application do not need to process this typically.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>An arbitrary length data.</td>
</tr>
</tbody>
</table>

Example:
```python
def send_echo_request(self, datapath, data):
ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPEchoRequest(datapath, data)
datapath.send_msg(req)
```

**Echo Reply**

```python
class ryu.ofproto.ofproto_v1_0_parser.OFPEchoReply (datapath, data=None)
```

Echo reply message

This message is handled by the Ryu framework, so the Ryu application do not need to process this typically.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>An arbitrary length data.</td>
</tr>
</tbody>
</table>

Example:
```python
@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**

```python
class ryu.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**

```python
class ryu.ofproto.ofproto_v1_0_parser.OFPPhyPort
```

Description of a port
### Attribute Description

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>port_no</td>
<td>Port number and it uniquely identifies a port within a switch.</td>
</tr>
<tr>
<td>hw_addr</td>
<td>MAC address for the port.</td>
</tr>
<tr>
<td>name</td>
<td>Null-terminated string containing a human-readable name for the interface.</td>
</tr>
<tr>
<td>config</td>
<td>Bitmap of port configuration flags.</td>
</tr>
<tr>
<td></td>
<td>OFPPC_PORT_DOWN</td>
</tr>
<tr>
<td></td>
<td>OFPPC_NO_STP</td>
</tr>
<tr>
<td></td>
<td>OFPPC_NO_RECV</td>
</tr>
<tr>
<td></td>
<td>OFPPC_NO_RECV_STP</td>
</tr>
<tr>
<td></td>
<td>OFPPC_NO_FLOOD</td>
</tr>
<tr>
<td></td>
<td>OFPPC_NO_FWD</td>
</tr>
<tr>
<td></td>
<td>OFPPC_NO_PACKET_IN</td>
</tr>
<tr>
<td>state</td>
<td>Bitmap of port state flags.</td>
</tr>
<tr>
<td></td>
<td>OFPPS_LINK_DOWN</td>
</tr>
<tr>
<td></td>
<td>OFPPS_STP_LISTEN</td>
</tr>
<tr>
<td></td>
<td>OFPPS_STP_LEARN</td>
</tr>
<tr>
<td></td>
<td>OFPPS_STP_FORWARD</td>
</tr>
<tr>
<td></td>
<td>OFPPS_STP_BLOCK</td>
</tr>
<tr>
<td></td>
<td>OFPPS_STP_MASK</td>
</tr>
<tr>
<td>curr</td>
<td>Current features.</td>
</tr>
<tr>
<td>advertised</td>
<td>Features being advertised by the port.</td>
</tr>
<tr>
<td>supported</td>
<td>Features supported by the port.</td>
</tr>
<tr>
<td>peer</td>
<td>Features advertised by peer.</td>
</tr>
</tbody>
</table>

### Flow Match Structure

```python
class ryu.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_proto=None, nw_src=None, nw_dst=None, tp_src=None, tp_dst=None, nw_proto_mask=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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wildcards</td>
<td>Wildcard fields.</td>
</tr>
<tr>
<td>(match fields)</td>
<td>For the available match fields, please refer to the following.</td>
</tr>
<tr>
<td>Argument</td>
<td>Value</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>in_port</td>
<td>Integer 16bit</td>
</tr>
<tr>
<td>dl_src</td>
<td>MAC address</td>
</tr>
<tr>
<td>dl_dst</td>
<td>MAC address</td>
</tr>
<tr>
<td>dl_vlan</td>
<td>Integer 16bit</td>
</tr>
<tr>
<td>dl_vlan_pcp</td>
<td>Integer 8bit</td>
</tr>
<tr>
<td>dl_type</td>
<td>Integer 16bit</td>
</tr>
<tr>
<td>nw_tos</td>
<td>Integer 8bit</td>
</tr>
<tr>
<td>nw_proto</td>
<td>Integer 8bit</td>
</tr>
<tr>
<td>nw_src</td>
<td>IPv4 address</td>
</tr>
<tr>
<td>nw_dst</td>
<td>IPv4 address</td>
</tr>
<tr>
<td>tp_src</td>
<td>Integer 16bit</td>
</tr>
<tr>
<td>tp_dst</td>
<td>Integer 16bit</td>
</tr>
<tr>
<td>nw_src_mask</td>
<td>Integer 8bit</td>
</tr>
<tr>
<td>nw_dst_mask</td>
<td>Integer 8bit</td>
</tr>
</tbody>
</table>

Example:

```python
>>> # compose
>>> match = parser.OFPMatch(  
    ... in_port=1,
    ... dl_type=0x0800,
    ... dl_src='aa:bb:cc:dd:ee:ff',
    ... nw_src='192.168.0.1')
```

```python
>>> # query
>>> if 'nw_src' in match:
    ...     print match['nw_src']
    ...
'192.168.0.1'
```

**Action Structures**

```python
class ryu.ofproto.ofproto_v1_0_parser.OFPActionHeader(type_, len_)
class ryu.ofproto.ofproto_v1_0_parser.OFPAction
```

```python
class ryu.ofproto.ofproto_v1_0_parser.OFPActionOutput (port, max_len=65509)
```

Output action

This action indicates output a packet to the switch port.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>port</td>
<td>Output port.</td>
</tr>
<tr>
<td>max_len</td>
<td>Max length to send to controller.</td>
</tr>
</tbody>
</table>

**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 of1.2 and of1.3 is used.

```python
class ryu.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.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vlan_vid</td>
<td>VLAN id.</td>
</tr>
</tbody>
</table>

```python
class ryu.ofproto.ofproto_v1_0_parser.OFPActionVlanPcp(vlan_pcp)
    Set the 802.1q priority action
    This action indicates the 802.1q priority to be set.
```

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vlan_pcp</td>
<td>VLAN priority.</td>
</tr>
</tbody>
</table>

```python
class ryu.ofproto.ofproto_v1_0_parser.OFPActionStripVlan
    Strip the 802.1q header action
    This action indicates the 802.1q priority to be striped.
```

```python
class ryu.ofproto.ofproto_v1_0_parser.OFPActionDlAddr(dl_addr)
class ryu.ofproto.ofproto_v1_0_parser.OFPActionSetDlSrc(dl_addr)
    Set the ethernet source address action
    This action indicates the ethernet source address to be set.
```

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dl_addr</td>
<td>Ethernet address.</td>
</tr>
</tbody>
</table>

```python
class ryu.ofproto.ofproto_v1_0_parser.OFPActionSetDlDst(dl_addr)
    Set the ethernet destination address action
    This action indicates the ethernet destination address to be set.
```

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dl_addr</td>
<td>Ethernet address.</td>
</tr>
</tbody>
</table>

```python
class ryu.ofproto.ofproto_v1_0_parser.OFPActionNwAddr(nw_addr)
class ryu.ofproto.ofproto_v1_0_parser.OFPActionSetNwSrc(nw_addr)
    Set the IP source address action
    This action indicates the IP source address to be set.
```

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nw_addr</td>
<td>IP address.</td>
</tr>
</tbody>
</table>

```python
class ryu.ofproto.ofproto_v1_0_parser.OFPActionSetNwDst(nw_addr)
    Set the IP destination address action
    This action indicates the IP destination address to be set.
```

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nw_addr</td>
<td>IP address.</td>
</tr>
</tbody>
</table>

```python
class ryu.ofproto.ofproto_v1_0_parser.OFPActionSetNwTos(tos)
    Set the IP ToS action
```

Chapter 2. Writing Your Ryu Application
This action indicates the IP ToS (DSCP field, 6 bits) to be set.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tos</td>
<td>IP ToS (DSCP field, 6 bits).</td>
</tr>
</tbody>
</table>

```python
class ryu.ofproto.ofproto_v1_0_parser.OFPActionTpPort(tp)
```

This action indicates the TCP/UDP source port to be set.

```python
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tp</td>
<td>TCP/UDP port.</td>
</tr>
</tbody>
</table>

```python
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tp</td>
<td>TCP/UDP port.</td>
</tr>
</tbody>
</table>

```python
class ryu.ofproto.ofproto_v1_0_parser.OFPActionEnqueue(port, queue_id)
```

Output to queue action

This action indicates send packets to given queue on port.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>port</td>
<td>Port that queue belongs.</td>
</tr>
<tr>
<td>queue_id</td>
<td>Where to enqueue the packets.</td>
</tr>
</tbody>
</table>

```python
class ryu.ofproto.ofproto_v1_0_parser.OFPActionVendor(vendor=None)
```

Vendor action

This action is an extensible action for the vendor.

### 2.5.3 OpenFlow v1.2 Messages and Structures

#### Controller-to-Switch Messages

##### Handshake

```python
class ryu.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 Ryu framework, so the Ryu application do not need to process this typically.

Example:
```python
def send_features_request(self, datapath):
    ofp_parser = datapath.ofproto_parser
    req = ofp_parser.OFPFeaturesRequest(datapath)
    datapath.send_msg(req)
```

JSON Example:
```
{
    "OFPFeaturesRequest": {}
}
```

class ryu.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 Ryu framework, so the Ryu 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
                }
            }
        }
    }
}
```

(continues on next page)
Switch Configuration

```python
class ryu.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 configuration parameters.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>One of the following configuration flags.</td>
</tr>
<tr>
<td>miss_send_len</td>
<td>Max bytes of new flow that datapath should send to the controller</td>
</tr>
</tbody>
</table>

Example:

```python
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:

```json
{
    "OFPSetConfig": {
    
    }
"
```
class ryu.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:

```python
def send_get_config_request(self, datapath):
    ofp_parser = datapath.ofproto_parser
    req = ofp_parser.OFPGetConfigRequest(datapath)
    datapath.send_msg(req)
```

JSON Example:

```json
{}
```

class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>One of the following configuration flags.</td>
</tr>
<tr>
<td>OFPC_FRAG_NORMAL</td>
<td></td>
</tr>
<tr>
<td>OFPC_FRAG_DROP</td>
<td></td>
</tr>
<tr>
<td>OFPC_FRAG_REASM</td>
<td></td>
</tr>
<tr>
<td>OFPC_FRAG_MASK</td>
<td></td>
</tr>
<tr>
<td>OFPC_INVALID_TTL_TO_CONTROLLER</td>
<td></td>
</tr>
<tr>
<td>miss_send_len</td>
<td>Max bytes of new flow that datapath should send to the controller</td>
</tr>
</tbody>
</table>

Example:

```python
@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:
```

(continues on next page)
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('OFPGetConfigReply received: flags=%s, miss_send_len=%d', flags, msg.miss_send_len)

JSON Example:

```json
{
    "OFPGetConfigReply": {
        "flags": 0,
        "miss_send_len": 128
    }
}
```

**Flow Table Configuration**

```python
class ryu.ofproto.ofproto_v1_2_parser.OFPTableMod(datapath, table_id, config):
    Flow table configuration message

    The controller sends this message to configure table state.
```

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_id</td>
<td>ID of the table (OFPTT_ALL indicates all tables)</td>
</tr>
<tr>
<td>config</td>
<td>Bitmap of the following flags.</td>
</tr>
</tbody>
</table>

- OFPTC_TABLE_MISS_CONTROLLER
- OFPTC_TABLE_MISS_CONTINUE
- OFPTC_TABLE_MISS_DROP
- OFPTC_TABLE_MISS_MASK

Example:

```python
def send_table_mod(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPTableMod(datapath, ofp.OFPTT_ALL,
                                  ofp.OFPTC_TABLE_MISS_DROP)
    datapath.send_msg(req)
```

JSON Example:

```json
{
    "OFPTableMod": {
        "config": 0,
        "table_id": 255
    }
}
```
Modify State Messages

class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cookie</td>
<td>Opaque controller-issued identifier</td>
</tr>
<tr>
<td>cookie_mask</td>
<td>Mask used to restrict the cookie bits that must match when the command is</td>
</tr>
<tr>
<td></td>
<td>OFPFC_MODIFY* or OFPFC_DELETE*</td>
</tr>
<tr>
<td>table_id</td>
<td>ID of the table to put the flow in</td>
</tr>
<tr>
<td>command</td>
<td>One of the following values.</td>
</tr>
<tr>
<td></td>
<td>OFPFC_ADD</td>
</tr>
<tr>
<td></td>
<td>OFPFC_MODIFY</td>
</tr>
<tr>
<td></td>
<td>OFPFC_MODIFY STRICT</td>
</tr>
<tr>
<td></td>
<td>OFPFC_DELETE</td>
</tr>
<tr>
<td></td>
<td>OFPFC_DELETE STRICT</td>
</tr>
<tr>
<td>idle_timeout</td>
<td>Idle time before discarding (seconds)</td>
</tr>
<tr>
<td>hard_timeout</td>
<td>Max time before discarding (seconds)</td>
</tr>
<tr>
<td>priority</td>
<td>Priority level of flow entry</td>
</tr>
<tr>
<td>buffer_id</td>
<td>Buffered packet to apply to (or OFP_NO_BUFFER)</td>
</tr>
<tr>
<td>out_port</td>
<td>For OFPFC_DELETE* commands, require matching entries to include this as an</td>
</tr>
<tr>
<td></td>
<td>output port</td>
</tr>
<tr>
<td>out_group</td>
<td>For OFPFC_DELETE* commands, require matching entries to include this as an</td>
</tr>
<tr>
<td></td>
<td>output group</td>
</tr>
<tr>
<td>flags</td>
<td>One of the following values.</td>
</tr>
<tr>
<td></td>
<td>OFPFF_SEND_FLOW_REM</td>
</tr>
<tr>
<td></td>
<td>OFPFF_CHECK_OVERLAP</td>
</tr>
<tr>
<td></td>
<td>OFPFF_RESET_COUNTS</td>
</tr>
<tr>
<td>match</td>
<td>Instance of OFPMatch</td>
</tr>
<tr>
<td>instructions</td>
<td>list of OFPInstruction* instance</td>
</tr>
</tbody>
</table>

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,
                                        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:

```json
{
    "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
                    }
                    },
                    {
                        "OFPActionOutput": {
                            "len": 16,
                            "max_len": 65535,
                            "port": 6,
                            "type": 0
                        }
                    }
                }
            }
        ]
    }
}
```
"len": 40,
"type": 3
},
},
"OFPInstructionActions": {
"actions": [
{
"OFPActionSetField": {
"field": {
"OXMField": {
"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": [
{
"OXMField": {
"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
}
}
"idle_timeout": 0,
"instructions": [
  {
    "OFPInstructionGotoTable": {
      "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 ryu.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.

2.5. OpenFlow protocol API Reference 171
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>command</td>
<td>One of the following values.</td>
</tr>
<tr>
<td></td>
<td>OFPGC_ADD</td>
</tr>
<tr>
<td></td>
<td>OFPGC_MODIFY</td>
</tr>
<tr>
<td></td>
<td>OFPGC_DELETE</td>
</tr>
<tr>
<td>type</td>
<td>One of the following values.</td>
</tr>
<tr>
<td></td>
<td>OFPGT_ALL</td>
</tr>
<tr>
<td></td>
<td>OFPGT_SELECT</td>
</tr>
<tr>
<td></td>
<td>OFPGT_INDIRECT</td>
</tr>
<tr>
<td></td>
<td>OFPGT_FF</td>
</tr>
<tr>
<td>group_id</td>
<td>Group identifier</td>
</tr>
<tr>
<td>buckets</td>
<td>list of OFPBucket</td>
</tr>
</tbody>
</table>

The `type` attribute corresponds to the `type_` parameter of `__init__`.

Example:

```python
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:

```json
{
    "OFPGroupMod": {
        "buckets": [
            "OFPBucket": {
                "actions": [
                    "OFPActionOutput": {
                        "len": 16,
                        "max_len": 65535,
                        "port": 2,
                        "type": 0
                    }
                ]
            }
        ]
    }
}
```
(continues on next page)
Port modification message

The controller sends this message to modify the behavior of the port.

```
class ryu.ofproto.ofproto_v1_2_parser.OFPPortMod (datapath, port_no=0, hw_addr='00:00:00:00:00:00', config=0, mask=0, advertise=0)
```
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>port_no</td>
<td>Port number to modify</td>
</tr>
<tr>
<td>hw_addr</td>
<td>The hardware address that must be the same as hw_addr of OFPPort of OFPSwitchFeatures</td>
</tr>
<tr>
<td>config</td>
<td>Bitmap of configuration flags.</td>
</tr>
<tr>
<td></td>
<td>OFPPC_PORT_DOWN</td>
</tr>
<tr>
<td></td>
<td>OFPPC_NO_RECV</td>
</tr>
<tr>
<td></td>
<td>OFPPC_NO_FWD</td>
</tr>
<tr>
<td></td>
<td>OFPPC_NO_PACKET_IN</td>
</tr>
<tr>
<td>mask</td>
<td>Bitmap of configuration flags above to be changed</td>
</tr>
<tr>
<td>advertise</td>
<td>Bitmap of the following flags.</td>
</tr>
<tr>
<td></td>
<td>OFPPF_10MB_HD</td>
</tr>
<tr>
<td></td>
<td>OFPPF_10MB_FD</td>
</tr>
<tr>
<td></td>
<td>OFPPF_100MB_HD</td>
</tr>
<tr>
<td></td>
<td>OFPPF_100MB_FD</td>
</tr>
<tr>
<td></td>
<td>OFPPF_1GB_HD</td>
</tr>
<tr>
<td></td>
<td>OFPPF_1GB_FD</td>
</tr>
<tr>
<td></td>
<td>OFPPF_10GB_FD</td>
</tr>
<tr>
<td></td>
<td>OFPPF_40GB_FD</td>
</tr>
<tr>
<td></td>
<td>OFPPF_100GB_FD</td>
</tr>
<tr>
<td></td>
<td>OFPPF_1TB_FD</td>
</tr>
<tr>
<td></td>
<td>OFPPF_OTHER</td>
</tr>
<tr>
<td></td>
<td>OFPPF_COPPER</td>
</tr>
<tr>
<td></td>
<td>OFPPF_FIBER</td>
</tr>
<tr>
<td></td>
<td>OFPPF_AUTONEG</td>
</tr>
<tr>
<td></td>
<td>OFPPF_PAUSE</td>
</tr>
<tr>
<td></td>
<td>OFPPF_PAUSEASYM</td>
</tr>
</tbody>
</table>

Example:

```python
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 |
                  ofp.OFPPF_AUTONEG | ofp.OFPPF_PAUSE |
                  ofp.OFPPF_PAUSEASYM)
    req = ofp_parser.OFPPortMod(datapath, port_no, hw_addr, config,
                                  mask, advertise)
    datapath.send_msg(req)
```

JSON Example:
```json
{
  "OFPPortMod": {
    "advertise": 4096,
    "config": 0,
    "hw_addr": "00-11-00-00-11-11",
    "mask": 0,
    "port_no": 1
  }
}
```

### Read State Messages

**class** [ryu.ofproto.ofproto_v1_2_parser.OFPDescStatsRequest](https://github.com/osrg/ryu/tree/master/ryu/ofproto/ofproto_v1_2_parser.py) (`datapath, flags=0`)

Description statistics request message

The controller uses this message to query description of the switch.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero (none yet defined in the spec)</td>
</tr>
</tbody>
</table>

**Example:**

```python
def send_desc_stats_request(self, datapath):
    ofp_parser = datapath.ofproto_parser
    req = ofp_parser.OFPDescStatsRequest(datapath)
    datapath.send_msg(req)
```

**JSON Example:**

```json
{
  "OFPDescStatsRequest": {
    "flags": 0
  }
}
```

**class** [ryu.ofproto.ofproto_v1_2_parser.OFPDescStats](https://github.com/osrg/ryu/tree/master/ryu/ofproto/ofproto_v1_2_parser.py)

Description statistics reply message

The switch responds with a stats reply that include this message to a description statistics request.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mfr_desc</td>
<td>Manufacturer description</td>
</tr>
<tr>
<td>hw_desc</td>
<td>Hardware description</td>
</tr>
<tr>
<td>sw_desc</td>
<td>Software description</td>
</tr>
<tr>
<td>serial_num</td>
<td>Serial number</td>
</tr>
<tr>
<td>dp_desc</td>
<td>Human readable description of datapath</td>
</tr>
</tbody>
</table>

**Example:**

```python
@set_ev_cls(ofp_event.EventOFPStatsReply, MAIN_DISPATCHER)
def stats_reply_handler(self, ev):
    msg = ev.msg
    # Handle stats reply message here
```

(continues on next page)
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:

```json
{
    "OFPStatsReply": {
        "body": {
            "OFPDescStats": {
                "dp_desc": "dp",
                "hw_desc": "hw",
                "mfr_desc": "mfr",
                "serial_num": "serial",
                "sw_desc": "sw"
            }
        },
        "flags": 0,
        "type": 0
    }
}
```

class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_id</td>
<td>ID of table to read</td>
</tr>
<tr>
<td>out_port</td>
<td>Require matching entries to include this as an output port</td>
</tr>
<tr>
<td>out_group</td>
<td>Require matching entries to include this as an output group</td>
</tr>
<tr>
<td>cookie</td>
<td>Require matching entries to contain this cookie value</td>
</tr>
<tr>
<td>cookie_mask</td>
<td>Mask used to restrict the cookie bits that must match</td>
</tr>
<tr>
<td>match</td>
<td>Instance of OFPMatch</td>
</tr>
<tr>
<td>flags</td>
<td>Zero (none yet defined in the spec)</td>
</tr>
</tbody>
</table>

Example:

```python
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:

```json
{
    "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
    }
}
```

```python
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_id</td>
<td>ID of table flow came from</td>
</tr>
<tr>
<td>duration_sec</td>
<td>Time flow has been alive in seconds</td>
</tr>
<tr>
<td>duration_nsec</td>
<td>Time flow has been alive in nanoseconds beyond duration_sec</td>
</tr>
<tr>
<td>priority</td>
<td>Priority of the entry</td>
</tr>
<tr>
<td>idle_timeout</td>
<td>Number of seconds idle before expiration</td>
</tr>
<tr>
<td>hard_timeout</td>
<td>Number of seconds before expiration</td>
</tr>
<tr>
<td>cookie</td>
<td>Opaque controller-issued identifier</td>
</tr>
<tr>
<td>packet_count</td>
<td>Number of packets in flow</td>
</tr>
<tr>
<td>byte_count</td>
<td>Number of bytes in flow</td>
</tr>
<tr>
<td>match</td>
<td>Instance of OFPMatch</td>
</tr>
<tr>
<td>instructions</td>
<td>list of OFPIInstruction* instance</td>
</tr>
</tbody>
</table>

Example:
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 = [
        'table_id=%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,
                           stat.match, stat.instructions))
    self.logger.debug('FlowStats: %s', flows)

JSON Example:

```json
{
    "OFPStatsReply": {
        "body": [
            {
                "OFPFlowStats": {
                    "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
                }
            },
            {
                "OFPFlowStats": {
                    "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
          }
        }
      ],
      "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": [
    {
      "OFPInstructionGotoTable": {
        "len": 8,
        "table_id": 1,
        "type": 1
      }
    }
  ]
}
"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,
  "hard_timeout": 0,
  "idle_timeout": 0,
  "instructions": [
    {
      "OFPInstructionActions": {
        "actions": [
          {
            "OFPActionOutput": {
              "len": 16,
              "max_len": 65535,
              "port": 4294967293,
              "type": 0
            }
          }
        ],
        "len": 24,
        "type": 3
      }
    }
  ]
};
Aggregate flow statistics request message

The controller uses this message to query aggregate flow statictics.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_id</td>
<td>ID of table to read</td>
</tr>
<tr>
<td>out_port</td>
<td>Require matching entries to include this as an output port</td>
</tr>
<tr>
<td>out_group</td>
<td>Require matching entries to include this as an output group</td>
</tr>
<tr>
<td>cookie</td>
<td>Require matching entries to contain this cookie value</td>
</tr>
<tr>
<td>cookie_mask</td>
<td>Mask used to restrict the cookie bits that must match</td>
</tr>
<tr>
<td>match</td>
<td>Instance of OFPMatch</td>
</tr>
<tr>
<td>flags</td>
<td>Zero (none yet defined in the spec)</td>
</tr>
</tbody>
</table>

Example:

```python
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:

```json
{
    "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 ryu.ofproto.ofproto_v1_2_parser.OFPAggregateStatsReply

Aggregate flow statistics reply message

The switch responds with a stats reply that include this message to an aggregate flow statistics request.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>packet_count</td>
<td>Number of packets in flows</td>
</tr>
<tr>
<td>byte_count</td>
<td>Number of bytes in flows</td>
</tr>
<tr>
<td>flow_count</td>
<td>Number of flows</td>
</tr>
</tbody>
</table>

Example:

```python
@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_AGGREGATE:
        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:

```json
{
    "OFPStatsReply": {
        "body": {
            "OFPAggregateStatsReply": {
                "byte_count": 574,
                "flow_count": 6,
                "packet_count": 7
            }
        }
    }
}
```
class ryu.ofproto.ofproto_v1_2_parser.OFPTableStatsRequest (datapath, flags=0)
Table statistics request message

The controller uses this message to query flow table statistics.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero (none yet defined in the spec)</td>
</tr>
</tbody>
</table>

Example:

```python
def send_table_stats_request(self, datapath):
    ofp_parser = datapath.ofproto_parser
    req = ofp_parser.OFPTableStatsRequest(datapath)
    datapath.send_msg(req)
```

JSON Example:

```json
{
   "OFPTableStatsRequest": {
      "flags": 0
   }
}
```

class ryu.ofproto.ofproto_v1_2_parser.OFPTTableStats
Table statistics reply message

The switch responds with a stats reply that include this message to a table statistics request.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_id</td>
<td>ID of table</td>
</tr>
<tr>
<td>name</td>
<td>table name</td>
</tr>
<tr>
<td>match</td>
<td>Bitmap of (1 &lt;&lt; OFPXMT_*) that indicate the fields the table can match on</td>
</tr>
<tr>
<td>wildcards</td>
<td>Bitmap of (1 &lt;&lt; OFPXMT_*) wildcards that are supported by the table</td>
</tr>
<tr>
<td>write_actions</td>
<td>Bitmap of OFPAT_* that are supported by the table with OFPIT_WRITE_ACTIONS</td>
</tr>
<tr>
<td>apply_actions</td>
<td>Bitmap of OFPAT_* that are supported by the table with OFPIT_APPLY_ACTIONS</td>
</tr>
<tr>
<td>write_setfields</td>
<td>Bitmap of (1 &lt;&lt; OFPXMT_*) header fields that can be set with OFPIT_WRITE_ACTIONS</td>
</tr>
<tr>
<td>apply_setfields</td>
<td>Bitmap of (1 &lt;&lt; OFPXMT_*) header fields that can be set with OFPIT_APPLY_ACTIONS</td>
</tr>
<tr>
<td>metadata_match</td>
<td>Bits of metadata table can match</td>
</tr>
<tr>
<td>metadata_write</td>
<td>Bits of metadata table can write</td>
</tr>
<tr>
<td>instructions</td>
<td>Bitmap of OFPAT_* values supported</td>
</tr>
<tr>
<td>config</td>
<td>Bitmap of OFPTC_* values</td>
</tr>
<tr>
<td>max_entries</td>
<td>Max number of entries supported</td>
</tr>
<tr>
<td>active_count</td>
<td>Number of active entries</td>
</tr>
<tr>
<td>lookup_count</td>
<td>Number of packets looked up in table</td>
</tr>
<tr>
<td>matched_count</td>
<td>Number of packets that hit table</td>
</tr>
</tbody>
</table>

Example:

```python
@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 = []
    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)
```

### Port statistics request message

The controller uses this message to query information about ports statistics.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>port_no</td>
<td>Port number to read (OFPP ANY to all ports)</td>
</tr>
<tr>
<td>flags</td>
<td>Zero (none yet defined in the spec)</td>
</tr>
</tbody>
</table>

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:

```json
{
    "OFPPortStatsRequest": {
        "flags": 0,
        "port_no": 4294967295
    }
}
```

class ryu.ofproto.ofproto_v1_2_parser.OFPPortStats
Port statistics reply message

The switch responds with a stats reply that include this message to a port statistics request.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>port_no</td>
<td>Port number</td>
</tr>
<tr>
<td>rx_packets</td>
<td>Number of received packets</td>
</tr>
<tr>
<td>tx_packets</td>
<td>Number of transmitted packets</td>
</tr>
<tr>
<td>rx_bytes</td>
<td>Number of received bytes</td>
</tr>
<tr>
<td>tx_bytes</td>
<td>Number of transmitted bytes</td>
</tr>
<tr>
<td>rx_dropped</td>
<td>Number of packets dropped by RX</td>
</tr>
<tr>
<td>tx_dropped</td>
<td>Number of packets dropped by TX</td>
</tr>
<tr>
<td>rx_errors</td>
<td>Number of receive errors</td>
</tr>
<tr>
<td>tx_errors</td>
<td>Number of transmit errors</td>
</tr>
<tr>
<td>rx_frame_err</td>
<td>Number of frame alignment errors</td>
</tr>
<tr>
<td>rx_over_err</td>
<td>Number of packet with RX overrun</td>
</tr>
<tr>
<td>rx_crc_err</td>
<td>Number of CRC errors</td>
</tr>
<tr>
<td>collisions</td>
<td>Number of collisions</td>
</tr>
</tbody>
</table>

Example:

```python
@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 ' %stat.port_no +
                      'rx_packets=%d tx_packets=%d ' %stat.rx_packets +
                      'tx_packets=%d tx_bytes=%d ' %stat.tx_packets +
                      'rx_bytes=%d tx_bytes=%d ' %stat.rx_bytes +
                      'rx_dropped=%d tx_dropped=%d ' %stat.rx_dropped +
                      'tx_dropped=%d tx_errors=%d ' %stat.tx_dropped +
                      'rx_errors=%d tx_errors=%d ' %stat.rx_errors +
                      'rx_frame_err=%d rx_over_err=%d ' %stat.rx_frame_err +
                      'rx_over_err=%d rx_crc_err=%d ' %stat.rx_over_err +
                      'collisions=%d' %stat.collisions)```

(continues on next page)
```python
self.logger.debug("PortStats: %s", ports)
```

JSON Example:

```json
{
    "OFPStatsReply": {
        "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 ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>port_no</td>
<td>Port number to read</td>
</tr>
<tr>
<td>queue_id</td>
<td>ID of queue to read</td>
</tr>
<tr>
<td>flags</td>
<td>Zero (none yet defined in the spec)</td>
</tr>
</tbody>
</table>

Example:

```python
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:

```json
{
    "OFPQueueStatsRequest": {
        "flags": 0,
        "port_no": 4294967295,
        "queue_id": 4294967295
    }
}
```

class ryu.ofproto.ofproto_v1_2_parser.OFPQueueStats

Queue statistics reply message

The switch responds with a stats reply that include this message to an aggregate flow statistics request.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>port_no</td>
<td>Port number</td>
</tr>
<tr>
<td>queue_id</td>
<td>ID of queue</td>
</tr>
<tr>
<td>tx_bytes</td>
<td>Number of transmitted bytes</td>
</tr>
<tr>
<td>tx_packets</td>
<td>Number of transmitted packets</td>
</tr>
<tr>
<td>tx_errors</td>
<td>Number of packets dropped due to overrun</td>
</tr>
</tbody>
</table>

Example:

```python
@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:
        self.queue_stats_reply_handler(body)
```

(continues on next page)
def queue_stats_reply_handler(self, body):
    queues = []
    for stat in body:
        queues.append('port_no=%d queue_id=%d ' %
                       (stat.port_no, stat.queue_id,
                        stat.tx_bytes, stat.tx_packets, stat.tx_errors))
    self.logger.debug('QueueStats: %s', queues)

JSON Example:

```json
{
    "OFPStatsReply": {
        "body": [
            {
                "OFPQueueStats": {
                    "port_no": 7,
                    "queue_id": 1,
                    "tx_bytes": 0,
                    "tx_errors": 0,
                    "tx_packets": 0
                }
            },
            {
                "OFPQueueStats": {
                    "port_no": 6,
                    "queue_id": 1,
                    "tx_bytes": 0,
                    "tx_errors": 0,
                    "tx_packets": 0
                }
            },
            {
                "OFPQueueStats": {
                    "port_no": 7,
                    "queue_id": 2,
                    "tx_bytes": 0,
                    "tx_errors": 0,
                    "tx_packets": 0
                }
            },
            {
                "flags": 0,
                "type": 5
            }
        ]
    }
}
```

class ryu.ofproto.ofproto_v1_2_parser.OFPGroupStatsRequest (datapath,
                                                             group_id=4294967292,
                                                             flags=0)

Group statistics request message

The controller uses this message to query statistics of one or more groups.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>group_id</td>
<td>ID of group to read (OFPG_ALL to all groups)</td>
</tr>
<tr>
<td>flags</td>
<td>Zero (none yet defined in the spec)</td>
</tr>
</tbody>
</table>
Example:

```python
def send_group_stats_request(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPGroupStatsRequest(datapath, ofp.OFPG_ALL)
    datapath.send_msg(req)
```

class ryu.ofproto.ofproto_v1_2_parser.OFPGroupStats:
    
Group statistics reply message

The switch responds with a stats reply that include this message to a group statistics request.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>group_id</td>
<td>Group identifier</td>
</tr>
<tr>
<td>ref_count</td>
<td>Number of flows or groups that directly forward to this group</td>
</tr>
<tr>
<td>packet_count</td>
<td>Number of packets processed by group</td>
</tr>
<tr>
<td>byte_count</td>
<td>Number of bytes processed by group</td>
</tr>
<tr>
<td>bucket_counters</td>
<td>List of OFPBucketCounter instance</td>
</tr>
</tbody>
</table>

Example:

```python
@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:
        self.group_stats_reply_handler(body)

def group_stats_reply_handler(self, body):
    groups = []
    for stat in body:
        groups.append('group_id=%d ref_count=%d packet_count=%d' %
                       (stat.group_id, stat.ref_count, stat.packet_count, stat.byte_count, stat.bucket_counters))

    self.logger.debug('GroupStats: %s', groups)
```

class ryu.ofproto.ofproto_v1_2_parser.OFPGroupDescStatsRequest:

Group description request message

The controller uses this message to list the set of groups on a switch.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero (none yet defined in the spec)</td>
</tr>
</tbody>
</table>

Example:
```python
def send_group_desc_stats_request(self, datapath):
    ofp_parser = datapath.ofproto_parser
    req = ofp_parser.OFPGroupDescStatsRequest(datapath)
    datapath.send_msg(req)
```

JSON Example:

```json
{
    "OFPGroupDescStatsRequest": {
        "flags": 0
    }
}
```

```python
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>One of OFPGT_*</td>
</tr>
<tr>
<td>group_id</td>
<td>Group identifier</td>
</tr>
<tr>
<td>buckets</td>
<td>List of OFPBucket instance</td>
</tr>
</tbody>
</table>

`type` attribute corresponds to `type_` parameter of `__init__`.

Example:

```python
@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:

```json
{
    "OFPStatsReply": {
        "body": [
            {
                "OFPGroupDescStats": {
                    "buckets": [
                        {
                            "OFPBucket": {
                                "actions": [
```

(continues on next page)
"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,
"type": 7
}
}
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero (none yet defined in the spec)</td>
</tr>
</tbody>
</table>

Example:

```python
def send_group_features_stats_request(self, datapath):
    ofp_parser = datapath.ofproto_parser
    req = ofp_parser.OFPGroupFeaturesStatsRequest(datapath)
    datapath.send_msg(req)
```

JSON Example:

```json
{
    "OFPGroupFeaturesStatsRequest": {
        "flags": 0
    }
}
```

class ryu.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.

2.5. OpenFlow protocol API Reference
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>types</td>
<td>Bitmap of OFPGT_* values supported</td>
</tr>
<tr>
<td>capabilities</td>
<td>Bitmap of OFPGFC_* capability supported</td>
</tr>
<tr>
<td>max_groups</td>
<td>Maximum number of groups for each type</td>
</tr>
<tr>
<td>actions</td>
<td>Bitmaps of OFPAT_* that are supported</td>
</tr>
</tbody>
</table>

Example:

```python
@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_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:

```json
{
    "OFPStatsReply": {
        "body": {
            "OFPGroupFeaturesStats": {
                "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** ryu.ofproto.ofproto_v1_2_parser.OFPQueueGetConfigRequest (datapath, port)
Queue configuration request message

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>port</td>
<td>Port to be queried (OFPP_ANY to all configured queues)</td>
</tr>
</tbody>
</table>

Example:

```python
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:
```json
{
    "OFPQueueGetConfigRequest": {
        "port": 4294967295
    }
}
```

**class** ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>port</td>
<td>Port which was queried</td>
</tr>
<tr>
<td>queues</td>
<td>list of OFPPacketQueue instance</td>
</tr>
</tbody>
</table>

Example:
```python
@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:
```json
{
    "OFPQueueGetConfigReply": {
        "port": 4294967295,
        "queues": [
            {
                "OFPPacketQueue": {
                    "len": 48,
```

(continues on next page)
Packet-Out Message

class ryu.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.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>buffer_id</td>
<td>ID assigned by datapath (OFP_NO_BUFFER if none)</td>
</tr>
<tr>
<td>in_port</td>
<td>Packet’s input port or OFPP_CONTROLLER</td>
</tr>
<tr>
<td>actions</td>
<td>list of OpenFlow action class</td>
</tr>
<tr>
<td>data</td>
<td>Packet data of a binary type value or an instances of packet.Packet.</td>
</tr>
</tbody>
</table>

Example:

```python
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:

```json
{
    "OFPPacketOut": {
        "actions": [
            {
                "OFPActionOutput": {
                    "len": 16,
                    "max_len": 65535,
                    "port": 4294967292,
                    "type": 0
                }
            }
        ],
        "actions_len": 16,
        "buffer_id": 4294967295,
        "data": "8guk0D9w8gukffjqCAABFAABU+BoAAP8Br4sKAAABAGAgaAgj3YAAAAMdYCAAAAAACrjS0xAAAAAABABchMUFRYXGBkaGxwdHh8gISIjJCUmJygpKissLS4vAAAAAAAAAAA=",
        "in_port": 4294967293
    }
}
```

**Barrier Message**

```python
class ryu.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:

```python
def send_barrier_request(self, datapath):
    ofp_parser = datapath.ofproto_parser
    req = ofp_parser.OFPBarrierRequest(datapath)
    datapath.send_msg(req)
```
JSON Example:

```
{
    "OFPBarrierRequest": {}
}
```

**class ryu.ofproto.ofproto_v1_2_parser.OFPBarrierReply (datapath)**

Barrier reply message

The switch responds with this message to a barrier request.

Example:

```python
@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 ryu.ofproto.ofproto_v1_2_parser.OFPRoleRequest (datapath, role, generation_id)**

Role request message

The controller uses this message to change its role.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>role</td>
<td>One of the following values.</td>
</tr>
<tr>
<td></td>
<td>OFPCR_ROLE_NOCHANGE</td>
</tr>
<tr>
<td></td>
<td>OFPCR_ROLE_EQUAL</td>
</tr>
<tr>
<td></td>
<td>OFPCR_ROLE_MASTER</td>
</tr>
<tr>
<td></td>
<td>OFPCR_ROLE_SLAVE</td>
</tr>
<tr>
<td>generation_id</td>
<td>Master Election Generation ID</td>
</tr>
</tbody>
</table>

Example:

```python
def send_role_request(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser
    req = ofp_parser.OFPRoleRequest(datapath, ofp.OFPCR_ROLE_EQUAL, 0)
    datapath.send_msg(req)
```

JSON Example:

```
{
    "OFPRoleRequest": {
        "generation_id": 17294086455919964160,
        "role": 2
    }
}
```
class ryu.ofproto.ofproto_v1_2_parser.OFPRoleReply(datapath, role=None, generation_id=None)

Role reply message

The switch responds with this message to a role request.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>role</td>
<td>One of the following values.</td>
</tr>
<tr>
<td></td>
<td>OFPCR_ROLE_NOCHANGE</td>
</tr>
<tr>
<td></td>
<td>OFPCR_ROLE_EQUAL</td>
</tr>
<tr>
<td></td>
<td>OFPCR_ROLE_MASTER</td>
</tr>
<tr>
<td></td>
<td>OFPCR_ROLE_SLAVE</td>
</tr>
<tr>
<td>generation_id</td>
<td>Master Election Generation ID</td>
</tr>
</tbody>
</table>

Example:

```python
@set_ev_cls(ofp_event.EventOFPRoleReply, 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('OFPRoleReply received: ' +
                      'role=%s generation_id=%d',
                      role, msg.generation_id)

JSON Example:

```json
{
    "OFPRoleReply": {
        "generation_id": 17294086455919964160,
        "role": 3
    }
}
```
Packet-In Message

```python
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>buffer_id</td>
<td>ID assigned by datapath</td>
</tr>
<tr>
<td>total_len</td>
<td>Full length of frame</td>
</tr>
<tr>
<td>reason</td>
<td>Reason packet is being sent.</td>
</tr>
<tr>
<td></td>
<td>OFPR_NO_MATCH</td>
</tr>
<tr>
<td></td>
<td>OFPR_ACTION</td>
</tr>
<tr>
<td></td>
<td>OFPR_INVALID_TTL</td>
</tr>
<tr>
<td>table_id</td>
<td>ID of the table that was looked up</td>
</tr>
<tr>
<td>match</td>
<td>Instance of OFPMatch</td>
</tr>
<tr>
<td>data</td>
<td>Ethernet frame</td>
</tr>
</tbody>
</table>

Example:

```python
@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 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:

```json
{
    "OFPPacketIn": {
        "buffer_id": 2,
        "data": "8gukffjqCAYAAQgABqQAAfILpH346goAAAEAAAAAAAKAAAD",
        "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"
    }
  },
  {
    "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"
    }
  }
]
Flow Removed Message

```python
class ryu.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.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cookie</td>
<td>Opaque controller-issued identifier</td>
</tr>
<tr>
<td>priority</td>
<td>Priority level of flow entry</td>
</tr>
<tr>
<td>reason</td>
<td>One of the following values.</td>
</tr>
<tr>
<td></td>
<td>OFPERR_IDLE_TIMEOUT</td>
</tr>
<tr>
<td></td>
<td>OFPERR_HARD_TIMEOUT</td>
</tr>
<tr>
<td></td>
<td>OFPERR_DELETE</td>
</tr>
<tr>
<td>table_id</td>
<td>ID of the table</td>
</tr>
<tr>
<td>duration_sec</td>
<td>Time flow was alive in seconds</td>
</tr>
<tr>
<td>duration_nsec</td>
<td>Time flow was alive in nanoseconds beyond duration_sec</td>
</tr>
<tr>
<td>idle_timeout</td>
<td>Idle timeout from original flow mod</td>
</tr>
<tr>
<td>hard_timeout</td>
<td>Hard timeout from original flow mod</td>
</tr>
<tr>
<td>packet_count</td>
<td>Number of packets that was associated with the flow</td>
</tr>
<tr>
<td>byte_count</td>
<td>Number of bytes that was associated with the flow</td>
</tr>
<tr>
<td>match</td>
<td>Instance of OFPMatch</td>
</tr>
</tbody>
</table>

Example:

```python
@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: '
                      '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:

```json
{
    "OFPFlowRemoved": {
        "byte_count": 86,
        (continues on next page)
```
"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 ryu.ofproto.ofproto_v1_2_parser.OFPPortStatus(datapath, reason=None, desc=None)

Port status message

The switch notifies controller of change of ports.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reason</td>
<td>One of the following values.</td>
</tr>
<tr>
<td></td>
<td>OFPPR_ADD</td>
</tr>
<tr>
<td></td>
<td>OFPPR_DELETE</td>
</tr>
<tr>
<td></td>
<td>OFPPR_MODIFY</td>
</tr>
<tr>
<td>desc</td>
<td>instance of OFPPort</td>
</tr>
</tbody>
</table>

Example:

```python
@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:
```

(continues on next page)
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": "e2:0b:a4:d0:3f:70",
                "max_speed": 5000,
                "name": "\u79c1\u306e\u30dd\u30fc\u30c8",
                "peer": 10248,
                "port_no": 7,
                "state": 4,
                "supported": 10248
            }
        },
        "reason": 0
    }
}
```

**Error Message**

```python
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>High level type of error</td>
</tr>
<tr>
<td>code</td>
<td>Details depending on the type</td>
</tr>
<tr>
<td>data</td>
<td>Variable length data depending on the type and code</td>
</tr>
</tbody>
</table>

*type* attribute corresponds to *type_* parameter of *__init__*.

Types and codes are defined in *ryu.ofproto.ofproto*.  

2.5. OpenFlow protocol API Reference 203
<table>
<thead>
<tr>
<th>Type</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>OFPET HELLO_FAILED</td>
<td>OFPHFC_*</td>
</tr>
<tr>
<td>OFPET BAD REQUEST</td>
<td>OFPBRRC_*</td>
</tr>
<tr>
<td>OFPET BAD ACTION</td>
<td>OFPBAC_*</td>
</tr>
<tr>
<td>OFPET BAD INSTRUCTION</td>
<td>OFPBIC_*</td>
</tr>
<tr>
<td>OFPET BAD MATCH</td>
<td>OFPBM_*</td>
</tr>
<tr>
<td>OFPET FLOW_MOD_FAILED</td>
<td>OFPFMFC_*</td>
</tr>
<tr>
<td>OFPET GROUP_MOD_FAILED</td>
<td>OFPGMFC_*</td>
</tr>
<tr>
<td>OFPET PORT_MOD_FAILED</td>
<td>OFPPMFC_*</td>
</tr>
<tr>
<td>OFPET TABLE_MOD_FAILED</td>
<td>OFPTMFC_*</td>
</tr>
<tr>
<td>OFPET QUEUE_OP_FAILED</td>
<td>OFPQOFC_*</td>
</tr>
<tr>
<td>OFPET SWITCH_CONFIG_FAILED</td>
<td>OFPSRFC_*</td>
</tr>
<tr>
<td>OFPET ROLE_REQUEST_FAILED</td>
<td>OFPRRFC_*</td>
</tr>
<tr>
<td>OFPET EXPERIMENTER</td>
<td>N/A</td>
</tr>
</tbody>
</table>

If `type == OFPET_EXPERIMENTER`, this message has also the following attributes.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>exp_type</td>
<td>Experimenter defined type</td>
</tr>
<tr>
<td>experimenter</td>
<td>Experimenter ID</td>
</tr>
</tbody>
</table>

Example:

```python
@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:

```json
{
    "OFPErrorMsg": {
        "code": 11,
        "data": "ZnVnYWZ1Z2E=",
        "type": 2
    }
}
```

```json
{
    "OFPErrorMsg": {
        "code": null,
        "data": "amlra2VuIGRhdGE=",
        "exp_type": 60000,
        "experimenter": 999999,
        "type": 65535
    }
}
```

Symmetric Messages
Hello

class ryu.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 Ryu framework, so the Ryu application do not need to process this typically.

    JSON Example:

    ```json
    {
        "OFPHello": {}
    }
    ```

Echo Request

class ryu.ofproto.ofproto_v1_2_parser.OFPEchoRequest(datapath, data=None)
    Echo request message

    This message is handled by the Ryu framework, so the Ryu application do not need to process this typically.

    | Attribute | Description |
    |-----------|-------------|
    | data      | An arbitrary length data |

    Example:

    ```python
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:

    ```json
    {
        "OFPEchoRequest": {
            "data": "aG9nZQ=="
        }
    }
    ```

Echo Reply

class ryu.ofproto.ofproto_v1_2_parser.OFPEchoReply(datapath, data=None)
    Echo reply message

    This message is handled by the Ryu framework, so the Ryu application do not need to process this typically.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>An arbitrary length data</td>
</tr>
</tbody>
</table>

Example:

```python
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:

```json
{
    "OFPEchoReply": {
        "data": "aG9nZQ=="
    }
}
```

**Experimenter**

```python
class ryu.ofproto.ofproto_v1_2_parser.OFPExperimenter(datapath, experimenter=None, exp_type=None, data=None)
```

Experimenter extension message

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>experimenter</td>
<td>Experimenter ID</td>
</tr>
<tr>
<td>exp_type</td>
<td>Experimenter defined</td>
</tr>
<tr>
<td>data</td>
<td>Experimenter defined arbitrary additional data</td>
</tr>
</tbody>
</table>

JSON Example:

```json
{
    "OFPExperimenter": {
        "data": "bmF6bw==",
        "exp_type": 123456789,
        "experimenter": 98765432
    }
}
```

**Port Structures**

```python
class ryu.ofproto.ofproto_v1_2_parser.OFPPort
```

Description of a port
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>port_no</td>
<td>Port number and it uniquely identifies a port within a switch.</td>
</tr>
<tr>
<td>hw_addr</td>
<td>MAC address for the port.</td>
</tr>
<tr>
<td>name</td>
<td>Null-terminated string containing a human-readable name for the interface.</td>
</tr>
<tr>
<td>config</td>
<td>Bitmap of port configuration flags.</td>
</tr>
<tr>
<td></td>
<td>OFPPC_PORT_DOWN</td>
</tr>
<tr>
<td></td>
<td>OFPPC_NO_RECV</td>
</tr>
<tr>
<td></td>
<td>OFPPC_NO_FWD</td>
</tr>
<tr>
<td></td>
<td>OFPPC_NO_PACKET_IN</td>
</tr>
<tr>
<td>state</td>
<td>Bitmap of port state flags.</td>
</tr>
<tr>
<td></td>
<td>OFPPS_LINK_DOWN</td>
</tr>
<tr>
<td></td>
<td>OFPPS_BLOCKED</td>
</tr>
<tr>
<td></td>
<td>OFPPS_LIVE</td>
</tr>
<tr>
<td>curr</td>
<td>Current features.</td>
</tr>
<tr>
<td>advertised</td>
<td>Features being advertised by the port.</td>
</tr>
<tr>
<td>supported</td>
<td>Features supported by the port.</td>
</tr>
<tr>
<td>peer</td>
<td>Features advertised by peer.</td>
</tr>
<tr>
<td>curr_speed</td>
<td>Current port bitrate in kbps.</td>
</tr>
<tr>
<td>max_speed</td>
<td>Max port bitrate in kbps.</td>
</tr>
</tbody>
</table>

Flow Match Structure

class ryu.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.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in_port</td>
<td>Integer 32bit</td>
<td>Switch input port</td>
</tr>
<tr>
<td>in_phy_port</td>
<td>Integer 32bit</td>
<td>Switch physical input port</td>
</tr>
<tr>
<td>metadata</td>
<td>Integer 64bit</td>
<td>Metadata passed between tables</td>
</tr>
<tr>
<td>eth_dst</td>
<td>MAC address</td>
<td>Ethernet destination address</td>
</tr>
<tr>
<td>eth_src</td>
<td>MAC address</td>
<td>Ethernet source address</td>
</tr>
<tr>
<td>eth_type</td>
<td>Integer 16bit</td>
<td>Ethernet frame type</td>
</tr>
<tr>
<td>vlan_vid</td>
<td>Integer 16bit</td>
<td>VLAN id</td>
</tr>
<tr>
<td>vlan_pcp</td>
<td>Integer 8bit</td>
<td>VLAN priority</td>
</tr>
<tr>
<td>ip_dscp</td>
<td>Integer 8bit</td>
<td>IP DSCP (6 bits in ToS field)</td>
</tr>
<tr>
<td>ip_ecn</td>
<td>Integer 8bit</td>
<td>IP ECN (2 bits in ToS field)</td>
</tr>
<tr>
<td>ip_proto</td>
<td>Integer 8bit</td>
<td>IP protocol</td>
</tr>
<tr>
<td>ipv4_src</td>
<td>IPv4 address</td>
<td>IPv4 source address</td>
</tr>
<tr>
<td>ipv4_dst</td>
<td>IPv4 address</td>
<td>IPv4 destination address</td>
</tr>
</tbody>
</table>
Table 1 – continued from previous page

<table>
<thead>
<tr>
<th>Argument</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tcp_src</td>
<td>Integer 16bit</td>
<td>TCP source port</td>
</tr>
<tr>
<td>tcp_dst</td>
<td>Integer 16bit</td>
<td>TCP destination port</td>
</tr>
<tr>
<td>udp_src</td>
<td>Integer 16bit</td>
<td>UDP source port</td>
</tr>
<tr>
<td>udp_dst</td>
<td>Integer 16bit</td>
<td>UDP destination port</td>
</tr>
<tr>
<td>sctp_src</td>
<td>Integer 16bit</td>
<td>SCTP source port</td>
</tr>
<tr>
<td>sctp_dst</td>
<td>Integer 16bit</td>
<td>SCTP destination port</td>
</tr>
<tr>
<td>icmpv4_type</td>
<td>Integer 8bit</td>
<td>ICMP type</td>
</tr>
<tr>
<td>icmpv4_code</td>
<td>Integer 8bit</td>
<td>ICMP code</td>
</tr>
<tr>
<td>arp_op</td>
<td>Integer 16bit</td>
<td>ARP opcode</td>
</tr>
<tr>
<td>arp_spa</td>
<td>IPv4 address</td>
<td>ARP source IPv4 address</td>
</tr>
<tr>
<td>arp_tpa</td>
<td>IPv4 address</td>
<td>ARP target IPv4 address</td>
</tr>
<tr>
<td>arp_sha</td>
<td>MAC address</td>
<td>ARP source hardware address</td>
</tr>
<tr>
<td>arp_tha</td>
<td>MAC address</td>
<td>ARP target hardware address</td>
</tr>
<tr>
<td>ipv6_src</td>
<td>IPv6 address</td>
<td>IPv6 source address</td>
</tr>
<tr>
<td>ipv6_dst</td>
<td>IPv6 address</td>
<td>IPv6 destination address</td>
</tr>
<tr>
<td>ipv6_flabel</td>
<td>Integer 32bit</td>
<td>IPv6 Flow Label</td>
</tr>
<tr>
<td>icmpv6_type</td>
<td>Integer 8bit</td>
<td>ICMPv6 type</td>
</tr>
<tr>
<td>icmpv6_code</td>
<td>Integer 8bit</td>
<td>ICMPv6 code</td>
</tr>
<tr>
<td>ipv6_nd_target</td>
<td>IPv6 address</td>
<td>Target address for ND</td>
</tr>
<tr>
<td>ipv6_nd_sll</td>
<td>MAC address</td>
<td>Source link-layer for ND</td>
</tr>
<tr>
<td>ipv6_nd_tll</td>
<td>MAC address</td>
<td>Target link-layer for ND</td>
</tr>
<tr>
<td>mpls_label</td>
<td>Integer 32bit</td>
<td>MPLS label</td>
</tr>
<tr>
<td>mpls_tc</td>
<td>Integer 8bit</td>
<td>MPLS TC</td>
</tr>
<tr>
<td>pbb_uca</td>
<td>Integer 8bit</td>
<td>PBB UCA header field (EXT-256 Old version of ONF Extension)</td>
</tr>
<tr>
<td>tcp_flags</td>
<td>Integer 16bit</td>
<td>TCP flags (EXT-109 ONF Extension)</td>
</tr>
<tr>
<td>actset_output</td>
<td>Integer 32bit</td>
<td>Output port from action set metadata (EXT-233 ONF Extension)</td>
</tr>
</tbody>
</table>

Example:

```python
>>> # compose
>>> match = parser.OFPMatch(
...   in_port=1,
...   eth_type=0x86dd,
...   ipv6_src=('2001:db8:bd05:1d2:288a:1fc0:1:10ee',
...   ipv6_dst='2001:db8:bd05:1d2:288a:1fc0:1:10ee')
```

```python
>>> # query
>>> if 'ipv6_src' in match:
...     print match['ipv6_src']
```

**Note:** For the list of the supported Nicira experimenter matches, please refer to `ryu.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

<table>
<thead>
<tr>
<th>non-VLAN-tagged</th>
<th>MATCH</th>
</tr>
</thead>
<tbody>
<tr>
<td>VLAN-tagged(vlan_id=3)</td>
<td>MATCH</td>
</tr>
<tr>
<td>VLAN-tagged(vlan_id=5)</td>
<td>MATCH</td>
</tr>
</tbody>
</table>

2) Only packets without a VLAN tag

• Example:

```python
match = parser.OFPMatch(vlan_vid=0x0000)
```

• Packet Matching

<table>
<thead>
<tr>
<th>non-VLAN-tagged</th>
<th>MATCH</th>
</tr>
</thead>
<tbody>
<tr>
<td>VLAN-tagged(vlan_id=3)</td>
<td>x</td>
</tr>
<tr>
<td>VLAN-tagged(vlan_id=5)</td>
<td>x</td>
</tr>
</tbody>
</table>

3) Only packets with a VLAN tag regardless of its value

• Example:

```python
match = parser.OFPMatch(vlan_vid=(0x1000, 0x1000))
```

• Packet Matching

<table>
<thead>
<tr>
<th>non-VLAN-tagged</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>VLAN-tagged(vlan_id=3)</td>
<td>MATCH</td>
</tr>
<tr>
<td>VLAN-tagged(vlan_id=5)</td>
<td>MATCH</td>
</tr>
</tbody>
</table>

4) Only packets with VLAN tag and VID equal

• Example:

```python
match = parser.OFPMatch(vlan_vid=(0x1000 | 3))
```

• Packet Matching

<table>
<thead>
<tr>
<th>non-VLAN-tagged</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>VLAN-tagged(vlan_id=3)</td>
<td>MATCH</td>
</tr>
<tr>
<td>VLAN-tagged(vlan_id=5)</td>
<td>x</td>
</tr>
</tbody>
</table>

**Flow Instruction Structures**

class ryu.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
--- | ---
table_id | Next table

```python
class ryu.ofproto.ofproto_v1_2_parser.OFPInstructionWriteMetadata(metadata, metadata_mask, type_=None, len_=None)
```

Write metadata instruction
This instruction writes the masked metadata value into the metadata field.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>metadata</td>
<td>Metadata value to write</td>
</tr>
<tr>
<td>metadata_mask</td>
<td>Metadata write bitmask</td>
</tr>
</tbody>
</table>

```python
class ryu.ofproto.ofproto_v1_2_parser.OFPInstructionActions(type_, actions=None, len_=None)
```

Actions instruction
This instruction writes/applies/clears the actions.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>One of following values.</td>
</tr>
<tr>
<td></td>
<td>OFPIT_WRITE_ACTIONS</td>
</tr>
<tr>
<td></td>
<td>OFPIT_APPLY_ACTIONS</td>
</tr>
<tr>
<td></td>
<td>OFPIT_CLEAR_ACTIONS</td>
</tr>
<tr>
<td>actions</td>
<td>list of OpenFlow action class</td>
</tr>
</tbody>
</table>

`type` attribute corresponds to `type_` parameter of `__init__`.

### Action Structures

```python
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>port</td>
<td>Output port</td>
</tr>
<tr>
<td>max_len</td>
<td>Max length to send to controller</td>
</tr>
</tbody>
</table>

```python
class ryu.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  

```python
class ryu.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  

```python
class ryu.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  

```python
class ryu.ofproto.ofproto_v1_2_parser.OFPActionDecMplsTtl (type_=None, len_=None)
```

Decrement MPLS TTL action
This action decrements the MPLS TTL.

```python
class ryu.ofproto.ofproto_v1_2_parser.OFPActionSetNwTtl (nw_ttl, type_=None, len_=None)
```

Set IP TTL action
This action sets the IP TTL.

| Attribute | Description  
--- | ---  
nw_ttl | IP TTL  

```python
class ryu.ofproto.ofproto_v1_2_parser.OFPActionDecNwTtl (type_=None, len_=None)
```

Decrement IP TTL action
This action decrements the IP TTL.

```python
class ryu.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.

```python
class ryu.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.

```python
class ryu.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.

2.5. OpenFlow protocol API Reference
### Attribute Description

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ethertype</td>
<td>Ether type. The default is 802.1Q (0x8100)</td>
</tr>
</tbody>
</table>

```python
class ryu.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.

```python
class ryu.ofproto.ofproto_v1_2_parser.OFPActionPopVlan(type_=None, len_=None)
```

Pop VLAN action

This action pops the outermost VLAN tag from the packet.

```python
class ryu.ofproto.ofproto_v1_2_parser.OFPActionPopMpls(ethertype=2048,
                                                        type_=None, len_=None)
```

Pop MPLS action

This action pops the MPLS header from the packet.

```python
class ryu.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:

```python
set_field = OFPActionSetField(eth_src="00:00:00:00:00:00")
```

```python
class ryu.ofproto.ofproto_v1_2_parser.OFPActionExperimenter(experimenter,
                                                              type_=None, len_=None)
```

Experimenter action

This action is an extensible action for the experimenter.

### 2.5.4 OpenFlow v1.3 Messages and Structures

**Controller-to-Switch Messages**

**Handshake**

```python
class ryu.ofproto.ofproto_v1_3_parser.OFPFeaturesRequest (datapath)
```

Features request message

---

**Note:** For the list of the supported Nicira experimenter actions, please refer to `ryu.ofproto.nx_actions`.

---

212 Chapter 2. Writing Your Ryu Application
The controller sends a feature request to the switch upon session establishment.
This message is handled by the Ryu framework, so the Ryu application do not need to process this typically.
Example:

```python
def send_features_request(self, datapath):
    ofp_parser = datapath.ofproto_parser
    req = ofp_parser.OFPFeaturesRequest(datapath)
    datapath.send_msg(req)
```

JSON Example:

```
{
    "OFPFeaturesRequest": {}
}
```

```python
class ryu.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 Ryu framework, so the Ryu application do not need to process this typically.
Example:

```python
@set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
def switch_features_handler(self, ev):
    msg = ev.msg
    self.logger.debug('OFPSwitchFeatures received: ' 'datapath_id=%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 ryu.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 configuration parameters.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Bitmap of the following flags.</td>
</tr>
<tr>
<td></td>
<td>OFPC_FRAG_NORMAL</td>
</tr>
<tr>
<td></td>
<td>OFPC_FRAG_DROP</td>
</tr>
<tr>
<td></td>
<td>OFPC_FRAG_REASM</td>
</tr>
<tr>
<td>miss_send_len</td>
<td>Max bytes of new flow that datapath should send to the controller</td>
</tr>
</tbody>
</table>

Example:

```python
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:

```json
{
"OFPSetConfig": {
  "flags": 0,
  "miss_send_len": 128
}
}
```

class ryu.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:

```python
def send_get_config_request(self, datapath):
ofp_parser = datapath.ofproto_parser
req = ofp_parser.OFPGetConfigRequest(datapath)
datapath.send_msg(req)
```

JSON Example:

```json
{
"OFPGetConfigRequest": {}
}
```

class ryu.ofproto.ofproto_v1_3_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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Bitmap of the following flags.</td>
</tr>
<tr>
<td></td>
<td>OFPC_FRAG_NORMAL</td>
</tr>
<tr>
<td></td>
<td>OFPC_FRAG_DROP</td>
</tr>
<tr>
<td></td>
<td>OFPC_FRAG_REASM</td>
</tr>
<tr>
<td></td>
<td>OFPC_FRAG_MASK</td>
</tr>
<tr>
<td>miss_send_len</td>
<td>Max bytes of new flow that datapath should send to the controller</td>
</tr>
</tbody>
</table>

Example:

```python
@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 = []
    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('OFPGetConfigReply received: ')
    self.logger.debug('flags=%s miss_send_len=%d',
                      ','.join(flags), msg.miss_send_len)
```

JSON Example:

```json
{
    "OFPGetConfigReply": {
        "flags": 0,
        "miss_send_len": 128
    }
}
```

**Flow Table Configuration**

```python
class ryu.ofproto.ofproto_v1_3_parser.OFPTableMod(datapath, table_id, config)
```

Flow table configuration message

The controller sends this message to configure table state.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_id</td>
<td>ID of the table (OFPTT_ALL indicates all tables)</td>
</tr>
<tr>
<td>config</td>
<td>Bitmap of the following flags. OFPTC_DEPRECATED_MASK (3)</td>
</tr>
</tbody>
</table>

Example:
def send_table_mod(self, datapath):
    ofp_parser = datapath.ofproto_parser
    req = ofp_parser.OFPTableMod(datapath, 1, 3)
    datapath.send_msg(req)

JSON Example:

```
{
    "OFPTableMod": {
        "config": 0,
        "table_id": 255
    }
}
```

Modify State Messages

```
class ryu.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.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cookie</td>
<td>Opaque controller-issued identifier</td>
</tr>
<tr>
<td>cookie_mask</td>
<td>Mask used to restrict the cookie bits that must match when the command is OPFFC_MODIFY* or OPFFC_DELETE*</td>
</tr>
<tr>
<td>table_id</td>
<td>ID of the table to put the flow in</td>
</tr>
<tr>
<td>command</td>
<td>One of the following values.</td>
</tr>
<tr>
<td></td>
<td>OFPFC_ADD</td>
</tr>
<tr>
<td></td>
<td>OFPFC_MODIFY</td>
</tr>
<tr>
<td></td>
<td>OFPFC_MODIFY_STRICT</td>
</tr>
<tr>
<td></td>
<td>OFPFC_DELETE</td>
</tr>
<tr>
<td></td>
<td>OFPFC_DELETE_STRICT</td>
</tr>
<tr>
<td>idle_timeout</td>
<td>Idle time before discarding (seconds)</td>
</tr>
<tr>
<td>hard_timeout</td>
<td>Max time before discarding (seconds)</td>
</tr>
<tr>
<td>priority</td>
<td>Priority level of flow entry</td>
</tr>
<tr>
<td>buffer_id</td>
<td>Buffered packet to apply to (or OFP_NO_BUFFER)</td>
</tr>
<tr>
<td>out_port</td>
<td>For OPFFC_DELETE* commands, require matching entries to include this as an output port</td>
</tr>
<tr>
<td>out_group</td>
<td>For OPFFC_DELETE* commands, require matching entries to include this as an output group</td>
</tr>
<tr>
<td>flags</td>
<td>Bitmap of the following flags.</td>
</tr>
<tr>
<td></td>
<td>OFPFF_SEND_FLOW_REM</td>
</tr>
<tr>
<td></td>
<td>OFPFF_CHECK_OVERLAP</td>
</tr>
<tr>
<td></td>
<td>OFPFF_RESET_COUNTS</td>
</tr>
<tr>
<td></td>
<td>OFPFF_NO_PKT_COUNTS</td>
</tr>
<tr>
<td></td>
<td>OFPFF_NO_BYT_COUNTS</td>
</tr>
<tr>
<td>match</td>
<td>Instance of OFPMatch</td>
</tr>
<tr>
<td>instructions</td>
<td>list of OFPInstruction* instance</td>
</tr>
</tbody>
</table>

Example:

```python
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, actions)]
    req = ofp_parser.OFPFlowMod(datapath, cookie, cookie_mask, table_id, ofp.OFPFC_ADD, idle_timeout, hard_timeout, priority, buffer_id,)
```

(continues on next page)
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
              }
            },
            {  
              "OFPActionCopyTtlIn": {  
                "len": 8,
                "type": 12
              }
            },
            {  
              "OFPActionPopPbb": {  
                "len": 8,
                "type": 27
              }
            }
          ]
        }
      ]
    }
  }
}
<table>
<thead>
<tr>
<th>Action Name</th>
<th>Ethertype</th>
<th>Length</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>OFPActionPushPbb</td>
<td>4660</td>
<td>8</td>
<td>26</td>
</tr>
<tr>
<td>OFPActionPopMpls</td>
<td>39030</td>
<td>8</td>
<td>20</td>
</tr>
<tr>
<td>OFPActionPushMpls</td>
<td>34887</td>
<td>8</td>
<td>19</td>
</tr>
<tr>
<td>OFPActionPopVlan</td>
<td></td>
<td>8</td>
<td>18</td>
</tr>
<tr>
<td>OFPActionPushVlan</td>
<td>33024</td>
<td>8</td>
<td>17</td>
</tr>
<tr>
<td>OFPActionDecMplsTtl</td>
<td></td>
<td>8</td>
<td>16</td>
</tr>
<tr>
<td>OFPActionSetMplsTtl</td>
<td></td>
<td>8</td>
<td>15</td>
</tr>
<tr>
<td>OFPActionDecNwTtl</td>
<td>8</td>
<td>8</td>
<td>24</td>
</tr>
<tr>
<td>OFPActionSetNwTtl</td>
<td>8</td>
<td>8</td>
<td>10</td>
</tr>
</tbody>
</table>
"type": 23,
}
,
"OFPActionExperimenterUnknown": {
  "data": "AAECaQFBgc=",
  "experimenter": 101,
  "len": 16,
  "type": 65535
},
,
"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
}
],
"len": 176,
"type": 3
}
),
{"OFPInstructionActions": {
  "actions": [
    {
      "OFPActionSetField": {
        "field": {
          "OXMTlv": {
            "field": "eth_src",
            "mask": null,
            "value": "01:02:03:04:05:06"
          }
        },
        "len": 16,
        "type": 25
      }
    },
    {
      "OFPActionSetField": {
        "field": {
          "OXMTlv": {
            "field": "eth_src",
            "mask": null,
            "value": "01:02:03:04:05:06"
          }
        },
        "len": 16,
        "type": 25
      }
    }
  ]
}}
"OXMTlv": {  
  "field": "pbb_uca",
  "mask": null,
  "value": 1
}
,
"len": 16,
"type": 25
}
,
"len": 40,
"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
}

{"OFPFlowMod": {
  "buffer_id": 65535,
  "command": 0,
  "cookie": 0,
  "cookie_mask": 0,
  "flags": 0,
  "hard_timeout": 0,
  "idle_timeout": 0,
  "instructions": [
    "OFPInstructionGotoTable": {
      "len": 8,
      "table_id": 1,
      "type": 1
    }
  ]
},
(continues on next page)
"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  
},  
"instructions": [  
  {  
    "OFPInstructionMeter": {  
      "len": 8,  
      "meter_id": 1,  
      "type": 6  
    }  
  },  
  {  
    "OFPInstructionActions": {  
      "actions": [  
        {  
          "OFPActionOutput": {  
            "len": 16,  
            "max_len": 65535,  
            "port": 6,  
            "type": 0  
          }  
        }  
      ]  
    }  
  }  
]
Modify group entry message

The controller sends this message to modify the group table.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>command</td>
<td>One of the following values.</td>
</tr>
<tr>
<td>type</td>
<td>One of the following values.</td>
</tr>
<tr>
<td>group_id</td>
<td>Group identifier</td>
</tr>
<tr>
<td>buckets</td>
<td>list of OFPBucket</td>
</tr>
</tbody>
</table>

type attribute corresponds to type parameter of __init__.

Example:
```python
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:

```json
{
    "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 ryu.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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>port_no</td>
<td>Port number to modify</td>
</tr>
<tr>
<td>hw_addr</td>
<td>The hardware address that must be the same as hw_addr of OFPPort of OFPSwitchFeatures</td>
</tr>
<tr>
<td>config</td>
<td>Bitmap of configuration flags.</td>
</tr>
<tr>
<td></td>
<td>OFPPC_PORT_DOWN</td>
</tr>
<tr>
<td></td>
<td>OFPPC_NO_RECV</td>
</tr>
<tr>
<td></td>
<td>OFPPC_NO_FWD</td>
</tr>
<tr>
<td></td>
<td>OFPPC_NO_PACKET_IN</td>
</tr>
<tr>
<td>mask</td>
<td>Bitmap of configuration flags above to be changed</td>
</tr>
<tr>
<td>advertise</td>
<td>Bitmap of the following flags.</td>
</tr>
<tr>
<td></td>
<td>OFPPF_10MB_HD</td>
</tr>
<tr>
<td></td>
<td>OFPPF_10MB_FD</td>
</tr>
<tr>
<td></td>
<td>OFPPF_100MB_HD</td>
</tr>
<tr>
<td></td>
<td>OFPPF_100MB_FD</td>
</tr>
<tr>
<td></td>
<td>OFPPF_1GB_HD</td>
</tr>
<tr>
<td></td>
<td>OFPPF_1GB_FD</td>
</tr>
<tr>
<td></td>
<td>OFPPF_10GB_FD</td>
</tr>
<tr>
<td></td>
<td>OFPPF_40GB_FD</td>
</tr>
<tr>
<td></td>
<td>OFPPF_100GB_FD</td>
</tr>
<tr>
<td></td>
<td>OFPPF_1TB_FD</td>
</tr>
<tr>
<td></td>
<td>OFPPF_AUTONEG</td>
</tr>
<tr>
<td></td>
<td>OFPPF_COPPER</td>
</tr>
<tr>
<td></td>
<td>OFPPF_FIBER</td>
</tr>
<tr>
<td></td>
<td>OFPPF_PAUSE</td>
</tr>
<tr>
<td></td>
<td>OFPPF_PAUSEASYM</td>
</tr>
</tbody>
</table>

#### Example:

```python
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 |
                ofp.OFPPF_AUTONEG | ofp.OFPPF_PAUSE |
                ofp.OFPPF_PAUSEASYM)
    req = ofp_parser.OFPPortMod(datapath, port_no, hw_addr, config,
                                 mask, advertise)
    datapath.send_msg(req)
```

#### JSON Example:
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>command</td>
<td>One of the following values.</td>
</tr>
<tr>
<td></td>
<td>OFPMC_ADD</td>
</tr>
<tr>
<td></td>
<td>OFPMC_MODIFY</td>
</tr>
<tr>
<td></td>
<td>OFPMC_DELETE</td>
</tr>
<tr>
<td>flags</td>
<td>Bitmap of the following flags.</td>
</tr>
<tr>
<td></td>
<td>OFPMF_KBPS</td>
</tr>
<tr>
<td></td>
<td>OFPMF_PKTPS</td>
</tr>
<tr>
<td></td>
<td>OFPMF_BURST</td>
</tr>
<tr>
<td></td>
<td>OFPMF_STATS</td>
</tr>
<tr>
<td>meter_id</td>
<td>Meter instance</td>
</tr>
<tr>
<td>bands</td>
<td>list of the following class instance.</td>
</tr>
<tr>
<td></td>
<td>OFPMeterBandDrop</td>
</tr>
<tr>
<td></td>
<td>OFPMeterBandDscpRemark</td>
</tr>
<tr>
<td></td>
<td>OFPMeterBandExperimenter</td>
</tr>
</tbody>
</table>

JSON Example:

```json
{
    "OFPMeterMod": {
        "advertise": 4096,
        "config": 0,
        "hw_addr": "00:11:00:00:11:11",
        "mask": 0,
        "port_no": 1
    }
}
```

(continues on next page)
Multipart Messages

class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero or OFPMPF_REQ_MORE</td>
</tr>
</tbody>
</table>

Example:

```python
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 ryu.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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>Instance of OFPDescStats</td>
</tr>
</tbody>
</table>

Example:

```python
@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:

```json
{
    "OFPDescStatsReply": {
        "body": {
            "OFPDescStats": {
                "dp_desc": "dp",
                "hw_desc": "hw",
                "mfr_desc": "mfr",
                "serial_num": "serial",
                "sw_desc": "sw"
            }
        },
        "flags": 0,
        "type": 0
    }
}
```

```python
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero or OFPMPF_REQ_MORE</td>
</tr>
<tr>
<td>table_id</td>
<td>ID of table to read</td>
</tr>
<tr>
<td>out_port</td>
<td>Require matching entries to include this as an output port</td>
</tr>
<tr>
<td>out_group</td>
<td>Require matching entries to include this as an output group</td>
</tr>
<tr>
<td>cookie</td>
<td>Require matching entries to contain this cookie value</td>
</tr>
<tr>
<td>cookie_mask</td>
<td>Mask used to restrict the cookie bits that must match</td>
</tr>
<tr>
<td>match</td>
<td>Instance of OFPMatch</td>
</tr>
</tbody>
</table>

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 ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>List of OFPFlowStats instance</td>
</tr>
</tbody>
</table>

Example:

@set_ev_cls(ofp_event.EventOFPFlowStatsReply, MAIN_DISPATCHER)
def flow_stats_reply_handler(self, ev):
    flows = []
    for stat in ev.msg.body:
        flows.append('table_id=%s' %
                      'duration_sec=%d duration_nsec=%d' %
                      'priority=%d' %
                      'idle_timeout=%d hard_timeout=%d flags=0x%x' %
                      '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))

(continues on next page)
(continued from previous page)

    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:

```json
{
    "OFPFlowStatsReply": {
        "body": [  
            {
                "OFPFlowStats": {
                    "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
                    },
                    "instructions": []
                },
                {  
                    "OFPFlowStats": {
                        "byte_count": 0,
                        "cookie": 0,
                        "duration_nsec": 115055000,
                        "duration_sec": 358,
                        "flags": 0,
                        "hard_timeout": 0,
                        "idle_timeout": 0,
                        "instructions": [
                            {"OFPInstructionActions": {
                                "actions": [
                                    {  
                                        "OFPActionOutput": {
                                            "len": 16,
                                            "max_len": 0,
                                            "port": 4294967290,
                                            "type": 0
                                        }
                                    }
                                ]
                            },
                            "len": 24,
                        }
                    }
                }
            }
        ]
    }
}
```

(continues on next page)
"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,
    "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
          }}
        ],
        "type": 4
      }
    }
  }
},
(continues on next page)
"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": {
"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": 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
},
"OFPActionDecNhTtl": {
  "len": 8,
"type": 24
},
{
  "OFPActionSetNwTtl": {
    "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
},
{
  "OFPInstructionActions": {
    "actions": [
      "actions": [
        "type": 24
      ],
      "OFPActionSetNwTtl": {
        "len": 8,
        "nw_ttl": 10,
        "type": 23
      }
    ],
    "len": 192,
    "type": 3
  }
}
{| "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 }, "OFPInstructionActions": { "actions": [ { "OFPActionOutput": { "len": 16, "max_len": 65535, "port": 4294967293, "type": 0 } }, ], "len": 24, "type": 3 } }, "length": 312, "match": { "OFPMatch": { "length": 4, "oxm_fields": [], "type": 1 } }, "packet_count": 1,} (continues on next page)
"priority": 0,
"table_id": 0
}
}
,"flags": 0,
,"type": 1
}
}

```python
class ryu.ofproto.ofproto_v1_3_parser.OFAggregateStatsRequest (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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero or OFMPF_REQ_MORE</td>
</tr>
<tr>
<td>table_id</td>
<td>ID of table to read</td>
</tr>
<tr>
<td>out_port</td>
<td>Require matching entries to include this as an output port</td>
</tr>
<tr>
<td>out_group</td>
<td>Require matching entries to include this as an output group</td>
</tr>
<tr>
<td>cookie</td>
<td>Require matching entries to contain this cookie value</td>
</tr>
<tr>
<td>cookie_mask</td>
<td>Mask used to restrict the cookie bits that must match</td>
</tr>
<tr>
<td>match</td>
<td>Instance of OFPMatch</td>
</tr>
</tbody>
</table>

Example:

```python
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.OFAggregateStatsRequest (datapath, 0,
                                            ofp.OFPTT_ALL, ofp.OFPP_ANY, ofp.OFPG_ANY,
                                            cookie, cookie_mask, match)
    datapath.send_msg (req)
```

JSON Example:

```json
{
    "OFAggregateStatsRequest": {
        "cookie": 0,
        "cookie_mask": 0,
        "flags": 0,
        "match": {
```

(continues on previous page)
"OFPMatch": {  
  "length": 4,  
  "oxm_fields": [],  
  "type": 1  
},  
"out_group": 4294967295,  
"out_port": 4294967295,  
"table_id": 255,  
"type": 2
}

```python
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>Instance of OFPAggregateStats</td>
</tr>
</tbody>
</table>

Example:

```python
@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:

```json
{
    "OFPAggregateStatsReply": {  
        "body": {  
            "OFPAggregateStats": {  
                "byte_count": 574,  
                "flow_count": 6,  
                "packet_count": 7  
            }  
        },  
        "flags": 0,  
        "type": 2
    }
}
```

```python
class ryu.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.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero or OFMPF_REQ_MORE</td>
</tr>
</tbody>
</table>

Example:

```python
def send_table_stats_request(self, datapath):
ofp_parser = datapath.ofproto_parser
req = ofp_parser.OFPTableStatsRequest(datapath, 0)
datapath.send_msg(req)
```

JSON Example:

```json
{
"OFPTableStatsRequest": {
  "flags": 0,
  "type": 3
}
}
```

```
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>List of OFPTableStats instance</td>
</tr>
</tbody>
</table>

Example:

```python
@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:

```json
{
"OFPTableStatsReply": {
  "body": [
    {
      "OFPTableStats": {
        "active_count": 4,
        "lookup_count": 4,
        "matched_count": 4,
        "table_id": 0
      }
    },
    {
      "OFPTableStats": {
```

(continues on next page)
Port statistics request message

The controller uses this message to query information about ports statistics.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero or OFPMPF_REQ_MORE</td>
</tr>
<tr>
<td>port_no</td>
<td>Port number to read (OFPP_ANY to all ports)</td>
</tr>
</tbody>
</table>

Example:

```python
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:

```json
{
    "OFPPortStatsRequest": {
        "flags": 0,
        "port_no": 4294967295,
        "type": 4
    }
}
```

Port statistics reply message

The switch responds with this message to a port statistics request.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>List of OFPPortStats instance</td>
</tr>
</tbody>
</table>

Example:

```python
@set_ev_cls(ofp_event.EventOFPPortStatsReply, MAIN_DISPATCHER)
def port_stats_reply_handler(self, ev):
    # process the port statistics reply
```
ports = []
for stat in ev.msg.body:
    ports.append('port_no=%d ' % (stat.port_no,)
                 'rx_packets=%d tx_packets=%d ' % (stat.rx_packets, stat.tx_packets,)
                 'rx_bytes=%d tx_bytes=%d ' % (stat.rx_bytes, stat.tx_bytes,)
                 'rx_dropped=%d tx_dropped=%d ' % (stat.rx_dropped, stat.tx_dropped,)
                 'rx_errors=%d tx_errors=%d ' % (stat.rx_errors, stat.tx_errors,)
                 'rx_frame_err=%d rx_over_err=%d rx_crc_err=%d ' % (stat.rx_frame_err, stat.rx_over_err, stat.rx_crc_err,)
                 'collisions=%d duration_sec=%d duration_nsec=%d' % (stat.collisions, stat.duration_sec, stat.duration_nsec))

self.logger.debug('PortStats: %s', ports)

JSON Example:

```json
{
    "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": 4,
                    "tx_bytes": 336,
                    "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_errors": 0,
                    "tx_packets": 4
                }
            }
        ]
    }
}
```


```
"tx_dropped": 0,
"tx_errors": 0,
"tx_packets": 4
}
},
"flags": 0,
"type": 4
}
```

**class** ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero or OFPMPF_REQ_MORE</td>
</tr>
</tbody>
</table>

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** ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>List of OFFPort instance</td>
</tr>
</tbody>
</table>

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%x dpid=%d'
```
(continued from previous page)

```python
'state=0x%08x curr=0x%08x advertised=0x%08x '
'supported=0x%08x peer=0x%08x curr_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:

```json
{
    "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
    }
}
```

```python
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero or OFPMPF_REQ_MORE</td>
</tr>
<tr>
<td>port_no</td>
<td>Port number to read</td>
</tr>
<tr>
<td>queue_id</td>
<td>ID of queue to read</td>
</tr>
</tbody>
</table>

Example:

```python
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:

```json
{
  "OFPQueueStatsRequest": {
    "flags": 0,
    "port_no": 4294967295,
    "queue_id": 4294967295,
    "type": 5
  }
}
```

class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>List of OFPQueueStats instance</td>
</tr>
</tbody>
</table>

Example:

```python
@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 ' %
                       (stat.port_no, stat.queue_id,)
                       'tx_bytes=%d tx_packets=%d tx_errors=%d ' %
                       (stat.tx_bytes, stat.tx_packets, stat.tx_errors,)
                       'duration_sec=%d duration_nsec=%d' %
                       (stat.duration_sec, stat.duration_nsec))
    self.logger.debug('QueueStats: %s', queues)
```

JSON Example:

```json
{
  "OFPQueueStatsReply": {
    "body": [
```

(continues on next page)
### Group statistics request message

The controller uses this message to query statistics of one or more groups.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero or OFPMPF_REQ_MORE</td>
</tr>
<tr>
<td>group_id</td>
<td>ID of group to read (OFPG_ALL to all groups)</td>
</tr>
</tbody>
</table>

**Example:**

```python
def send_group_stats_request(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser
```

(continues on next page)
class ryu.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.

Example:

```python
@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 ryu.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.

Example:

```python
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:

```json
{
    "OFPGroupDescStatsRequest": {
        "flags": 0,
        "type": 7
    }
}
```
class Ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>List of OFPGpUpGroupDescStats instance</td>
</tr>
</tbody>
</table>

Example:

```python
@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 ' %
                      (stat.length, stat.type, stat.group_id, 
                       stat.bucket))
    self.logger.debug('GroupDescStats: %s', descs)
```

JSON Example:

```json
{
    "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,
                "type": 7
            }
        ],
        "flags": 0,
        "type": 7
    }
}  
```

(continues on next page)
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero or OFPMPF_REQ_MORE</td>
</tr>
</tbody>
</table>

Example:
```python
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:
```json
{
    "OFPGroupFeaturesStatsRequest": {
        "flags": 0,
        "type": 8
    }
}
```

class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>Instance of OFPGroupFeaturesStats</td>
</tr>
</tbody>
</table>

Example:
```python
@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%x max_groups=%s ' \
                      'actions=%s',
                      body.types, body.capabilities, \
                      body.max_groups, body.actions)
```

JSON Example:
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero or OFPMPF_REQ_MORE</td>
</tr>
<tr>
<td>meter_id</td>
<td>ID of meter to read (OFPM_ALL to all meters)</td>
</tr>
</tbody>
</table>

Example:

```python
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:

```json
{
    "OFPMeterStatsRequest": {
        "flags": 0,
        "meter_id": 4294967295,
        "type": 9
    }
}
```

class ryu.ofproto.ofproto_v1_3_parser.OFPMeterStatsReply (datapath, **kwargs)
Meter statistics reply message

The switch responds with this message to a meter statistics request.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>List of OFPMeterStats instance</td>
</tr>
</tbody>
</table>

Example:

```python
@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'
(setqstatmeter_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:

```json
{
   "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 ryu.ofproto.ofproto_v1_3_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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero or OFPMPF_REQ_MORE</td>
</tr>
<tr>
<td>meter_id</td>
<td>ID of meter to read (OFPM_ALL to all meters)</td>
</tr>
</tbody>
</table>

Example:

```python
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:

```json
{
    "OFPMeterConfigStatsRequest": {
        "flags": 0,
        "meter_id": 4294967295,
        "type": 10
    }
}
```

class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>List of OFPMeterConfigStats instance</td>
</tr>
</tbody>
</table>

Example:

```python
@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 ' %
                        (stat.length, stat.flags, stat.meter_id,
                         (stat.bands))
        self.logger.debug('MeterConfigStats: $s', configs)
```

JSON Example:

```json
{
    "OFPMeterConfigStatsReply": {
        "body": [
            "OFPMeterConfigStats": {
                "bands": [ ...
```

(continues on next page)
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero or OFMPF_REQ_MORE</td>
</tr>
</tbody>
</table>

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:

```json
{
    "OFPMeterFeaturesStatsRequest": {
        "flags": 0,
        "type": 11
    }
}
```

class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>List of OFPMeterFeaturesStats instance</td>
</tr>
</tbody>
</table>
Example:

```python
@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 ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>List of OFPTableFeaturesStats instances. The default is [].</td>
</tr>
</tbody>
</table>

class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>List of OFPTableFeaturesStats instance</td>
</tr>
</tbody>
</table>

JSON Example:
See an example in:

```
ryu/tests/unit/ofproto/json/of13/4-56-ofp_table_features_reply.
packet.json
```

### Queue Configuration Messages

**class ryu.ofproto.ofproto_v1_3_parser.OFPQueueGetConfigRequest** *(datapath, port)*  
Queue configuration request message

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>port</td>
<td>Port to be queried (OFPP_ANY to all configured queues)</td>
</tr>
</tbody>
</table>

Example:

```python
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:

```json
{
    "OFPQueueGetConfigRequest": {
        "port": 4294967295
    }
}
```

**class ryu.ofproto.ofproto_v1_3_parser.OFPQueueGetConfigReply** *(datapath, queues=None, port=None)*  
Queue configuration reply message

The switch responds with this message to a queue configuration request.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queues</td>
<td>list of OFPPacketQueue instance</td>
</tr>
<tr>
<td>port</td>
<td>Port which was queried</td>
</tr>
</tbody>
</table>

Example:

```python
@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": 64,
        "port": 77,
        "properties": [  
        {  "OFPQueuePropMinRate": {  
            "len": 16,
            "property": 1,
            "rate": 10
        },  
        {  "OFPQueuePropMaxRate": {  
            "len": 16,
            "property": 2,
            "rate": 900
        },  
        {  "OFPQueuePropExperimenter": {  
            "data": [],
            "experimenter": 999,
            "len": 16,
            "property": 65535
        }
        ]
    },  
    "queue_id": 99
    }  
    },  
    "OFPPacketQueue": {  
        "len": 65,
        "port": 77,
        "properties": [  
        {  "OFPQueuePropMinRate": {  
            "len": 16,
            "property": 1,
            "rate": 100
        },  
        {  "OFPQueuePropMaxRate": {  
            "len": 16,
            "property": 2,
            "rate": 200
        },  
        {  "OFPQueuePropExperimenter": {  
            "experimenter": 999,
            }}
    ]
    },
    "OFPQueuePropExperimenter": {  
    "data": [],
    "experimenter": 999,
    "len": 16,
    "property": 65535
  }
}  
}  
}  
(continues on next page)
Packet-Out Message

```python
class ryu.ofproto.ofproto_v1_3_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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>buffer_id</td>
<td>ID assigned by datapath (OFP_NO_BUFFER if none)</td>
</tr>
<tr>
<td>in_port</td>
<td>Packet's input port or OFPP_CONTROLLER</td>
</tr>
<tr>
<td>actions</td>
<td>list of OpenFlow action class</td>
</tr>
<tr>
<td>data</td>
<td>Packet data of a binary type value or an instances of packet.Packet.</td>
</tr>
</tbody>
</table>

Example:

```python
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:

```json
{
    "OFPPacketOut": {
        "actions": [
            {
                "OFPActionOutput": {
                    "len": 16,
                    "max_len": 65535,
                    "port": 4294967292,
                    "type": 0
                }
            }
        ],
        "actions_len": 16,
        "buffer_id": 4294967295,
        "data": "8guk0D9w8gukffjqCABFAABU+BoAAP8Br4sKAAABCgAAAggAAACrjS0xAAAAABArCBmMUFRYXGkBaGw",
        "in_port": 4294967293
    }
}
```

**Barrier Message**

class ryu.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:

```python
def send_barrier_request(self, datapath):
ofp_parser = datapath.ofproto_parser
```

(continues on next page)
ryu Documentation, Release 4.34

(continued from previous page)

```python
req = ofp_parser.OFPBarrierRequest(datapath)
datapath.send_msg(req)
```

JSON Example:

```json
{
  "OFPBarrierRequest": {}
}
```

class `ryu.ofproto.ofproto_v1_3_parser.OFPBarrierReply`(*datapath*)

Barrier reply message

The switch responds with this message to a barrier request.

Example:

```python
@set_ev_cls(ofp_event.EventOFPBarrierReply, MAIN_DISPATCHER)
def barrier_reply_handler(self, ev):
    self.logger.debug('OFPBarrierReply received')
```

JSON Example:

```json
{
  "OFPBarrierReply": {}
}
```

Role Request Message

class `ryu.ofproto.ofproto_v1_3_parser.OFPRoleRequest`(*datapath*, *role=None, generation_id=None*)

Role request message

The controller uses this message to change its role.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>role</td>
<td>One of the following values.</td>
</tr>
<tr>
<td></td>
<td>OFPCR_ROLE_NOCHANGE</td>
</tr>
<tr>
<td></td>
<td>OFPCR_ROLE_EQUAL</td>
</tr>
<tr>
<td></td>
<td>OFPCR_ROLE_MASTER</td>
</tr>
<tr>
<td></td>
<td>OFPCR_ROLE_SLAVE</td>
</tr>
<tr>
<td>generation_id</td>
<td>Master Election Generation ID</td>
</tr>
</tbody>
</table>

Example:

```python
def send_role_request(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPRoleRequest(datapath, ofp.OFPCR_ROLE_EQUAL, 0)
    datapath.send_msg(req)
```

JSON Example:
class ryu.ofproto.ofproto_v1_3_parser.OFPRoleReply(datapath, role=None, generation_id=None)

Role reply message

The switch responds with this message to a role request.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>role</td>
<td>One of the following values.</td>
</tr>
<tr>
<td>generation_id</td>
<td>Master Election Generation ID</td>
</tr>
</tbody>
</table>

Example:

```python
@set_ev_cls(ofp_event.EventOFPRoleReply, 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('OFPRoleReply received: '
                   'role=%s generation_id=%d',
                   role, msg.generation_id)
```

JSON Example:

```json
{
    "OFPRoleReply": {
        "generation_id": 17294086455919964160,
        "role": 3
    }
}
```
Set Asynchronous Configuration Message

```python
class ryu.ofproto.ofproto_v1_3_parser.OFPSetsAsync(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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>packet_in_mask</td>
<td>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</td>
</tr>
<tr>
<td>port_status_mask</td>
<td>2-element array. Bitmasks of following values. OFPPR_ADD OFPPR_DELETE OFPPR_MODIFY</td>
</tr>
<tr>
<td>flow_removed_mask</td>
<td>2-element array. Bitmasks of following values. OFP_PR_IDE_TIMEOUT OFPRR_HARD_TIMEOUT OFPRR_DELETE OFPRR_GROUP_DELETE</td>
</tr>
</tbody>
</table>

Example:

```python
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.OFPSetsAsync(datapath,
                             [packet_in_mask, 0],
                             [port_status_mask, 0],
                             [flow_removed_mask, 0])
    datapath.send_msg(req)
```
JSON Example:

```json
{
    "OFPSetAsync": {
        "flow_removed_mask": [15, 3],
        "packet_in_mask": [5, 1],
        "port_status_mask": [7, 3]
    }
}
```

```python
class ryu.ofproto.ofproto_v1_3_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:

```json
{
    "OFPGetAsyncRequest": {}
}
```

```python
class ryu.ofproto.ofproto_v1_3_parser.OFPGetAsyncReply (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.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>packet_in_mask</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>OFPR_NO_MATCH</td>
</tr>
<tr>
<td></td>
<td>OFPR_ACTION</td>
</tr>
<tr>
<td></td>
<td>OFPR_INVALID_TTL</td>
</tr>
<tr>
<td>port_status_mask</td>
<td>2-element array. Bitmasks of following values.</td>
</tr>
<tr>
<td></td>
<td>OFPPR_ADD</td>
</tr>
<tr>
<td></td>
<td>OFPPR_DELETE</td>
</tr>
<tr>
<td></td>
<td>OFPPR_MODIFY</td>
</tr>
<tr>
<td>flow_removed_mask</td>
<td>2-element array. Bitmasks of following values.</td>
</tr>
<tr>
<td></td>
<td>OFPERR_IDLE_TIMEOUT</td>
</tr>
<tr>
<td></td>
<td>OFPERR_HARD_TIMEOUT</td>
</tr>
<tr>
<td></td>
<td>OFPERR_DELETE</td>
</tr>
<tr>
<td></td>
<td>OFPERR_GROUP_DELETE</td>
</tr>
</tbody>
</table>

Example:

```python
@set_ev_cls(ofp_event.EventOFPGetAsyncReply, MAIN_DISPATCHER)
def get_async_reply_handler(self, ev):
    msg = ev.msg

    self.logger.debug('OFPGetAsyncReply received: ')
    self.logger.debug('  packet_in_mask=0x%08x:0x%08x' % (msg.packet_in_mask[0], msg.packet_in_mask[1]))
    self.logger.debug('  port_status_mask=0x%08x:0x%08x' % (msg.port_status_mask[0], msg.port_status_mask[1]))
    self.logger.debug('  flow_removed_mask=0x%08x:0x%08x' % (msg.flow_removed_mask[0], msg.flow_removed_mask[1]))
```

JSON Example:

```json
{
    "OFPGetAsyncReply": {
        "flow_removed_mask": [15, 3],
        "packet_in_mask": [5, 1]
    }
}
```

(continues on next page)
Asynchronous Messages

Packet-In Message

```python
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>buffer_id</td>
<td>ID assigned by datapath</td>
</tr>
<tr>
<td>total_len</td>
<td>Full length of frame</td>
</tr>
<tr>
<td>reason</td>
<td>Reason packet is being sent.</td>
</tr>
<tr>
<td></td>
<td>OFPR_NO_MATCH</td>
</tr>
<tr>
<td></td>
<td>OFPR_ACTION</td>
</tr>
<tr>
<td></td>
<td>OFPR_INVALID_TTL</td>
</tr>
<tr>
<td>table_id</td>
<td>ID of the table that was looked up</td>
</tr>
<tr>
<td>cookie</td>
<td>Cookie of the flow entry that was looked up</td>
</tr>
<tr>
<td>match</td>
<td>Instance of OFPMetach</td>
</tr>
<tr>
<td>data</td>
<td>Ethernet frame</td>
</tr>
</tbody>
</table>

Example:

```python
@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: ' 
                      'reason=%s
```

(continues on next page)
'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:

```json
{
  "OFPPacketIn": {
    "buffer_id": 2,
    "cookie": 283686884868096,
    "data": "////////8gukffjqCAAYAAQgABgQAAfILpH346g0AAEAAAAAAAAAAAD",
    "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"
          }
        },{
          "OXMTlv": {
            "field": "arp_op",
            "mask": null,
            "value": 1
          }
        },{
          "OXMTlv": {
            "field": "arp_spa",
            "mask": null,
            "value": "10.0.0.1"
          }
        }
      }
    }
  }
}
```

(continues on next page)
Flow Removed Message

```python
class ryu.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.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cookie</td>
<td>Opaque controller-issued identifier</td>
</tr>
<tr>
<td>priority</td>
<td>Priority level of flow entry</td>
</tr>
<tr>
<td>reason</td>
<td>One of the following values.</td>
</tr>
<tr>
<td></td>
<td>OFP_PRR_IDLE_TIMEOUT</td>
</tr>
<tr>
<td></td>
<td>OFP_PRR_HARD_TIMEOUT</td>
</tr>
<tr>
<td></td>
<td>OFP_PRR_DELETE</td>
</tr>
<tr>
<td></td>
<td>OFP_PRR_GROUP_DELETE</td>
</tr>
<tr>
<td>table_id</td>
<td>ID of the table</td>
</tr>
<tr>
<td>duration_sec</td>
<td>Time flow was alive in seconds</td>
</tr>
<tr>
<td>duration_nsec</td>
<td>Time flow was alive in nanoseconds beyond duration_sec</td>
</tr>
<tr>
<td>idle_timeout</td>
<td>Idle timeout from original flow mod</td>
</tr>
<tr>
<td>hard_timeout</td>
<td>Hard timeout from original flow mod</td>
</tr>
<tr>
<td>packet_count</td>
<td>Number of packets that was associated with the flow</td>
</tr>
<tr>
<td>byte_count</td>
<td>Number of bytes that was associated with the flow</td>
</tr>
<tr>
<td>match</td>
<td>Instance of OFPMatch</td>
</tr>
</tbody>
</table>

Example:

```python
@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.OFP_PRR_IDLE_TIMEOUT:
        reason = 'IDLE TIMEOUT'
    elif msg.reason == ofp.OFP_PRR_HARD_TIMEOUT:
        reason = 'HARD TIMEOUT'
    elif msg.reason == ofp.OFP_PRR_DELETE:
        reason = 'DELETE'
    elif msg.reason == ofp.OFP_PRR_GROUP_DELETE:
        reason = 'GROUP DELETE'
    else:
        reason = 'unknown'

    self.logger.debug('OFPFlowRemoved 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:

```json
{
    "OFPFlowRemoved": {
        "byte_count": 86,
        (continues on next page)
    }
}
```
Port Status Message

class ryu.ofproto.ofproto_v1_3_parser.OFPPortStatus(datapath, reason=None, desc=None)

Port status message

The switch notifies controller of change of ports.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
</table>
| reason    | One of the following values.  
|           | OFPPR_ADD  
|           | OFPPR_DELETE  
|           | OFPPR_MODIFY  |
| desc      | instance of OFPPort |

Example:

```python
@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:
        pass
```

(continues on next page)
reason = 'ADD'
eelif msg.reason == ofp.OFPPR_DELETE:
    reason = 'DELETE'
eelif msg.reason == ofp.OFPPR_MODIFY:
    reason = 'MODIFY'
eelse:
    reason = 'unknown'
    self.logger.debug('OFPPortStatus received: reason=%s desc=%s', reason, msg.desc)

JSON Example:

```json
{
    "OFPPortStatus": {
        "desc": {
            "OFFPort": {
                "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",
                "peer": 10248,
                "port_no": 7,
                "state": 4,
                "supported": 10248
            }
        },
        "reason": 0
    }
}
```

### Error Message

class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>High level type of error</td>
</tr>
<tr>
<td>code</td>
<td>Details depending on the type</td>
</tr>
<tr>
<td>data</td>
<td>Variable length data depending on the type and code</td>
</tr>
</tbody>
</table>

The `type` attribute corresponds to the `type_` parameter of `__init__`. Types and codes are defined in `ryu.ofproto.ofproto`. 

2.5. OpenFlow protocol API Reference 267
### Type Codes

<table>
<thead>
<tr>
<th>Type</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>OFPET_HELLO_FAILED</td>
<td>OFPHFC_*</td>
</tr>
<tr>
<td>OFPET_BAD_REQUEST</td>
<td>OFPBRC_*</td>
</tr>
<tr>
<td>OFPET_BAD_ACTION</td>
<td>OFPBAC_*</td>
</tr>
<tr>
<td>OFPET_BAD_INSTRUCTION</td>
<td>OFPBIC_*</td>
</tr>
<tr>
<td>OFPET_BAD_MATCH</td>
<td>OFPBMC_*</td>
</tr>
<tr>
<td>OFPET_FLOW_MOD_FAILED</td>
<td>OFPFMFC_*</td>
</tr>
<tr>
<td>OFPET_GROUP_MOD_FAILED</td>
<td>OFPGMFC_*</td>
</tr>
<tr>
<td>OFPET_PORT_MOD_FAILED</td>
<td>OFPPMFC_*</td>
</tr>
<tr>
<td>OFPET_TABLE_MOD_FAILED</td>
<td>OFPTMFC_*</td>
</tr>
<tr>
<td>OFPET_QUEUE_OP_FAILED</td>
<td>OFPQOFC_*</td>
</tr>
<tr>
<td>OFPET_SWITCH_CONFIG_FAILED</td>
<td>OFPSFC_*</td>
</tr>
<tr>
<td>OFPET_ROLE_REQUEST_FAILED</td>
<td>OFPRRFC_*</td>
</tr>
<tr>
<td>OFPET_METER_MOD_FAILED</td>
<td>OFPMMF_*</td>
</tr>
<tr>
<td>OFPET_TABLE_FEATURES_FAILED</td>
<td>OFPTIFC_*</td>
</tr>
<tr>
<td>OFPET_EXPERIMENTER</td>
<td>N/A</td>
</tr>
</tbody>
</table>

If `type == OFPET_EXPERIMENTER`, this message has also the following attributes.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>exp_type</td>
<td>Experimenter defined type</td>
</tr>
<tr>
<td>experimenter</td>
<td>Experimenter ID</td>
</tr>
</tbody>
</table>

Example:

```python
@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:

```json
{
    "OFPErrorMsg": {
        "code": 11,
        "data": "ZnVnYWZ1Z2E=",
        "type": 2
    }
}
```

### Symmetric Messages

#### Hello

```python
class ryu.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 Ryu framework, so the Ryu application do not need to process this typically.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>elements</td>
<td>list of OFPHelloElemVersionBitmap instance</td>
</tr>
</tbody>
</table>

JSON Example:

```json
{
  "OFPHello": {
    "elements": [
      {
        "OFPHelloElemVersionBitmap": {
          "length": 8,
          "type": 1,
          "versions": [
            1,
            2,
            3,
            9,
            10,
            30
          ]
        }
      }
    ]
  }
}
```

class ryu.ofproto.ofproto_v1_3_parser.OFPHelloElemVersionBitmap (versions, type_=None, length=None)

Version bitmap Hello Element

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>versions</td>
<td>list of versions of OpenFlow protocol a device supports</td>
</tr>
</tbody>
</table>

Echo Request

class ryu.ofproto.ofproto_v1_3_parser.OFPEchoRequest (datapath, data=None)

Echo request message

This message is handled by the Ryu framework, so the Ryu application do not need to process this typically.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>An arbitrary length data</td>
</tr>
</tbody>
</table>

Example:

```python
def send_echo_request(self, datapath, data):
    ofp_parser = datapath.ofproto_parser
    req = ofp_parser.OFPEchoRequest(datapath, data)
    datapath.send_msg(req)
```

(continues on next page)
@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:

```json
{
    "OFPEchoRequest": {
        "data": "aG9nZQ=="
    }
}
```

Echo Reply

```python
class ryu.ofproto.ofproto_v1_3_parser.OFPEchoReply (datapath, data=None)
```

Echo reply message

This message is handled by the Ryu framework, so the Ryu application do not need to process this typically.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>An arbitrary length data</td>
</tr>
</tbody>
</table>

Example:

```python
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:

```json
{
    "OFPEchoReply": {
        "data": "aG9nZQ=="
    }
}
```

Experimenter

```python
class ryu.ofproto.ofproto_v1_3_parser.OFPEXperimenter (datapath, exper-
imenter=None, exp_type=None, data=None)
```

Experimenter extension message
### Attribute Description

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>experimenter</td>
<td>Experimenter ID</td>
</tr>
<tr>
<td>exp_type</td>
<td>Experimenter defined</td>
</tr>
<tr>
<td>data</td>
<td>Experimenter defined arbitrary additional data</td>
</tr>
</tbody>
</table>

JSON Example:

```json
{
    "OFPExperimenter": {
        "data": "bmF6bw==",
        "exp_type": 123456789,
        "experimenter": 98765432
    }
}
```

### Port Structures

**class ryu.ofproto.ofproto_v1_3_parser.OFPPort**

Description of a port

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>port_no</td>
<td>Port number and it uniquely identifies a port within a switch.</td>
</tr>
<tr>
<td>hw_addr</td>
<td>MAC address for the port.</td>
</tr>
<tr>
<td>name</td>
<td>Null-terminated string containing a human-readable name for the interface.</td>
</tr>
<tr>
<td>config</td>
<td>Bitmap of port configuration flags.</td>
</tr>
<tr>
<td>state</td>
<td>Bitmap of port state flags.</td>
</tr>
<tr>
<td>curr</td>
<td>Current features.</td>
</tr>
<tr>
<td>advertised</td>
<td>Features being advertised by the port.</td>
</tr>
<tr>
<td>supported</td>
<td>Features supported by the port.</td>
</tr>
<tr>
<td>peer</td>
<td>Features advertised by peer.</td>
</tr>
<tr>
<td>curr_speed</td>
<td>Current port bitrate in kbps.</td>
</tr>
<tr>
<td>max_speed</td>
<td>Max port bitrate in kbps.</td>
</tr>
</tbody>
</table>

### Flow Match Structure

**class ryu.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.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in_port</td>
<td>Integer 32bit</td>
<td>Switch input port</td>
</tr>
<tr>
<td>in_phy_port</td>
<td>Integer 32bit</td>
<td>Switch physical input port</td>
</tr>
<tr>
<td>metadata</td>
<td>Integer 64bit</td>
<td>Metadata passed between tables</td>
</tr>
<tr>
<td>eth_dst</td>
<td>MAC address</td>
<td>Ethernet destination address</td>
</tr>
<tr>
<td>eth_src</td>
<td>MAC address</td>
<td>Ethernet source address</td>
</tr>
<tr>
<td>eth_type</td>
<td>Integer 16bit</td>
<td>Ethernet frame type</td>
</tr>
<tr>
<td>vlan_vid</td>
<td>Integer 16bit</td>
<td>VLAN id</td>
</tr>
<tr>
<td>vlan_pcp</td>
<td>Integer 8bit</td>
<td>VLAN priority</td>
</tr>
<tr>
<td>ip_dscp</td>
<td>Integer 8bit</td>
<td>IP DSCP (6 bits in ToS field)</td>
</tr>
<tr>
<td>ip_ecn</td>
<td>Integer 8bit</td>
<td>IP ECN (2 bits in ToS field)</td>
</tr>
<tr>
<td>ip_proto</td>
<td>Integer 8bit</td>
<td>IP protocol</td>
</tr>
<tr>
<td>ipv4_src</td>
<td>IPv4 address</td>
<td>IPv4 source address</td>
</tr>
<tr>
<td>ipv4_dst</td>
<td>IPv4 address</td>
<td>IPv4 destination address</td>
</tr>
<tr>
<td>tcp_src</td>
<td>Integer 16bit</td>
<td>TCP source port</td>
</tr>
<tr>
<td>tcp_dst</td>
<td>Integer 16bit</td>
<td>TCP destination port</td>
</tr>
<tr>
<td>udp_src</td>
<td>Integer 16bit</td>
<td>UDP source port</td>
</tr>
<tr>
<td>udp_dst</td>
<td>Integer 16bit</td>
<td>UDP destination port</td>
</tr>
<tr>
<td>sctp_src</td>
<td>Integer 16bit</td>
<td>SCTP source port</td>
</tr>
<tr>
<td>sctp_dst</td>
<td>Integer 16bit</td>
<td>SCTP destination port</td>
</tr>
<tr>
<td>icmpv4_type</td>
<td>Integer 8bit</td>
<td>ICMP type</td>
</tr>
<tr>
<td>icmpv4_code</td>
<td>Integer 8bit</td>
<td>ICMP code</td>
</tr>
<tr>
<td>arp_op</td>
<td>Integer 16bit</td>
<td>ARP opcode</td>
</tr>
<tr>
<td>arp_spa</td>
<td>IPv4 address</td>
<td>ARP source IPv4 address</td>
</tr>
<tr>
<td>arp_tpa</td>
<td>IPv4 address</td>
<td>ARP target IPv4 address</td>
</tr>
<tr>
<td>arp_sha</td>
<td>MAC address</td>
<td>ARP source hardware address</td>
</tr>
<tr>
<td>arp_tha</td>
<td>MAC address</td>
<td>ARP target hardware address</td>
</tr>
<tr>
<td>ipv6_src</td>
<td>IPv6 address</td>
<td>IPv6 source address</td>
</tr>
<tr>
<td>ipv6_dst</td>
<td>IPv6 address</td>
<td>IPv6 destination address</td>
</tr>
<tr>
<td>ipv6_flabel</td>
<td>Integer 32bit</td>
<td>IPv6 Flow Label</td>
</tr>
<tr>
<td>icmpv6_type</td>
<td>Integer 8bit</td>
<td>ICMPv6 type</td>
</tr>
<tr>
<td>icmpv6_code</td>
<td>Integer 8bit</td>
<td>ICMPv6 code</td>
</tr>
<tr>
<td>ipv6_nd_target</td>
<td>IPv6 address</td>
<td>Target address for ND</td>
</tr>
<tr>
<td>ipv6_nd_sll</td>
<td>MAC address</td>
<td>Source link-layer for ND</td>
</tr>
<tr>
<td>ipv6_nd_tll</td>
<td>MAC address</td>
<td>Target link-layer for ND</td>
</tr>
<tr>
<td>mpls_label</td>
<td>Integer 32bit</td>
<td>MPLS label</td>
</tr>
<tr>
<td>mpls_tc</td>
<td>Integer 8bit</td>
<td>MPLS TC</td>
</tr>
<tr>
<td>mpls_bos</td>
<td>Integer 8bit</td>
<td>MPLS BoS bit</td>
</tr>
<tr>
<td>pbb_isid</td>
<td>Integer 24bit</td>
<td>PBB I-SID</td>
</tr>
<tr>
<td>tunnel_id</td>
<td>Integer 64bit</td>
<td>Logical Port Metadata</td>
</tr>
<tr>
<td>ipv6_exthdr</td>
<td>Integer 16bit</td>
<td>IPv6 Extension Header pseudo-field</td>
</tr>
<tr>
<td>pbb_uca</td>
<td>Integer 8bit</td>
<td>PBB UCA header field (EXT-256 Old version of ONF Extension)</td>
</tr>
<tr>
<td>tcp_flags</td>
<td>Integer 16bit</td>
<td>TCP flags (EXT-109 ONF Extension)</td>
</tr>
<tr>
<td>actset_output</td>
<td>Integer 32bit</td>
<td>Output port from action set metadata (EXT-233 ONF Extension)</td>
</tr>
</tbody>
</table>

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 ryu.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:
     ```python
     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:
     ```python
     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:
     ```python
     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:

```python
match = parser.OFPMatch(vlan_vid=(0x1000 | 3))
```

- Packet Matching

<table>
<thead>
<tr>
<th>Non-VLAN-tagged</th>
<th>VLAN-tagged(vlan_id=3)</th>
<th>VLAN-tagged(vlan_id=5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>MATCH</td>
<td>x</td>
</tr>
</tbody>
</table>

---

**Flow Instruction Structures**

```python
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_id</td>
<td>Next table</td>
</tr>
</tbody>
</table>

```python
class ryu.ofproto.ofproto_v1_3_parser.OFPInstructionWriteMetadata (metadata, metadata_mask, type_=None, len_=None)
```

Write metadata instruction

This instruction writes the masked metadata value into the metadata field.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>metadata</td>
<td>Metadata value to write</td>
</tr>
<tr>
<td>metadata_mask</td>
<td>Metadata write bitmask</td>
</tr>
</tbody>
</table>

```python
class ryu.ofproto.ofproto_v1_3_parser.OFPInstructionActions (type_, actions=None, len_=None)
```

Actions instruction

This instruction writes/applies/clears the actions.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>One of following values.</td>
</tr>
<tr>
<td></td>
<td>OFPIT_WRITE_ACTIONS</td>
</tr>
<tr>
<td></td>
<td>OFPIT_APPLY_ACTIONS</td>
</tr>
<tr>
<td></td>
<td>OFPIT_CLEAR_ACTIONS</td>
</tr>
<tr>
<td>actions</td>
<td>list of OpenFlow action class</td>
</tr>
</tbody>
</table>
type attribute corresponds to type_ parameter of __init__.

class ryu.ofproto.ofproto_v1_3_parser.OFPInstructionMeter(meter_id=1,
    type_=None,
    len_=None)

Meter instruction
This instruction applies the meter.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>meter_id</td>
<td>Meter instance</td>
</tr>
</tbody>
</table>

**Action Structures**

class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>port</td>
<td>Output port</td>
</tr>
<tr>
<td>max_len</td>
<td>Max length to send to controller</td>
</tr>
</tbody>
</table>

class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>group_id</td>
<td>Group identifier</td>
</tr>
</tbody>
</table>

class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_id</td>
<td>Queue ID for the packets</td>
</tr>
</tbody>
</table>

class ryu.ofproto.ofproto_v1_3_parser.OFPActionSetMplsTtl(mpls_ttl, 
    type_=None, len_=None)

Set MPLS TTL action
This action sets the MPLS TTL.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpls_ttl</td>
<td>MPLS TTL</td>
</tr>
</tbody>
</table>

class ryu.ofproto.ofproto_v1_3_parser.OFPActionDecMplsTtl(type_=None, 
    len_=None)

Decrement MPLS TTL action
This action decrements the MPLS TTL.
class ryu.ofproto.ofproto_v1_3_parser.OFPActionSetNwTtl(nw_ttl, type_=None, len_=None)

Set IP TTL action
This action sets the IP TTL.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nw_ttl</td>
<td>IP TTL</td>
</tr>
</tbody>
</table>

class ryu.ofproto.ofproto_v1_3_parser.OFPActionDecNwTtl(type_=None, len_=None)

Decrement IP TTL action
This action decrements the IP TTL.

class ryu.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 ryu.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 ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ethertype</td>
<td>Ether type. The default is 802.1Q. (0x8100)</td>
</tr>
</tbody>
</table>

class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ethertype</td>
<td>Ether type</td>
</tr>
</tbody>
</table>

class ryu.ofproto.ofproto_v1_3_parser.OFPActionPopVlan(type_=None, len_=None)

Pop VLAN action
This action pops the outermost VLAN tag from the packet.

class ryu.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 ryu.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:

```python
set_field = OFPActionSetField(eth_src="00:00:00:00:00:00")
```

class ryu.ofproto.ofproto_v1_3_parser.OFPActionExperimenter(experimenter)

Experimenter action

This action is an extensible action for the experimenter.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>experimenter</td>
<td>Experimenter ID</td>
</tr>
</tbody>
</table>

Note: For the list of the supported Nicira experimenter actions, please refer to `ryu.ofproto.nx_actions`.

2.5.5 OpenFlow v1.4 Messages and Structures

Controller-to-Switch Messages

Handshake

class ryu.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 Ryu framework, so the Ryu application do not need to process this typically.

Example:

```python
def send_features_request(self, datapath):
    ofp_parser = datapath.ofproto_parser
    req = ofp_parser.OFPFeaturesRequest(datapath)
    datapath.send_msg(req)
```

JSON Example:

```json
{
    "OFPFeaturesRequest": {}
}
```

class ryu.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 Ryu framework, so the Ryu application do not need to process this typically.

Example:
```python
@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',
    msgdatapath_id, msgn_buffers, msgn_tables,
    msgauxiliary_id, msgcapabilities)

JSON Example:
{
    "OFPSwitchFeatures": {
        "auxiliary_id": 99,
        "capabilities": 79,
        "datapath_id": 9210263729383,
        "n_buffers": 0,
        "n_tables": 255
    }
}

Switch Configuration

class ryu.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 configuration parameters.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Bitmap of the following flags.</td>
</tr>
<tr>
<td></td>
<td>OFPC_FRAG_NORMAL</td>
</tr>
<tr>
<td></td>
<td>OFPC_FRAG_DROP</td>
</tr>
<tr>
<td></td>
<td>OFPC_FRAG_REASM</td>
</tr>
<tr>
<td>miss_send_len</td>
<td>Max bytes of new flow that datapath should send to the controller</td>
</tr>
</tbody>
</table>

Example:
```python
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:
```json
{
    "OFPSetConfig": {
```
(continues on next page)
class ryu.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:

```python
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 ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Bitmap of the following flags.</td>
</tr>
<tr>
<td></td>
<td>OFPC_FRAG_NORMAL</td>
</tr>
<tr>
<td></td>
<td>OFPC_FRAG_DROP</td>
</tr>
<tr>
<td></td>
<td>OFPC_FRAG_REASM</td>
</tr>
<tr>
<td>miss_send_len</td>
<td>Max bytes of new flow that datapath should send to the controller</td>
</tr>
</tbody>
</table>

Example:

```python
@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 = []

    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('OFPGetConfigReply received: '}
```
Modify State Messages

class ryu.ofproto.ofproto_v1_4_parser.OFPTableMod(datapath, table_id, config, properties)

Flow table configuration message

The controller sends this message to configure table state.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_id</td>
<td>ID of the table (OFPTT_ALL indicates all tables)</td>
</tr>
<tr>
<td>config</td>
<td>Bitmap of configuration flags.</td>
</tr>
<tr>
<td>properties</td>
<td>List of OFPTableModProp subclass instance</td>
</tr>
</tbody>
</table>

Example:

```python
def send_table_mod(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPTableMod(datapath, 1, 3)
    flags = ofp.OFPTC_VACANCY_EVENTS
    properties = [ofp_parser.OFPTableModPropEviction(flags)]
    req = ofp_parser.OFPTableMod(datapath, 1, 3, properties)
    datapath.send_msg(req)
```

JSON Example:

```json
{
    "OFPTableMod": {
        "config": 0,
        "properties": [
            {
                "OFPTableModPropEviction": {
                    "flags": 0,
                    "length": 8,
                    "type": 2
                }
            }
        ]
    }
}
```
Modify Flow entry message

```python
class ryu.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)
```
The controller sends this message to modify the flow table.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cookie</td>
<td>Opaque controller-issued identifier</td>
</tr>
<tr>
<td>cookie_mask</td>
<td>Mask used to restrict the cookie bits that must match when the command is OFPFC_MODIFY* or OFPFC_DELETE*</td>
</tr>
<tr>
<td>table_id</td>
<td>ID of the table to put the flow in</td>
</tr>
<tr>
<td>command</td>
<td>One of the following values.</td>
</tr>
<tr>
<td></td>
<td>OFPFC_ADD</td>
</tr>
<tr>
<td></td>
<td>OFPFC_MODIFY</td>
</tr>
<tr>
<td></td>
<td>OFPFC_MODIFY_STRICT</td>
</tr>
<tr>
<td></td>
<td>OFPFC_DELETE</td>
</tr>
<tr>
<td></td>
<td>OFPFC_DELETE_STRICT</td>
</tr>
<tr>
<td>idle_timeout</td>
<td>Idle time before discarding (seconds)</td>
</tr>
<tr>
<td>hard_timeout</td>
<td>Max time before discarding (seconds)</td>
</tr>
<tr>
<td>priority</td>
<td>Priority level of flow entry</td>
</tr>
<tr>
<td>buffer_id</td>
<td>Buffered packet to apply to (or OFP_NO_BUFFER)</td>
</tr>
<tr>
<td>out_port</td>
<td>For OFPFC_DELETE* commands, require matching entries to include this as an output port</td>
</tr>
<tr>
<td>out_group</td>
<td>For OFPFC_DELETE* commands, require matching entries to include this as an output group</td>
</tr>
<tr>
<td>flags</td>
<td>Bitmap of the following flags.</td>
</tr>
<tr>
<td></td>
<td>OFPFF_SEND_FLOW_REM</td>
</tr>
<tr>
<td></td>
<td>OFPFF_CHECK_OVERLAP</td>
</tr>
<tr>
<td></td>
<td>OFPFF_RESET_COUNTS</td>
</tr>
<tr>
<td></td>
<td>OFPFF_NO_PKT_COUNTS</td>
</tr>
<tr>
<td></td>
<td>OFPFF_NO_BYT_COUNTS</td>
</tr>
<tr>
<td>importance</td>
<td>Eviction precedence</td>
</tr>
<tr>
<td>match</td>
<td>Instance of OFPMatch</td>
</tr>
<tr>
<td>instructions</td>
<td>list of OFPIInstruction* instance</td>
</tr>
</tbody>
</table>

Example:

```python
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
    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)]
```

(continues on next page)
ryu Documentation, Release 4.34

(continued from previous page)

```python
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:

```json
{
   "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
                    }
                  },

                  {
                    "OFPActionCopyTtlIn": {
                       "len": 8,
                       "type": 12
                    }
                  },

                  {
                    "OFPActionCopyTtlIn": {
                       "len": 8,
                       "type": 12
                    }
                  }
                ]
            }
        }
    }
}
```

(continues on next page)
{ "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
  }
},
{ "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": {
    "len": 16,
    "max_len": 65535,
    "port": 6,
    "type": 0
}
],
"len": 176,
"type": 3
},
"OFPInstructionActions": {
"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
}
},
"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
}
}
}


```json
"OFPInstructionGotoTable": {
    "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
}

{
    "OFPFlowMod": {
        "buffer_id": 65535,
        "command": 0,
        "cookie": 0,
        "cookie_mask": 0,
        "flags": 0,
        "hard_timeout": 0,
        "idle_timeout": 0,
        "importance": 0,
        "instructions": [
            { "OFPInstructionMeter": {
                "len": 8,
                "meter_id": 1,
                "type": 6
            }
        ],
        { "OFPInstructionActions": {

```
"actions": [  
  {  
    "OFPActionOutput": {  
      "len": 16,  
      "max_len": 65535,  
      "port": 6,  
      "type": 0  
    }  
  },  
  {  
    "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,  
              "value": "f2:0b:a4:7d:f8:ea"  
            }  
          }  
        ]  
      }  
    }  
  }  
}
"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,
  "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,
    "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": 55936
  }
}
"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,
    "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,
    "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,
        "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
            }
          }
        }
      }
    }
  }
},
"type": 1
},
"out_group": 4294967295,
"out_port": 4294967295,
"priority": 123,
"table_id": 1
}
(continues on next page)
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>command</td>
<td>One of the following values.</td>
</tr>
<tr>
<td></td>
<td>OFPGC_ADD</td>
</tr>
<tr>
<td></td>
<td>OFPGC_MODIFY</td>
</tr>
<tr>
<td></td>
<td>OFPGC_DELETE</td>
</tr>
<tr>
<td>type</td>
<td>One of the following values.</td>
</tr>
<tr>
<td></td>
<td>OFPGT_ALL</td>
</tr>
<tr>
<td></td>
<td>OFPGT_SELECT</td>
</tr>
<tr>
<td></td>
<td>OFPGT_INDIRECT</td>
</tr>
<tr>
<td></td>
<td>OFPGT_FF</td>
</tr>
<tr>
<td>group_id</td>
<td>Group identifier</td>
</tr>
<tr>
<td>buckets</td>
<td>list of OFPBucket</td>
</tr>
</tbody>
</table>

The type attribute corresponds to the type parameter of \_\_init\_.

Example:

```python
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,

    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:

```json
{
    "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 ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>port_no</td>
<td>Port number to modify</td>
</tr>
<tr>
<td>hw_addr</td>
<td>The hardware address that must be the same as hw_addr of OFPPort of OFPSwitchFeatures</td>
</tr>
<tr>
<td>config</td>
<td>Bitmap of configuration flags.</td>
</tr>
<tr>
<td></td>
<td>OFPPC_PORT_DOWN</td>
</tr>
<tr>
<td></td>
<td>OFPPC_NO_RECV</td>
</tr>
<tr>
<td></td>
<td>OFPPC_NO_FWD</td>
</tr>
<tr>
<td></td>
<td>OFPPC_NO_PACKET_IN</td>
</tr>
<tr>
<td>mask</td>
<td>Bitmap of configuration flags above to be changed</td>
</tr>
<tr>
<td>properties</td>
<td>List of OFPPortModProp subclass instance</td>
</tr>
</tbody>
</table>

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 |
ofp.OFPPF_AUTONEG | ofp.OFPPF_PAUSE |
ofp.OFPPF_PAUSEASYM)
properties = [ofp_parser.OFPPortModPropEthernet(advertise)]
req = ofp_parser.OFPPortMod(datapath, port_no, hw_addr, config,
mask, properties)
datapath.send_msg(req)

JSON Example:

```json
{
    "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
    }
}
```

(continues on next page)
class ryu.ofproto.ofproto_v1_4_parser.OFMeterMod(datapath, command=0, flags=1, meter_id=1, bands=None)

Meter modification message

The controller sends this message to modify the meter.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>command</td>
<td>One of the following values.</td>
</tr>
<tr>
<td></td>
<td>OFPMC_ADD</td>
</tr>
<tr>
<td></td>
<td>OFPMC_MODIFY</td>
</tr>
<tr>
<td></td>
<td>OFPMC_DELETE</td>
</tr>
<tr>
<td>flags</td>
<td>Bitmap of the following flags.</td>
</tr>
<tr>
<td></td>
<td>OFPMF_KBPS</td>
</tr>
<tr>
<td></td>
<td>OFPMF_PKTPS</td>
</tr>
<tr>
<td></td>
<td>OFPMF_BURST</td>
</tr>
<tr>
<td></td>
<td>OFPMF_STATS</td>
</tr>
<tr>
<td>meter_id</td>
<td>Meter instance</td>
</tr>
<tr>
<td>bands</td>
<td>list of the following class instance.</td>
</tr>
<tr>
<td></td>
<td>OFPMeterBandDrop</td>
</tr>
<tr>
<td></td>
<td>OFPMeterBandDscpRemark</td>
</tr>
<tr>
<td></td>
<td>OFPMeterBandExperimenter</td>
</tr>
</tbody>
</table>

JSON Example:

```json
{
    "OFPMeterMod": {
        "bands": [
            {
                "OFPMeterBandDrop": {
                    "burst_size": 10,
                }
            }
        ]
    }
}
```
Multipart Messages

```python
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero or OFFMPF_REQ_MORE</td>
</tr>
</tbody>
</table>

Example:

```python
def send_desc_stats_request(self, datapath):
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPDescStatsRequest(datapath, 0)
    datapath.send_msg(req)
```

JSON Example:

```json
{
    "OFPDescStatsRequest": {
        "flags": 0,
```
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>Instance of OFPDescStats</td>
</tr>
</tbody>
</table>

Example:

```python
@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 ' %
                      (body.mfr_desc, body.hw_desc, body.sw_desc,)
                      'serial_num=%s dp_desc=%s',
                      body.serial_num, body.dp_desc)
```

JSON Example:

```json
{
    "OFPDescStatsReply": {
        "body": {
            "OFPDescStats": {
                "dp_desc": "dp",
                "hw_desc": "hw",
                "mfr_desc": "mfr",
                "serial_num": "serial",
                "sw_desc": "sw"
            }
        },
        "flags": 0,
        "type": 0
    }
}
```

class ryu.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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero or OFPMPF_REQ_MORE</td>
</tr>
<tr>
<td>table_id</td>
<td>ID of table to read</td>
</tr>
<tr>
<td>out_port</td>
<td>Require matching entries to include this as an output port</td>
</tr>
<tr>
<td>out_group</td>
<td>Require matching entries to include this as an output group</td>
</tr>
<tr>
<td>cookie</td>
<td>Require matching entries to contain this cookie value</td>
</tr>
<tr>
<td>cookie_mask</td>
<td>Mask used to restrict the cookie bits that must match</td>
</tr>
<tr>
<td>match</td>
<td>Instance of OFPMatch</td>
</tr>
</tbody>
</table>

Example:

```python
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:

```json
{
  "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
  }
}
```

### ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>List of OFPFlowStats instance</td>
</tr>
</tbody>
</table>

Example:
@set_ev_cls(ofp_event.EventOFPFlowStatsReply, MAIN_DISPATCHER)
def flow_stats_reply_handler(self, ev):
    flows = []
    for stat in ev.msg.body:
        flows.append('table_id=%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:

```json
{
    "OFPFlowStatsReply": {
        "body": [
            {
                "OFPFlowStats": {
                    "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
                    }
                }
            },
            {
                "OFPFlowStats": {
                    "byte_count": 0,
                    "cookie": 0,
                    "duration_nsec": 115055000,
                    "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
                    }
                }
            }
        ]
    }
}
```

(continues on next page)
"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,
    "flags": 0,
    "hard_timeout": 0,
    "idle_timeout": 0,
    "importance": 0,
    "instructions": [
      {"OFPInstructionGotoTable": {
        "len": 8,
        "table_id": 1,
        "type": 1
      }
    ]
  }}]


```json
{
  "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,
    "importance": 0,
    "instructions": [
      {
        "OFPInstructionActions": {
          "actions": [
            {
              "OFPInstructionSetField": {
                "field": {
                  "OXMTlv": {
                    "field": "vlan_vid",
                    "mask": null,
                    "value": 258
                  }
                },
                "len": 16,
                "type": 25
              }
            }
          ]
        }
      }
    ]
  }
}
```

(continues on next page)
"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": 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 }
},
{ "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
},

"OFPInstructionActions": {
    "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
}
},

"OFPInstructionActions": {
    "actions": [
    {
      "OFPActionOutput": {
        "len": 16,
        "max_len": 65535,
        "port": 4294967293,
        "type": 0
      }
    }
  ]
}
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero or OFPMPF_REQ_MORE</td>
</tr>
<tr>
<td>table_id</td>
<td>ID of table to read</td>
</tr>
<tr>
<td>out_port</td>
<td>Require matching entries to include this as an output port</td>
</tr>
<tr>
<td>out_group</td>
<td>Require matching entries to include this as an output group</td>
</tr>
<tr>
<td>cookie</td>
<td>Require matching entries to contain this cookie value</td>
</tr>
<tr>
<td>cookie_mask</td>
<td>Mask used to restrict the cookie bits that must match</td>
</tr>
<tr>
<td>match</td>
<td>Instance of OFPMatch</td>
</tr>
</tbody>
</table>

Example:

```python
def send_aggregate_stats_request(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser
    cookie = cookie_mask = 0
    ofp_parser.OFPAggregateStatsRequest(datapath, flags, table_id, out_port, out_group, cookie, cookie_mask, match)
```

(continues on next 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:

```json
{
  "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
  }
}
```

```python
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>Instance of OFPAggregateStats</td>
</tr>
</tbody>
</table>

Example:

```python
@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:

```json
{
  "OFPAggregateStatsReply": {
    "body": {
```
```
"OFPAggregateStats": {  
  "byte_count": 574,  
  "flow_count": 6,  
  "packet_count": 7  
},  
"flags": 0,  
"type": 2  
}  

```python
class ryu.ofproto.ofproto_v1_4_parser.OFPTableStatsRequest(datapath, flags, type_=None)
```

Table statistics request message

The controller uses this message to query flow table statictics.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero or OFMPF_REQ_MORE</td>
</tr>
</tbody>
</table>

Example:

```python
def send_table_stats_request(self, datapath):
    ofp_parser = datapath.ofproto_parser
    req = ofp_parser.OFPTableStatsRequest(datapath, 0)
    datapath.send_msg(req)
```

JSON Example:

```json
{
    "OFPTableStatsRequest": {
        "flags": 0,
        "type": 3
    }
}
```

```python
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>List of OFPTableStats instance</td>
</tr>
</tbody>
</table>

Example:

```python
@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 ' %
                       (stat.table_id, stat.active_count,
                        stat.matched_count))
```

(continues on next page)
```python
stat.lookup_count, stat.matched_count)

self.logger.debug('TableStats: %s', tables)
```

**JSON Example:**

```json
{
   "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** `ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero or OFPPMF_REQ_MORE</td>
</tr>
</tbody>
</table>

**Example:**

```python
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:**

```json
{
   "OFPTableDescStatsRequest": {
      "flags": 0,
      "type": 14
   }
}
```
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>List of OFPTableDesc instance</td>
</tr>
</tbody>
</table>

Example:

```python
@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:

```json
{
    "OFPTableDescStatsReply": {
        "body": [
            {
                "OFPTableDesc": {
                    "config": 0,
                    "length": 24,
                    "properties": [
                        {
                            "OFPTableModPropExperimenter": {
                                "data": [],
                                "exp_type": 0,
                                "experimenter": 101,
                                "length": 12,
                                "type": 65535
                            }
                        }
                    ]
                },
                "table_id": 7
            }
        ]
    }
}
```

(continues on next page)
class ryu.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.
### OFTableFeaturesStatsReply

#### class ryu.ofproto.ofproto_v1_4_parser.OFPTableFeaturesStatsReply

```python
class ryu.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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>List of OFPTableFeaturesStats instances. The default is [].</td>
</tr>
</tbody>
</table>

#### JSON Example:

See an example in:

```
ryu/tests/unit/ofproto/json/of14/5-53-ofp_table_features_request.packet.json
```

### OFPPortStatsRequest

#### class ryu.ofproto.ofproto_v1_4_parser.OFPPortStatsRequest

```python
class ryu.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.

#### Example:

```python
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:

```json
{
    "OFPPortStatsRequest": {
        "flags": 0,
        "port_no": 4294967295,
        "type": 4
    }
}
```
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>List of OFPPortStats instance</td>
</tr>
</tbody>
</table>

Example:

```python
@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:

```json
{
    "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,
    "tx_freq_lmda": 1500,
    "tx_pwr": 2000,
    "temperature": 273
    }
    }
    }
    }
```

(continues on next page)
"tx_grid_span": 500,
"tx_offset": 700,
"tx_pwr": 2000,
"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,
  "duration_sec": 0,
  "length": 120,
  "port_no": 6,
  "properties": [
    } (continues on next page)
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero or OFPMPF_REQ_MORE</td>
</tr>
</tbody>
</table>

Example:

```python
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:

```json
{
    "OFPPortDescStatsRequest": {
        "flags": 0,
        "type": 13
    }
}
```

class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>List of OFPPort instance</td>
</tr>
</tbody>
</table>

Example:

```python
@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 ' %
                      (p.port_no, p.hw_addr, p.name, p.config, p.state, repr(p.properties)))
    self.logger.debug('OFPPortDescStatsReply received: %s', ports)
```

JSON Example:

```json
{
    "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,
                            "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
                        }
                    }
                ]
            }
        }
    }
}
```

(continues on next page)
{
    "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,
        "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
        }]
}
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero or OFPMPF_REQ_MORE</td>
</tr>
<tr>
<td>port_no</td>
<td>Port number to read</td>
</tr>
<tr>
<td>queue_id</td>
<td>ID of queue to read</td>
</tr>
</tbody>
</table>

Example:
```python
def send_queue_stats_request(self, datapath):
ofp = datapath.ofproto
oxp_parser = datapath.ofproto_parser

req = ofp_parser.OFPQueueStatsRequest(datapath, 0, ofp.OFPP_ANY, ofp.OFPQ_ALL)
datapath.send_msg(req)
```

JSON Example:
```json
{
    "OFPQueueStatsRequest": {
        "flags": 0,
        "port_no": 4294967295,
        "queue_id": 4294967295,
        "type": 5
    }
}
```

class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>List of OFPQueueStats instance</td>
</tr>
</tbody>
</table>

Example:
```python
@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 ' %
                       (stat.port_no, stat.queue_id,
                        'tx_bytes=%d tx_packets=%d tx_errors=%d ' %
                        (stat.tx_bytes, stat.tx_packets, stat.tx_errors,
                        'duration_sec=%d duration_nsec=%d ' %
                        (stat.duration_sec, stat.duration_nsec,
                        'properties=%s' %
                        (stat.properties))))
    self.logger.debug('QueueStats: %s', queues)
```

JSON Example:

```json
{
    "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
                            }
                        }
                    ]
                }
            }
        ]
    }
}
```

(continues on next page)
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero or OFMPF_REQ_MORE</td>
</tr>
<tr>
<td>port_no</td>
<td>Port number to read (OFPP_ANY for all ports)</td>
</tr>
<tr>
<td>queue_id</td>
<td>ID of queue to read (OFPQ_ALL for all queues)</td>
</tr>
</tbody>
</table>

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:

```json
{
    "OFPQueueDescStatsRequest": {
        "flags": 0,
        "port_no": 7,
        "queue_id": 4294967295,
        "type": 15
    }
}
```

class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>List of OFPQueueDesc instance</td>
</tr>
</tbody>
</table>

Example:

```python
@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:

```json
{
    "OFPQueueDescStatsReply": {
        "body": [
            {
                "OFPQueueDesc": {
                    "len": 32,
                    "port_no": 7,
                    "properties": [
                        {
                            "OFPQueueDescPropExperimenter": {
                                "data": [],
                                "exp_type": 0,
                                "experimenter": 101,
                                "length": 12,
                                "type": 65535
                            }
                        }
                    ]
                }
            }
        ]
    }
}
```

(continues on next page)
```json
{
    "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,
                    "length": 16,
                    "type": 65535
                }
            },
            {
                "OFPQueueDescPropExperimenter": {
                    "data": [1, 2],
                    "exp_type": 2,
                    "experimenter": 101,
                    "length": 20,
                    "type": 65535
                }
            }
        ]
    }
}
(continues on next page)
```
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero or OFMPF_REQ_MORE</td>
</tr>
<tr>
<td>group_id</td>
<td>ID of group to read (OFPG_ALL to all groups)</td>
</tr>
</tbody>
</table>

Example:

```python
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:

```json
{
    "OFPGroupStatsRequest": {
        "flags": 0,
        "group_id": 4294967292,
        "type": 6
    }
}
```

class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>List of OFPGStats instance</td>
</tr>
</tbody>
</table>

Example:

```python
@set_ev_cls(ofp_event.EventOFPGroupStatsReply, MAIN_DISPATCHER)
def group_stats_reply_handler(self, ev):
    groups = []
    for stat in ev.msg.body:
        groups.append(stat)
```

(continues on next page)
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,
                "duration_sec": 9,
                "group_id": 1,
                "length": 56,
                "packet_count": 123,
                "ref_count": 2
            }
        }
        "flags": 0,
        "type": 6
    }
```

```ruby

class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero or OFPMPF_REQ_MORE</td>
</tr>
</tbody>
</table>

Example:

```python
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 ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>List of OFPGroupDescStats instance</td>
</tr>
</tbody>
</table>

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 ' %
                      (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,
"type": 7
}
}

```python
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero or OFPMPF_REQ_MORE</td>
</tr>
</tbody>
</table>

Example:
```python
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
   }
}
```

```python
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>Instance of OFPGroupFeaturesStats</td>
</tr>
</tbody>
</table>

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=%08x max_groups=%s ' +
                       'actions=%s',
                       body.types, body.capabilities, body.max_groups, body.actions)

JSON Example:

```json
{
    "OFPGroupFeaturesStatsReply": {
        "body": {
            "OFPGroupFeaturesStats": {
                "actions": [67082241, 67082241, 67082241, 67082241],
                "capabilities": 5,
                "max_groups": [16777216, 16777216, 16777216, 16777216],
                "types": 15
            }
        },
        "flags": 0,
        "type": 8
    }
}
```

class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero or OFPMPF_REQ_MORE</td>
</tr>
<tr>
<td>meter_id</td>
<td>ID of meter to read (OFPM_ALL to all meters)</td>
</tr>
</tbody>
</table>

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:

```json
{
    "OFPMeterStatsRequest": {
        "flags": 0,
        "meter_id": 4294967295,
        "type": 9
    }
}
```

class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>List of OFPMeterStats instance</td>
</tr>
</tbody>
</table>

Example:

```python
@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:

```json
{
    "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
            }
        ]
    }
}
```

(continues on next page)
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero or OFPMPPF_REQ_MORE</td>
</tr>
<tr>
<td>meter_id</td>
<td>ID of meter to read (OFPM_ALL to all meters)</td>
</tr>
</tbody>
</table>

Example:

```python
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:

```json
{
    "OFPMeterConfigStatsRequest": {
        "flags": 0,
        "meter_id": 4294967295,
        "type": 10
    }
}
```

class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>List of OFPMeterConfigStats instance</td>
</tr>
</tbody>
</table>

Example:

```python
@set_ev_cls(ofp_event.EventOFPMeterConfigStatsReply, MAIN_DISPATCHER)
def meter_config_stats_reply_handler(self, ev):
    configs = []
```

(continues on next page)
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:

```json
{
    "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
    }
}
```

```python
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero or OFMPF_REQ_MORE</td>
</tr>
</tbody>
</table>

Example:

```python
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:
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>List of OFPMeterFeaturesStats instance</td>
</tr>
</tbody>
</table>

Example:

```python
@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 ' %
                         (stat.max_meter, stat.band_types) +
                         'capabilities=0x%08x max_bands=%d ' %
                         (stat.capabilities, stat.max_bands) +
                         'max_color=%d ' %
                         (stat.max_color, stat.max_bands) +
                         'max_color=%d' %
                         (stat.max_color))
    self.logger.debug('MeterFeaturesStats: %s', features)
```

JSON Example:

```json
{
    "OFPMeterFeaturesStatsReply": {
        "body": {
            "OFPMeterFeaturesStats": {
                "band_types": 2147483654,
                "capabilities": 15,
                "max_bands": 255,
                "max_color": 0,
                "max_meter": 16777216
            }
        },
        "flags": 0,
        "type": 11
    }
}
```
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero or OFPFMF_REQ_MORE</td>
</tr>
<tr>
<td>monitor_id</td>
<td>Controller-assigned ID for this monitor</td>
</tr>
<tr>
<td>out_port</td>
<td>Require matching entries to include this as an output port</td>
</tr>
<tr>
<td>out_group</td>
<td>Require matching entries to include this as an output group</td>
</tr>
<tr>
<td>monitor_flags</td>
<td>Bitmap of the following flags.</td>
</tr>
<tr>
<td>table_id</td>
<td>ID of table to monitor</td>
</tr>
<tr>
<td>command</td>
<td>One of the following values.</td>
</tr>
<tr>
<td>match</td>
<td>Instance of OFPMatch</td>
</tr>
</tbody>
</table>

Example:

```python
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.OFP_ANY, ofp.OFPG_ANY,
                                           monitor_flags,
                                           ofp.OFPT_ALL,
                                           ofp.OFPFC_ADD, match)

    datapath.send_msg(req)
```
JSON Example:

```json
{
    "OFPFlowMonitorRequest": {
        "command": 0,
        "flags": 0,
        "match": {
            "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": 100000000,
        "out_group": 4294967295,
        "out_port": 22,
        "table_id": 33,
        "type": 16
    }
}
```

class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>List of list of the following class instance.</td>
</tr>
<tr>
<td></td>
<td>OFPFlowMonitorFull</td>
</tr>
<tr>
<td></td>
<td>OFPFlowMonitorAbbrev</td>
</tr>
<tr>
<td></td>
<td>OFPFlowMonitorPaused</td>
</tr>
</tbody>
</table>

Example:

```python
@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' % (update.length, update.event)
        if (update.event == ofp.OFPFME_INITIAL or
```

(continues on next page)
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 ' %
                  (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:

```json
{
    "OFPFlowMonitorReply": {
        "body": {
            "OFPFlowUpdateFull": {
                "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,
                        "oxm_fields": [
                            {
                                "OXMTlv": {
                                    "field": "eth_type",
                                    "mask": null,
                                    "value": 2054
                                }
                            }
                        ],
                        "length": 10
                    }
                }
            }
        }
    }
}
```

(continues on next page)
class ryu.ofproto.ofproto_v1_4_parser.OFPExperimenterStatsRequest ( datapath, flags, experimenter, exp_type, data, type_=None )

Experimenter multipart request message

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero or OFPMPF_REQ_MORE</td>
</tr>
<tr>
<td>experimenter</td>
<td>Experimenter ID</td>
</tr>
<tr>
<td>exp_type</td>
<td>Experimenter defined</td>
</tr>
<tr>
<td>data</td>
<td>Experimenter defined additional data</td>
</tr>
</tbody>
</table>

JSON Example:

```json
{
    "OFPExperimenterStatsRequest": {
        "data": "aG9nZWhvZ2U=",
        "exp_type": 3405678728,
        "experimenter": 3735928495,
        "flags": 0,
        "type": 65535
    }
}
```

class ryu.ofproto.ofproto_v1_4_parser.OFPExperimenterStatsReply ( datapath, type_=None, **kwargs )
Experimenter multipart reply message

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>An OFPExperimenterMultipart instance</td>
</tr>
</tbody>
</table>

JSON Example:

```
{
    "OFPExperimenterStatsReply": {
        "body": {
            "OFPExperimenterMultipart": {
                "data": "dGVzdGRhdGE5OTk5OTk5OQ==",
                "exp_type": 3405674359,
                "experimenter": 3735928495
            }
        },
        "flags": 0,
        "type": 65535
    }
}
```

Packet-Out Message

**class** `ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>buffer_id</td>
<td>ID assigned by datapath (OFP_NO_BUFFER if none)</td>
</tr>
<tr>
<td>in_port</td>
<td>Packet’s input port or OFPP_CONTROLLER</td>
</tr>
<tr>
<td>actions</td>
<td>list of OpenFlow action class</td>
</tr>
<tr>
<td>data</td>
<td>Packet data of a binary type value or an instances of packet.Packet.</td>
</tr>
</tbody>
</table>

Example:

```python
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": {
            ...
        }
    }
}
```
Barrier Message

class ryu.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:

```python
def send_barrier_request(self, datapath):
ofp_parser = datapath.ofproto_parser
req = ofp_parser.OFPBarrierRequest(datapath)
datapath.send_msg(req)
```

JSON Example:

```json
{
    "OFPBarrierRequest": {}
}
```

class ryu.ofproto.ofproto_v1_4_parser.OFPBarrierReply (datapath)
Barrier reply message

The switch responds with this message to a barrier request.

Example:

```python
@set_ev_cls(ofp_event.EventOFPBarrierReply, MAIN_DISPATCHER)
def barrier_reply_handler(self, ev):
    self.logger.debug('OFPBarrierReply received')
```

JSON Example:

```json
{
    "OFPBarrierReply": {}
}
```
Role Request Message

class ryu.ofproto.ofproto_v1_4_parser.OFPRoleRequest (datapath, role=None, generation_id=None)

Role request message

The controller uses this message to change its role.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>role</td>
<td>One of the following values.</td>
</tr>
<tr>
<td></td>
<td>OFPCR_ROLE_NOCHANGE</td>
</tr>
<tr>
<td></td>
<td>OFPCR_ROLE_EQUAL</td>
</tr>
<tr>
<td></td>
<td>OFPCR_ROLE_MASTER</td>
</tr>
<tr>
<td></td>
<td>OFPCR_ROLE_SLAVE</td>
</tr>
<tr>
<td>generation_id</td>
<td>Master Election Generation ID</td>
</tr>
</tbody>
</table>

Example:

def send_role_request(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser
    req = ofp_parser.OFPRoleRequest(datapath, ofp.OFPCR_ROLE_EQUAL, 0)
    datapath.send_msg(req)

JSON Example:

{
    "OFPRoleRequest": {
        "generation_id": 17294086455919964160,
        "role": 2
    }
}

class ryu.ofproto.ofproto_v1_4_parser.OFPRoleReply (datapath, role=None, generation_id=None)

Role reply message

The switch responds with this message to a role request.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>role</td>
<td>One of the following values.</td>
</tr>
<tr>
<td></td>
<td>OFPCR_ROLE_NOCHANGE</td>
</tr>
<tr>
<td></td>
<td>OFPCR_ROLE_EQUAL</td>
</tr>
<tr>
<td></td>
<td>OFPCR_ROLE_MASTER</td>
</tr>
<tr>
<td></td>
<td>OFPCR_ROLE_SLAVE</td>
</tr>
<tr>
<td>generation_id</td>
<td>Master Election Generation ID</td>
</tr>
</tbody>
</table>

Example:
@set_ev_cls(ofp_event.EventOFPRoleReply, 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('OFPRoleReply received: '
                      'role=%s generation_id=%d',
                      role, msg.generation_id)

JSON Example:

{
    "OFPRoleReply": {
        "generation_id": 17294086455919964160,
        "role": 3
    }
}

Bundle Messages

class ryu.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
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bundle_id</td>
<td>Id of the bundle</td>
</tr>
<tr>
<td>type</td>
<td>One of the following values.</td>
</tr>
<tr>
<td></td>
<td>OFPBCT_OPEN_REQUEST</td>
</tr>
<tr>
<td></td>
<td>OFPBCT_OPEN_REPLY</td>
</tr>
<tr>
<td></td>
<td>OFPBCT_CLOSE_REQUEST</td>
</tr>
<tr>
<td></td>
<td>OFPBCT_CLOSE_REPLY</td>
</tr>
<tr>
<td></td>
<td>OFPBCT_COMMIT_REQUEST</td>
</tr>
<tr>
<td></td>
<td>OFPBCT_COMMIT_REPLY</td>
</tr>
<tr>
<td></td>
<td>OFPBCT_DISCARD_REQUEST</td>
</tr>
<tr>
<td></td>
<td>OFPBCT_DISCARD_REPLY</td>
</tr>
<tr>
<td>flags</td>
<td>Bitmap of the following flags.</td>
</tr>
<tr>
<td></td>
<td>OFPBF_ATOMIC</td>
</tr>
<tr>
<td></td>
<td>OFPBF_ORDERED</td>
</tr>
<tr>
<td>properties</td>
<td>List of OFPBundleProp subclass instance</td>
</tr>
</tbody>
</table>

Example:

```python
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:

```json
{
    "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": 12,
                    "type": 65535
                }
            }
        ]
    }
}
```

(continues on next page)
class ryu.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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bundle_id</td>
<td>Id of the bundle</td>
</tr>
<tr>
<td>flags</td>
<td>Bitmap of the following flags.</td>
</tr>
<tr>
<td>message</td>
<td>MsgBase subclass instance</td>
</tr>
<tr>
<td>properties</td>
<td>List of OFPBundleProp subclass instance</td>
</tr>
</tbody>
</table>

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:

```json
{
    "OFPBundleAddMsg": {
        "bundle_id": 1234,
        "flags": 1,
        ...
    }
}
```
"message": {
    "OFPEchoRequest": {
        "data": null
    },
    "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
            }
        }
    ]
}

Set Asynchronous Configuration Message

**class** ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>properties</td>
<td>List of OFPAsyncConfigProp subclass instances</td>
</tr>
</tbody>
</table>

Example:
```python
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:

```json
{
    "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
            }
        }
    ]
}
```

(continues on next page)
"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": [],
    "exp_type": 0,
    "experimenter": 101,
    "length": 12,
    "type": 65534
},

"OFPAsyncConfigPropExperimenter": {
    "data": [1]
}
class ryu.ofproto.ofproto_v1_4_parser.OFPGetAsyncRequest(datapath)
Get asynchronous configuration request message

The controller uses this message to query the asynchronous message.

Example:

```python
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 ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>properties</td>
<td>List of OFPAsyncConfigProp subclass instances</td>
</tr>
</tbody>
</table>

Example:

```python
@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
            }
         },
         {
            "OFPAsyncConfigPropReasons": {
               "length": 8,
               "mask": 3,
               "type": 4
            }
         },
         {
            "OFPAsyncConfigPropReasons": {
               "length": 8,
               "mask": 3,
               "type": 5
            }
         },
         {
            "OFPAsyncConfigPropReasons": {
               "length": 8,
               "mask": 3,
               "type": 6
            }
         }
      ]
   }
```

(continues on next page)
{ "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": [],
    "exp_type": 0,
    "experimenter": 101,
    "length": 12,
    "type": 65534
  }
},
{ "OFPAsyncConfigPropExperimenter": {
    "data": [
      1
    ],
    "exp_type": 1,
    "experimenter": 101,
    "length": 16,
    "type": 65535
  }
},
{ "OFPAsyncConfigPropExperimenter": {
  (continues on next page)
Asynchronous Messages

Packet-In Message

class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>buffer_id</td>
<td>ID assigned by datapath</td>
</tr>
<tr>
<td>total_len</td>
<td>Full length of frame</td>
</tr>
<tr>
<td>reason</td>
<td>Reason packet is being sent.</td>
</tr>
<tr>
<td></td>
<td>OFPR_TABLE_MISS</td>
</tr>
<tr>
<td></td>
<td>OFPR_APPLY_ACTION</td>
</tr>
<tr>
<td></td>
<td>OFPR_INVALID_TTL</td>
</tr>
<tr>
<td></td>
<td>OFPR_ACTION_SET</td>
</tr>
<tr>
<td></td>
<td>OFPR_GROUP</td>
</tr>
<tr>
<td></td>
<td>OFPR_PACKET_OUT</td>
</tr>
<tr>
<td>table_id</td>
<td>ID of the table that was looked up</td>
</tr>
<tr>
<td>cookie</td>
<td>Cookie of the flow entry that was looked up</td>
</tr>
<tr>
<td>match</td>
<td>Instance of OFPMatch</td>
</tr>
<tr>
<td>data</td>
<td>Ethernet frame</td>
</tr>
</tbody>
</table>

Example:

```python
@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.OFPTABLE_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'
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:

```json
{
    "OFPPacketIn": {
        "buffer_id": 2,
        "cookie": 283688884868096,
        "data": \"/////////8gukffjqCAYAAQgABgQAAfILpH346goAAAEAAAAAAAKAAD\",
        "match": {
            "OFPMacMatch": {
                "length": 80,
                "ofx_mact_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": "ff:ff:ff:ff:ff:ff"
                    }}
                ]
            }
        }
    }
}
```

(continues on next 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": 3,
"table_id": 1,
"total_len": 42
}

{
  "OFPacketIn": {
    "buffer_id": 4026531840,
    "cookie": 283686884868096,
    "data": 
    "match": {
      "OFPMatch": {
        "in_port": 1,
        "eth_type": 0x0800,
        "ip_proto": 0,
        "ip_dscp": 0,
        "ip_ecn": 0,
        "ip_vrf": 0,
        "ip_src": "10.0.0.1",
        "ip_dst": "10.0.0.3",
        "tcp_src": 80,
        "tcp_dst": 80,
        "udp_src": 0,
        "udp_dst": 0,
        "icmpv4_type": 0,
        "icmpv4_code": 0,
        "arp_op": 1,
        "arp_spa": "10.0.0.1",
        "arp_tpa": "10.0.0.3",
        "arp_sha": "f2:0b:a4:7d:f8:ea",
        "arp_tha": "00:00:00:00:00:00"
      }
    }
  }
}
"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",
      "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",
    "mask": null,
    "value": 18080
  }
},
{
  "OXMTlv":{
    "field": "udp_src",
    "mask": null,
    "value": 28080
  }
},
{
  "OXMTlv":{
    "field": "udp_dst",
    "mask": null,
    "value": 55936
  }
}}
(continues on next page)
{ "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,
  "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,
      "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_sll",
      "mask": null,
      "value": "00:00:00:00:02:9a"
    }
  }
"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,
  "value": 1
},

"OXMTlv": {
  "field": "pbb_isid",
  "mask": null,
  "value": 11259375
},

"OXMTlv": {
  "field": "tunnel_id",
  "mask": null,
  "value": 65106155542690057
},

"OXMTlv": {
  "field": "ipv6_exthdr",
  "mask": null,
  "value": 500
},

"OXMTlv": {
  "field": "pbb_uca",
  "mask": null,
  "value": 1
}

"type": 1
}
Flow Removed Message

class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cookie</td>
<td>Opaque controller-issued identifier</td>
</tr>
<tr>
<td>priority</td>
<td>Priority level of flow entry</td>
</tr>
<tr>
<td>reason</td>
<td>One of the following values.</td>
</tr>
<tr>
<td></td>
<td>OFPRR_IDLE_TIMEOUT</td>
</tr>
<tr>
<td></td>
<td>OFPRR_HARD_TIMEOUT</td>
</tr>
<tr>
<td></td>
<td>OFPRR_DELETE</td>
</tr>
<tr>
<td></td>
<td>OFPRR_GROUP_DELETE</td>
</tr>
<tr>
<td></td>
<td>OFPRR_METER_DELETE</td>
</tr>
<tr>
<td></td>
<td>OFPRR_EVICTION</td>
</tr>
<tr>
<td>table_id</td>
<td>ID of the table</td>
</tr>
<tr>
<td>duration_sec</td>
<td>Time flow was alive in seconds</td>
</tr>
<tr>
<td>duration_nsec</td>
<td>Time flow was alive in nanoseconds beyond duration_sec</td>
</tr>
<tr>
<td>idle_timeout</td>
<td>Idle timeout from original flow mod</td>
</tr>
<tr>
<td>hard_timeout</td>
<td>Hard timeout from original flow mod</td>
</tr>
<tr>
<td>packet_count</td>
<td>Number of packets that was associated with the flow</td>
</tr>
<tr>
<td>byte_count</td>
<td>Number of bytes that was associated with the flow</td>
</tr>
<tr>
<td>match</td>
<td>Instance of OFPMatch</td>
</tr>
</tbody>
</table>

Example:

```python
@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: '
                  '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:

```json
{
    "OFPFlowRemoved": {
        "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

```python
class ryu.ofproto.ofproto_v1_4_parser.OFPPortStatus(datapath, reason=None, desc=None)
```

Port status message
The switch notifies controller of change of ports.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reason</td>
<td>One of the following values.</td>
</tr>
<tr>
<td></td>
<td>OFPPR_ADD</td>
</tr>
<tr>
<td></td>
<td>OFPPR_DELETE</td>
</tr>
<tr>
<td></td>
<td>OFPPR_MODIFY</td>
</tr>
<tr>
<td>desc</td>
<td>instance of OFPPort</td>
</tr>
</tbody>
</table>

Example:
```python
@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:
```json
{
    "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,
                        }
                    }
                ]
            }
        }
    }
}
```

(continues on next page)
"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,
    "length": 20,
    "type": 65535
  }
},
"state": 4
},
"reason": 0
Controller Role Status Message

class ryu.ofproto.ofproto_v1_4_parser.OFPRoleStatus (datapath, role=None, reason=None, generation_id=None, properties=None)

Role status message

The switch notifies controller of change of role.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>role</td>
<td>One of the following values.</td>
</tr>
<tr>
<td></td>
<td>OFPCR_ROLE_NOCHANGE</td>
</tr>
<tr>
<td></td>
<td>OFPCR_ROLE_EQUAL</td>
</tr>
<tr>
<td></td>
<td>OFPCR_ROLE_MASTER</td>
</tr>
<tr>
<td>reason</td>
<td>One of the following values.</td>
</tr>
<tr>
<td></td>
<td>OFPCRR_MASTER_REQUEST</td>
</tr>
<tr>
<td></td>
<td>OFPCRR_CONFIG</td>
</tr>
<tr>
<td></td>
<td>OFPCRR_EXPERIMENTER</td>
</tr>
<tr>
<td>generation_id</td>
<td>Master Election Generation ID</td>
</tr>
<tr>
<td>properties</td>
<td>List of OFPRoleProp subclass instance</td>
</tr>
</tbody>
</table>

Example:

```python
@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:
```

(continues on next page)
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:

```json
{
  "OFPRoleStatus": {
    "generation_id": 7,
    "properties": [
      {"OFPRolePropExperimenter": {  
        "data": [],  
        "exp_type": 0,  
        "experimenter": 101,  
        "length": 12,  
        "type": 65535
      }},  
      {"OFPRolePropExperimenter": {  
        "data": [1],  
        "exp_type": 1,  
        "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**

```python
class ryu.ofproto.ofproto_v1_4_parser.OFPTableStatus(datapath, reason=None, table=None)
```

Table status message
The switch notifies controller of change of table status.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reason</td>
<td>One of the following values.</td>
</tr>
<tr>
<td></td>
<td>OFPTR_VACANCY_DOWN</td>
</tr>
<tr>
<td></td>
<td>OFPTR_VACANCY_UP</td>
</tr>
<tr>
<td>instance</td>
<td>OFPTableDesc instance</td>
</tr>
</tbody>
</table>

Example:

```python
@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=
                     %d config=%08x properties=%s',
                     reason, msg.table.table_id, msg.table.config,
                     repr(msg.table.properties))
```

JSON Example:

```json
{
    "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
                        }
                    }
                ]
            }
        }
    }
}
```

(continues on next page)
Request Forward Message

class ryu.ofproto.ofproto_v1_4_parser.OFPRequestForward(datapath, request=None)

Forwarded request message

The switch forwards request messages from one controller to other controllers.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>request</td>
<td>OFPGPGroupMod or OFPMeterMod instance</td>
</tr>
</tbody>
</table>

Example:

```python
@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, buckets=%s')',
        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('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:

```json
{
    "OFPRequestForward": {
        "request": {
            "OFPGroupMod": {
                "buckets": [
                    "OFPBucket": {
                        "actions": [
                            "OFPActionOutput": {
                                "len": 16,
                                "max_len": 65535,
                                "port": 2,
                                "type": 0
                            }
                        ]
                    },
                    "len": 32,
                    "watch_group": 1,
                    "watch_port": 1,
                    "weight": 1
                }
            }
        }
    }
}
```

Symmetric Messages
Hello

```python
class ryu.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 Ryu framework, so the Ryu application do not need to process this typically.
```

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>elements</td>
<td>list of OFPHelloElemVersionBitmap instance</td>
</tr>
</tbody>
</table>

JSON Example:

```
{
    "OFPHello": {
        "elements": [
            {
                "OFPHelloElemVersionBitmap": {
                    "length": 8,
                    "type": 1,
                    "versions": [
                        1,
                        2,
                        3,
                        9,
                        10,
                        30
                    ]
                }
            }
        ]
    }
}
```

```python
class ryu.ofproto.ofproto_v1_4_parser.OFPHelloElemVersionBitmap(versions, type_=None, length=None)
    Version bitmap Hello Element
```

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>versions</td>
<td>list of versions of OpenFlow protocol a device supports</td>
</tr>
</tbody>
</table>

Echo Request

```python
class ryu.ofproto.ofproto_v1_4_parser.OFPEchoRequest(datapath, data=None)
    Echo request message
    This message is handled by the Ryu framework, so the Ryu application do not need to process this typically.
```

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>An arbitrary length data</td>
</tr>
</tbody>
</table>

Example:
```python
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 ryu.ofproto.ofproto_v1_4_parser.OFPEchoReply (datapath, data=None)
Echo reply message
This message is handled by the Ryu framework, so the Ryu application do not need to process this typically.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>An arbitrary length data</td>
</tr>
</tbody>
</table>

Example:
```python
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=="
    }
}
Error Message

class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>High level type of error</td>
</tr>
<tr>
<td>code</td>
<td>Details depending on the type</td>
</tr>
<tr>
<td>data</td>
<td>Variable length data depending on the type and code</td>
</tr>
</tbody>
</table>

type attribute corresponds to type_ parameter of __init__.

Types and codes are defined in ryu.ofproto.ofproto.

<table>
<thead>
<tr>
<th>Type</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>OFPET_HELLO_FAILED</td>
<td>OFPHFC_*</td>
</tr>
<tr>
<td>OFPET_BAD_REQUEST</td>
<td>OFPBRC_*</td>
</tr>
<tr>
<td>OFPET_BAD_ACTION</td>
<td>OFPBAC_*</td>
</tr>
<tr>
<td>OFPET_BAD_INSTRUCTION</td>
<td>OFPBIC_*</td>
</tr>
<tr>
<td>OFPET_BAD_MATCH</td>
<td>OFPBMC_*</td>
</tr>
<tr>
<td>OFPET_FLOW_MOD_FAILED</td>
<td>OFPFMFC_*</td>
</tr>
<tr>
<td>OFPET_GROUP_MOD_FAILED</td>
<td>OFPGMFC_*</td>
</tr>
<tr>
<td>OFPET_PORT_MOD_FAILED</td>
<td>OFPPMFC_*</td>
</tr>
<tr>
<td>OFPET_TABLE_MOD_FAILED</td>
<td>OFPTMFC_*</td>
</tr>
<tr>
<td>OFPET_QUEUE_OP_FAILED</td>
<td>OFPQOFC_*</td>
</tr>
<tr>
<td>OFPET_SWITCH_CONFIG_FAILED</td>
<td>OFPSFC_*</td>
</tr>
<tr>
<td>OFPET_ROLE_REQUEST_FAILED</td>
<td>OFPRRFC_*</td>
</tr>
<tr>
<td>OFPET_METER_MOD_FAILED</td>
<td>OFPMMFC_*</td>
</tr>
<tr>
<td>OFPET_TABLE_FEATURES_FAILED</td>
<td>OFPTFFC_*</td>
</tr>
<tr>
<td>OFPET_EXPERIMENTER</td>
<td>N/A</td>
</tr>
</tbody>
</table>

If type == OFPET_EXPERIMENTER, this message has also the following attributes.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>exp_type</td>
<td>Experimenter defined type</td>
</tr>
<tr>
<td>experimenter</td>
<td>Experimenter ID</td>
</tr>
</tbody>
</table>

Example:

```python
@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:
Experimenter

```python
class ryu.ofproto.ofproto_v1_4_parser.OFPExperimenter(datapath, experimenter=None, exp_type=None, data=None)
```

Experimenter extension message

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>experimenter</td>
<td>Experimenter ID</td>
</tr>
<tr>
<td>exp_type</td>
<td>Experimenter defined</td>
</tr>
<tr>
<td>data</td>
<td>Experimenter defined arbitrary additional data</td>
</tr>
</tbody>
</table>

JSON Example:

```json
{
    "OFPExperimenter": {
        "data": "bmF6bw==",
        "exp_type": 123456789,
        "experimenter": 98765432
    }
}
```

Port Structures

```python
class ryu.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
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>port_no</td>
<td>Port number and it uniquely identifies a port within a switch.</td>
</tr>
<tr>
<td>length</td>
<td>Length of ofp_port (excluding padding).</td>
</tr>
<tr>
<td>hw_addr</td>
<td>MAC address for the port.</td>
</tr>
<tr>
<td>name</td>
<td>Null-terminated string containing a human-readable name for the interface.</td>
</tr>
<tr>
<td>config</td>
<td>Bitmap of port configuration flags.</td>
</tr>
<tr>
<td></td>
<td>OFPPC_PORT_DOWN</td>
</tr>
<tr>
<td></td>
<td>OFPPC_NO_RECV</td>
</tr>
<tr>
<td></td>
<td>OFPPC_NO_FWD</td>
</tr>
<tr>
<td></td>
<td>OFPPC_NO_PACKET_IN</td>
</tr>
<tr>
<td>state</td>
<td>Bitmap of port state flags.</td>
</tr>
<tr>
<td></td>
<td>OFPPS_LINK_DOWN</td>
</tr>
<tr>
<td></td>
<td>OFPPS_BLOCKED</td>
</tr>
<tr>
<td></td>
<td>OFPPS_LIVE</td>
</tr>
<tr>
<td>properties</td>
<td>List of OFPPortDescProp subclass instance</td>
</tr>
</tbody>
</table>

Flow Match Structure

```python
class ryu.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.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in_port</td>
<td>Integer 32bit</td>
<td>Switch input port</td>
</tr>
<tr>
<td>in_phy_port</td>
<td>Integer 32bit</td>
<td>Switch physical input port</td>
</tr>
<tr>
<td>metadata</td>
<td>Integer 64bit</td>
<td>Metadata passed between tables</td>
</tr>
<tr>
<td>eth_dst</td>
<td>MAC address</td>
<td>Ethernet destination address</td>
</tr>
<tr>
<td>eth_src</td>
<td>MAC address</td>
<td>Ethernet source address</td>
</tr>
<tr>
<td>eth_type</td>
<td>Integer 16bit</td>
<td>Ethernet frame type</td>
</tr>
<tr>
<td>vlan_vid</td>
<td>Integer 16bit</td>
<td>VLAN id</td>
</tr>
<tr>
<td>vlan_pcp</td>
<td>Integer 8bit</td>
<td>VLAN priority</td>
</tr>
<tr>
<td>ip_dscp</td>
<td>Integer 8bit</td>
<td>IP DSCP (6 bits in ToS field)</td>
</tr>
<tr>
<td>ip_ecn</td>
<td>Integer 8bit</td>
<td>IP ECN (2 bits in ToS field)</td>
</tr>
<tr>
<td>ip_proto</td>
<td>Integer 8bit</td>
<td>IP protocol</td>
</tr>
<tr>
<td>ipv4_src</td>
<td>IPv4 address</td>
<td>IPv4 source address</td>
</tr>
<tr>
<td>ipv4_dst</td>
<td>IPv4 address</td>
<td>IPv4 destination address</td>
</tr>
<tr>
<td>tcp_src</td>
<td>Integer 16bit</td>
<td>TCP source port</td>
</tr>
<tr>
<td>tcp_dst</td>
<td>Integer 16bit</td>
<td>TCP destination port</td>
</tr>
<tr>
<td>udp_src</td>
<td>Integer 16bit</td>
<td>UDP source port</td>
</tr>
<tr>
<td>udp_dst</td>
<td>Integer 16bit</td>
<td>UDP destination port</td>
</tr>
</tbody>
</table>

Continued on next page
### Table 3 – continued from previous page

<table>
<thead>
<tr>
<th>Argument</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sctp_src</td>
<td>Integer 16bit</td>
<td>SCTP source port</td>
</tr>
<tr>
<td>sctp_dst</td>
<td>Integer 16bit</td>
<td>SCTP destination port</td>
</tr>
<tr>
<td>icmpv4_type</td>
<td>Integer 8bit</td>
<td>ICMP type</td>
</tr>
<tr>
<td>icmpv4_code</td>
<td>Integer 8bit</td>
<td>ICMP code</td>
</tr>
<tr>
<td>arp_op</td>
<td>Integer 16bit</td>
<td>ARP opcode</td>
</tr>
<tr>
<td>arp_spa</td>
<td>IPv4 address</td>
<td>ARP source IPv4 address</td>
</tr>
<tr>
<td>arp_tpa</td>
<td>IPv4 address</td>
<td>ARP target IPv4 address</td>
</tr>
<tr>
<td>arp_sha</td>
<td>MAC address</td>
<td>ARP source hardware address</td>
</tr>
<tr>
<td>arp_tha</td>
<td>MAC address</td>
<td>ARP target hardware address</td>
</tr>
<tr>
<td>ipv6_src</td>
<td>IPv6 address</td>
<td>IPv6 source address</td>
</tr>
<tr>
<td>ipv6_dst</td>
<td>IPv6 address</td>
<td>IPv6 destination address</td>
</tr>
<tr>
<td>ipv6_flabel</td>
<td>Integer 32bit</td>
<td>IPv6 Flow Label</td>
</tr>
<tr>
<td>icmpv6_type</td>
<td>Integer 8bit</td>
<td>ICMPv6 type</td>
</tr>
<tr>
<td>icmpv6_code</td>
<td>Integer 8bit</td>
<td>ICMPv6 code</td>
</tr>
<tr>
<td>ipv6_nd_target</td>
<td>IPv6 address</td>
<td>Target address for ND</td>
</tr>
<tr>
<td>ipv6_nd_sll</td>
<td>MAC address</td>
<td>Source link-layer for ND</td>
</tr>
<tr>
<td>ipv6_nd_tll</td>
<td>MAC address</td>
<td>Target link-layer for ND</td>
</tr>
<tr>
<td>mpls_label</td>
<td>Integer 32bit</td>
<td>MPLS label</td>
</tr>
<tr>
<td>mpls_tc</td>
<td>Integer 8bit</td>
<td>MPLS TC</td>
</tr>
<tr>
<td>mpls_bos</td>
<td>Integer 8bit</td>
<td>MPLS BoS bit</td>
</tr>
<tr>
<td>pbb_isid</td>
<td>Integer 24bit</td>
<td>PBB I-SID</td>
</tr>
<tr>
<td>tunnel_id</td>
<td>Integer 64bit</td>
<td>Logical Port Metadata</td>
</tr>
<tr>
<td>ipv6_extthdr</td>
<td>Integer 16bit</td>
<td>IPv6 Extension Header pseudo-field</td>
</tr>
<tr>
<td>pbb_uca</td>
<td>Integer 8bit</td>
<td>PBB UCA header field</td>
</tr>
<tr>
<td>tcp_flags</td>
<td>Integer 16bit</td>
<td>TCP flags (EXT-109 ONF Extension)</td>
</tr>
<tr>
<td>actset_output</td>
<td>Integer 32bit</td>
<td>Output port from action set metadata (EXT-233 ONF Extension)</td>
</tr>
</tbody>
</table>

**Example:**

```python
>>> # 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 `ryu.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

<table>
<thead>
<tr>
<th></th>
<th>MATCH</th>
</tr>
</thead>
<tbody>
<tr>
<td>non-VLAN-tagged</td>
<td></td>
</tr>
<tr>
<td>VLAN-tagged(vlan_id=3)</td>
<td></td>
</tr>
<tr>
<td>VLAN-tagged(vlan_id=5)</td>
<td></td>
</tr>
</tbody>
</table>

2) Only packets without a VLAN tag

• Example:

```python
match = parser.OFPMatch(vlan_vid=0x0000)
```

• Packet Matching

<table>
<thead>
<tr>
<th></th>
<th>MATCH</th>
</tr>
</thead>
<tbody>
<tr>
<td>non-VLAN-tagged</td>
<td></td>
</tr>
<tr>
<td>VLAN-tagged(vlan_id=3)</td>
<td>x</td>
</tr>
<tr>
<td>VLAN-tagged(vlan_id=5)</td>
<td>x</td>
</tr>
</tbody>
</table>

3) Only packets with a VLAN tag regardless of its value

• Example:

```python
match = parser.OFPMatch(vlan_vid=(0x1000, 0x1000))
```

• Packet Matching

<table>
<thead>
<tr>
<th></th>
<th>MATCH</th>
</tr>
</thead>
<tbody>
<tr>
<td>non-VLAN-tagged</td>
<td>x</td>
</tr>
<tr>
<td>VLAN-tagged(vlan_id=3)</td>
<td></td>
</tr>
<tr>
<td>VLAN-tagged(vlan_id=5)</td>
<td></td>
</tr>
</tbody>
</table>

4) Only packets with VLAN tag and VID equal

• Example:

```python
match = parser.OFPMatch(vlan_vid=(0x1000 | 3))
```

• Packet Matching

<table>
<thead>
<tr>
<th></th>
<th>MATCH</th>
</tr>
</thead>
<tbody>
<tr>
<td>non-VLAN-tagged</td>
<td>x</td>
</tr>
<tr>
<td>VLAN-tagged(vlan_id=3)</td>
<td></td>
</tr>
<tr>
<td>VLAN-tagged(vlan_id=5)</td>
<td>x</td>
</tr>
</tbody>
</table>

Flow Instruction Structures

```python
class ryu.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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_id</td>
<td>Next table</td>
</tr>
</tbody>
</table>

**class** `ryu.ofproto.ofproto_v1_4_parser.OFPInstructionWriteMetadata(metadata, metadata_mask, type_=None, len_=None)`

Write metadata instruction

This instruction writes the masked metadata value into the metadata field.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>metadata</td>
<td>Metadata value to write</td>
</tr>
<tr>
<td>metadata_mask</td>
<td>Metadata write bitmask</td>
</tr>
</tbody>
</table>

**class** `ryu.ofproto.ofproto_v1_4_parser.OFPInstructionActions(type_, actions=None, len_=None)`

Actions instruction

This instruction writes/applies/clears the actions.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>One of following values.</td>
</tr>
<tr>
<td></td>
<td>OFPIT_WRITE_ACTIONS</td>
</tr>
<tr>
<td></td>
<td>OFPIT_APPLY_ACTIONS</td>
</tr>
<tr>
<td></td>
<td>OFPIT_CLEAR_ACTIONS</td>
</tr>
<tr>
<td>actions</td>
<td>list of OpenFlow action class</td>
</tr>
</tbody>
</table>

`type` attribute corresponds to `type_` parameter of `__init__`.

**class** `ryu.ofproto.ofproto_v1_4_parser.OFPInstructionMeter(meter_id=1, type_=None, len_=None)`

Meter instruction

This instruction applies the meter.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>meter_id</td>
<td>Meter instance</td>
</tr>
</tbody>
</table>

### Action Structures

**class** `ryu.ofproto.ofproto_v1_4_parser.OFPActionOutput(port, max_len=65509, type_=None, len_=None)`

Output action

This action indicates output a packet to the switch port.
### Attribute Description

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>port</td>
<td>Output port</td>
</tr>
<tr>
<td>max_len</td>
<td>Max length to send to controller</td>
</tr>
</tbody>
</table>

```python
class ryu.ofproto.ofproto_v1_4_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 ryu.ofproto.ofproto_v1_4_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 ryu.ofproto.ofproto_v1_4_parser.OFPActionSetMplsTtl (mpls_ttl, type_=None, len_=None)
    Set MPLS TTL action
    This action sets the MPLS TTL.
```

### Attribute Description

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpls_ttl</td>
<td>MPLS TTL</td>
</tr>
</tbody>
</table>

```python
class ryu.ofproto.ofproto_v1_4_parser.OFPActionDecMplsTtl (type_=None, len_=None)
    Decrement MPLS TTL action
    This action decrements the MPLS TTL.

class ryu.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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ethertype</td>
<td>Ether type. The default is 802.1Q (0x8100)</td>
</tr>
</tbody>
</table>

```python
class ryu.ofproto.ofproto_v1_4_parser.OFPActionPopVlan (ethertype=2048, type_=None, len_=None)
    Pop VLAN action
    This action pops the outermost VLAN tag from the packet.

class ryu.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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ethertype</td>
<td>Ether type.</td>
</tr>
</tbody>
</table>

```python
class ryu.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 ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_id</td>
<td>Queue ID for the packets</td>
</tr>
</tbody>
</table>

**class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>group_id</td>
<td>Group identifier</td>
</tr>
</tbody>
</table>

**class ryu.ofproto.ofproto_v1_4_parser.OFPActionSetNwTtl(nw_ttl, type_=None, len_=None)**

Set IP TTL action

This action sets the IP TTL.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nw_ttl</td>
<td>IP TTL</td>
</tr>
</tbody>
</table>

**class ryu.ofproto.ofproto_v1_4_parser.OFPActionDecNwTtl(type_=None, len_=None)**

Decrement IP TTL action

This action decrements the IP TTL.

**class ryu.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:

```python
set_field = OFPActionSetField(eth_src="00:00:00:00:00:00")
```

**class ryu.ofproto.ofproto_v1_4_parser.OFPActionPushPbb(ethertype, type_=None, len_=None)**

Push PBB action

This action pushes a new PBB header to the packet.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ethertype</td>
<td>Ether type</td>
</tr>
</tbody>
</table>

**class ryu.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 ryu.ofproto.ofproto_v1_4_parser.OFPActionExperimenter***(experimenter) Experimenter action

This action is an extensible action for the experimenter.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>experimenter</td>
<td>Experimenter ID</td>
</tr>
</tbody>
</table>

**Note:** For the list of the supported Nicira experimenter actions, please refer to *ryu.ofproto.nx_actions*.

### 2.5.6 OpenFlow v1.5 Messages and Structures

#### Controller-to-Switch Messages

##### Handshake

**class ryu.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 Ryu framework, so the Ryu application do not need to process this typically.

**Example:**

```python
def send_features_request(self, datapath):
ofp_parser = datapath.ofproto_parser
req = ofp_parser.OFPFeaturesRequest(datapath)
datapath.send_msg(req)
```

**JSON Example:**

```json
{
"OFPFeaturesRequest": {}
}
```

**class ryu.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 Ryu framework, so the Ryu application do not need to process this typically.

**Example:**

```python
@set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
def switch_features_handler(self, ev):
    msg = ev.msg
```

(continues on next page)
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:

```json
{
"OFPSwitchFeatures": {
  "auxiliary_id": 0,
  "capabilities": 79,
  "datapath_id": 1,
  "n_buffers": 255,
  "n_tables": 255
}
}
```

Switch Configuration

```python
class ryu.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 configuration parameters.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Bitmap of the following flags.</td>
</tr>
<tr>
<td></td>
<td>OFPC_FRAG_NORMAL</td>
</tr>
<tr>
<td></td>
<td>OFPC_FRAG_DROP</td>
</tr>
<tr>
<td></td>
<td>OFPC_FRAG_REASM</td>
</tr>
<tr>
<td>miss_send_len</td>
<td>Max bytes of new flow that datapath should send to the controller</td>
</tr>
</tbody>
</table>

Example:

```python
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:

```json
{
  "OFPSetConfig": {
    "flags": 0,
    "miss_send_len": 128
  }
}
```
class ryu.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:

```python
def send_get_config_request(self, datapath):
    ofp_parser = datapath.ofproto_parser
    req = ofp_parser.OFPGetConfigRequest(datapath)
datapath.send_msg(req)
```

JSON Example:

```json
{
    "OFPGetConfigRequest": {}
}
```

class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Bitmap of the following flags.</td>
</tr>
<tr>
<td></td>
<td>OFPC_FRAG_NORMAL</td>
</tr>
<tr>
<td></td>
<td>OFPC_FRAG_DROP</td>
</tr>
<tr>
<td></td>
<td>OFPC_FRAG_REASM</td>
</tr>
<tr>
<td>miss_send_len</td>
<td>Max bytes of new flow that datapath should send to the controller</td>
</tr>
</tbody>
</table>

Example:

```python
@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 = []

    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('OFPGetConfigReply received: flags=%s, miss_send_len=%d',
                      flags, msg.miss_send_len)
```

JSON Example:
Modify State Messages

class ryu.ofproto.ofproto_v1_5_parser.OFPTableMod(datapath, table_id, config, properties)

Flow table configuration message

The controller sends this message to configure table state.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_id</td>
<td>ID of the table (OFPTT_ALL indicates all tables)</td>
</tr>
<tr>
<td>config</td>
<td>Bitmap of configuration flags.</td>
</tr>
<tr>
<td></td>
<td>OFPTC_EVICTION</td>
</tr>
<tr>
<td></td>
<td>OFPTC_VACANCY_EVENTS</td>
</tr>
<tr>
<td>properties</td>
<td>List of OFPTableModProp subclass instance</td>
</tr>
</tbody>
</table>

Example:

```python
def send_table_mod(self, datapath):
ofp = datapath.ofproto
ofp_parser = datapath.ofproto_parser

req = ofp_parser.OFPTableMod(datapath, 1, 3)
flags = ofp.OFPTC_VACANCY_EVENTS
properties = [ofp_parser.OFPTableModPropEviction(flags)]
req = ofp_parser.OFPTableMod(datapath, 1, 3, properties)
datapath.send_msg(req)
```

JSON Example:

```json
{
   "OFPTableMod": {
      "config": 4,
      "properties": [
         { "OFPTableModPropEviction": {
            "flags": 2,
            "length": 8,
            "type": 2
         }
      ],
      "table_id": 255
   }
}
```
class ryu.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=0xffffffff, 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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cookie</td>
<td>Opaque controller-issued identifier</td>
</tr>
<tr>
<td>cookie_mask</td>
<td>Mask used to restrict the cookie bits that must match when the command is OFPFC_MODIFY* or OFPFC_DELETE*</td>
</tr>
<tr>
<td>table_id</td>
<td>ID of the table to put the flow in</td>
</tr>
<tr>
<td>command</td>
<td>One of the following values.</td>
</tr>
<tr>
<td></td>
<td>OFPFC_ADD</td>
</tr>
<tr>
<td></td>
<td>OFPFC_MODIFY</td>
</tr>
<tr>
<td></td>
<td>OFPFC_MODIFY_STRICT</td>
</tr>
<tr>
<td></td>
<td>OFPFC_DELETE</td>
</tr>
<tr>
<td></td>
<td>OFPFC_DELETE_STRICT</td>
</tr>
<tr>
<td>idle_timeout</td>
<td>Idle time before discarding (seconds)</td>
</tr>
<tr>
<td>hard_timeout</td>
<td>Max time before discarding (seconds)</td>
</tr>
<tr>
<td>priority</td>
<td>Priority level of flow entry</td>
</tr>
<tr>
<td>buffer_id</td>
<td>Buffered packet to apply to (or OFP_NO_BUFFER)</td>
</tr>
<tr>
<td>out_port</td>
<td>For OFPFC_DELETE* commands, require matching entries to include this as an output port</td>
</tr>
<tr>
<td>out_group</td>
<td>For OFPFC_DELETE* commands, require matching entries to include this as an output group</td>
</tr>
<tr>
<td>flags</td>
<td>Bitmap of the following flags.</td>
</tr>
<tr>
<td></td>
<td>OFPFF_SEND_FLOW_REM</td>
</tr>
<tr>
<td></td>
<td>OFPFF_CHECK_OVERLAP</td>
</tr>
<tr>
<td></td>
<td>OFPFF_RESET_COUNTS</td>
</tr>
<tr>
<td></td>
<td>OFPFF_NO_PKT_COUNTS</td>
</tr>
<tr>
<td></td>
<td>OFPFF_NO_BYT_COUNTS</td>
</tr>
<tr>
<td>importance</td>
<td>Eviction precedence</td>
</tr>
<tr>
<td>match</td>
<td>Instance of OFPMatch</td>
</tr>
<tr>
<td>instructions</td>
<td>list of OFPIInstruction* instance</td>
</tr>
</tbody>
</table>

Example:

def send_flow_mod(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

(continues on next page)
cookie = cookie_mask = 0
table_id = 0
idle_timeout = hard_timeout = 0
priority = 32768
buffer_id = ofp.OFP_NO_BUFFER
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:

```json
{
    "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
                            }
                        }
                    ]
                }
            }
        ]
    }
}
```
"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
        }
    },
    {
        "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},

"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,
    "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
}

{ "OFPFlowMod": {
  "buffer_id": 0,
  "command": 0,
  "cookie": 1311768467463790320,
  "cookie_mask": 18446744073709551615,
  "flags": 0,
  "hard_timeout": 0,
  "idle_timeout": 0,
  "importance": 39032,
  "instructions": [
    
  ]
}
```json
{
  "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": [
      {
        "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"
  },

"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
}

"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": {

(continues on next page)
"field": {  
  "OXMTlv": {  
    "field": "ipv4_dst",  
    "mask": null,  
    "value": "192.168.2.9"  
  },  
  "len": 16,  
  "type": 25  
},  
"len": 32,  
"type": 4  
},  
"OFPInstructionGotoTable": {  
  "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  
  }  
}  
}

```python

class ryu.ofproto.ofproto_v1_5_parser.OFPGroupMod(datapath, command=0, type_=0, group_id=0, command_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.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>command</td>
<td>One of the following values.</td>
</tr>
<tr>
<td></td>
<td>OFPGC_ADD</td>
</tr>
<tr>
<td></td>
<td>OFPGC_MODIFY</td>
</tr>
<tr>
<td></td>
<td>OFPGC_DELETE</td>
</tr>
<tr>
<td></td>
<td>OFPGC_INSERT_BUCKET</td>
</tr>
<tr>
<td></td>
<td>OFPGC_REMOVE_BUCKET</td>
</tr>
<tr>
<td>type</td>
<td>One of the following values.</td>
</tr>
<tr>
<td></td>
<td>OFPGT_ALL</td>
</tr>
<tr>
<td></td>
<td>OFPGT_SELECT</td>
</tr>
<tr>
<td></td>
<td>OFPGT_INDIRECT</td>
</tr>
<tr>
<td></td>
<td>OFPGT_FF</td>
</tr>
<tr>
<td>group_id</td>
<td>Group identifier.</td>
</tr>
<tr>
<td>command_bucket_id</td>
<td>Bucket Id used as part of OFPGC_INSERT_BUCKET and OFPGC_REMOVE_BUCKET commands execution.</td>
</tr>
<tr>
<td>buckets</td>
<td>List of OFPBucket instance</td>
</tr>
<tr>
<td>properties</td>
<td>List of OFPGroupProp instance</td>
</tr>
</tbody>
</table>

```
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": [}
```
```json
{
    "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,
                    "watch": 4008636142
                }
            }
        ]
    }
},
"command": 3,
"command_bucket_id": 3149642683,
"group_id": 2863311530,
"properties": [],
"type": 1
}
```
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>port_no</td>
<td>Port number to modify</td>
</tr>
<tr>
<td>hw_addr</td>
<td>The hardware address that must be the same as hw_addr of OFPPort of OFPSwitchFeatures</td>
</tr>
<tr>
<td>config</td>
<td>Bitmap of configuration flags.</td>
</tr>
<tr>
<td>mask</td>
<td>Bitmap of configuration flags above to be changed</td>
</tr>
<tr>
<td>properties</td>
<td>List of OFPPortModProp subclass instance</td>
</tr>
</tbody>
</table>

Example:

```python
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 | ofp.OFPPF_AUTONEG | ofp.OFPPF_PAUSE | ofp.OFPPF_PAUSEASYM)
    properties = [ofp_parser.OFPPortModPropEthernet(advertise)]
    req = ofp_parser.OFPPortMod(datapath, port_no, hw_addr, config, mask, properties)
    datapath.send_msg(req)
```

JSON Example:

```json
{
    "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": [1],
  "exp_type": 1,
  "experimenter": 101,
  "length": 16,
  "type": 65535
},
"OFPPortModPropExperimenter": {
  "data": [1, 2],
  "exp_type": 2,
  "experimenter": 101,
  "length": 20,
  "type": 65535
}

```python
class ryu.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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>command</td>
<td>One of the following values.</td>
</tr>
<tr>
<td></td>
<td>OFPMC_ADD</td>
</tr>
<tr>
<td></td>
<td>OFPMC_MODIFY</td>
</tr>
<tr>
<td></td>
<td>OFPMC_DELETE</td>
</tr>
<tr>
<td>flags</td>
<td>Bitmap of the following flags.</td>
</tr>
<tr>
<td></td>
<td>OFPMF_KBPS</td>
</tr>
<tr>
<td></td>
<td>OFPMF_PKTTPS</td>
</tr>
<tr>
<td></td>
<td>OFPMF_BURST</td>
</tr>
<tr>
<td></td>
<td>OFPMF_STATS</td>
</tr>
<tr>
<td>meter_id</td>
<td>Meter instance</td>
</tr>
<tr>
<td>bands</td>
<td>list of the following class instance.</td>
</tr>
<tr>
<td></td>
<td>OFPMeterBandDrop</td>
</tr>
<tr>
<td></td>
<td>OFPMeterBandDscpRemark</td>
</tr>
<tr>
<td></td>
<td>OFPMeterBandExperimenter</td>
</tr>
</tbody>
</table>

### JSON Example:

```json
{
    "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
    }
}
```
Multipart Messages

class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero or OFMPF_REQ_MORE</td>
</tr>
</tbody>
</table>

Example:

```python
def send_desc_stats_request(self, datapath):
ofp_parser = datapath.ofproto_parser
req = ofp_parser.OFPDescStatsRequest(datapath, 0)
datapath.send_msg(req)
```  

JSON Example:

```json
{
"OFPDescStatsRequest": {
"flags": 0,
"type": 0
}
}
```

class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>Instance of OFPDescStats</td>
</tr>
</tbody>
</table>

Example:

```python
@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:

```json
{
"OFPDescStatsReply": {
"body": {
"OFPDescStats": {
"dp_desc": "dp",
"hw_desc": "hw",
```

(continues on next page)
Individual flow descriptions request message

The controller uses this message to query individual flow descriptions.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero or OFPMPF_REQ_MORE</td>
</tr>
<tr>
<td>table_id</td>
<td>ID of table to read</td>
</tr>
<tr>
<td>out_port</td>
<td>Require matching entries to include this as an output port</td>
</tr>
<tr>
<td>out_group</td>
<td>Require matching entries to include this as an output group</td>
</tr>
<tr>
<td>cookie</td>
<td>Require matching entries to contain this cookie value</td>
</tr>
<tr>
<td>cookie_mask</td>
<td>Mask used to restrict the cookie bits that must match</td>
</tr>
<tr>
<td>match</td>
<td>Instance of OFPMatch</td>
</tr>
</tbody>
</table>

Example:

```python
def send_flow_desc_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.OFPFlowDescStatsRequest(datapath, 0,
                                            ofp.OFPTT_ALL,
                                            ofp.OFPP_ANY,
                                            ofp.OFPG_ANY,
                                            cookie, cookie_mask,
                                            match)

    datapath.send_msg(req)
```

JSON Example:

```json
{
    "OFPFlowDescStatsRequest": {
        "cookie": 1234605616436508552,
        "cookie_mask": 18446744073709551615,
        "flags": 0,
    }
}
```
class ryu.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.

Example:

```
@set_ev_cls(ofp_event.EventOFPFlowDescStatsReply, 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:

```
{
    "OFPFlowDescStatsReply": {
        "body": {
            "OFPFlowDesc": {
```
"cookie": 1234605616436508552,
"flags": 1,
"hard_timeout": 255,
"idle_timeout": 255,
"importance": 43690,
"instructions": [
    {
        "OFPInstructionGotoTable": {
            "len": 8,
            "table_id": 2,
            "type": 1
        }
    }
],
"length": 64,
"match": {
    "OFPMatch": {
        "length": 12,
        "oxm_fields": [
            {
                "OXMTlv": {
                    "field": "in_port",
                    "mask": null,
                    "value": 1
                }
            },
            {
                "type": 1
            }
        ]
    }
},
"priority": 5,
"stats": {
    "OFPStats": {
        "length": 12,
        "oxs_fields": [
            {
                "OXSTlv": {
                    "field": "flow_count",
                    "value": 1
                }
            }
        ]
    },
    "table_id": 1
},
"flags": 0,
"type": 1
}
Individual flow statistics request message

The controller uses this message to query individual flow statistics.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero or OFPMPF_REQ_MORE</td>
</tr>
<tr>
<td>table_id</td>
<td>ID of table to read</td>
</tr>
<tr>
<td>out_port</td>
<td>Require matching entries to include this as an output port</td>
</tr>
<tr>
<td>out_group</td>
<td>Require matching entries to include this as an output group</td>
</tr>
<tr>
<td>cookie</td>
<td>Require matching entries to contain this cookie value</td>
</tr>
<tr>
<td>cookie_mask</td>
<td>Mask used to restrict the cookie bits that must match</td>
</tr>
<tr>
<td>match</td>
<td>Instance of OFPMatch</td>
</tr>
</tbody>
</table>

Example:

```python
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:

```json
{
    "OFPFlowStatsRequest": {
        "cookie": 0,
        "cookie_mask": 0,
        "flags": 0,
        "match": {
            "OFPMatch": {
                "type": 1
            }
        },
        "out_group": 4294967295,
        "out_port": 4294967295,
        "table_id": 0,
        "type": 17
    }
}
```
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>List of OFPFlowStats instance</td>
</tr>
</tbody>
</table>

Example:

```
@set_ev_cls(ofp_event.EventOFPFlowStatsReply, MAIN_DISPATCHER)
def flow_stats_reply_handler(self, ev):
    flows = []
    for stat in ev.msg.body:
        flows.append('table_id=%s reason=%d priority=%d ' %
                      (stat.table_id, stat.reason, stat.priority,
                       stat.match, stat.stats))
    self.logger.debug('FlowStats: %s', flows)
```

JSON Example:

```
{
  "OFPFlowStatsReply": {
    "body": {
      "OFPFlowStats": {
        "length": 40,
        "match": {
          "OFPMatch": {
            "length": 12,
            "oxm_fields": [
              {"OXMTlv": {
                "field": "in_port",
                  "mask": null,
                  "value": 1
              }]
            },
            "type": 1
          },
          "priority": 1,
          "reason": 0,
          "stats": {
            "OFPStats": {
              "length": 12,
              "oxs_fields": [
                {"OXSTlv": {
                  "field": "flow_count",
                  "value": 1
                }
              }
            }
          }
        }
      }
    }
  }
```

(continues on next page)
Aggregate flow statistics request message

The controller uses this message to query aggregate flow statistics.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero or OFPMPF_REQ_MORE</td>
</tr>
<tr>
<td>table_id</td>
<td>ID of table to read</td>
</tr>
<tr>
<td>out_port</td>
<td>Require matching entries to include this as an output port</td>
</tr>
<tr>
<td>out_group</td>
<td>Require matching entries to include this as an output group</td>
</tr>
<tr>
<td>cookie</td>
<td>Require matching entries to contain this cookie value</td>
</tr>
<tr>
<td>cookie_mask</td>
<td>Mask used to restrict the cookie bits that must match</td>
</tr>
<tr>
<td>match</td>
<td>Instance of OFPMatch</td>
</tr>
</tbody>
</table>

Example:

```python
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:

```json
{
    "OFPAggregateStatsRequest": {
        "cookie": 0,
        "cookie_mask": 0,
        "flags": 0,
    }
}
```
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>Instance of OFPAggregateStats</td>
</tr>
</tbody>
</table>

Example:

```python
@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:

```json
{
    "OFPAggregateStatsReply": {
        "body": {
            "OFPAggregateStats": {
                "length": 16,
                "stats": {
                    "OFPStats": {
                        "length": 12,
                        "oxs_fields": [
                            {
                                "OXSTlv": {
                                    "field": "flow_count",
                                    "value": 1
                                }
                            }
                        ]
                    }
                }
            }
        }
    }
}
```

(continues on next page)
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero or OFPPMPP_REQ_MORE</td>
</tr>
<tr>
<td>port_no</td>
<td>Port number to read (OFPP_ANY to all ports)</td>
</tr>
</tbody>
</table>

Example:

```python
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:

```json
{
  "OFPPortStatsRequest": {
    "flags": 0,
    "port_no": 4294967295,
    "type": 4
  }
}
```

class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>List of OFPPortStats instance</td>
</tr>
</tbody>
</table>

Example:

```python
@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:

```json
{
    "OFPStatsReply": {
        "body": [
            {
                "OFPStats": {
                    "duration_nsec": 0,
                    "duration_sec": 0,
                    "length": 224,
                    "port_no": 7,
                    "properties": [
                        {
                            "OFPStatsPropEthernet": {
                                "collisions": 0,
                                "length": 40,
                                "rx_crc_err": 0,
                                "rx_frame_err": 0,
                                "rx_over_err": 0,
                                "type": 0
                            }
                        },
                        {
                            "OFPStatsPropOptical": {
                                "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
                            }
                        },
                        {
                            "OFPStatsPropExperimenter": {
                                "data": [],
                                "exp_type": 0,
                                "experimenter": 101,
                                "length": 12,
                                "type": 65535
                            }
                        },
                        {
                            "OFPStatsPropExperimenter": {
                                "data": [1],
                                "exp_type": 1,
                                "experimenter": 101,
                                "length": 16,
                                "type": 65535
                            }
                        }
                    ]
                }
            }
        ]
    }
}
```

(continues on next page)
"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,
    "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 ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero or OFPMPF_REQ_MORE</td>
</tr>
<tr>
<td>port_no</td>
<td>Port number to read (OFPP_ANY to all ports)</td>
</tr>
</tbody>
</table>

Example:

```python
def send_port_desc_stats_request(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPPortDescStatsRequest(datapath, 0, ofp.OFPP_ANY)
    datapath.send_msg(req)
```

JSON Example:

```json
{
    "OFPPortDescStatsRequest": {
        "flags": 0,
        "port_no": 48346,
        "type": 13
    }
}
```

class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>List of OFPPort instance</td>
</tr>
</tbody>
</table>

Example:

```python
@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 ' %
                      (p.port_no, p.hw_addr, p.name, p.config,
                       'state=0x%08x properties=%s' % (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,
              "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
            }
          }
        ]
      }
    }
  ]
}


```python
class ryu.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.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero or OFPMPF_REQ_MORE</td>
</tr>
<tr>
<td>port_no</td>
<td>Port number to read</td>
</tr>
<tr>
<td>queue_id</td>
<td>ID of queue to read</td>
</tr>
</tbody>
</table>

Example:

```python
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:

```json
{
    "OFPQueueStatsRequest": {
        "flags": 0,
        "port_no": 43981,
        "queue_id": 4294967295,
        "type": 5
    }
}
```

**class ryu.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.

Example:

```python
@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 ' %
                       (stat.port_no, stat.queue_id,
                        'tx_bytes=%d tx_packets=%d tx_errors=%d ' %
                        (stat.tx_bytes, stat.tx_packets, stat.tx_errors,
                        'duration_sec=%d duration_nsec=%d ' %
                        (stat.duration_sec, stat.duration_nsec,
                        'properties=%s' %
                        (stat.properties))
            self.logger.debug('QueueStats: %s', queues)
```

JSON Example:

```json
{
    "OFPQueueStatsReply": {
        "body": [
```

(continues on next page)
```json
{
    "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,
        "tx_errors": 0,
        "tx_packets": 0
    }
}
```
Queue description request message
The controller uses this message to query description of all the queues.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero or OFMPF_REQ_MORE</td>
</tr>
<tr>
<td>port_no</td>
<td>Port number to read (OFPP_ANY for all ports)</td>
</tr>
<tr>
<td>queue_id</td>
<td>ID of queue to read (OFPQ_ALL for all queues)</td>
</tr>
</tbody>
</table>

Example:
```python
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:
```json
{
    "OFPQueueDescStatsRequest": {
        "flags": 0,
        "port_no": 52651,
        "queue_id": 57020,
        "type": 15
    }
}
```
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>List of OFPQueueDesc instance</td>
</tr>
</tbody>
</table>

Example:

```python
@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:

```json
{
"OFPQueueDescStatsReply": {
"body": [

"OFPQueueDesc": {
"len": 32,
"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": {

(continues on next page)```
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero or OFMPF_REQ_MORE</td>
</tr>
<tr>
<td>group_id</td>
<td>ID of group to read (OFPG_ALL to all groups)</td>
</tr>
</tbody>
</table>
Example:

```python
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:

```json
{
    "OFPGroupStatsRequest": {
        "flags": 0,
        "group_id": 4294967292,
        "type": 6
    }
}
```

```python
class ryu.ofproto.ofproto_v1_5_parser.OFPGroupStatsReply(datapath
```

Group statistics reply message

The switch responds with this message to a group statistics request.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>List of OFGroupStats instance</td>
</tr>
</tbody>
</table>

Example:

```python
@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
' % (stat.length, stat.group_id, 'ref_count=%d packet_count=%d byte_count=%d
' % (stat.ref_count, stat.packet_count, stat.byte_count, stat.duration_sec, stat.duration_nsec))

    self.logger.debug('GroupStats: %s', groups)
```

JSON Example:

```json
{
    "OFPGroupStatsReply": {
        "body": {
            "OFPGroupStats": {
                "bucket_stats": [
                    "OFPBucketCounter": {
                        "byte_count": 2345,
                        "packet_count": 234
                    }
                ]
            }
        }
    }
}
```
class ryu.ofproto.ofproto_v1_5_parser.OFPGroupDescStatsRequest (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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero or OFMPF_REQ_MORE</td>
</tr>
<tr>
<td>group_id</td>
<td>ID of group to read (OFPG_ALL to all groups)</td>
</tr>
</tbody>
</table>

Example:

```python
def send_group_desc_stats_request(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPGroupDescStatsRequest(datapath, 0, ofp.OFPG_ALL)
    datapath.send_msg(req)
```

JSON Example:

```json
{
    "OFPGroupDescStatsRequest": {
        "flags": 0,
        "group_id": 52651,
        "type": 7
    }
}
```

class ryu.ofproto.ofproto_v1_5_parser.OFPGroupDescStatsReply (datapath, 
type_=None, 
**kwargs)

Group description reply message

The switch responds with this message to a group description request.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>List of OFPGGroupDescStats instance</td>
</tr>
</tbody>
</table>
Example:

```python
@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:

```json
{
    "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
                                        }
                                    }
                                ]
                            }
                        }
                    ],
                    "group_id": 1,
                    "length": 48,
                    "properties": [],
                    "type": 1
                }
            }
        ],
        "flags": 0,
        "type": 7
    }
}"
```
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero or OFMPF_REQ_MORE</td>
</tr>
</tbody>
</table>

Example:
```python
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:
```json
{
    "OFPGroupFeaturesStatsRequest": {
        "flags": 0,
        "type": 8
    }
}
```

class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>Instance of OFPGroupFeaturesStats</td>
</tr>
</tbody>
</table>

Example:
```python
@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:
```json
{
    "OFPGroupFeaturesStatsReply": {
```
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero or OFPMPF_REQ_MORE</td>
</tr>
<tr>
<td>meter_id</td>
<td>ID of meter to read (OFPM_ALL to all meters)</td>
</tr>
</tbody>
</table>

Example:

```python
def send_meter_stats_request(self, datapath):
ofp = datapath.ofproto
ofp_parser = datapath.ofproto_parser

req = ofp_parser.OFPMeterStatsRequest(datapath, 0, ofp.OFPM_ALL)
datatpath.send_msg(req)
```

JSON Example:

```json
{
   "OFPMeterStatsRequest": {
      "flags": 0,
      "meter_id": 4294967295,
      "type": 9
   }
}
```

class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>List of OFPMeterStats instance</td>
</tr>
</tbody>
</table>

Example:

```python
@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 ' %
               (stat.meter_id, stat.len, stat.ref_count,
                'packet_in_count=%d byte_in_count=%d ' %
                (stat.packet_in_count, stat.byte_in_count,
                'duration_sec=%d duration_nsec=%d ' %
                (stat.duration_sec, stat.duration_nsec,
                'band_stats=%s' %
                (stat.band_stats))))
self.logger.debug('MeterStats: %s', meters)
```

JSON Example:

```json
{
  "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
  }
}
```

```python
class ryu.ofproto.ofproto_v1_5_parser.OFPMeterDescStatsRequest (datapath,
flags=0, meter_id=4294967295, type_=None)
```

Meter description statistics request message

The controller uses this message to query configuration for one or more meters.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero or OFPMPF_REQ_MORE</td>
</tr>
<tr>
<td>meter_id</td>
<td>ID of meter to read (OFPM_ALL to all meters)</td>
</tr>
</tbody>
</table>

Example:

```python
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:

```json
{
  "OFPMeterDescStatsRequest": {
    "flags": 0,
    "meter_id": 4294967295,
    "type": 10
  }
}
```

class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>List of OFPMeterDescStats instance</td>
</tr>
</tbody>
</table>

Example:

```python
@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 ' %
                        (stat.length, stat.flags, stat.meter_id,
                         'bands=%s' % (stat.bands))

    self.logger.debug('MeterDescStats: %s', configs)
```

JSON Example:

```json
{
  "OFPMeterDescStatsReply": {
    "body": [
    "OFPMeterDescStats": {
      "bands": [
      "OFPMeterBandDrop": {
```
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero or OFMPF_REQ_MORE</td>
</tr>
</tbody>
</table>

Example:

```python
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:

```json
{
    "OFPMeterFeaturesStatsRequest": {
        "flags": 0,
        "type": 11
    }
}
```

class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>List of OFPMeterFeaturesStats instance</td>
</tr>
</tbody>
</table>

Example:

```python
```
@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 ' % (stat.max_meter, stat.band_types, stat.capabilities, stat.max_bands, stat.max_color))
    self.logger.debug('MeterFeaturesStats: %s', features)

JSON Example:

```json
{
    "OFPMeterFeaturesStatsReply": {
        "body": [ {
            "OFPMeterFeaturesStats": {
                "band_types": 2147483654,
                "capabilities": 15,
                "features": 3,
                "max_bands": 255,
                "max_color": 0,
                "max_meter": 16777216
            }
        }],
        "flags": 0,
        "type": 11
    }
}
```

class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero or OFMPMF_REQ_MORE</td>
</tr>
</tbody>
</table>

Example:

```python
def send_controller_status.multipart_request(self, datapath):
    ofp_parser = datapath.ofproto.ofproto_parser

    req = ofp_parser.OFPPortDescStatsRequest(datapath, 0)
    datapath.send_msg(req)
```

JSON Example:

```json
{
    "OFPControllerStatusStatsRequest": {
(continues on next page)```
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>List of OFPControllerStatus instance</td>
</tr>
</tbody>
</table>

Example:

```python
@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 ' %
                       (s.short_id, s.role, s.reason,
                        s.channel_status, repr(s.properties)))
    self.logger.debug('OFPControllerStatusStatsReply received: %s', status)
```

JSON Example:

```json
{
    "OFPControllerStatusStatsReply": {
        "body": [
        {
            "OFPControllerStatusStats": {
                "channel_status": 1,
                "length": 48,
                "properties": [
                {
                    "OFPControllerStatusPropUri": {
                        "length": 26,
                        "type": 0,
                        "uri": "tls:192.168.34.23:6653"
                    }
                },
                "reason": 1,
                "role": 1,
                "short_id": 65535
                }
        }
        ],
        "flags": 0,
        "type": 18
    }
}
class ryu.ofproto.ofproto_v1_5_parser.OFPTableStatsRequest (datapath, flags, type_=None)

Table statistics request message
The controller uses this message to query flow table statistics.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero or OFMPF_REQ_MORE</td>
</tr>
</tbody>
</table>

Example:

```python
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 ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>List of OFPTableStats instance</td>
</tr>
</tbody>
</table>

Example:

```python
@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 ' %
                       (stat.table_id, stat.active_count, stat.matched_count))
    self.logger.debug('TableStats: %s', tables)
```

JSON Example:
```
{
    "OFPTableStatsReply": {
        "body": [
```
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero or OFMPF_REQ_MORE</td>
</tr>
</tbody>
</table>

Example:

```python
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:

```json
{
  "OFPTableDescStatsRequest": {
    "flags": 0,
    "type": 14
  }
}
```

class ryu.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.
Example:

```python
@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)
```

Example:

```json
{
    "OFPTableDescStatsReply": {
        "body": [
            {
                "OFPTableDesc": {
                    "config": 0,
                    "length": 24,
                    "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
                }
            }
        ]
    }
}
```

(continues on next page)
Table features statistics request message

The controller uses this message to query table features.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>List of OFPTableFeaturesStats instances. The default is [].</td>
</tr>
</tbody>
</table>

JSON Example:
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>List of OFPTableFeaturesStats instance</td>
</tr>
</tbody>
</table>

JSON Example:

```json
{
  "OFPTableFeaturesStatsReply": {
    "body": [
      {
        "OFPTableFeaturesStats": {
          "capabilities": 4,
          "command": 1,
          "features": 1,
          "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
        }
      }
    ]
  }
}
```
class ryu.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.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero or OFPFMF_REQ_MORE</td>
</tr>
<tr>
<td>monitor_id</td>
<td>Controller-assigned ID for this monitor</td>
</tr>
<tr>
<td>out_port</td>
<td>Require matching entries to include this as an output port</td>
</tr>
<tr>
<td>out_group</td>
<td>Require matching entries to include this as an output group</td>
</tr>
<tr>
<td>monitor_flags</td>
<td>Bitmap of the following flags.</td>
</tr>
<tr>
<td></td>
<td>OFPFMF_INITIAL</td>
</tr>
<tr>
<td></td>
<td>OFPFMF_ADD</td>
</tr>
<tr>
<td></td>
<td>OFPFMF_REMOVED</td>
</tr>
<tr>
<td></td>
<td>OFPFMF_MODIFY</td>
</tr>
<tr>
<td></td>
<td>OFPFMF_INSTRUCTIONS</td>
</tr>
<tr>
<td></td>
<td>OFPFMF_NO_ABBREV</td>
</tr>
<tr>
<td></td>
<td>OFPFMF_ONLY_OWN</td>
</tr>
<tr>
<td>table_id</td>
<td>ID of table to monitor</td>
</tr>
<tr>
<td>command</td>
<td>One of the following values.</td>
</tr>
<tr>
<td></td>
<td>OFPFMC_ADD</td>
</tr>
<tr>
<td></td>
<td>OFPFMC_MODIFY</td>
</tr>
<tr>
<td></td>
<td>OFPFMC_DELETE</td>
</tr>
<tr>
<td>match</td>
<td>Instance of OFPMatch</td>
</tr>
</tbody>
</table>

Example:

```python
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.OFFP_ANY, ofp.OFPG_ANY,
                                           monitor_flags,
                                           ofp.OFPPT_ALL,
                                           ofp.OFPFM_ADD, match)

    datapath.send_msg(req)
```

JSON Example:

```
{
    "OFPFlowMonitorRequest": { 
        "command": 0,
        "flags": 0,
        "match": { 
            "OFPMatch": { 
                "length": 14,
                "oxm_fields": [ 
                    "OXMTlv": { 
```
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>List of list of the following class instance.</td>
</tr>
<tr>
<td></td>
<td>OFPFlowMonitorFull</td>
</tr>
<tr>
<td></td>
<td>OFPFlowMonitorAbbrev</td>
</tr>
<tr>
<td></td>
<td>OFPFlowMonitorPaused</td>
</tr>
</tbody>
</table>

Example:

```python
@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' %
                     (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 ' %
                         (update.table_id, update.reason,
                          update.idle_timeout, update.hard_timeout,
                          update.priority, update.cookie,
                          update.match, update.instructions)
```

(continues on next page)
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:

```json
{
    "OFPFlowMonitorReply": {
        "body": {
            "OFPFlowUpdateFull": {
                "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,
                        "oxm_fields": {
                            "OXMTlv": {
                                "field": "eth_type",
                                "mask": null,
                                "value": 2054
                            }
                        },
                        "type": 1
                    },
                    "priority": 3,
                    "reason": 0,
                    "table_id": 0
                }
            }
        }
    }
}
```
class ryu.ofproto.ofproto_v1_5_parser.OFPBundleFeaturesStatsRequest (datapath, flags=0, feature_request_flags=0, properties=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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero or OFPMFF_REQ_MORE</td>
</tr>
<tr>
<td>feature_request_flags</td>
<td>Bitmap of the following flags.</td>
</tr>
<tr>
<td></td>
<td>OFPBF_TIMESTAMP</td>
</tr>
<tr>
<td></td>
<td>OFPBF_TIME_SET_SCHED</td>
</tr>
<tr>
<td>properties</td>
<td>List of OFPBundleFeaturesProp subclass instance</td>
</tr>
</tbody>
</table>

Example:

```python
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:

```json
{
    "OFPBundleFeaturesStatsRequest": {
        "feature_request_flags": 3,
        "flags": 0,
    }
}
```
"properties": [
    {
        "OFPBundleFeaturesPropTime": {
            "length": 72,
            "sched_accuracy": {
                "OFTTime": {
                    "nanoseconds": 1717986918,
                    "seconds": 6148914691236517205
                }
            },
            "sched_max_future": {
                "OFTTime": {
                    "nanoseconds": 2290649224,
                    "seconds": 8608480567731124087
                }
            },
            "sched_max_past": {
                "OFTTime": {
                    "nanoseconds": 2863311530,
                    "seconds": 11068046444225730969
                }
            },
            "timestamp": {
                "OFTTime": {
                    "nanoseconds": 3435973836,
                    "seconds": 13527612320720337851
                }
            }
        }
    },
    "type": 1
]
}

class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>Instance of OFPBundleFeaturesStats</td>
</tr>
</tbody>
</table>

Example:

```python
@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=%0x08x'
                      'properties=%s',
                      body.capabilities, repr(body.properties))
```

JSON Example:
```python
{
    "OFPBundleFeaturesStatsReply": {
        "body": {
            "OFPBundleFeaturesStats": {
                "capabilities": 7,
                "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": 1106804644225730969
                            }
                        },
                        "timestamp": {
                            "OFPTime": {
                                "nanoseconds": 3435973836,
                                "seconds": 13527612320720337851
                            }
                        },
                        "type": 1
                    }
                }
            }
        },
        "flags": 0,
        "type": 19
    }
}
```

**class** ryu.ofproto.ofproto_v1_5_parser.OFPExperimenterStatsRequest  
(datapath, flags, experimenter, exp_type, data, type_=None)

Experimenter multipart request message

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero or OFPMPF_REQ_MORE</td>
</tr>
<tr>
<td>experimenter</td>
<td>Experimenter ID</td>
</tr>
<tr>
<td>exp_type</td>
<td>Experimenter defined</td>
</tr>
<tr>
<td>data</td>
<td>Experimenter defined extra data</td>
</tr>
</tbody>
</table>

2.5. OpenFlow protocol API Reference 433
JSON Example:

```json
{
    "OFPExperimenterStatsRequest": {
        "data": "aG9nZWhvZ2U=",
        "exp_type": 3405678728,
        "experimenter": 3735928495,
        "flags": 0,
        "type": 65535
    }
}
```

```python
class ryu.ofproto.ofproto_v1_5_parser.OFPExperimenterStatsReply (datapath, type_=None, **kwargs)
```

Experimenter multipart reply message

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>An OFPExperimenterMultipart instance</td>
</tr>
</tbody>
</table>

JSON Example:

```json
{
    "OFPExperimenterStatsReply": {
        "body": {
            "OFPExperimenterMultipart": {
                "data": "dGVzdGRhdGE5OTk5OTk5OQ=",
                "exp_type": 3405674359,
                "experimenter": 3735928495
            },
            "flags": 0,
            "type": 65535
        }
    }
}
```

 Packet-Out Message

```python
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>buffer_id</td>
<td>ID assigned by datapath (OFP_NO_BUFFER if none)</td>
</tr>
<tr>
<td>match</td>
<td>Instance of OFPMatch (in_port is mandatory in the match field)</td>
</tr>
<tr>
<td>actions</td>
<td>list of OpenFlow action class</td>
</tr>
<tr>
<td>data</td>
<td>Packet data of a binary type value or an instances of packet.Packet.</td>
</tr>
</tbody>
</table>

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:

```json
{
    "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

```python
class Ryu.ofproto.ofproto_v1_5_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 ryu.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 ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>role</td>
<td>One of the following values.</td>
</tr>
<tr>
<td></td>
<td>OFPCR_ROLE_NOCHANGE</td>
</tr>
<tr>
<td></td>
<td>OFPCR_ROLE_EQUAL</td>
</tr>
<tr>
<td></td>
<td>OFPCR_ROLE_MASTER</td>
</tr>
<tr>
<td></td>
<td>OFPCR_ROLE_SLAVE</td>
</tr>
<tr>
<td>short_id</td>
<td>ID number for the controller. The default is OFP-CID_UNDEFINED.</td>
</tr>
<tr>
<td>generation_id</td>
<td>Master Election Generation ID</td>
</tr>
</tbody>
</table>

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 ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>role</td>
<td>One of the following values.</td>
</tr>
<tr>
<td>short_id</td>
<td>ID number for the controller. The default is OFPCID_UNDEFINED.</td>
</tr>
<tr>
<td>generation_id</td>
<td>Master Election Generation ID</td>
</tr>
</tbody>
</table>

Example:

@set_ev_cls(ofp_event.EventOFPRoleReply, 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('OFPRoleReply received: ' +
    'role=%s short_id=%d, generation_id=%d',
    role, msg.short_id, msg.generation_id)

JSON Example:

```json
{
    "OFPRoleReply": {
        "generation_id": 1234605616436508552,
        "role": 1,
        "short_id": 43690
    }
}
```

### Bundle Messages

**class ryu.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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bundle_id</td>
<td>Id of the bundle</td>
</tr>
<tr>
<td>type</td>
<td>One of the following values.</td>
</tr>
<tr>
<td></td>
<td>OFPBCT_OPEN_REQUEST</td>
</tr>
<tr>
<td></td>
<td>OFPBCT_OPEN_REPLY</td>
</tr>
<tr>
<td></td>
<td>OFPBCT_CLOSE_REQUEST</td>
</tr>
<tr>
<td></td>
<td>OFPBCT_CLOSE_REPLY</td>
</tr>
<tr>
<td></td>
<td>OFPBCT_COMMIT_REQUEST</td>
</tr>
<tr>
<td></td>
<td>OFPBCT_COMMIT_REPLY</td>
</tr>
<tr>
<td></td>
<td>OFPBCT_DISCARD_REQUEST</td>
</tr>
<tr>
<td></td>
<td>OFPBCT_DISCARD_REPLY</td>
</tr>
<tr>
<td>flags</td>
<td>Bitmap of the following flags.</td>
</tr>
<tr>
<td></td>
<td>OFPBF_ATOMIC</td>
</tr>
<tr>
<td></td>
<td>OFPBF_ORDERED</td>
</tr>
<tr>
<td>properties</td>
<td>List of OFPBundleProp subclass instance</td>
</tr>
</tbody>
</table>

Example:

```python
def send_bundle_control(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    req = ofp_parser.OFPBundleCtrlMsg(datapath, 7,
        ofp.OFPBCT_OPEN_REQUEST,
```

(continues on next page)
ofp.OFPBF_ATOMIC, []

datapath.send_msg(req)

JSON Example:

```python
{
    "OFPBundleCtrlMsg": {
        "bundle_id": 99999999,
        "flags": 1,
        "properties": [],
        "type": 1
    }
}
```

class ryu.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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bundle_id</td>
<td>Id of the bundle</td>
</tr>
<tr>
<td>flags</td>
<td>Bitmap of the following flags.</td>
</tr>
<tr>
<td>properties</td>
<td>List of OFPBundleProp subclass instance</td>
</tr>
<tr>
<td>message</td>
<td>MsgBase subclass instance</td>
</tr>
</tbody>
</table>

Example:

```python
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:

```python
{
    "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": [
  {
    "OFPInstructionActions": {
      "actions": [
        {
          "OFPActionPopVlan": {
            "len": 8,
            "type": 18
          }
        },
        {
          "OFPActionSetField": {
            "field": {
              "OXMTlv": {
                "field": "ipv4_dst",
                "mask": null,
                "value": "192.168.2.9"
              }
            },
            "len": 16,
            "type": 25
          }
        }
      ]
    },
    "NXPActionLearn": {
      "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
},

"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
},

"OFPInstructionGotoTable": {
  "len": 8,
  "table_id": 100,
  "type": 1
}

(continues on next page)
```json
{
  "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,
            "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"
          }
        }
      ]
    }
  }
}
```
Set Asynchronous Configuration Message

```python
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>properties</td>
<td>List of <code>OFPAsyncConfigProp</code> subclass instances</td>
</tr>
</tbody>
</table>

Example:

```python
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:

```json
{
    "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, "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": [],
    "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 ryu.ofproto.ofproto_v1_5_parser.OFPGetAsyncRequest (datapath)
Get asynchronous configuration request message

The controller uses this message to query the asynchronous message.

Example:

```python
def send_get_async_request(self, datapath):
ofp_parser = datapath.ofproto_parser
req = ofp_parser.OFPGetAsyncRequest(datapath)
datapath.send_msg(req)
```

JSON Example:

```json
{
    "OFPGetAsyncRequest": {}
}
```

class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>properties</td>
<td>List of OFPAsyncConfigProp subclass instances</td>
</tr>
</tbody>
</table>

Example:

```python
@set_ev_cls(ofp_event.EventOFPGetAsyncReply, MAIN_DISPATCHER)
def get_async_reply_handler(self, ev):
    msg = ev.msg
    self.logger.debug('OFPGetAsyncReply received: ')
    self.logger.debug('properties=%s', repr(msg.properties))
```

JSON Example:

```json
{
    "OFPGetAsyncReply": {
        "properties": [
            {
                "OFPAsyncConfigPropReasons": {
                    "length": 8,
                    "mask": 3,
                    "type": 0
                }
            },
            {
                "OFPAsyncConfigPropReasons": {
                    "length": 8,
                    "mask": 3,
                    "type": 1
                }
            },
            {
                "OFPAsyncConfigPropReasons": {
                    "length": 8,
```

(continues on next page)
"mask": 3,
"type": 2
},
{
"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
}
},
{
"OFPAsyncConfigPropReasons": {
"length": 8,
"mask": 3
}
}
Asynchronous Messages

Packet-In Message

class ryu.ofproto.ofproto_v1_5_parser.OFPPacketIn (datapath, buffer_id=None, total_len=None, reason=None, table_id=None, cookie=None, match=None, data=None)
The switch sends the packet that received to the controller by this message.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>buffer_id</td>
<td>ID assigned by datapath</td>
</tr>
<tr>
<td>total_len</td>
<td>Full length of frame</td>
</tr>
<tr>
<td>reason</td>
<td>Reason packet is being sent.</td>
</tr>
<tr>
<td></td>
<td>OFPR_TABLE_MISS</td>
</tr>
<tr>
<td></td>
<td>OFPR_APPLY_ACTION</td>
</tr>
<tr>
<td></td>
<td>OFPR_INVALID_TTL</td>
</tr>
<tr>
<td></td>
<td>OFPR_ACTION_SET</td>
</tr>
<tr>
<td></td>
<td>OFPR_GROUP</td>
</tr>
<tr>
<td></td>
<td>OFPR_PACKET_OUT</td>
</tr>
<tr>
<td>table_id</td>
<td>ID of the table that was looked up</td>
</tr>
<tr>
<td>cookie</td>
<td>Cookie of the flow entry that was looked up</td>
</tr>
<tr>
<td>match</td>
<td>Instance of OFPMatch</td>
</tr>
<tr>
<td>data</td>
<td>Ethernet frame</td>
</tr>
</tbody>
</table>

Example:

```python
@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'
    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:

```json
{
    "OFPPacketIn": {
        "buffer_id": 200,
        "cookie": 0,
```
Flow Removed Message

```python
class ryu.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.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_id</td>
<td>ID of the table</td>
</tr>
<tr>
<td>reason</td>
<td>One of the following values.</td>
</tr>
<tr>
<td></td>
<td>OFPRR_IDLE_TIMEOUT</td>
</tr>
<tr>
<td></td>
<td>OFPRR_HARD_TIMEOUT</td>
</tr>
<tr>
<td></td>
<td>OFPRR_DELETE</td>
</tr>
<tr>
<td></td>
<td>OFPRR_GROUP_DELETE</td>
</tr>
<tr>
<td></td>
<td>OFPRR_METER_DELETE</td>
</tr>
<tr>
<td></td>
<td>OFPRR_EVICTION</td>
</tr>
<tr>
<td>priority</td>
<td>Priority level of flow entry</td>
</tr>
<tr>
<td>idle_timeout</td>
<td>Idle timeout from original flow mod</td>
</tr>
<tr>
<td>hard_timeout</td>
<td>Hard timeout from original flow mod</td>
</tr>
<tr>
<td>cookie</td>
<td>Opaque controller-issued identifier</td>
</tr>
<tr>
<td>match</td>
<td>Instance of OFPMatch</td>
</tr>
<tr>
<td>stats</td>
<td>Instance of OFPStats</td>
</tr>
</tbody>
</table>

Example:

```python
@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'
    elif msg.reason == ofp.OFPRR_METER_DELETE:
        reason = 'METER DELETE'
    elif msg.reason == ofp.OFPRR_EVICTION:
        reason = 'EVICTION'
    else:
        reason = 'unknown'

    self.logger.debug('OFPFlowRemoved 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:

```json
{
    "OFPFlowRemoved": {
        "cookie": 1234605616436508552,
```

(continues on next page)
Port Status Message

```python
class ryu.ofproto.ofproto_v1_5_parser.OFPPortStatus ( datapath, reason=None, desc=None )
```

Port status message

The switch notifies controller of change of ports.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reason</td>
<td>One of the following values.</td>
</tr>
<tr>
<td></td>
<td>OFPPR_ADD</td>
</tr>
<tr>
<td></td>
<td>OFPPR_DELETE</td>
</tr>
<tr>
<td></td>
<td>OFPPR_MODIFY</td>
</tr>
<tr>
<td>desc</td>
<td>instance of OFPPort</td>
</tr>
</tbody>
</table>

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:

```json
{
    "OFPPortStatus": {
        "desc": {
            "OFPPort": {
                "config": 0,
                "hw_addr": "f2:0b:a4:d0:3f:70",
                "length": 168,
                "name": "私のポート",
                "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
                        }
                    }
                ]
            }
        }
    }
}
```

(continues on next page)
Controller Role Status Message

class ryu.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.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>role</td>
<td>One of the following values.</td>
</tr>
<tr>
<td></td>
<td>OFPCR_ROLE_NOCHANGE</td>
</tr>
<tr>
<td></td>
<td>OFPCR_ROLE_EQUAL</td>
</tr>
<tr>
<td></td>
<td>OFPCR_ROLE_MASTER</td>
</tr>
<tr>
<td>reason</td>
<td>One of the following values.</td>
</tr>
<tr>
<td></td>
<td>OFPCRR_MASTER_REQUEST</td>
</tr>
<tr>
<td></td>
<td>OFPCRR_CONFIG</td>
</tr>
<tr>
<td></td>
<td>OFPCRR_EXPERIMENTER</td>
</tr>
<tr>
<td>generation_id</td>
<td>Master Election Generation ID</td>
</tr>
<tr>
<td>properties</td>
<td>List of OFPRoleProp subclass instance</td>
</tr>
</tbody>
</table>

Example:

```python
@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:

```json
{
    "OFPRoleStatus": {
        "generation_id": 17356517385562371090,
        "properties": [],
        "reason": 0,
        "role": 3
    }
}
```

(continues on next page)
Table Status Message

**class** ryu.ofproto.ofproto_v1_5_parser.OFPTableStatus (datapath, reason=None, table=None)

Table status message

The switch notifies controller of change of table status.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reason</td>
<td>One of the following values.</td>
</tr>
<tr>
<td></td>
<td>OFPTR_VACANCY_DOWN</td>
</tr>
<tr>
<td></td>
<td>OFPTR_VACANCY_UP</td>
</tr>
<tr>
<td>table</td>
<td>OFPTableDesc instance</td>
</tr>
</tbody>
</table>

Example:

```python
@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:

```json
{
    "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
    ],
    "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
}
```

Request Forward Message

class ryu.ofproto.ofproto_v1_5_parser.OFPRequestForward(datapath, request=None)

Forwarded request message

The switch forwards request messages from one controller to other controllers.

2.5. OpenFlow protocol API Reference
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>request</td>
<td>OFPGGroupMod or OFPMeterMod instance</td>
</tr>
</tbody>
</table>

Example:

```python
@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:

```json
{
    "OFPRequestForward": {
        "request": {
            "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,
                            }
                        }
                    }
                }
            }
        }
    }
}
```

(continues on next page)
Controller Status Message

class ryu.ofproto.ofproto_v1_5_parser.OFPControllerStatus(datapath, status=\nNone)

Controller status message

The switch informs the controller about the status of the control channel it maintains with each controller.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>status</td>
<td>OFPControllerStatusStats instance</td>
</tr>
</tbody>
</table>

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:
        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:

```json
{
    "OFPControllerStatus": {
        "status": {
            "OFPControllerStatusStats": {
                "channel_status": 1,
                "length": 48,
                "properties": []
            }
        }
    }
}
```

(continues on next page)
Symmetric Messages

Hello

```python
class ryu.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 Ryu framework, so the Ryu application do not need to process this typically.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>elements</td>
<td>list of OFPHelloElemVersionBitmap instance</td>
</tr>
</tbody>
</table>

JSON Example:

```json
{
    "OFPHello": {
        "elements": [
            {
                "OFPHelloElemVersionBitmap": {
                    "length": 8,
                    "type": 1,
                    "versions": [6]
                }
            }
        ]
    }
}
```

class ryu.ofproto.ofproto_v1_5_parser.OFPHelloElemVersionBitmap(versions, type_=None, length=None)

Version bitmap Hello Element

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>versions</td>
<td>list of versions of OpenFlow protocol a device supports</td>
</tr>
</tbody>
</table>
Echo Request

class ryu.ofproto.ofproto_v1_5_parser.OFPEchoRequest (datapath, data=None)

Echo request message

This message is handled by the Ryu framework, so the Ryu application do not need to process this typically.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>An arbitrary length data</td>
</tr>
</tbody>
</table>

Example:

```python
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:

```json
{
    "OFPEchoRequest": {
    "data": ""
    }
}
```

Echo Reply

class ryu.ofproto.ofproto_v1_5_parser.OFPEchoReply (datapath, data=None)

Echo reply message

This message is handled by the Ryu framework, so the Ryu application do not need to process this typically.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>An arbitrary length data</td>
</tr>
</tbody>
</table>

Example:

```python
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))
```
Error Message

class ryu.ofproto.ofproto_v1_5_parser.OFErrorMsg (datapath, type=None, code=None, data=None, **kwargs)

Error message
The switch notifies controller of problems by this message.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>High level type of error</td>
</tr>
<tr>
<td>code</td>
<td>Details depending on the type</td>
</tr>
<tr>
<td>data</td>
<td>Variable length data depending on the type and code</td>
</tr>
</tbody>
</table>

*type* attribute corresponds to *type_* parameter of __init__.

Types and codes are defined in *ryu.ofproto.ofproto*.

<table>
<thead>
<tr>
<th>Type</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>OFPET_HELLO_FAILED</td>
<td>OFPHFC_*</td>
</tr>
<tr>
<td>OFPET_BAD_REQUEST</td>
<td>OFPBRC_*</td>
</tr>
<tr>
<td>OFPET_BAD_ACTION</td>
<td>OFPBAC_*</td>
</tr>
<tr>
<td>OFPET_BAD_INSTRUCTION</td>
<td>OFPBIC_*</td>
</tr>
<tr>
<td>OFPET_BAD_MATCH</td>
<td>OFPBMC_*</td>
</tr>
<tr>
<td>OFPET_FLOW_MOD_FAILED</td>
<td>OFPPMFC_*</td>
</tr>
<tr>
<td>OFPET_GROUP_MOD_FAILED</td>
<td>OFPGMFC_*</td>
</tr>
<tr>
<td>OFPET_PORT_MOD_FAILED</td>
<td>OFPPMFC_*</td>
</tr>
<tr>
<td>OFPET_TABLE_MOD_FAILED</td>
<td>OFPTMFC_*</td>
</tr>
<tr>
<td>OFPET_QUEUE_OP_FAILED</td>
<td>OFPQOF_C_*</td>
</tr>
<tr>
<td>OFPET_SWITCH_CONFIG_FAILED</td>
<td>OFPSFC_*</td>
</tr>
<tr>
<td>OFPET_ROLE_REQUEST_FAILED</td>
<td>OFPRRFC_*</td>
</tr>
<tr>
<td>OFPET_METER_MOD_FAILED</td>
<td>OFPMFC_*</td>
</tr>
<tr>
<td>OFPET_TABLE_FEATURES_FAILED</td>
<td>OFPTFFC_*</td>
</tr>
<tr>
<td>OFPET_EXPERIMENTER</td>
<td>N/A</td>
</tr>
</tbody>
</table>

If *type* == OFPET_EXPERIMENTER, this message has also the following attributes.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>exp_type</td>
<td>Experimenter defined type</td>
</tr>
<tr>
<td>experimenter</td>
<td>Experimenter ID</td>
</tr>
</tbody>
</table>

Example:

```json
{
"OFPEchoReply": {
"data": ""
}
}
```
@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": "Bg4ACAAAAAA=",
        "type": 4
    }
}

Experimenter

```

Experimenter extension message

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>experimenter</td>
<td>Experimenter ID</td>
</tr>
<tr>
<td>exp_type</td>
<td>Experimenter defined</td>
</tr>
<tr>
<td>data</td>
<td>Experimenter defined arbitrary additional data</td>
</tr>
</tbody>
</table>

```

JSON Example:
{
    "OFPErrorMsg": {
        "code": null,
        "data": "amlra2VuIGRhdGE=",
        "exp_type": 60000,
        "experimenter": 999999,
        "type": 65535
    }
}

Port Structures

```

Port Structures

```

```

```

```

class ryu.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

```
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>port_no</td>
<td>Port number and it uniquely identifies a port within a switch.</td>
</tr>
<tr>
<td>length</td>
<td>Length of ofp_port (excluding padding).</td>
</tr>
<tr>
<td>hw_addr</td>
<td>MAC address for the port.</td>
</tr>
<tr>
<td>name</td>
<td>Null-terminated string containing a human-readable name for the interface.</td>
</tr>
<tr>
<td>config</td>
<td>Bitmap of port configuration flags.</td>
</tr>
<tr>
<td></td>
<td>OFPPC_PORT_DOWN</td>
</tr>
<tr>
<td></td>
<td>OFPPC_NO_RECV</td>
</tr>
<tr>
<td></td>
<td>OFPPC_NO_FWD</td>
</tr>
<tr>
<td></td>
<td>OFPPC_NO_PACKET_IN</td>
</tr>
<tr>
<td>state</td>
<td>Bitmap of port state flags.</td>
</tr>
<tr>
<td></td>
<td>OFPPS_LINK_DOWN</td>
</tr>
<tr>
<td></td>
<td>OFPPS_BLOCKED</td>
</tr>
<tr>
<td></td>
<td>OFPPS_LIVE</td>
</tr>
<tr>
<td>properties</td>
<td>List of OFPPortDescProp subclass instance</td>
</tr>
</tbody>
</table>

## Flow Match Structure

```python
class ryu.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.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in_port</td>
<td>Integer 32bit</td>
<td>Switch input port</td>
</tr>
<tr>
<td>in_phy_port</td>
<td>Integer 32bit</td>
<td>Switch physical input port</td>
</tr>
<tr>
<td>metadata</td>
<td>Integer 64bit</td>
<td>Metadata passed between tables</td>
</tr>
<tr>
<td>eth_dst</td>
<td>MAC address</td>
<td>Ethernet destination address</td>
</tr>
<tr>
<td>eth_src</td>
<td>MAC address</td>
<td>Ethernet source address</td>
</tr>
<tr>
<td>eth_type</td>
<td>Integer 16bit</td>
<td>Ethernet frame type</td>
</tr>
<tr>
<td>vlan_vid</td>
<td>Integer 16bit</td>
<td>VLAN id</td>
</tr>
<tr>
<td>vlan_pcp</td>
<td>Integer 8bit</td>
<td>VLAN priority</td>
</tr>
<tr>
<td>ip_dscp</td>
<td>Integer 8bit</td>
<td>IP DSCP (6 bits in ToS field)</td>
</tr>
<tr>
<td>ip_ecn</td>
<td>Integer 8bit</td>
<td>IP ECN (2 bits in ToS field)</td>
</tr>
<tr>
<td>ip_proto</td>
<td>Integer 8bit</td>
<td>IP protocol</td>
</tr>
<tr>
<td>ipv4_src</td>
<td>IPv4 address</td>
<td>IPv4 source address</td>
</tr>
<tr>
<td>ipv4_dst</td>
<td>IPv4 address</td>
<td>IPv4 destination address</td>
</tr>
<tr>
<td>tcp_src</td>
<td>Integer 16bit</td>
<td>TCP source port</td>
</tr>
<tr>
<td>tcp_dst</td>
<td>Integer 16bit</td>
<td>TCP destination port</td>
</tr>
<tr>
<td>udp_src</td>
<td>Integer 16bit</td>
<td>UDP source port</td>
</tr>
<tr>
<td>udp_dst</td>
<td>Integer 16bit</td>
<td>UDP destination port</td>
</tr>
</tbody>
</table>

Continued on next page
Table 4 – continued from previous page

<table>
<thead>
<tr>
<th>Argument</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sctp_src</td>
<td>Integer 16bit</td>
<td>SCTP source port</td>
</tr>
<tr>
<td>sctp_dst</td>
<td>Integer 16bit</td>
<td>SCTP destination port</td>
</tr>
<tr>
<td>icmpv4_type</td>
<td>Integer 8bit</td>
<td>ICMP type</td>
</tr>
<tr>
<td>icmpv4_code</td>
<td>Integer 8bit</td>
<td>ICMP code</td>
</tr>
<tr>
<td>arp_op</td>
<td>Integer 16bit</td>
<td>ARP opcode</td>
</tr>
<tr>
<td>arp_spa</td>
<td>IPv4 address</td>
<td>ARP source IPv4 address</td>
</tr>
<tr>
<td>arp_tpa</td>
<td>IPv4 address</td>
<td>ARP target IPv4 address</td>
</tr>
<tr>
<td>arp_sha</td>
<td>MAC address</td>
<td>ARP source hardware address</td>
</tr>
<tr>
<td>arp_tha</td>
<td>MAC address</td>
<td>ARP target hardware address</td>
</tr>
<tr>
<td>ipv6_src</td>
<td>IPv6 address</td>
<td>IPv6 source address</td>
</tr>
<tr>
<td>ipv6_dst</td>
<td>IPv6 address</td>
<td>IPv6 destination address</td>
</tr>
<tr>
<td>ipv6_flabel</td>
<td>Integer 32bit</td>
<td>IPv6 Flow Label</td>
</tr>
<tr>
<td>icmpv6_type</td>
<td>Integer 8bit</td>
<td>ICMPv6 type</td>
</tr>
<tr>
<td>icmpv6_code</td>
<td>Integer 8bit</td>
<td>ICMPv6 code</td>
</tr>
<tr>
<td>ipv6_nd_target</td>
<td>IPv6 address</td>
<td>Target address for ND</td>
</tr>
<tr>
<td>ipv6_nd_sll</td>
<td>MAC address</td>
<td>Source link-layer for ND</td>
</tr>
<tr>
<td>ipv6_nd_tll</td>
<td>MAC address</td>
<td>Target link-layer for ND</td>
</tr>
<tr>
<td>mpls_label</td>
<td>Integer 32bit</td>
<td>MPLS label</td>
</tr>
<tr>
<td>mpls_tc</td>
<td>Integer 8bit</td>
<td>MPLS TC</td>
</tr>
<tr>
<td>mpls bos</td>
<td>Integer 8bit</td>
<td>MPLS Bos bit</td>
</tr>
<tr>
<td>pbb_isid</td>
<td>Integer 24bit</td>
<td>PBB I-SID</td>
</tr>
<tr>
<td>tunnel_id</td>
<td>Integer 64bit</td>
<td>Logical Port Metadata</td>
</tr>
<tr>
<td>ipv6_exthdr</td>
<td>Integer 16bit</td>
<td>IPv6 Extension Header pseudo-field</td>
</tr>
<tr>
<td>pbb_uca</td>
<td>Integer 8bit</td>
<td>PBB UCA header field</td>
</tr>
<tr>
<td>tcp_flags</td>
<td>Integer 16bit</td>
<td>TCP flags</td>
</tr>
<tr>
<td>actset_output</td>
<td>Integer 32bit</td>
<td>Output port from action set metadata</td>
</tr>
<tr>
<td>packet_type</td>
<td>Integer 32bit</td>
<td>Packet type value</td>
</tr>
</tbody>
</table>

Example:

```python
>>> # 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')
```

```python
>>> # 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 `ryu.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

<table>
<thead>
<tr>
<th>Type</th>
<th>MATCH</th>
</tr>
</thead>
<tbody>
<tr>
<td>non-VLAN-tagged</td>
<td></td>
</tr>
<tr>
<td>VLAN-tagged(vlan_id=3)</td>
<td>MATCH</td>
</tr>
<tr>
<td>VLAN-tagged(vlan_id=5)</td>
<td>MATCH</td>
</tr>
</tbody>
</table>

2) Only packets without a VLAN tag

• Example:

match = parser.OFPMatch(vlan_vid=0x0000)

• Packet Matching

<table>
<thead>
<tr>
<th>Type</th>
<th>MATCH</th>
</tr>
</thead>
<tbody>
<tr>
<td>non-VLAN-tagged</td>
<td></td>
</tr>
<tr>
<td>VLAN-tagged(vlan_id=3)</td>
<td>x</td>
</tr>
<tr>
<td>VLAN-tagged(vlan_id=5)</td>
<td>x</td>
</tr>
</tbody>
</table>

3) Only packets with a VLAN tag regardless of its value

• Example:

match = parser.OFPMatch(vlan_vid=(0x1000, 0x1000))

• Packet Matching

<table>
<thead>
<tr>
<th>Type</th>
<th>MATCH</th>
</tr>
</thead>
<tbody>
<tr>
<td>non-VLAN-tagged</td>
<td>x</td>
</tr>
<tr>
<td>VLAN-tagged(vlan_id=3)</td>
<td>MATCH</td>
</tr>
<tr>
<td>VLAN-tagged(vlan_id=5)</td>
<td>MATCH</td>
</tr>
</tbody>
</table>

4) Only packets with VLAN tag and VID equal

• Example:

match = parser.OFPMatch(vlan_vid=(0x1000 | 3))

• Packet Matching

<table>
<thead>
<tr>
<th>Type</th>
<th>MATCH</th>
</tr>
</thead>
<tbody>
<tr>
<td>non-VLAN-tagged</td>
<td>x</td>
</tr>
<tr>
<td>VLAN-tagged(vlan_id=3)</td>
<td>MATCH</td>
</tr>
<tr>
<td>VLAN-tagged(vlan_id=5)</td>
<td>x</td>
</tr>
</tbody>
</table>

Flow Stats Structures

class ryu.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.
<table>
<thead>
<tr>
<th>Argument</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>duration</td>
<td>Integer 32bit*2</td>
<td>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.</td>
</tr>
<tr>
<td>idle_time</td>
<td>Integer 32bit *2</td>
<td>Time flow entry has been idle.</td>
</tr>
<tr>
<td>flow_count</td>
<td>Integer 32bit</td>
<td>Number of aggregated flow entries.</td>
</tr>
<tr>
<td>packet_count</td>
<td>Integer 64bit</td>
<td>Number of packets matched by a flow entry.</td>
</tr>
<tr>
<td>byte_count</td>
<td>Integer 64bit</td>
<td>Number of bytes matched by a flow entry.</td>
</tr>
</tbody>
</table>

Example:

```python
>>> # compose
>>> stats = parser.OFPStats(
...    packet_count=100,
...    duration=(100, 200)
>>> # query
>>> if 'duration' in stats:
...    print stats['duration']
...    (100, 200)
```

Flow Instruction Structures

```python
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_id</td>
<td>Next table</td>
</tr>
</tbody>
</table>

```python
class ryu.ofproto.ofproto_v1_5_parser.OFPInstructionWriteMetadata (metadata, metadata_mask, type_=None, len_=None)
```

Write metadata instruction

This instruction writes the masked metadata value into the metadata field.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>metadata</td>
<td>Metadata value to write</td>
</tr>
<tr>
<td>metadata_mask</td>
<td>Metadata write bitmask</td>
</tr>
</tbody>
</table>

```python
class ryu.ofproto.ofproto_v1_5_parser.OFPInstructionActions (type_, actions=None, len_=None)
```

Actions instruction
This instruction writes/applies/clears the actions.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>One of following values.</td>
</tr>
<tr>
<td></td>
<td>OFPIT_WRITE_ACTIONS</td>
</tr>
<tr>
<td></td>
<td>OFPIT_APPLY_ACTIONS</td>
</tr>
<tr>
<td></td>
<td>OFPIT_CLEAR_ACTIONS</td>
</tr>
<tr>
<td>actions</td>
<td>list of OpenFlow action class</td>
</tr>
</tbody>
</table>

`type` attribute corresponds to `type_` parameter of `__init__`.

```python
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Bitmap of the following flags.</td>
</tr>
<tr>
<td></td>
<td>OFPSTF_PERIODIC</td>
</tr>
<tr>
<td></td>
<td>OFPSTF_ONLY_FIRST</td>
</tr>
<tr>
<td>thresholds</td>
<td>Instance of <code>OFPStats</code></td>
</tr>
</tbody>
</table>

**Action Structures**

```python
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>port</td>
<td>Output port</td>
</tr>
<tr>
<td>max_len</td>
<td>Max length to send to controller</td>
</tr>
</tbody>
</table>

```python
class ryu.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.

```python
class ryu.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.

```python
class ryu.ofproto.ofproto_v1_5_parser.OFPActionSetMplsTtl(mpls_ttl, type_=None, len_=None)
```

Set MPLS TTL action
This action sets the MPLS TTL.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpls_ttl</td>
<td>MPLS TTL</td>
</tr>
</tbody>
</table>

```python
class ryu.ofproto.ofproto_v1_5_parser.OFPActionDecMplsTtl(type_=None, len_=None)
```

Decrement MPLS TTL action

This action decrements the MPLS TTL.

```python
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ethertype</td>
<td>Ether type. The default is 802.1Q. (0x8100)</td>
</tr>
</tbody>
</table>

```python
class ryu.ofproto.ofproto_v1_5_parser.OFPActionPopVlan(type_=None, len_=None)
```

Pop VLAN action

This action pops the outermost VLAN tag from the packet.

```python
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ethertype</td>
<td>Ether type</td>
</tr>
</tbody>
</table>

```python
class ryu.ofproto.ofproto_v1_5_parser.OFPActionPopMpls(ethertype=2048, type_=None, len_=None)
```

Pop MPLS action

This action pops the MPLS header from the packet.

```python
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_id</td>
<td>Queue ID for the packets</td>
</tr>
</tbody>
</table>

```python
class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>group_id</td>
<td>Group identifier</td>
</tr>
</tbody>
</table>
class ryu.ofproto.ofproto_v1_5_parser.OFPActionSetNwTtl(nw_ttl, type_=None, len_=None)

Set IP TTL action
This action sets the IP TTL.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nw_ttl</td>
<td>IP TTL</td>
</tr>
</tbody>
</table>

class ryu.ofproto.ofproto_v1_5_parser.OFPActionDecNwTtl(type_=None, len_=None)

Decrement IP TTL action
This action decrements the IP TTL.

class ryu.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:

```python
def set_field():
    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 ryu.ofproto.ofproto_v1_5_parser.OFPActionPushPbb(ethertype, type_=None, len_=None)

Push PBB action
This action pushes a new PBB header to the packet.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ethertype</td>
<td>Ether type</td>
</tr>
</tbody>
</table>

class ryu.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 ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n_bits</td>
<td>Number of bits to copy.</td>
</tr>
<tr>
<td>src_offset</td>
<td>Starting bit offset in source.</td>
</tr>
<tr>
<td>dst_offset</td>
<td>Starting bit offset in destination.</td>
</tr>
<tr>
<td>oxm_ids</td>
<td>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 [].</td>
</tr>
</tbody>
</table>
class ryu.ofproto.ofproto_v1_5_parser.OFPActionMeter(meter_id, type_=None, len_=None)

Meter action

This action applies meter (rate limiter)

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>meter_id</td>
<td>Meter instance</td>
</tr>
</tbody>
</table>

class ryu.ofproto.ofproto_v1_5_parser.OFPActionExperimenter(experimenter)

Experimenter action

This action is an extensible action for the experimenter.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>experimenter</td>
<td>Experimenter ID</td>
</tr>
</tbody>
</table>

Note: For the list of the supported Nicira experimenter actions, please refer to ryu.ofproto.nx_actions.

Controller Status Structure

class ryu.ofproto.ofproto_v1_5_parser.OFPControllerStatusStats(short_id=None, role=None, reason=None, channel_status=None, properties=None, length=None)

Controller status structure
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>Length of this entry.</td>
</tr>
<tr>
<td>short_id</td>
<td>ID number which identifies the controller.</td>
</tr>
<tr>
<td>role</td>
<td>Bitmap of controller’s role flags.</td>
</tr>
<tr>
<td></td>
<td>OFPCR_ROLE_NOCHANGE</td>
</tr>
<tr>
<td></td>
<td>OFPCR_ROLE_EQUAL</td>
</tr>
<tr>
<td></td>
<td>OFPCR_ROLE_MASTER</td>
</tr>
<tr>
<td></td>
<td>OFPCR_ROLE_SLAVE</td>
</tr>
<tr>
<td>reason</td>
<td>Bitmap of controller status reason flags.</td>
</tr>
<tr>
<td></td>
<td>OFPCSR_REQUEST</td>
</tr>
<tr>
<td></td>
<td>OFPCSR_CHANNEL_STATUS</td>
</tr>
<tr>
<td></td>
<td>OFPCSR_ROLE</td>
</tr>
<tr>
<td></td>
<td>OFPCSR_CONTROLLER_ADDED</td>
</tr>
<tr>
<td></td>
<td>OFPCSR_CONTROLLER_REMOVED</td>
</tr>
<tr>
<td></td>
<td>OFPCSR_SHORT_ID</td>
</tr>
<tr>
<td></td>
<td>OFPCSR_EXPERIMENTER</td>
</tr>
<tr>
<td>channel_status</td>
<td>Bitmap of control channel status flags.</td>
</tr>
<tr>
<td></td>
<td>OFPCT_STATUS_UP</td>
</tr>
<tr>
<td></td>
<td>OFPCT_STATUS_DOWN</td>
</tr>
<tr>
<td>properties</td>
<td>List of OFPControllerStatusProp subclass instance</td>
</tr>
</tbody>
</table>

### 2.6 Nicira Extension Structures

#### 2.6.1 Nicira Extension Actions Structures

The followings shows the supported NXAction classes only in OpenFlow1.0

```python
class ryu.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.

```bash
set_queue:queue
```

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_id</td>
<td>Queue ID for the packets</td>
</tr>
</tbody>
</table>
**Note:** This actions is supported by `OFPActionSetQueue` in OpenFlow 1.2 or later.

Example:

```python
actions += [parser.NXActionSetQueue(queue_id=10)]
```

```python
class ryu.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.

```python
dec_ttl
```

**Note:** This actions is supported by `OFPActionDecNwTtl` in OpenFlow 1.2 or later.

Example:

```python
actions += [parser.NXActionDecTtl()]
```

```python
class ryu.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.

```python
push_mpls:ethertype
```

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ethertype</td>
<td>Ether type (The value must be either 0x8847 or 0x8848)</td>
</tr>
</tbody>
</table>

**Note:** This actions is supported by `OFPActionPushMpls` in OpenFlow 1.2 or later.

Example:

```python
match = parser.OFPMatch(dl_type=0x0800)
actions += [parser.NXActionPushMpls(ethertype=0x8847)]
```

```python
class ryu.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**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ethtype</td>
<td>Ether type</td>
</tr>
</tbody>
</table>

**Note:** This actions is supported by `OFPActionPopMpls` in OpenFlow1.2 or later.

Example:

```python
match = parser.OFPMatch(dl_type=0x8847)
actions += [parser.NXActionPushMpls(ethertype=0x0800)]
```

```python
class ryu.ofproto.ofproto_v1_0_parser.NXActionSetMplsTtl(ttl, type_=None,
            len_=None, vendor=None, sub_type=None)
```

Set MPLS TTL action
This action sets the MPLS TTL.
And equivalent to the followings action of ovs-ofctl command.

```
set_mpls_ttl:ttl
```

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ttl</td>
<td>MPLS TTL</td>
</tr>
</tbody>
</table>

**Note:** This actions is supported by `OFPActionSetMplsTtl` in OpenFlow1.2 or later.

Example:

```python
actions += [parser.NXActionSetMplsTtl(ttl=128)]
```

```python
class ryu.ofproto.ofproto_v1_0_parser.NXActionDecMplsTtl(type_=None, len_=None,
            vendor=None, sub_type=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:
def actions += [parser.NXActionDecMplsTil()]

class ryu.ofproto.ofproto_v1_0_parser.NXActionSetMplsLabel:
    label, type_=None, len_=None, vendor=None, subtype=None

    Set MPLS Label 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 ryu.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 OFPActionSetField(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 ryu.ofproto.ofproto_v1_3_parser.NXActionPopQueue:
    type_=None, len_=None, experimenter=None, subtype=None

    Pop queue action
This action restors 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:

```python
actions += [parser.NXActionPopQueue()]
```

class ryu.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]
```

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ofsnbits</td>
<td>Start and End for the OXM/NXM field. Setting method refer to the nicira_ext. ofsnbits</td>
</tr>
<tr>
<td>dst</td>
<td>OXM/NXM header for destination field</td>
</tr>
<tr>
<td>value</td>
<td>OXM/NXM value to be loaded</td>
</tr>
</tbody>
</table>

Example:

```python
actions += [parser.NXActionRegLoad(
    ofs_nbits=nicira_ext.ofs_nbits(4, 31),
    dst="eth_dst",
    value=0x112233)]
```

class ryu.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
```

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>OXM/NXM value to be loaded</td>
</tr>
<tr>
<td>mask</td>
<td>Mask for destination field</td>
</tr>
<tr>
<td>dst</td>
<td>OXM/NXM header for destination field</td>
</tr>
</tbody>
</table>

Example:
actions += [parser.NXActionRegLoad2(dst="tun_ipv4_src",
value="192.168.10.0",
mask="255.255.255.0")]

class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>note</td>
<td>A list of integer type values</td>
</tr>
</tbody>
</table>

Example:

actions += [parser.NXActionNote(note=[0xaa,0xbb,0xcc,0xdd])]

class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tun_id</td>
<td>Tunnel ID(32bits)</td>
</tr>
</tbody>
</table>

Example:

actions += [parser.NXActionSetTunnel(tun_id=0xa)]

class ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tun_id</td>
<td>Tunnel ID(64bits)</td>
</tr>
</tbody>
</table>

Example:

```python
actions += [parser.NXActionSetTunnel64(tun_id=0xa)]
```

```python
class ryu.ofproto.ofproto_v1_3_parser.NXActionRegMove(src_field, dst_field, n_bits,
src_ofs=0, dst_ofs=0, type_=None, len_=None, experimenter=None, sub-
type=None)
```

Move register action

This action copies the src to dst.

And equivalent to the followings action of ovs-ofctl command.

```python
move:src[start..end]->dst[start..end]
```

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>src_field</td>
<td>OXM/NXM header for source field</td>
</tr>
<tr>
<td>dst_field</td>
<td>OXM/NXM header for destination field</td>
</tr>
<tr>
<td>n_bits</td>
<td>Number of bits</td>
</tr>
<tr>
<td>src_ofs</td>
<td>Starting bit offset in source</td>
</tr>
<tr>
<td>dst_ofs</td>
<td>Starting bit offset in destination</td>
</tr>
</tbody>
</table>

**Caution:**

`src_start` and `src_end` difference and `dst_start` and `dst_end` difference must be the same.

Example:

```python
actions += [parser.NXActionRegMove(src_field="reg0",
dst_field="reg1",
n_bits=5,
src_ofs=0
dst_ofs=10)]
```

```python
class ryu.ofproto.ofproto_v1_3_parser.NXActionResubmit(in_port=65528,
type_=None, len_=None, experimenter=None, sub-
type=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>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in_port</td>
<td>New in_port for checking flow table</td>
</tr>
</tbody>
</table>

Example:

```
actions += [parser.NXActionResubmit(in_port=8080)]
```

class `ryu.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])
```

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in_port</td>
<td>New in_port for checking flow table</td>
</tr>
<tr>
<td>table_id</td>
<td>Checking flow tables</td>
</tr>
</tbody>
</table>

Example:

```
actions += [parser.NXActionResubmitTable(in_port=8080, table_id=10)]
```

class `ryu.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]
```

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ofs_nbits</td>
<td>Start and End for the OXM/NXM field Setting method refer to the nicira_ext.</td>
</tr>
<tr>
<td>src</td>
<td>OXM/NXM header for source field</td>
</tr>
<tr>
<td>max_len</td>
<td>Max length to send to controller</td>
</tr>
</tbody>
</table>
Example:

```python
actions += [parser.NXActionOutputReg2(
    ofs_nbits=nicira_ext.ofs_nbits(4, 31),
    src="reg0",
    max_len=1024)]
```

class ryu.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[...]
```

**Note:** Like the NXActionOutputReg but organized so that there is room for a 64-bit experimenter OXM as 'src'.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ofs_nbits</td>
<td>Start and End for the OXM/NXM field. Setting method refer to the nicira_ext.ofs_nbits</td>
</tr>
<tr>
<td>src</td>
<td>OXM/NXM header for source field</td>
</tr>
<tr>
<td>max_len</td>
<td>Max length to send to controller</td>
</tr>
</tbody>
</table>

Example:

```python
actions += [parser.NXActionOutputReg2(
    ofs_nbits=nicira_ext.ofs_nbits(4, 31),
    src="reg0",
    max_len=1024)]
```

class ryu.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]...)
```

2.6. Nicira Extension Structures
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_id</td>
<td>The table in which the new flow should be inserted</td>
</tr>
<tr>
<td>specs</td>
<td>Adds a match criterion to the new flow</td>
</tr>
<tr>
<td></td>
<td>Please use the NXFlowSpecMatch in order to set the following format</td>
</tr>
<tr>
<td></td>
<td><code>field=value</code></td>
</tr>
<tr>
<td></td>
<td><code>field[start..end] =</code></td>
</tr>
<tr>
<td></td>
<td>`src[start..end]</td>
</tr>
<tr>
<td></td>
<td>Please use the NXFlowSpecLoad in order to set the following format</td>
</tr>
<tr>
<td></td>
<td><code>load:value-&gt;dst</code></td>
</tr>
<tr>
<td></td>
<td>`[start..end]</td>
</tr>
<tr>
<td>idle_timeout</td>
<td>Idle time before discarding(seconds)</td>
</tr>
<tr>
<td>hard_timeout</td>
<td>Max time before discarding(seconds)</td>
</tr>
<tr>
<td>priority</td>
<td>Priority level of flow entry</td>
</tr>
<tr>
<td>cookie</td>
<td>Cookie for new flow</td>
</tr>
<tr>
<td>flags</td>
<td>send_flow_rem</td>
</tr>
<tr>
<td>fin_idle_timeout</td>
<td>Idle timeout after FIN(seconds)</td>
</tr>
<tr>
<td>fin_hard_timeout</td>
<td>Hard timeout after FIN(seconds)</td>
</tr>
</tbody>
</table>

**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:

```python
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),
        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),
```
class ryu.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.

```python
actions += [parser.NXActionExit()]
```

class ryu.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.

```python
controller(key=value...)
```

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>max_len</td>
<td>Max length to send to controller</td>
</tr>
<tr>
<td>controller_id</td>
<td>Controller ID to send packet-in</td>
</tr>
<tr>
<td>reason</td>
<td>Reason for sending the message</td>
</tr>
</tbody>
</table>

Example:

```python
actions += [config.NXActionController(max_len=1024, controller_id=1, reason=ofproto.OFPR_INVALID_TTL)]
```

class ryu.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...)
```

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>max_len</td>
<td>Max length to send to controller</td>
</tr>
<tr>
<td>controller_id</td>
<td>Controller ID to send packet-in</td>
</tr>
<tr>
<td>reason</td>
<td>Reason for sending the message</td>
</tr>
<tr>
<td>userdata</td>
<td>Additional data to the controller in the packet-in message</td>
</tr>
<tr>
<td>pause</td>
<td>Flag to pause pipeline to resume later</td>
</tr>
</tbody>
</table>

Example:

```python
actions += [
    parser.NXActionController(max_len=1024,
        controller_id=1,
        reason=ofproto.OFPR_INVALID_TTL,
        userdata=[0xa,0xb,0xc],
        pause=True)
]
```

class ryu.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]...)```

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cnt_ids</td>
<td>Controller ids</td>
</tr>
</tbody>
</table>

Example:

```python
actions += [parser.NXActionDecTtlCntIds(cnt_ids=[1,2,3])]
```

**Note:** If you want to set the following ovs-ofctl command. Please use OFPActionDecNwTtl.

```
dec_ttl```

class ryu.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.

\[
pop:dst[start...end]
\]

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>field</td>
<td>OXM/NXM header for source field</td>
</tr>
<tr>
<td>start</td>
<td>Start bit for source field</td>
</tr>
<tr>
<td>end</td>
<td>End bit for source field</td>
</tr>
</tbody>
</table>

Example:

```python
genesys += [parser.NXActionStackPush(field="reg2",  
start=0,  
end=5)]
```

class ryu.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.

\[
pop:src[start...end]
\]

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>field</td>
<td>OXM/NXM header for destination field</td>
</tr>
<tr>
<td>start</td>
<td>Start bit for destination field</td>
</tr>
<tr>
<td>end</td>
<td>End bit for destination field</td>
</tr>
</tbody>
</table>

Example:

```python
genesys += [parser.NXActionStackPop(field="reg2",  
start=0,  
end=5)]
```

class ryu.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.

\[
sample(argument[argument]...)
\]
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>probability</td>
<td>The number of sampled packets</td>
</tr>
<tr>
<td>collector_set_id</td>
<td>The unsigned 32-bit integer identifier of the set of sample collectors to send sampled packets to</td>
</tr>
<tr>
<td>obs_domain_id</td>
<td>The Unsigned 32-bit integer Observation Domain ID</td>
</tr>
<tr>
<td>obs_point_id</td>
<td>The unsigned 32-bit integer Observation Point ID</td>
</tr>
</tbody>
</table>

Example:

```python
actions += [parser.NXActionSample(probability=3,
        collector_set_id=1,
        obs_domain_id=2,
        obs_point_id=3,)]
```

class ryu.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]...]
```

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>probability</td>
<td>The number of sampled packets</td>
</tr>
<tr>
<td>collector_set_id</td>
<td>The unsigned 32-bit integer identifier of the set of sample collectors to send sampled packets to</td>
</tr>
<tr>
<td>obs_domain_id</td>
<td>The Unsigned 32-bit integer Observation Domain ID</td>
</tr>
<tr>
<td>obs_point_id</td>
<td>The unsigned 32-bit integer Observation Point ID</td>
</tr>
<tr>
<td>sampling_port</td>
<td>Sampling port number</td>
</tr>
</tbody>
</table>

Example:

```python
actions += [parser.NXActionSample2(probability=3,
        collector_set_id=1,
        obs_domain_id=2,
        obs_point_id=3,
        sampling_port=8080)]
```

class ryu.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]...)
```

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fin_idle_timeout</td>
<td>Causes the flow to expire after the given number of seconds of inactivity</td>
</tr>
<tr>
<td>fin_idle_timeout</td>
<td>Causes the flow to expire after the given number of second, regardless of activity</td>
</tr>
</tbody>
</table>

Example:

```python
match = parser.OFPMatch(ip_proto=6, eth_type=0x0800)
actions += [parser.NXActionFinTimeout(fin_idle_timeout=30,
                                       fin_hard_timeout=60)]
```

```python
class ryu.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,kln)
```

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clause</td>
<td>Number assigned to the flow’s dimension</td>
</tr>
<tr>
<td>n_clauses</td>
<td>Specify the conjunctive flow’s match condition</td>
</tr>
<tr>
<td>id_</td>
<td>Conjunction ID</td>
</tr>
</tbody>
</table>

Example:

```python
actions += [parser.NXActionConjunction(clause=1,
                                       n_clauses=2,
                                       id_=10)]
```

```python
class ryu.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.

2.6. Nicira Extension Structures
multipath(fields, basis, algorithm, n_links, arg, dst[start..end])

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fields</td>
<td>One of NX_HASH_FIELDS_*</td>
</tr>
<tr>
<td>basis</td>
<td>Universal hash parameter</td>
</tr>
<tr>
<td>algorithm</td>
<td>One of NX_MP_ALG_*</td>
</tr>
<tr>
<td>max_link</td>
<td>Number of output links</td>
</tr>
<tr>
<td>arg</td>
<td>Algorithm-specific argument</td>
</tr>
<tr>
<td>ofs_nbits</td>
<td>Start and End for the OXM/NXM field. Setting method refer to the nicira_ext.</td>
</tr>
<tr>
<td>dst</td>
<td>OXM/NXM header for source field</td>
</tr>
</tbody>
</table>

Example:

```python
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 ryu.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,...])

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>algorithm</td>
<td>One of NX_MP_ALG_*</td>
</tr>
<tr>
<td>fields</td>
<td>One of NX_HASH_FIELDS_*</td>
</tr>
<tr>
<td>basis</td>
<td>Universal hash parameter</td>
</tr>
<tr>
<td>slave_type</td>
<td>Type of slaves (must be NXM_OF_IN_PORT)</td>
</tr>
<tr>
<td>n_slaves</td>
<td>Number of slaves</td>
</tr>
<tr>
<td>ofs_nbits</td>
<td>Start and End for the OXM/NXM field. (must be zero)</td>
</tr>
<tr>
<td>dst</td>
<td>OXM/NXM header for source field (must be zero)</td>
</tr>
<tr>
<td>slaves</td>
<td>List of slaves</td>
</tr>
</tbody>
</table>

Example:

```python
actions += [parser.NXActionBundle(
    algorithm=nicira_ext.NX_MP_ALG_HRW,
    fields=nicira_ext.NX_HASH_FIELDS_ETH_SRC,
    basis=0,
    (continues on next page)```
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.

```python
class ryu.ofproto.ofproto_v1_3_parser.NXActionBundleLoad(algorithm, fields, basis, slave_type, n_slaves, ofs_nbits, dst, slaves)
```

**Attribute**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>algorithm</td>
<td>One of NX_MP_ALG_*</td>
</tr>
<tr>
<td>fields</td>
<td>One of NX_HASH_FIELDS_*</td>
</tr>
<tr>
<td>basis</td>
<td>Universal hash parameter</td>
</tr>
<tr>
<td>slave_type</td>
<td>Type of slaves (must be NXM_OF_IN_PORT)</td>
</tr>
<tr>
<td>n_slaves</td>
<td>Number of slaves</td>
</tr>
<tr>
<td>ofs_nbits</td>
<td>Start and End for the OXM/NXM field. Setting method refer to the nicira_ext.ofs_nbits</td>
</tr>
<tr>
<td>dst</td>
<td>OXM/NXM header for source field</td>
</tr>
<tr>
<td>slaves</td>
<td>List of slaves</td>
</tr>
</tbody>
</table>

Example:

```python
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])]
```

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.

```python
 ctl(argument[,argument]...)
```
### Attribute Description

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero or more(Unspecified flag bits must be zero.)</td>
</tr>
<tr>
<td>zone_src</td>
<td>OXM/NXM header for source field</td>
</tr>
<tr>
<td>zone_ofs</td>
<td>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, zone_src must be set to None or empty character string.</td>
</tr>
<tr>
<td>recirc_table</td>
<td>Recirculate to a specific table</td>
</tr>
<tr>
<td>alg</td>
<td>Well-known port number for the protocol</td>
</tr>
<tr>
<td>actions</td>
<td>Zero or more actions may immediately follow this action</td>
</tr>
</tbody>
</table>

**Note:** If you set number to zone_src, Traceback occurs when you run the to_jsondict.

Example:

```python
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 = [])]
```

```python
class ryu.ofproto.ofproto_v1_3_parser.NXActionNAT(flags, range_ipv4_min=", range_ipv4_max="), range_ipv6_min=", range_ipv6_max="), range_proto_min=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.

```
net(src=ip_min-ip_max : proto_minproto-max)
```
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Zero or more(Unspecified flag bits must be zero.)</td>
</tr>
<tr>
<td>range_ipv4_min</td>
<td>Range ipv4 address minimum</td>
</tr>
<tr>
<td>range_ipv4_max</td>
<td>Range ipv4 address maximum</td>
</tr>
<tr>
<td>range_ipv6_min</td>
<td>Range ipv6 address minimum</td>
</tr>
<tr>
<td>range_ipv6_max</td>
<td>Range ipv6 address maximum</td>
</tr>
<tr>
<td>range_proto_min</td>
<td>Range protocol minimum</td>
</tr>
<tr>
<td>range_proto_max</td>
<td>Range protocol maximum</td>
</tr>
</tbody>
</table>

**Caution:**  
NXActionNAT must be defined in the actions in the NXActionCT.

Example:

```python
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 = [
            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**  
ryu.ofproto.ofproto_v1_3_parser.NXActionOutputTrunc(port, max_len, type_=None, len_=None, experimenter=None, subtype=None)

Truncate output action

This action truncates 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)
```

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>port</td>
<td>Output port</td>
</tr>
<tr>
<td>max_len</td>
<td>Max bytes to send</td>
</tr>
</tbody>
</table>

Example:
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.

```
external
actions += [parser.NXActionDecNshTtl()]
```

Example:

```
external
actions += [parser.NXActionDecNshTtl()]
```

**Class** ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>src</td>
<td>OXM/NXM header and Start bit for source field</td>
</tr>
<tr>
<td>dst</td>
<td>OXM/NXM header and Start bit for destination field</td>
</tr>
<tr>
<td>n_bits</td>
<td>The number of bits from the start bit</td>
</tr>
</tbody>
</table>

**Class** ryu.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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>src</td>
<td>OXM/NXM header and Start bit for source field</td>
</tr>
<tr>
<td>dst</td>
<td>OXM/NXM header and Start bit for destination field</td>
</tr>
<tr>
<td>n_bits</td>
<td>The number of bits from the start bit</td>
</tr>
</tbody>
</table>

**Class** ryu.ofproto.ofproto_v1_3_parser.NXFlowSpecOutput(src, n_bits, dst=“”)

Add an OFPAT_OUTPUT action

This class is used by NXActionLearn.

For the usage of this class, please refer to NXActionLearn.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>src</td>
<td>OXM/NXM header and Start bit for source field</td>
</tr>
<tr>
<td>dst</td>
<td>Must be “”</td>
</tr>
<tr>
<td>n_bits</td>
<td>The number of bits from the start bit</td>
</tr>
</tbody>
</table>
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:

\[
\text{ofs_nbits} = (\text{start} << 6) + (\text{end} - \text{start})
\]

The parameter start/end means the OXM/NXM field of ovs-ofctl command.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>Start bit for OXM/NXM field</td>
</tr>
<tr>
<td>end</td>
<td>End bit for OXM/NXM field</td>
</tr>
</tbody>
</table>

### 2.6.2 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.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in_port_nxm</td>
<td>Integer 16bit</td>
<td>OpenFlow port number.</td>
</tr>
<tr>
<td>eth_dst_nxm</td>
<td>MAC address</td>
<td>Ethernet destination address.</td>
</tr>
<tr>
<td>eth_src_nxm</td>
<td>MAC address</td>
<td>Ethernet source address.</td>
</tr>
<tr>
<td>eth_type_nxm</td>
<td>Integer 16bit</td>
<td>Ethernet type. Needed to support Nicira extensions that require the eth_type to be set. (i.e. tcp_flags_nxm)</td>
</tr>
<tr>
<td>vlan_tci</td>
<td>Integer 16bit</td>
<td>VLAN TCI. Basically same as vlan_vid plus vlan_pcp.</td>
</tr>
<tr>
<td>nw_tos</td>
<td>Integer 8bit</td>
<td>IP ToS or IPv6 traffic class field dscp. Requires setting fields: eth_type_nxm = [0x0800 (IPv4)] and ip_proto_nxm = 6 (TCP)</td>
</tr>
<tr>
<td>ip_proto_nxm</td>
<td>Integer 8bit</td>
<td>IP protocol. Needed to support Nicira extensions that require the ip_proto to be set. (i.e. tcp_flags_nxm) Requires setting fields: eth_type_nxm = [0x0800 (IPv4)] and ip_proto_nxm = 6 (TCP)</td>
</tr>
<tr>
<td>ipv4_src_nxm</td>
<td>IPv4 address</td>
<td>IPv4 source address. Requires setting fields: eth_type_nxm = 0x0800 (IPv4)</td>
</tr>
<tr>
<td>ipv4_dst_nxm</td>
<td>IPv4 address</td>
<td>IPv4 destination address.</td>
</tr>
<tr>
<td>tcp_src_nxm</td>
<td>Integer 16bit</td>
<td>TCP source port. Requires setting fields: eth_type_nxm = [0x0800 (IPv4)]0x86dd (IPv6) and ip_proto_nxm = 6 (TCP)</td>
</tr>
<tr>
<td>tcp_dst_nxm</td>
<td>Integer 16bit</td>
<td>TCP destination port. Requires setting fields: eth_type_nxm = [0x0800 (IPv4)]0x86dd (IPv6) and ip_proto_nxm = 6 (TCP)</td>
</tr>
<tr>
<td>udp_src_nxm</td>
<td>Integer 16bit</td>
<td>UDP source port. Requires setting fields: eth_type_nxm = [0x0800 (IPv4)]0x86dd (IPv6) and ip_proto_nxm = 17 (UDP)</td>
</tr>
<tr>
<td>udp_dst_nxm</td>
<td>Integer 16bit</td>
<td>UDP destination port. eth_type_nxm = [0x0800 (IPv4)]0x86dd (IPv6) and ip_proto_nxm = 17 (UDP)</td>
</tr>
<tr>
<td>icmpv4_type_nxm</td>
<td>Integer 8bit</td>
<td>Type matches the ICMP type and code matches the ICMP code. Requires setting fields: eth_type_nxm = [0x0800 (IPv4)] and ip_proto_nxm = 1 (ICMP)</td>
</tr>
<tr>
<td>icmpv4_code_nxm</td>
<td>Integer 8bit</td>
<td>Type matches the ICMP type and code matches the ICMP code. Requires setting fields: eth_type_nxm = [0x0800 (IPv4)] and ip_proto_nxm = 1 (ICMP)</td>
</tr>
<tr>
<td>arp_op_nxm</td>
<td>Integer 16bit</td>
<td>Only ARP opcodes between 1 and 255 should be specified for matching. Requires setting fields: eth_type_nxm = 0x0806 (ARP)</td>
</tr>
<tr>
<td>arp_spa_nxm</td>
<td>IPv4 address</td>
<td>An address may be specified as an IP address or host name. Requires setting fields: eth_type_nxm = 0x0806 (ARP)</td>
</tr>
<tr>
<td>arp_tha_nxm</td>
<td>IPv4 address</td>
<td>An address may be specified as an IP address or host name. Requires setting fields: eth_type_nxm = 0x0806 (ARP)</td>
</tr>
<tr>
<td>ipv6_src_nxm</td>
<td>IPv6 address</td>
<td>IPv6 source address. Requires setting fields: eth_type_nxm = 0x86dd (IPv6)</td>
</tr>
<tr>
<td>ipv6_dst_nxm</td>
<td>IPv6 address</td>
<td>IPv6 destination address. Requires setting fields: eth_type_nxm = 0x86dd (IPv6)</td>
</tr>
<tr>
<td>icmpv6_type_nxm</td>
<td>Integer 8bit</td>
<td>Type matches the ICMP type and code matches the ICMP code. Requires setting fields: eth_type_nxm = [0x0800 (IPv4)] and ip_proto_nxm = 58 (ICMP for IPv6)</td>
</tr>
<tr>
<td>icmpv6_code_nxm</td>
<td>Integer 8bit</td>
<td>Type matches the ICMP type and code matches the ICMP code. Requires setting fields: eth_type_nxm = [0x0800 (IPv4)] and ip_proto_nxm = 58 (ICMP for IPv6)</td>
</tr>
<tr>
<td>nd_target</td>
<td>IPv6 address</td>
<td>The target address ipv6. Requires setting fields: eth_type_nxm = 0x86dd (IPv6) and ip_proto_nxm = 17 (UDP)</td>
</tr>
</tbody>
</table>

### 2.6. Nicira Extension Structures

493
<table>
<thead>
<tr>
<th>Argument</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nd_sll</td>
<td>MAC address</td>
<td>The source link-layer address option. Requires setting fields: eth_type_nxm = 0x86dd (IPv6) and ip_proto_nxm = 58 (ICMP for IPv6) and icmpv6_type_nxm = 135 (Neighbor solicitation)</td>
</tr>
<tr>
<td>nd_tll</td>
<td>MAC address</td>
<td>The target link-layer address option. Requires setting fields: eth_type_nxm = 0x86dd (IPv6) and ip_proto_nxm = 58 (ICMP for IPv6) and icmpv6_type_nxm = 136 (Neighbor advertisement)</td>
</tr>
<tr>
<td>ip_frag</td>
<td>Integer 8bit</td>
<td>The fragment type specifies what kind of IP fragments or non-fragments to match. Requires setting fields: eth_type_nxm = [0x0800 (IPv4)</td>
</tr>
<tr>
<td>ipv6_label</td>
<td>Integer 32bit</td>
<td>Matches IPv6 flow label. Requires setting fields: eth_type_nxm = 0x86dd (IPv6)</td>
</tr>
<tr>
<td>ip_ecn_nxm</td>
<td>Integer 8bit</td>
<td>Matches ecn bits in IP TOS or IPv6 traffic class fields. Requires setting fields: eth_type_nxm = [0x0800 (IPv4)</td>
</tr>
<tr>
<td>nw_ttl</td>
<td>Integer 8bit</td>
<td>IP TTL or IPv6 hop limit value. Requires setting fields: eth_type_nxm = [0x0800 (IPv4)</td>
</tr>
<tr>
<td>mpls_ttl</td>
<td>Integer 8bit</td>
<td>The TTL of the outer MPLS label stack entry of a packet. Requires setting fields: eth_type_nxm = 0x8847 (MPLS Unicast)</td>
</tr>
<tr>
<td>tun_ipv4_src</td>
<td>IPv4 address</td>
<td>Tunnel IPv4 source address. Requires setting fields: eth_type_nxm = 0x0800 (IPv4)</td>
</tr>
<tr>
<td>tun_ipv4_dst</td>
<td>IPv4 address</td>
<td>Tunnel IPv4 destination address. Requires setting fields: eth_type_nxm = 0x0800 (IPv4)</td>
</tr>
<tr>
<td>pkt_mark</td>
<td>Integer 32bit</td>
<td>Packet metadata mark.</td>
</tr>
<tr>
<td>tcp_flags_nxm</td>
<td>Integer 16bit</td>
<td>TCP Flags. Requires setting fields: eth_type_nxm = [0x0800 (IPv4)</td>
</tr>
<tr>
<td>conj_id</td>
<td>Integer 32bit</td>
<td>Conjunction ID used only with the conjunction action</td>
</tr>
<tr>
<td>tun_gbp_id</td>
<td>Integer 16bit</td>
<td>The group policy identifier in the VXLAN header.</td>
</tr>
<tr>
<td>tun_gbp_flags</td>
<td>Integer 8bit</td>
<td>The group policy flags in the VXLAN header.</td>
</tr>
<tr>
<td>tun_flags</td>
<td>Integer 16bit</td>
<td>Flags indicating various aspects of the tunnel encapsulation.</td>
</tr>
<tr>
<td>ct_state</td>
<td>Integer 32bit</td>
<td>Conntrack state.</td>
</tr>
<tr>
<td>ct_zone</td>
<td>Integer 16bit</td>
<td>Conntrack zone.</td>
</tr>
<tr>
<td>ct_mark</td>
<td>Integer 32bit</td>
<td>Conntrack mark.</td>
</tr>
<tr>
<td>ct_label</td>
<td>Integer 128bit</td>
<td>Conntrack label.</td>
</tr>
<tr>
<td>tun_ipv6_src</td>
<td>IPv6 address</td>
<td>Tunnel IPv6 source address. Requires setting fields: eth_type_nxm = 0x86dd (IPv6)</td>
</tr>
<tr>
<td>tun_ipv6_dst</td>
<td>IPv6 address</td>
<td>Tunnel IPv6 destination address. Requires setting fields: eth_type_nxm = 0x86dd (IPv6)</td>
</tr>
<tr>
<td>_recirc_id</td>
<td>Integer 32bit</td>
<td>ID for recirculation.</td>
</tr>
<tr>
<td>_dp_hash</td>
<td>Integer 32bit</td>
<td>Flow hash computed in Datapath.</td>
</tr>
<tr>
<td>nsh_flags</td>
<td>Integer 8bit</td>
<td>Flags field in NSH Base Header. Requires eth_type_nxm = 0x894f (NSH). Since OpenFlow 1.3.</td>
</tr>
<tr>
<td>nsh_mdtype</td>
<td>Integer 8bit</td>
<td>Metadata Type in NSH Base Header. Requires eth_type_nxm = 0x894f (NSH). Since OpenFlow 1.3.</td>
</tr>
<tr>
<td>nsh_np</td>
<td>Integer 8bit</td>
<td>Next Protocol type in NSH Base Header. Requires eth_type_nxm = 0x894f (NSH). Since OpenFlow 1.3.</td>
</tr>
<tr>
<td>nsh_spi</td>
<td>Integer 32bit</td>
<td>Service Path Identifier in NSH Service Path Header. Requires eth_type_nxm = 0x894f (NSH). Since OpenFlow 1.3.</td>
</tr>
<tr>
<td>nsh_si</td>
<td>Integer 8bit</td>
<td>Service Index in NSH Service Path Header. Requires eth_type_nxm = 0x894f (NSH). Since OpenFlow 1.3.</td>
</tr>
<tr>
<td>nsh_&lt;N&gt;</td>
<td>Integer 32bit</td>
<td>Context fields in NSH Context Header. &lt;N&gt; is a number of 0-4. Requires eth_type_nxm = 0x894f (NSH). Since OpenFlow 1.3.</td>
</tr>
<tr>
<td>reg&lt;idx&gt;</td>
<td>Integer 32bit</td>
<td>Packet register. &lt;idx&gt; is register number 0-15.</td>
</tr>
<tr>
<td>xxreg&lt;idx&gt;</td>
<td>Integer 128bit</td>
<td>Packet extended-extended register. &lt;idx&gt; is register number 0-3.</td>
</tr>
</tbody>
</table>

**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:**

```python
# 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)
```

494 Chapter 2. Writing Your Ryu Application
2.7 Ryu API Reference

class ryu.base.app_manager.RyuApp(*_args, **_kwargs):
    The base class for Ryu applications.

    RyuApp subclasses are instantiated after ryu-manager loaded all requested Ryu application modules. __init__ should call RyuApp.__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 Ryu applications. (Cf. send_event) It’s set to __class__.__name__ by RyuApp.__init__. It’s discouraged for subclasses to override this.

OFP_VERSIONS = None
    A list of supported OpenFlow versions for this RyuApp. The default is all versions supported by the framework.

    Examples:

    ```python
    OFP_VERSIONS = [ofproto_v1_0.OFP_VERSION,
                    ofproto_v1_2.OFP_VERSION]
    ```

    If multiple Ryu applications are loaded in the system, the intersection of their OFP_VERSIONS is used.

_CONTEXTS = {}
    A dictionary to specify contexts which this Ryu 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 RyuApp subclasses which has _CONTEXTS member with the same key. A RyuApp subclass can obtain a reference to the instance via its __init__’s kwargs as the following.

    Example:

    ```python
    _CONTEXTS = {
        'network': network.Network
    }
    def __init__(self, *args, **kwags):
        self.network = kwags['network']
    ```

_EVENTS = []
    A list of event classes which this RyuApp subclass would generate. This should be specified if and only if event classes are defined in a different python module from the RyuApp subclass is.

close()
    teardown method. The method name, close, is chosen for python context manager

classmethod context_iteritems()
    Return iterator over the (key, context 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 RyuApp instance specified by name.

send_event_to_observers (ev, state=None)
    Send the specified event to all observers of this RyuApp.
send_request(req)
    Make a synchronous request. Set req.sync to True, send it to a Ryu 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 ryu.controller.dpset.DPSet(*args, **kwargs)
    DPSet application manages a set of switches (datapaths) connected to this controller.

Usage Example:

```python
from ryu.controller import dpset

class MyApp(app_manager.RyuApp):
    _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 ryu.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 ryu.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 ryu.controller.dpset.PortState instance for the given Datapath ID and the port number. Raises ryu_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 ryu.controller.dpset.PortState instances for the given Datapath ID. Raises KeyError if no such a datapath connected to this controller.
3.1 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 Ryu to connect to the Open vSwitch over TLS.

3.1.1 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
```

cntl-privkey.pem and cntl-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.
### 3.1.2 Testing TLS Connection

Configuring ovs-vswitchd to use CA files using the ovs-vsctl "set-ssl" command, e.g.:

```bash
% 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 Ryu with CA files:

```bash
% ryu-manager --ctl-privkey ctl-privkey.pem \
   --ctl-cert ctl-cert.pem \
   --ca-certs /usr/local/var/lib/openvswitch/pki/switchca/cacert.pem \
   --verbose
```

You can see something like:

```bash
loading app ryu.controller.ofp_handler
instantiating app ryu.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> address:('127.0.0.1', 61302)
hello ev <ryu.controller.ofp_event.EventOFPHello object at 0x1047806d0> move onto config mode
move onto main mode
```

### 3.2 Topology Viewer

ryu.app.gui_topology.gui_topology provides topology visualization.

This depends on following ryu applications.

<table>
<thead>
<tr>
<th>Application</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ryu.app.rest_toplogy</td>
<td>Get node and link data.</td>
</tr>
<tr>
<td>ryu.app.ws_toplogy</td>
<td>Being notified change of link up/down.</td>
</tr>
<tr>
<td>ryu.app.ofctl_rest</td>
<td>Get flows of datapaths.</td>
</tr>
</tbody>
</table>

#### 3.2.1 Usage

Run mininet (or join your real environment):

```bash
$ sudo mn --controller remote --topo tree,depth=3
```
Run GUI application:

```
$ PYTHONPATH=. ./bin/ryu run --observe-links ryu/app/gui_topology/gui_topology.py
```

Access http://<ip address of ryu host>:8080 with your web browser.

### 3.2.2 Screenshot

![Ryu Topology Viewer](image)

- ["actions": ["OUTPUT:65533"], "idle_timeout": 0, "cookie": 0, "packet_count": 18270, "hard_timeout": 0, "byte_count": 931770, "duration_nsec": 119000000, "priority": 65535, "duration sec": 6114, "table id": 0, "match": {
  "dl_type": 35020, "nw_dst": "0.0.0.0", "dl_vlan pcb": 0, "dl_src": "00:00:00:00:00:00", "tp_src": 0, "dl_vlan": 0, "nw_src": "0.0.0.0", "nw_proto": 0, "tp_dst": 0, "dl_dst": "01:80:c2:00:00:00", "in_port": 0 } ]}
4.1 Testing VRRP Module

This page describes how to test Ryu VRRP service

4.1.1 Running integrated tests

Some testing scripts are available.

- ryu/tests/integrated/test_vrrp_linux_multi.py
- ryu/tests/integrated/test_vrrp_multi.py

Each files include how to run in the comment. Please refer to it.

4.1.2 Running multiple Ryu VRRP in network namespace

The following command lines set up necessary bridges and interfaces.

And then run RYU-VRRP:

```
# ip netns add gateway1
# ip netns add gateway2
# brctl addbr vrrp-br0
# brctl addbr vrrp-br1
# 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
```
# brctl addif vrrp-br0 veth0-br0
# brctl addif vrrp-br0 veth1-br0
# brctl addif vrrp-br0 veth2-br0
# brctl addif vrrp-br1 veth3-br1
# brctl addif vrrp-br1 veth4-br1
# brctl addif vrrp-br1 veth5-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 .ryu-vrrp veth1 '10.0.0.2' 254
# ip netns exec gateway2 .ryu-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.
Here's the helper executable, ryu-vrrp:

```python
#!/usr/bin/env python
#
# 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 ryu.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,
# use it
#
# NOTE: this modifies sys.path and thus affects the following imports.
# eg. oslo.config.cfg.
import ryu.contrib

from oslo.config import cfg
import logging
import netaddr
import sys
import time

from ryu import log
log.early_init_log(logging.DEBUG)

from ryu import flags
from ryu import version
from ryu.base import app_manager
from ryu.controller import controller
from ryu.lib import mac as lib_mac
from ryu.lib.packet import vrrp
from ryu.services.protocols.vrrp import api as vrrp_api
from ryu.services.protocols.vrrp import event as vrrp_event
```

(continues on next page)
CONF = cfg.CONF

_VRID = 7
_IP_ADDRESS = '10.0.0.1'
_PRIORITY = 100

class VRRPTestRouter(app_manager.RyuApp):
    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)
        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='ryu', version='ryu-vrrp %s' % version)
    log.init_log()
    # always enable ofp for now.
    app_lists = ['ryu.services.protocols.vrrp.manager',
                 'ryu.services.protocols.vrrp.dumper',
                 'ryu.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()

4.2 Testing OF-config support with LINC

This page describes how to setup LINC and test Ryu 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 Ryu test_of_config app

For getting/installing Ryu itself, please refer to https://ryu-sdn.org/

4.2.1 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.

4.2.2 build LINC

install necessary packages for build

install necessary build tools

On Ubuntu:

```bash
# apt-get install git-core bridge-utils libpcap0.8 libpcap-dev libcap2-bin uml-utilities
```

On RedHat/CentOS:

```bash
# yum install git sudo bridge-utils libpcap libpcap-devel libcap tunctl
```

Note that on RedHat/CentOS 5.x you need a newer version of libpcap:

4.2. Testing OF-config support with LINC 505
# 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:

```bash
% git clone git://github.com/FlowForwarding/LINC-Switch.git
```

Then compile everything:

```bash
% 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.

```bash
% cd LINC-Switch
% make
% cd deps/of_config
% git reset --hard f772af4b765984381ad024ca8e5b5b8c54362638
% cd ../..
% make offline
```

### 4.2.3 Setup LINC

edit LINC switch configuration file. rel/linc/releases/0.1/sys.config

Here is the sample sys.config for test_of_config.py to run.

```json
[{
  "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],
     (continues on next page)
```
4.2.4 setup for LINC

As the above sys.config requires some network interface, create them:

```bash
# 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:

```bash
# ip link delete linc-port
# ip link delete linc-port2
# ip link delete linc-port3
# ip link delete linc-port4
```
4.2.5 Starting LINC OpenFlow switch

Then run LINC:

```
# rel/linc/bin/linc console
```

4.2.6 Run Ryu test_of_config app

Run test_of_config app:

```
# ryu-manager --verbose ryu.tests.integrated.test_of_config ryu.app.rest
```

If you don’t install ryu and are working in the git repo directly:

```
# PYTHONPATH=. ./bin/ryu-manager --verbose ryu.tests.integrated.test_of_config ryu.app.rest
```
This document describes how to integrate Ryu with Snort.

5.1 Overview

There are two options can send alert to Ryu 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] Ryu and Snort are on the same machine

```
+---------------------+
| unixsock            |
|  Ryu == snort       |
| +-----eth0-----eth1----+
|                |     |
+----eth0-----eth1----+
```

The above depicts Ryu and Snort architecture. Ryu 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] Ryu and Snort are on the different machines

```
+---------------------+
| Snort   eth0--|    |
| Sniffer         |    |
| +-----eth1------+
|                |     |
+---------------------+
```

(continues on next page)
The above depicts Ryu and Snort architecture. Ryu receives Snort alert packet via Network Socket. To monitor packets between HostA and HostB, installing a flow that mirrors packets to Snort.

5.2 Installation Snort

Snort is an open source network intrusion prevention and detectionsystem developed by Sourcefire. If you are not familiar with installing/setting up Snort, please refer to snort setup guides.

http://www.snort.org/documents

5.3 Configure Snort

The configuration example is below:

- Add a snort rules file into /etc/snort/rules named Myrules.rules

```plaintext
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

```plaintext
include $RULE_PATH/Myrules.rules
```

Configure NIC as a promiscuous mode.

```bash
$ sudo ifconfig eth1 promisc
```

5.4 Usage

[Option 1]

1. Modify the simple_switch_snort.py:

```python
socket_config = {'unixsock': True}
# True: Unix Domain Socket Server [Option1]
# False: Network Socket Server [Option2]
```

2. Run Ryu with sample application:

```bash
$ sudo ./bin/ryu-manager ryu/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:
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`:

   ```python
   socket_config = {'unixsock': False}
   # True: Unix Domain Socket Server [Option1]
   # False: Network Socket Server [Option2]
   ```

2. Run Ryu with sample application (On the Controller):

   ```
   $ ./bin/ryu-manager ryu/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 Ryu using network socket.

You can clone the source code from this repo. [https://github.com/John-Lin/pigrelay](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]), 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]), 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)
```

(continues on next page)
Ryu has some built-in Ryu applications. Some of them are examples. Others provide some functionalities to other Ryu applications.

## 6.1 ryu.app.ofctl

ryu.app.ofctl provides a convenient way to use OpenFlow messages synchronously.

OfctlService ryu application is automatically loaded if your Ryu application imports ofctl.api module.

Example:

```python
import ryu.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.

### 6.1.1 api module

ryu.app.ofctl.api.get_datapath(app, dpid=None)

Get datapath object by dpid.

**Parameters**

- **app** – Client RyuApp 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 ryu.app.ofctl.api as ofctl_api

class MyApp(app_manager.RyuApp):
    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)

ryu.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 RyuApp 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 ryu.app.ofctl.api as ofctl_api
class MyApp(app_manager.RyuApp):
    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)

6.1.2 exceptions

- **exception** ryu.app.ofctl.exception.InvalidDatapath(result)
  Datapath is invalid.
  This can happen when the bridge disconnects.

- **exception** ryu.app.ofctl.exception.OFError(result)
  OFErrorMsg is received.

- **exception** ryu.app.ofctl.exception.UnexpectedMultiReply(result)
  Two or more replies are received for reply_multi=False request.
6.2 ryu.app.ofctl_rest

ryu.app.ofctl_rest provides REST APIs for retrieving the switch stats and Updating the switch stats. This application helps you debug your application and get various statistics.

This application supports OpenFlow version 1.0, 1.2, 1.3, 1.4 and 1.5.

Contents

- ryu.app.ofctl_rest
  - Retrieve the switch stats
    * Get all switches
    * Get the desc stats
    * Get all flows stats
    * Get flows stats filtered by fields
    * Get aggregate flow stats
    * Get aggregate flow stats filtered by fields
    * Get table stats
    * Get table features
    * Get ports stats
    * Get ports description
    * Get queues stats
    * Get queues config
    * Get queues description
    * Get groups stats
    * Get group description stats
    * Get group features stats
    * Get meters stats
    * Get meter config stats
    * Get meter description stats
    * Get meter features stats
    * Get role
  - Update the switch stats
    * Add a flow entry
    * Modify all matching flow entries
    * Modify flow entry strictly
    * Delete all matching flow entries
    * Delete flow entry strictly
6.2.1 Retrieve the switch stats

Get all switches

Get the list of all switches which connected to the controller.

Usage:

<table>
<thead>
<tr>
<th>Method</th>
<th>GET</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
<td>/stats switches</td>
</tr>
</tbody>
</table>

Response message body:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>dpid</td>
<td>Datapath ID</td>
<td>1</td>
</tr>
</tbody>
</table>

Example of use:

```
$ curl -X GET http://localhost:8080/stats/switches
```

```
[1, 2, 3]
```

**Note:** The result of the REST command is formatted for easy viewing.
Get the desc stats

Get the desc stats of the switch which specified with Datapath ID in URI.

Usage:

<table>
<thead>
<tr>
<th>Method</th>
<th>GET</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
<td>/stats/desc/&lt;dpid&gt;</td>
</tr>
</tbody>
</table>

Response message body:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>dpid</td>
<td>Datapath ID</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>mfr_desc</td>
<td>Manufacturer description</td>
<td>&quot;Nicira, Inc.&quot;</td>
</tr>
<tr>
<td>hw_desc</td>
<td>Hardware description</td>
<td>&quot;Open vSwitch&quot;</td>
</tr>
<tr>
<td>sw_desc</td>
<td>Software description</td>
<td>&quot;2.3.90&quot;</td>
</tr>
<tr>
<td>serial_num</td>
<td>Serial number</td>
<td>&quot;None&quot;</td>
</tr>
<tr>
<td>dp_desc</td>
<td>Human readable description of datapath</td>
<td>&quot;None&quot;</td>
</tr>
</tbody>
</table>

Example of use:

```bash
$ curl -X GET http://localhost:8080/stats/desc/1
```

```
{
  "1": {
    "mfr_desc": "Nicira, Inc.",
    "hw_desc": "Open vSwitch",
    "sw_desc": "2.3.90",
    "serial_num": "None",
    "dp_desc": "None"
  }
}
```

Get all flows stats

Get all flows stats of the switch which specified with Datapath ID in URI.

Usage:

<table>
<thead>
<tr>
<th>Method</th>
<th>GET</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
<td>/stats/flow/&lt;dpid&gt;</td>
</tr>
</tbody>
</table>

Response message body (OpenFlow 1.3 or earlier):
### Attribute Description Example

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>dpid</td>
<td>Datapath ID</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>length</td>
<td>Length of this entry</td>
<td>88</td>
</tr>
<tr>
<td>table_id</td>
<td>Table ID</td>
<td>0</td>
</tr>
<tr>
<td>duration_sec</td>
<td>Time flow has been alive in seconds</td>
<td>2</td>
</tr>
<tr>
<td>duration_nsec</td>
<td>Time flow has been alive in nanoseconds beyond duration_sec</td>
<td>6.76e+08</td>
</tr>
<tr>
<td>priority</td>
<td>Priority of the entry</td>
<td>11111</td>
</tr>
<tr>
<td>idle_timeout</td>
<td>Number of seconds idle before expiration</td>
<td>0</td>
</tr>
<tr>
<td>hard_timeout</td>
<td>Number of seconds before expiration</td>
<td>0</td>
</tr>
<tr>
<td>flags</td>
<td>Bitmap of OFPFF_* flags</td>
<td>1</td>
</tr>
<tr>
<td>cookie</td>
<td>Opaque controller-issued identifier</td>
<td>1</td>
</tr>
<tr>
<td>packet_count</td>
<td>Number of packets in flow</td>
<td>0</td>
</tr>
<tr>
<td>byte_count</td>
<td>Number of bytes in flow</td>
<td>0</td>
</tr>
<tr>
<td>match</td>
<td>Fields to match</td>
<td>{&quot;in_port&quot;: 1}</td>
</tr>
<tr>
<td>actions</td>
<td>Instruction set</td>
<td>[&quot;OUT:PUT:2&quot;]</td>
</tr>
</tbody>
</table>

Response message body (OpenFlow1.4 or later):

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>dpid</td>
<td>Datapath ID</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>length</td>
<td>Length of this entry</td>
<td>88</td>
</tr>
<tr>
<td>table_id</td>
<td>Table ID</td>
<td>0</td>
</tr>
<tr>
<td>duration_sec</td>
<td>Time flow has been alive in seconds</td>
<td>2</td>
</tr>
<tr>
<td>duration_nsec</td>
<td>Time flow has been alive in nanoseconds beyond duration_sec</td>
<td>6.76e+08</td>
</tr>
<tr>
<td>priority</td>
<td>Priority of the entry</td>
<td>11111</td>
</tr>
<tr>
<td>idle_timeout</td>
<td>Number of seconds idle before expiration</td>
<td>0</td>
</tr>
<tr>
<td>hard_timeout</td>
<td>Number of seconds before expiration</td>
<td>0</td>
</tr>
<tr>
<td>flags</td>
<td>Bitmap of OFPFF_* flags</td>
<td>1</td>
</tr>
<tr>
<td>cookie</td>
<td>Opaque controller-issued identifier</td>
<td>1</td>
</tr>
<tr>
<td>packet_count</td>
<td>Number of packets in flow</td>
<td>0</td>
</tr>
<tr>
<td>byte_count</td>
<td>Number of bytes in flow</td>
<td>0</td>
</tr>
<tr>
<td>importance</td>
<td>Eviction precedence</td>
<td>0</td>
</tr>
<tr>
<td>match</td>
<td>Fields to match</td>
<td>{&quot;eth_type&quot;: 2054}</td>
</tr>
<tr>
<td>actions</td>
<td>struct ofp_instruction_header</td>
<td>[&quot;type&quot;:GOTO_TABLE, &quot;table_id&quot;:1]</td>
</tr>
</tbody>
</table>

Example of use:

```bash
$ curl -X GET http://localhost:8080/stats/flow/1
```

Response (OpenFlow1.3 or earlier):

```json
{
   "1": [
   {
      "length": 88,
      "
```
"table_id": 0,
"duration_sec": 2,
"duration_nsec": 6.76e+08,
"priority": 11111,
"idle_timeout": 0,
"hard_timeout": 0,
"flags": 1,
"cookie": 1,
"packet_count": 0,
"byte_count": 0,
"match": {
   "in_port": 1
},
"actions": [
   "OUTPUT:2"
]}
)

Response (OpenFlow1.4 or later):

{
   "1": [
   {
   "length": 88,
   "table_id": 0,
   "duration_sec": 2,
   "duration_nsec": 6.76e+08,
   "priority": 11111,
   "idle_timeout": 0,
   "hard_timeout": 0,
   "flags": 1,
   "cookie": 1,
   "packet_count": 0,
   "byte_count": 0,
   "match": {
      "eth_type": 2054
   },
   "importance": 0,
   "instructions": [
   {"type": "APPLY_ACTIONS",
    "actions": [
    {"port": 2,
     "max_len": 0,
     "type": "OUTPUT"
    }
    ]
   ]
   }]
}
)
Get flows stats filtered by fields

Get flows stats of the switch filtered by the OFPFlowStats fields. This is POST method version of Get all flows stats.

Usage:

Method: POST
URI: /stats/flow/<dpid>

Request message body:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Example</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_id</td>
<td>Table ID (int)</td>
<td>0</td>
<td>OF-PFTT_ALL</td>
</tr>
<tr>
<td>out_port</td>
<td>Require matching entries to include this as an output port (int)</td>
<td>2</td>
<td>OFPP_ANY</td>
</tr>
<tr>
<td>out_group</td>
<td>Require matching entries to include this as an output group (int)</td>
<td>1</td>
<td>OFPG_ANY</td>
</tr>
<tr>
<td>cookie</td>
<td>Require matching entries to contain this cookie value (int)</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>cookie_mask</td>
<td>Mask used to restrict the cookie bits that must match (int)</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>match</td>
<td>Fields to match (dict)</td>
<td>{&quot;in_port&quot;:1}</td>
<td>#wild-carded</td>
</tr>
<tr>
<td>priority</td>
<td>Priority of the entry (int) (See Note)</td>
<td>11111</td>
<td>#wild-carded</td>
</tr>
</tbody>
</table>

Note: OpenFlow Spec does not allow to filter flow entries by priority, but when with a large amount of flow entries, filtering by priority is convenient to get statistics efficiently. So, this app provides priority field for filtering.

Response message body: The same as Get all flows stats

Example of use:

```
$ curl -X POST -d '{
   "table_id": 0,
   "out_port": 2,
   "cookie": 1,
   "cookie_mask": 1,
   "match":{
      "in_port":1
   }
}' http://localhost:8080/stats/flow/1
```

Response (OpenFlow1.3 or earlier):

```
{
   "1": [
   {
      "length": 88,
      "table_id": 0,
```

(continues on next page)
"duration_sec": 2,
"duration_nsec": 6.76e+08,
"priority": 11111,
"idle_timeout": 0,
"hard_timeout": 0,
"flags": 1,
"cookie": 1,
"packet_count": 0,
"byte_count": 0,
"match": {
    "in_port": 1
},
"actions": [
    "OUTPUT:2"
]}
]
]}

Response (OpenFlow1.4 or later):

{
    "1": [
        {
            "length": 88,
            "table_id": 0,
            "duration_sec": 2,
            "duration_nsec": 6.76e+08,
            "priority": 11111,
            "idle_timeout": 0,
            "hard_timeout": 0,
            "flags": 1,
            "cookie": 1,
            "packet_count": 0,
            "byte_count": 0,
            "match": {
                "eth_type": 2054
            },
            "importance": 0,
            "instructions": [
                {
                    "type": "APPLY_ACTIONS",
                    "actions": [
                        {
                            "port": 2,
                            "max_len": 0,
                            "type": "OUTPUT"
                        }
                    ]
                }
            ]
        }
    ]
}
Get aggregate flow stats

Get aggregate flow stats of the switch which specified with Datapath ID in URI.

Usage:

<table>
<thead>
<tr>
<th>Method</th>
<th>GET</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
<td>/stats/aggregateflow/&lt;dpid&gt;</td>
</tr>
</tbody>
</table>

Response message body:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>dpid</td>
<td>Datapath ID</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>packet_count</td>
<td>Number of packets in flows</td>
<td>18</td>
</tr>
<tr>
<td>byte_count</td>
<td>Number of bytes in flows</td>
<td>756</td>
</tr>
<tr>
<td>flow_count</td>
<td>Number of flows</td>
<td>3</td>
</tr>
</tbody>
</table>

Example of use:

```bash
$ curl -X GET http://localhost:8080/stats/aggregateflow/1
```

```json
{
   "1": [
      {
         "packet_count": 18,
         "byte_count": 756,
         "flow_count": 3
      }
   ]
}
```

Get aggregate flow stats filtered by fields

Get aggregate flow stats of the switch filtered by the OFPAggregateStats fields. This is POST method version of Get aggregate flow stats.

Usage:

<table>
<thead>
<tr>
<th>Method</th>
<th>POST</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
<td>/stats/aggregateflow/&lt;dpid&gt;</td>
</tr>
</tbody>
</table>

Request message body:
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Example</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_id</td>
<td>Table ID (int)</td>
<td>0</td>
<td>OF-PTT_ALL</td>
</tr>
<tr>
<td>out_port</td>
<td>Require matching entries to include this as an output port (int)</td>
<td>2</td>
<td>OFPP_ANY</td>
</tr>
<tr>
<td>out_group</td>
<td>Require matching entries to include this as an output group (int)</td>
<td>1</td>
<td>OFPG_ANY</td>
</tr>
<tr>
<td>cookie</td>
<td>Require matching entries to contain this cookie value (int)</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>cookie_mask</td>
<td>Mask used to restrict the cookie bits that must match (int)</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>match</td>
<td>Fields to match (dict)</td>
<td>{} #wild-carded</td>
<td></td>
</tr>
</tbody>
</table>

**Response message body:** The same as *Get aggregate flow stats*

Example of use:

```
$ curl -X POST -d '{
  "table_id": 0,
  "out_port": 2,
  "cookie": 1,
  "cookie_mask": 1,
  "match":{
    "in_port":1
  }
}' http://localhost:8080/stats/aggregateflow/1
```

```json
{
  "1": [
    {
      "packet_count": 18,
      "byte_count": 756,
      "flow_count": 3
    }
  ]
}
```

**Get table stats**

Get table stats of the switch which specified with Datapath ID in URI.

Usage:

<table>
<thead>
<tr>
<th>Method</th>
<th>GET</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
<td>/stats/table/&lt;dpid&gt;</td>
</tr>
</tbody>
</table>

Response message body(OpenFlow1.0):
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>dpid</td>
<td>Datapath ID</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>table_id</td>
<td>Table ID</td>
<td>0</td>
</tr>
<tr>
<td>name</td>
<td>Name of Table</td>
<td>&quot;classifier&quot;</td>
</tr>
<tr>
<td>max_entries</td>
<td>Max number of entries supported</td>
<td>1e+06</td>
</tr>
<tr>
<td>wildcards</td>
<td>Bitmap of OFPTFW_* wildcards that are supported by the table</td>
<td>[&quot;IN_PORT&quot;,&quot;DL_VLAN&quot;]</td>
</tr>
<tr>
<td>active_count</td>
<td>Number of active entries</td>
<td>0</td>
</tr>
<tr>
<td>lookup_count</td>
<td>Number of packets looked up in table</td>
<td>8</td>
</tr>
<tr>
<td>matched_count</td>
<td>Number of packets that hit table</td>
<td>0</td>
</tr>
</tbody>
</table>

Response message body(OpenFlow1.2):

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>dpid</td>
<td>Datapath ID</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>table_id</td>
<td>Table ID</td>
<td>0</td>
</tr>
<tr>
<td>name</td>
<td>Name of Table</td>
<td>&quot;classifier&quot;</td>
</tr>
<tr>
<td>match</td>
<td>Bitmap of (1 &lt;&lt; OFPXMT_*) that indicate the fields the table can match on</td>
<td>[&quot;OFB_IN_PORT&quot;,&quot;OFB_METADATA&quot;]</td>
</tr>
<tr>
<td>wildcards</td>
<td>Bitmap of (1 &lt;&lt; OFPXMT_*) wildcards that are supported by the table</td>
<td>[&quot;OFB_IN_PORT&quot;,&quot;OFB_METADATA&quot;]</td>
</tr>
<tr>
<td>write_actions</td>
<td>Bitmap of OFPAT_* that are supported by the table with OFPIT_WRITE_ACTIONS</td>
<td>[&quot;OUT-PUT&quot;,&quot;SET_MPLS_TTL&quot;]</td>
</tr>
<tr>
<td>apply_actions</td>
<td>Bitmap of OFPAT_* that are supported by the table with OFPIT_APPLY_ACTIONS</td>
<td>[&quot;OUT-PUT&quot;,&quot;SET_MPLS_TTL&quot;]</td>
</tr>
<tr>
<td>write_setfields</td>
<td>Bitmap of (1 &lt;&lt; OFPXMT_*) header fields that can be set with OFPIT_WRITE_ACTIONS</td>
<td>[&quot;OFB_IN_PORT&quot;,&quot;OFB_METADATA&quot;]</td>
</tr>
<tr>
<td>apply_setfields</td>
<td>Bitmap of (1 &lt;&lt; OFPXMT_*) header fields that can be set with OFPIT_APPLY_ACTIONS</td>
<td>[&quot;OFB_IN_PORT&quot;,&quot;OFB_METADATA&quot;]</td>
</tr>
<tr>
<td>metadata_match</td>
<td>Bits of metadata table can match</td>
<td>18446744073709552000</td>
</tr>
<tr>
<td>metadata_write</td>
<td>Bits of metadata table can write</td>
<td>18446744073709552000</td>
</tr>
<tr>
<td>instructions</td>
<td>Bitmap of OFPIT_* values supported</td>
<td>[&quot;GOTO_TABLE&quot;,&quot;WRITE_METADATA&quot;]</td>
</tr>
<tr>
<td>config</td>
<td>Bitmap of OFPTC_* values</td>
<td>[]</td>
</tr>
<tr>
<td>max_entries</td>
<td>Max number of entries supported</td>
<td>1e+06</td>
</tr>
<tr>
<td>active_count</td>
<td>Number of active entries</td>
<td>0</td>
</tr>
<tr>
<td>lookup_count</td>
<td>Number of packets looked up in table</td>
<td>0</td>
</tr>
<tr>
<td>matched_count</td>
<td>Number of packets that hit table</td>
<td>8</td>
</tr>
</tbody>
</table>

Response message body(OpenFlow1.3):
Attribute | Description | Example
--- | --- | ---
dpid | Datapath ID | "1"
table_id | Table ID | 0
active_count | Number of active entries | 0
lookup_count | Number of packets looked up in table | 8
matched_count | Number of packets that hit table | 0

Example of use:

```bash
$ curl -X GET http://localhost:8080/stats/table/1
```

Response (OpenFlow1.0):

```json
{
  "1": [
    {
      "table_id": 0,
      "lookup_count": 8,
      "max_entries": 1e+06,
      "active_count": 0,
      "name": "classifier",
      "matched_count": 0,
      "wildcards": [
        "IN_PORT",
        "DL_VLAN"
      ]
    },
    ...
    {
      "table_id": 253,
      "lookup_count": 0,
      "max_entries": 1e+06,
      "active_count": 0,
      "name": "table253",
      "matched_count": 0,
      "wildcards": [
        "IN_PORT",
        "DL_VLAN"
      ]
    }
  ]
}
```

Response (OpenFlow1.2):

```json
{
  "1": [
    {
      "apply_setfields": [
        "OFB_IN_PORT",
        "OFB_METADATA"
      ],
      "match": [
        "OFB_IN_PORT",
        "OFB_METADATA"
      ]
    },
    ...
    {
      "apply_setfields": [
        "OFB_IN_PORT",
        "OFB_METADATA"
      ],
      "match": [
        "OFB_IN_PORT",
        "OFB_METADATA"
      ]
    }
  ]
}
```
"metadata_write": 18446744073709552000,
"config": [],
"instructions": [
  "GOTO_TABLE",
  "WRITE_METADATA"
],
"table_id": 0,
"metadata_match": 18446744073709552000,
"lookup_count": 8,
"wildcards": [
  "OFB_IN_PORT",
  "OFB_METADATA"
],
"write_setfields": [
  "OFB_IN_PORT",
  "OFB_METADATA"
],
"write_actions": [
  "OUTPUT",
  "SET_MPLS_TTL"
],
"name": "classifier",
"matched_count": 0,
"apply_actions": [
  "OUTPUT",
  "SET_MPLS_TTL"
],
"active_count": 0,
"max_entries": 1e+06
},
...
{
  "apply_setfields": [
    "OFB_IN_PORT",
    "OFB_METADATA"
  ],
  "match": [
    "OFB_IN_PORT",
    "OFB_METADATA"
  ],
  "metadata_write": 18446744073709552000,
  "config": [],
  "instructions": [
    "GOTO_TABLE",
    "WRITE_METADATA"
  ],
  "table_id": 253,
  "metadata_match": 18446744073709552000,
  "lookup_count": 0,
  "wildcards": [
    "OFB_IN_PORT",
    "OFB_METADATA"
  ],
  "write_setfields": [
    "OFB_IN_PORT",
    "OFB_METADATA"
  ]
}
"write_actions": [ 
  "OUTPUT",
  "SET_MPLS_TTL"
],
"name": "table253",
"matched_count": 0,
"apply_actions": [ 
  "OUTPUT",
  "SET_MPLS_TTL"
],
"active_count": 0,
"max_entries": 1e+06
}]
}

Response (OpenFlow1.3):

{
  "1": [ 
    {
      "active_count": 0,
      "table_id": 0,
      "lookup_count": 8,
      "matched_count": 0
    },
    ...
    {
      "active_count": 0,
      "table_id": 253,
      "lookup_count": 0,
      "matched_count": 0
    }
  ]
}

Get table features

Get table features of the switch which specified with Datapath ID in URI.

Usage:

<table>
<thead>
<tr>
<th>Method</th>
<th>GET</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
<td>/stats/tablefeatures/&lt;dpid&gt;</td>
</tr>
</tbody>
</table>

Response message body:
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>dpid</td>
<td>Datapath ID</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>table_id</td>
<td>Table ID</td>
<td>0</td>
</tr>
<tr>
<td>name</td>
<td>Name of Table</td>
<td>&quot;table_0&quot;</td>
</tr>
<tr>
<td>meta-data_match</td>
<td>Bits of metadata table can match</td>
<td>18446744073709552000</td>
</tr>
<tr>
<td>meta-data_write</td>
<td>Bits of metadata table can write</td>
<td>18446744073709552000</td>
</tr>
<tr>
<td>config</td>
<td>Bitmap of OFPTC_* values</td>
<td>0</td>
</tr>
<tr>
<td>max_entries</td>
<td>Max number of entries supported</td>
<td>4096</td>
</tr>
<tr>
<td>properties</td>
<td>struct ofp_table_feature_prop_header</td>
<td>[{&quot;type&quot;: &quot;INSTRUCTIONS&quot;, &quot;instruction_ids&quot;: [...]}]</td>
</tr>
</tbody>
</table>

Example of use:

```bash
$ curl -X GET http://localhost:8080/stats/tablefeatures/1
```

```json
{
    "1": [
        {
            "metadata_write": 18446744073709552000,
            "config": 0,
            "table_id": 0,
            "metadata_match": 18446744073709552000,
            "max_entries": 4096,
            "properties": [
                {
                    "type": "INSTRUCTIONS",
                    "instruction_ids": [
                        {
                            "len": 4,
                            "type": 1
                        },
                        ...
                    ]
                },
                ...
            ],
            "name": "table_0"
        },
        {
            "metadata_write": 18446744073709552000,
            "config": 0,
            "table_id": 1,
            "metadata_match": 18446744073709552000,
            "max_entries": 4096,
            "properties": [
                {
                    "type": "INSTRUCTIONS",
                    "instruction_ids": [
                        {
                            "len": 4,
                            "type": 1
                        }
                    ]
                }
            ],
            "name": "table_1"
        }
    ]
}
```

(continues on next page)


...}{
  },
  ...
],
  "name": "table_1"
],
  ...
}

Get ports stats

Get ports stats of the switch which specified with Datapath ID in URI.

Usage:

<table>
<thead>
<tr>
<th>Method</th>
<th>GET</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
<td>/stats/port/&lt;dpid&gt;/[&lt;port&gt;]</td>
</tr>
</tbody>
</table>

**Note:** Specification of port number is optional.

Response message body(OpenFlow1.3 or earlier):

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>dpid</td>
<td>Datapath ID</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>port_no</td>
<td>Port number</td>
<td>1</td>
</tr>
<tr>
<td>rx_packets</td>
<td>Number of received packets</td>
<td>9</td>
</tr>
<tr>
<td>tx_packets</td>
<td>Number of transmitted packets</td>
<td>6</td>
</tr>
<tr>
<td>rx_bytes</td>
<td>Number of received bytes</td>
<td>738</td>
</tr>
<tr>
<td>tx_bytes</td>
<td>Number of transmitted bytes</td>
<td>252</td>
</tr>
<tr>
<td>rx_dropped</td>
<td>Number of packets dropped by RX</td>
<td>0</td>
</tr>
<tr>
<td>tx_dropped</td>
<td>Number of packets dropped by TX</td>
<td>0</td>
</tr>
<tr>
<td>rx_errors</td>
<td>Number of receive errors</td>
<td>0</td>
</tr>
<tr>
<td>tx_errors</td>
<td>Number of transmit errors</td>
<td>0</td>
</tr>
<tr>
<td>rx_frame_err</td>
<td>Number of frame alignment errors</td>
<td>0</td>
</tr>
<tr>
<td>rx_over_err</td>
<td>Number of packets with RX overrun</td>
<td>0</td>
</tr>
<tr>
<td>rx_crc_err</td>
<td>Number of CRC errors</td>
<td>0</td>
</tr>
<tr>
<td>collisions</td>
<td>Number of collisions</td>
<td>0</td>
</tr>
<tr>
<td>duration_sec</td>
<td>Time port has been alive in seconds</td>
<td>12</td>
</tr>
<tr>
<td>duration_nsec</td>
<td>Time port has been alive in nanoseconds beyond duration_sec</td>
<td>9.76e+08</td>
</tr>
</tbody>
</table>

Response message body(OpenFlow1.4 or later):
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>dpid</td>
<td>Datapath ID</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>port_no</td>
<td>Port number</td>
<td>1</td>
</tr>
<tr>
<td>rx_packets</td>
<td>Number of received packets</td>
<td>9</td>
</tr>
<tr>
<td>tx_packets</td>
<td>Number of transmitted packets</td>
<td>6</td>
</tr>
<tr>
<td>rx_bytes</td>
<td>Number of received bytes</td>
<td>738</td>
</tr>
<tr>
<td>tx_bytes</td>
<td>Number of transmitted bytes</td>
<td>252</td>
</tr>
<tr>
<td>rx_dropped</td>
<td>Number of packets dropped by RX</td>
<td>0</td>
</tr>
<tr>
<td>tx_dropped</td>
<td>Number of packets dropped by TX</td>
<td>0</td>
</tr>
<tr>
<td>rx_errors</td>
<td>Number of receive errors</td>
<td>0</td>
</tr>
<tr>
<td>tx_errors</td>
<td>Number of transmit errors</td>
<td>0</td>
</tr>
<tr>
<td>duration_sec</td>
<td>Time port has been alive in seconds</td>
<td>12</td>
</tr>
<tr>
<td>duration_nsec</td>
<td>Time port has been alive in nanoseconds beyond duration_sec</td>
<td>9.76e+08</td>
</tr>
</tbody>
</table>

Properties:
```
struct ofp_port_desc_prop_header
```
```
["rx_frame_err": 0, "rx_over_err": 0, "rx_crc_err": 0, "collisions": 0,...],...
```

Example of use:
```
$ curl -X GET http://localhost:8080/stats/port/1
```

Response (OpenFlow1.3 or earlier):
```
{
   "1": [
      {
         "port_no": 1,
         "rx_packets": 9,
         "tx_packets": 6,
         "rx_bytes": 738,
         "tx_bytes": 252,
         "rx_dropped": 0,
         "tx_dropped": 0,
         "rx_errors": 0,
         "tx_errors": 0,
         "rx_frame_err": 0,
         "rx_over_err": 0,
         "rx_crc_err": 0,
         "collisions": 0,
         "duration_sec": 12,
         "duration_nsec": 9.76e+08
      },
      {
         
      }
   ]
}
```

Response (OpenFlow1.4 or later):
Get ports description

Get ports description of the switch which specified with Datapath ID in URI.

Usage (OpenFlow1.4 or earlier):

6.2. ryu.app.ofctl_rest 531
Method | GET  
---|---
URI  | /stats/portdesc/<dpid>

Usage (OpenFlow 1.5 or later):

Method | GET  
---|---
URI  | /stats/portdesc/<dpid>/{<port>}

**Note:** Specification of port number is optional.

Response message body (OpenFlow 1.3 or earlier):

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>dpid</td>
<td>Datapath ID</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>port_no</td>
<td>Port number</td>
<td>1</td>
</tr>
<tr>
<td>hw_addr</td>
<td>Ethernet hardware address</td>
<td>&quot;0a:b6:d0:0c:e1:d7&quot;</td>
</tr>
<tr>
<td>name</td>
<td>Name of port</td>
<td>&quot;s1-eth1&quot;</td>
</tr>
<tr>
<td>config</td>
<td>Bitmap of OFPPC_* flags</td>
<td>0</td>
</tr>
<tr>
<td>state</td>
<td>Bitmap of OFPPS_* flags</td>
<td>0</td>
</tr>
<tr>
<td>curr</td>
<td>Current features</td>
<td>2112</td>
</tr>
<tr>
<td>advertised</td>
<td>Features being advertised by the port</td>
<td>0</td>
</tr>
<tr>
<td>supported</td>
<td>Features supported by the port</td>
<td>0</td>
</tr>
<tr>
<td>peer</td>
<td>Features advertised by peer</td>
<td>0</td>
</tr>
<tr>
<td>curr_speed</td>
<td>Current port bitrate in kbps</td>
<td>1e+07</td>
</tr>
<tr>
<td>max_speed</td>
<td>Max port bitrate in kbps</td>
<td>0</td>
</tr>
</tbody>
</table>

Response message body (OpenFlow 1.4 or later):

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>dpid</td>
<td>Datapath ID</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>port_no</td>
<td>Port number</td>
<td>1</td>
</tr>
<tr>
<td>hw_addr</td>
<td>Ethernet hardware address</td>
<td>&quot;0a:b6:d0:0c:e1:d7&quot;</td>
</tr>
<tr>
<td>name</td>
<td>Name of port</td>
<td>&quot;s1-eth1&quot;</td>
</tr>
<tr>
<td>config</td>
<td>Bitmap of OFPPC_* flags</td>
<td>0</td>
</tr>
<tr>
<td>state</td>
<td>Bitmap of OFPPS_* flags</td>
<td>0</td>
</tr>
<tr>
<td>length</td>
<td>Length of this entry</td>
<td>168</td>
</tr>
<tr>
<td>properties</td>
<td>struct ofp_port_desc_prop_header</td>
<td>[{&quot;length&quot;: 32, &quot;curr&quot;: 10248,...}]</td>
</tr>
</tbody>
</table>

Example of use:

$ curl -X GET http://localhost:8080/stats/portdesc/1

Response (OpenFlow 1.3 or earlier):

```json
{
  "1": [
    {
      "port_no": 1,
      "hw_addr": "0a:b6:d0:0c:e1:d7",
      (continues on next page)
    }
  ]
}
```
Response (OpenFlow 1.4 or later):

```json
{
  "1": [
    {
      "port_no": 1,
      "hw_addr": "0a:b6:d0:0c:e1:d7",
      "name": "s1-eth1",
      "config": 0,
      "state": 0,
      "length": 168,
      "properties": [
        {
          "length": 32,
          "curr": 10248,
          "advertised": 10240,
          "supported": 10248,
          "peer": 10248,
          "curr_speed": 5000,
          "max_speed": 5000,
          "type": "ETHERNET"
        },
        {
          "length": 40,
          "rx_grid_freq_lmda": 1500,
          "tx_grid_freq_lmda": 1500,
          "rx_max_freq_lmda": 2000,
          "tx_max_freq_lmda": 2000,
          "rx_min_freq_lmda": 1000,
          "tx_min_freq_lmda": 1000,
          "tx_pwr_max": 2000,
          "tx_pwr_min": 1000,
          "supported": 1,
          "type": "OPTICAL"
        },
        {
          "data": [],
          "exp_type": 0,
          "experimenter": 101,
          "length": 12,
```

(continues on next page)
"type": "EXPERIMENTER",
),
{
 :
 :

"

Get queues stats

Get queues stats of the switch which specified with Datapath ID in URI.

Usage:

<table>
<thead>
<tr>
<th>Method</th>
<th>GET</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
<td>/stats/queue/&lt;dpid&gt;[/&lt;port&gt;[/&lt;queue_id&gt;]]</td>
</tr>
</tbody>
</table>

**Note:** Specification of port number and queue id are optional.

If you want to omitting the port number and setting the queue id, please specify the keyword "ALL" to the port number.

e.g. GET http://localhost:8080/stats/queue/1/ALL/1

Response message body(OpenFlow1.3 or earlier):

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>dpid</td>
<td>Datapath ID</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>port_no</td>
<td>Port number</td>
<td>1</td>
</tr>
<tr>
<td>queue_id</td>
<td>Queue ID</td>
<td>0</td>
</tr>
<tr>
<td>tx_bytes</td>
<td>Number of transmitted bytes</td>
<td>0</td>
</tr>
<tr>
<td>tx_packets</td>
<td>Number of transmitted packets</td>
<td>0</td>
</tr>
<tr>
<td>tx_errors</td>
<td>Number of packets dropped due to overrun</td>
<td>0</td>
</tr>
<tr>
<td>duration_sec</td>
<td>Time queue has been alive in seconds</td>
<td>4294963425</td>
</tr>
<tr>
<td>duration_nsec</td>
<td>Time queue has been alive in nanoseconds beyond duration_sec</td>
<td>3912967296</td>
</tr>
</tbody>
</table>

Response message body(OpenFlow1.4 or later):
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>dpid</td>
<td>Datapath ID</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>port_no</td>
<td>Port number</td>
<td>1</td>
</tr>
<tr>
<td>queue_id</td>
<td>Queue ID</td>
<td>0</td>
</tr>
<tr>
<td>tx_bytes</td>
<td>Number of transmitted bytes</td>
<td>0</td>
</tr>
<tr>
<td>tx_packets</td>
<td>Number of transmitted packets</td>
<td>0</td>
</tr>
<tr>
<td>tx_errors</td>
<td>Number of packets dropped due to overrun</td>
<td>0</td>
</tr>
<tr>
<td>duration_sec</td>
<td>Time queue has been alive in seconds</td>
<td>4294963425</td>
</tr>
<tr>
<td>duration_nsec</td>
<td>Time queue has been alive in nanoseconds beyond duration_sec</td>
<td>3912967296</td>
</tr>
<tr>
<td>length</td>
<td>Length of this entry</td>
<td>104</td>
</tr>
<tr>
<td>properties</td>
<td>struct ofp_queue_stats_prop_header</td>
<td></td>
</tr>
</tbody>
</table>

Example of use:

```bash
$ curl -X GET http://localhost:8080/stats/queue/1
```

Response (OpenFlow1.3 or earlier):

```
{
  "1": [
    {
      "port_no": 1,
      "queue_id": 0,
      "tx_bytes": 0,
      "tx_packets": 0,
      "tx_errors": 0,
      "duration_sec": 4294963425,
      "duration_nsec": 3912967296
    },
    {
      "port_no": 1,
      "queue_id": 1,
      "tx_bytes": 0,
      "tx_packets": 0,
      "tx_errors": 0,
      "duration_sec": 4294963425,
      "duration_nsec": 3912967296
    }
  ]
}
```

Response (OpenFlow1.4 or later):

```
{
  "1": [
    {
      "port_no": 1,
      "queue_id": 0,
      "tx_bytes": 0,
      "tx_packets": 0,
      "tx_errors": 0,
      "duration_sec": 4294963425,
      "duration_nsec": 3912967296
    }
  ]
}
```

(continues on next page)
"duration_nsec": 3912967296,
"length": 104,
"properties": [
  {
    "OFPQueueStatsPropExperimenter": {
      "type": 65535,
      "length": 16,
      "data": [1],
      "exp_type": 1,
      "experimenter": 101
    }
  }
],
{
  "port_no": 2,
  "queue_id": 1,
  "tx_bytes": 0,
  "tx_packets": 0,
  "tx_errors": 0,
  "duration_sec": 4294967295,
  "duration_nsec": 3912967296,
  "length": 48,
  "properties": []
}
]

Get queues config

Get queues config of the switch which specified with Datapath ID and Port in URI.

Usage:

<table>
<thead>
<tr>
<th>Method</th>
<th>GET</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
<td>/stats/queueconfig/&lt;dpid&gt;[/&lt;port&gt;]</td>
</tr>
</tbody>
</table>

Note: Specification of port number is optional.

Caution: This message is deprecated in Openflow1.4. If OpenFlow 1.4 or later is in use, please refer to Get queues description instead.

Response message body:
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>dpid</td>
<td>Datapath ID</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>port</td>
<td>Port which was queried</td>
<td>1</td>
</tr>
<tr>
<td>queues</td>
<td>struct ofp_packet_queue</td>
<td></td>
</tr>
<tr>
<td>queue_id</td>
<td>ID for the specific queue</td>
<td>2</td>
</tr>
<tr>
<td>port</td>
<td>Port this queue is attached to</td>
<td>0</td>
</tr>
<tr>
<td>properties</td>
<td>struct ofp_queue_prop_header properties</td>
<td>[&quot;property&quot;: &quot;MIN_RATE&quot;,&quot;rate&quot;: 80]</td>
</tr>
</tbody>
</table>

Example of use:

```bash
$ curl -X GET http://localhost:8080/stats/queueconfig/1/1
```

```json
{
    "1": [
        {
            "port": 1,
            "queues": [
                {
                    "properties": [
                        {
                            "property": "MIN_RATE",
                            "rate": 80
                        }
                    ],
                    "port": 0,
                    "queue_id": 1
                },
                {
                    "properties": [
                        {
                            "property": "MAX_RATE",
                            "rate": 120
                        }
                    ],
                    "port": 2,
                    "queue_id": 2
                },
                {
                    "properties": [
                        {
                            "property": "EXPERIMENTER",
                            "data": [],
                            "experimenter": 999
                        }
                    ],
                    "port": 3,
                    "queue_id": 3
                }
            ]
        }
    ]
}
```
Get queues description

Get queues description of the switch which specified with Datapath ID, Port and Queue_id in URI.

Usage:

<table>
<thead>
<tr>
<th>Method</th>
<th>GET</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
<td>/stats/queuedesc/&lt;dpid&gt;[/&lt;port&gt;[/&lt;queue_id&gt;]]</td>
</tr>
</tbody>
</table>

Note: Specification of port number and queue id are optional.

If you want to omitting the port number and setting the queue id, please specify the keyword "ALL" to the port number.

e.g. GET http://localhost:8080/stats/queuedesc/1/ALL/1

Caution: This message is available in OpenFlow1.4 or later. If Openflow1.3 or earlier is in use, please refer to Get queues config instead.

Response message body:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>dpid</td>
<td>Datapath ID</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>len</td>
<td>Length in bytes of this queue desc</td>
<td>88</td>
</tr>
<tr>
<td>port_no</td>
<td>Port which was queried</td>
<td>1</td>
</tr>
<tr>
<td>queue_id</td>
<td>Queue ID</td>
<td>1</td>
</tr>
<tr>
<td>properties</td>
<td>struct ofp_queue_desc_prop_header</td>
<td>[{&quot;length&quot;: 8, ...}, ...]</td>
</tr>
</tbody>
</table>

Example of use:

$ curl -X GET http://localhost:8080/stats/queuedesc/1/1/1

```
{
  "1": [
    {
      "len": 88,
      "port_no": 1,
      "queue_id": 1,
      "properties": [
        {
          "length": 8,
          "rate": 300,
          "type": "MIN_RATE"
        },
        {
          "length": 8,
          "rate": 900,
          "type": "MAX_RATE"
        },
        {
          "length": 16,
          "exp_type": 0,
```

(continues on next page)
Get groups stats

Get groups stats of the switch which specified with Datapath ID in URI.

Usage:

<table>
<thead>
<tr>
<th>Method</th>
<th>URI</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>/stats/group/&lt;dpid&gt;</td>
</tr>
<tr>
<td></td>
<td>[/&lt;group_id&gt;]</td>
</tr>
</tbody>
</table>

**Note:** Specification of group id is optional.

Response message body:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>dpid</td>
<td>Datapath ID</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>length</td>
<td>Length of this entry</td>
<td>56</td>
</tr>
<tr>
<td>group_id</td>
<td>Group ID</td>
<td>1</td>
</tr>
<tr>
<td>ref_count</td>
<td>Number of flows or groups that directly forward to this group</td>
<td>1</td>
</tr>
<tr>
<td>packet_count</td>
<td>Number of packets processed by group</td>
<td>0</td>
</tr>
<tr>
<td>byte_count</td>
<td>Number of bytes processed by group</td>
<td>0</td>
</tr>
<tr>
<td>duration_sec</td>
<td>Time group has been alive in seconds</td>
<td>161</td>
</tr>
<tr>
<td>duration_nsec</td>
<td>Time group has been alive in nanoseconds beyond duration_sec</td>
<td>3.03e+08</td>
</tr>
<tr>
<td>bucket_stats</td>
<td>struct ofp_bucket_counter</td>
<td></td>
</tr>
<tr>
<td>– packet_count</td>
<td>Number of packets processed by bucket</td>
<td>0</td>
</tr>
<tr>
<td>– byte_count</td>
<td>Number of bytes processed by bucket</td>
<td>0</td>
</tr>
</tbody>
</table>

Example of use:

```
$ curl -X GET http://localhost:8080/stats/group/1
```
Get group description stats

Get group description stats of the switch which specified with Datapath ID in URI.

Usage (Openflow1.4 or earlier):

<table>
<thead>
<tr>
<th>Method</th>
<th>GET</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
<td>/stats/groupdesc/&lt;dpid&gt;</td>
</tr>
</tbody>
</table>

Usage (Openflow1.5 or later):

<table>
<thead>
<tr>
<th>Method</th>
<th>GET</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
<td>/stats/groupdesc/&lt;dpid&gt;/&lt;group_id&gt;</td>
</tr>
</tbody>
</table>

**Note:** Specification of group id is optional.

Response message body (Openflow1.3 or earlier):

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>dpid</td>
<td>Datapath ID</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>type</td>
<td>One of OFPGT_*</td>
<td>&quot;ALL&quot;</td>
</tr>
<tr>
<td>group_id</td>
<td>Group ID</td>
<td>1</td>
</tr>
<tr>
<td>buckets</td>
<td>struct ofp_bucket</td>
<td></td>
</tr>
<tr>
<td>– weight</td>
<td>Relative weight of bucket (Only defined for select groups)</td>
<td>0</td>
</tr>
<tr>
<td>– watch_port</td>
<td>Port whose state affects whether this bucket is live (Only required for fast failover groups)</td>
<td>4294967295</td>
</tr>
<tr>
<td>– watch_group</td>
<td>Group whose state affects whether this bucket is live (Only required for fast failover groups)</td>
<td>4294967295</td>
</tr>
<tr>
<td>– actions</td>
<td>0 or more actions associated with the bucket</td>
<td>[&quot;OUTPUT:1&quot;]</td>
</tr>
</tbody>
</table>
Response message body (Openflow1.4 or later):

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>dpid</td>
<td>Datapath ID</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>type</td>
<td>One of OFPGT_*</td>
<td>&quot;ALL&quot;</td>
</tr>
<tr>
<td>group_id</td>
<td>Group ID</td>
<td>1</td>
</tr>
<tr>
<td>length</td>
<td>Length of this entry</td>
<td>40</td>
</tr>
<tr>
<td>buckets</td>
<td>struct ofp_bucket</td>
<td></td>
</tr>
<tr>
<td>– weight</td>
<td>Relative weight of bucket (Only defined for select groups)</td>
<td>0</td>
</tr>
<tr>
<td>– watch_port</td>
<td>Port whose state affects whether this bucket is live (Only required for fast failover groups)</td>
<td>4294967295</td>
</tr>
<tr>
<td>– watch_group</td>
<td>Group whose state affects whether this bucket is live (Only required for fast failover groups)</td>
<td>4294967295</td>
</tr>
<tr>
<td>– len</td>
<td>Length the bucket in bytes, including this header and any adding to make it 64-bit aligned.</td>
<td>32</td>
</tr>
<tr>
<td>– actions</td>
<td>0 or more actions associated with the bucket</td>
<td></td>
</tr>
</tbody>
</table>

   

   

Example of use:

```
$ curl -X GET http://localhost:8080/stats/groupdesc/1
```

Response (Openflow1.3 or earlier):

```
{
   "1": [
   {
      "type": "ALL",
      "group_id": 1,
      "buckets": [
      {
         "weight": 0,
         "watch_port": 4294967295,
         "watch_group": 4294967295,
         "actions": ["OUTPUT:1"]
      }
   }
   ]
}
```

Response (Openflow1.4 or later):

```
{
   "1": [
   {
      "type": "ALL",
      "group_id": 1,
      "length": 40,
      "buckets": [
      ]
   }
```

(continues on next page)
Get group features stats

Get group features stats of the switch which specified with Datapath ID in URI.

Usage:

<table>
<thead>
<tr>
<th>Method</th>
<th>GET</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
<td>/stats/groupfeatures/&lt;dpid&gt;</td>
</tr>
</tbody>
</table>

Response message body:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>dpid</td>
<td>Datapath ID</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>types</td>
<td>Bitmap of (1 &lt;&lt; OFPGT_*)</td>
<td>[]</td>
</tr>
<tr>
<td>capabilities</td>
<td>Bitmap of OFPGFC_* capability supported</td>
<td>[&quot;SELECT_WEIGHT&quot;,&quot;SELECT_LIVENESS&quot;,&quot;CHAINING&quot;]</td>
</tr>
<tr>
<td>max_groups</td>
<td>Maximum number of groups for each type</td>
<td>[&quot;ALL&quot;: 4294967040],...</td>
</tr>
<tr>
<td>actions</td>
<td>Bitmaps of (1 &lt;&lt; OFPAT_*)</td>
<td>[&quot;ALL&quot;: [&quot;OUTPUT&quot;],...],...</td>
</tr>
</tbody>
</table>

Example of use:

$ curl -X GET http://localhost:8080/stats/groupfeatures/1

```
{
    "1": [
    {
        "types": [],
        "capabilities": [
            "SELECT_WEIGHT",
            "SELECT_LIVENESS",
            "CHAINING"
        ],
        "actions": [
            {
                "type": "OUTPUT",
                "max_len": 65535,
                "port": 2
            }
        ]
    }
]
```

(continues on next page)
"SELECT_LIVENESS",
"CHAINING"
],
"max_groups": [
{
  "ALL": 4294967040
},
{
  "SELECT": 4294967040
},
{
  "INDIRECT": 4294967040
},
{
  "FF": 4294967040
}
],
"actions": [
{
  "ALL": [
    "OUTPUT",
    "COPY_TTL_OUT",
    "COPY_TTL_IN",
    "SET_MPLS_TTL",
    "DEC_MPLS_TTL",
    "PUSH_VLAN",
    "POP_VLAN",
    "PUSH_MPLS",
    "POP_MPLS",
    "SET_QUEUE",
    "GROUP",
    "SET_NW_TTL",
    "DEC_NW_TTL",
    "SET_FIELD"
  ]
},
{
  "SELECT": []
},
{
  "INDIRECT": []
},
{
  "FF": []
}
]
}
}

Get meters stats

Get meters stats of the switch which specified with Datapath ID in URI.

Usage:
Method: GET
URI: /stats/meter/<dpid>[/<meter_id>]

**Note:** Specification of meter id is optional.

Response message body:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>dpid</td>
<td>Datapath ID</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>meter_id</td>
<td>Meter ID</td>
<td>1</td>
</tr>
<tr>
<td>len</td>
<td>Length in bytes of this stats</td>
<td>56</td>
</tr>
<tr>
<td>flow_count</td>
<td>Number of flows bound to meter</td>
<td>0</td>
</tr>
<tr>
<td>packet_in_count</td>
<td>Number of packets in input</td>
<td>0</td>
</tr>
<tr>
<td>byte_in_count</td>
<td>Number of bytes in input</td>
<td>0</td>
</tr>
<tr>
<td>duration_sec</td>
<td>Time meter has been alive in seconds</td>
<td>37</td>
</tr>
<tr>
<td>duration_nsec</td>
<td>Time meter has been alive in nanoseconds beyond duration_sec</td>
<td>988000</td>
</tr>
<tr>
<td>band_stats</td>
<td>struct ofp_meter_band_stats</td>
<td></td>
</tr>
<tr>
<td>– packet_band_count</td>
<td>Number of packets in band</td>
<td>0</td>
</tr>
<tr>
<td>– byte_band_count</td>
<td>Number of bytes in band</td>
<td>0</td>
</tr>
</tbody>
</table>

Example of use:

```
$ curl -X GET http://localhost:8080/stats/meter/1
```

```
{
  "1": [
    {
      "meter_id": 1,
      "len": 56,
      "flow_count": 0,
      "packet_in_count": 0,
      "byte_in_count": 0,
      "duration_sec": 37,
      "duration_nsec": 988000,
      "band_stats": [
        {
          "packet_band_count": 0,
          "byte_band_count": 0
        }
      ]
    }
  ]
}
```

**Get meter config stats**
Get meter description stats

Get meter config stats of the switch which specified with Datapath ID in URI.

**Caution:** This message has been renamed in Openflow 1.5. If Openflow 1.4 or earlier is in use, please used as Get meter description stats. If Openflow 1.5 or later is in use, please used as Get meter description stats.

Usage (Openflow 1.4 or earlier):

<table>
<thead>
<tr>
<th>Method</th>
<th>GET</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
<td>/stats/meterconfig/&lt;dpid&gt;[/&lt;meter_id&gt;]</td>
</tr>
</tbody>
</table>

Usage (Openflow 1.5 or later):

<table>
<thead>
<tr>
<th>Method</th>
<th>GET</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
<td>/stats/meterdesc/&lt;dpid&gt;[/&lt;meter_id&gt;]</td>
</tr>
</tbody>
</table>

**Note:** Specification of meter id is optional.

Response message body:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>dpid</td>
<td>Datapath ID</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>flags</td>
<td>All OFPMC_* that apply</td>
<td>&quot;KBPS&quot;</td>
</tr>
<tr>
<td>meter_id</td>
<td>Meter ID</td>
<td>1</td>
</tr>
<tr>
<td>bands</td>
<td>struct ofp_meter_band_header</td>
<td></td>
</tr>
<tr>
<td>– type</td>
<td>One of OFPMBT_*</td>
<td>&quot;DROP&quot;</td>
</tr>
<tr>
<td>– rate</td>
<td>Rate for this band</td>
<td>1000</td>
</tr>
<tr>
<td>– burst_size</td>
<td>Size of bursts</td>
<td>0</td>
</tr>
</tbody>
</table>

Example of use:

```bash
$ curl -X GET http://localhost:8080/stats/meterconfig/1
```

```json
{
  "1": [
    {
      "flags": [
        "KBPS"
      ],
      "meter_id": 1,
      "bands": [
        {
          "type": "DROP",
          "rate": 1000,
          "burst_size": 0
        }
      ]
    }
  ]
}
```

(continues on next page)
Get meter features stats

Get meter features stats of the switch which specified with Datapath ID in URI.

Usage:

<table>
<thead>
<tr>
<th>Method</th>
<th>GET</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
<td>/stats/meterfeatures/&lt;dpid&gt;</td>
</tr>
</tbody>
</table>

Response message body:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>dpid</td>
<td>Datapath ID</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>max_meter</td>
<td>Maximum number of meters</td>
<td>256</td>
</tr>
<tr>
<td>band_types</td>
<td>Bitmaps of (1 &lt;&lt; OFPMBT_*) values supported</td>
<td>[&quot;DROP&quot;]</td>
</tr>
<tr>
<td>capabilities</td>
<td>Bitmaps of &quot;ofp_meter_flags&quot;</td>
<td>[&quot;KBPS&quot;, &quot;BURST&quot;, &quot;STATS&quot;]</td>
</tr>
<tr>
<td>max_bands</td>
<td>Maximum bands per meters</td>
<td>16</td>
</tr>
<tr>
<td>max_color</td>
<td>Maximum color value</td>
<td>8</td>
</tr>
</tbody>
</table>

Example of use:

```
$ curl -X GET http://localhost:8080/stats/meterfeatures/1
```

```
{
    "1": [
        {
            "max_meter": 256,
            "band_types": ["DROP"],
            "capabilities": ["KBPS", "BURST", "STATS"],
            "max_bands": 16,
            "max_color": 8
        }
    ]
}
```

Get role

Get the current role of the controller from the switch.

Usage:
**Method**
GET

**URI**
/stats/role/<dpid>

Response message body (Openflow 1.4 or earlier):

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>dpid</td>
<td>Datapath ID</td>
<td>1</td>
</tr>
<tr>
<td>role</td>
<td>One of OFPCR_ROLE_*</td>
<td>&quot;EQUAL&quot;</td>
</tr>
<tr>
<td>generation_id</td>
<td>Master Election Generation Id</td>
<td>0</td>
</tr>
</tbody>
</table>

Response message body (Openflow 1.5 or later):

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>dpid</td>
<td>Datapath ID</td>
<td>1</td>
</tr>
<tr>
<td>role</td>
<td>One of OFPCR_ROLE_*</td>
<td>&quot;EQUAL&quot;</td>
</tr>
<tr>
<td>short_id</td>
<td>ID number for the controller</td>
<td>0</td>
</tr>
<tr>
<td>generation_id</td>
<td>Master Election Generation Id</td>
<td>0</td>
</tr>
</tbody>
</table>

Example of use:

```
$ curl -X GET http://localhost:8080/stats/role/1
```

Response (Openflow 1.4 or earlier):

```json
{
   "1": [
      {
         "generation_id": 0,
         "role": "EQUAL"
      }
   ]
}
```

Response (Openflow 1.5 or later):

```json
{
   "1": [
      {
         "generation_id": 0,
         "role": "EQUAL",
         "short_id": 0
      }
   ]
}
```

### 6.2.2 Update the switch stats

#### Add a flow entry

Add a flow entry to the switch.

**Usage:**

```bash
6.2. ryu.app.ofctl_rest
```
Method | POST
---|---
URI | /stats/flowentry/add

Request message body (Openflow1.3 or earlier):

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Example</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>dpid</td>
<td>Datapath ID (int)</td>
<td>1</td>
<td>(Mandatory)</td>
</tr>
<tr>
<td>cookie</td>
<td>Opaque controller-issued identifier  (int)</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>cookie_mask</td>
<td>Mask used to restrict the cookie bits (int)</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>table_id</td>
<td>Table ID to put the flow in (int)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>idle_timeout</td>
<td>Idle time before discarding (seconds) (int)</td>
<td>30</td>
<td>0</td>
</tr>
<tr>
<td>hard_timeout</td>
<td>Max time before discarding (seconds) (int)</td>
<td>30</td>
<td>0</td>
</tr>
<tr>
<td>priority</td>
<td>Priority level of flow entry (int)</td>
<td>11111</td>
<td>0</td>
</tr>
<tr>
<td>buffer_id</td>
<td>Buffered packet to apply to, or OFP_NO_BUFFER (int)</td>
<td>1</td>
<td>OFP_NO_BUFFER</td>
</tr>
<tr>
<td>flags</td>
<td>Bitmap of OFPFF_* flags (int)</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>match</td>
<td>Fields to match (dict)</td>
<td>{}</td>
<td>#wild-carded</td>
</tr>
<tr>
<td>actions</td>
<td>Instruction set (list of dict)</td>
<td>[]</td>
<td>#DROP</td>
</tr>
</tbody>
</table>

Request message body (Openflow1.4 or later):

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Example</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>dpid</td>
<td>Datapath ID (int)</td>
<td>1</td>
<td>(Mandatory)</td>
</tr>
<tr>
<td>cookie</td>
<td>Opaque controller-issued identifier  (int)</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>cookie_mask</td>
<td>Mask used to restrict the cookie bits (int)</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>table_id</td>
<td>Table ID to put the flow in (int)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>idle_timeout</td>
<td>Idle time before discarding (seconds) (int)</td>
<td>30</td>
<td>0</td>
</tr>
<tr>
<td>hard_timeout</td>
<td>Max time before discarding (seconds) (int)</td>
<td>30</td>
<td>0</td>
</tr>
<tr>
<td>priority</td>
<td>Priority level of flow entry (int)</td>
<td>11111</td>
<td>0</td>
</tr>
<tr>
<td>buffer_id</td>
<td>Buffered packet to apply to, or OFP_NO_BUFFER (int)</td>
<td>1</td>
<td>OFP_NO_BUFFER</td>
</tr>
<tr>
<td>flags</td>
<td>Bitmap of OFPFF_* flags (int)</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>match</td>
<td>Fields to match (dict)</td>
<td>{}</td>
<td>#wild-carded</td>
</tr>
<tr>
<td>instructions</td>
<td>Instruction set (list of dict)</td>
<td>[]</td>
<td>#DROP</td>
</tr>
</tbody>
</table>
Note: For description of match and actions, please see Reference: Description of Match and Actions.

Example of use (Openflow1.3 or earlier):

```bash
$ curl -X POST -d '{
    "dpid": 1,
    "cookie": 1,
    "cookie_mask": 1,
    "table_id": 0,
    "idle_timeout": 30,
    "hard_timeout": 30,
    "priority": 11111,
    "flags": 1,
    "match":{
        "in_port":1
    },
    "actions":[
    {
        "type":"OUTPUT",
        "port": 2
    }
    ]
}' http://localhost:8080/stats/flowentry/add

$ curl -X POST -d '{
    "dpid": 1,
    "priority": 22222,
    "match":{
        "in_port":1
    },
    "actions":[
    {
        "type":"GOTO_TABLE",
        "table_id": 1
    }
    ]
}' http://localhost:8080/stats/flowentry/add

$ curl -X POST -d '{
    "dpid": 1,
    "priority": 33333,
    "match":{
        "in_port":1
    },
    "actions":[
    {
        "type":"WRITE_METADATA",
        "metadata": 1,
        "metadata_mask": 1
    }
    ]
}' http://localhost:8080/stats/flowentry/add

$ curl -X POST -d '{
    "dpid": 1,
    "cookie": 1,
    "cookie_mask": 1,
    "table_id": 0,
    "idle_timeout": 30,
    "hard_timeout": 30,
    "priority": 11111,
    "flags": 1,
    "match":{
        "in_port":1
    },
    "actions":[
    {
        "type":"OUTPUT",
        "port": 2
    }
    ]
}' http://localhost:8080/stats/flowentry/add
```

(continues on next page)
"priority": 44444,
"match": {
    "in_port": 1
},
"actions": [
    {
        "type": "METER",
        "meter_id": 1
    }
]}
}' http://localhost:8080/stats/flowentry/add

Example of use (Openflow 1.4 or later):

$ curl -X POST -d '{
    "dpid": 1,
    "cookie": 1,
    "cookie_mask": 1,
    "table_id": 0,
    "idle_timeout": 30,
    "hard_timeout": 30,
    "priority": 11111,
    "flags": 1,
    "match": {
        "in_port": 1
    },
    "instructions": [
        {
            "type": "APPLY_ACTIONS",
            "actions": [
                {
                    "max_len": 65535,
                    "port": 2,
                    "type": "OUTPUT"
                }
            ]
        }
    ]
}' http://localhost:8080/stats/flowentry/add

$ curl -X POST -d '{
    "dpid": 1,
    "priority": 22222,
    "match": {
        "in_port": 1
    },
    "instructions": [
        {
            "type": "GOTO_TABLE",
            "table_id": 1
        }
    ]
}' http://localhost:8080/stats/flowentry/add

$ curl -X POST -d '{
    "dpid": 1,
    "priority": 22222,
    "match": {
        "in_port": 1
    },
    "instructions": [
        {
            "type": "GOTO_TABLE",
            "table_id": 1
        }
    ]
}' http://localhost:8080/stats/flowentry/add

(continues on next page)
"priority": 33333,
"match":{
   "in_port":1
},
"instructions": [
   {
      "type":"WRITE_METADATA",
      "metadata": 1,
      "metadata_mask": 1
   }
]}' http://localhost:8080/stats/flowentry/add

$ curl -X POST -d '{
   "dpid": 1,
   "priority": 44444,
   "match":{
      "in_port":1
   },
   "instructions": [
      {
         "type": "METER",
         "meter_id": 1
      }
   ]}' http://localhost:8080/stats/flowentry/add

**Note:** To confirm flow entry registration, please see *Get all flows stats* or *Get flows stats filtered by fields*.

**Modify all matching flow entries**

Modify all matching flow entries of the switch.

Usage:

<table>
<thead>
<tr>
<th>Method</th>
<th>POST</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
<td>/stats/flowentry/modify</td>
</tr>
</tbody>
</table>

Request message body:
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Example</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>dpid</td>
<td>Datapath ID (int)</td>
<td>1</td>
<td>(Mandatory)</td>
</tr>
<tr>
<td>cookie</td>
<td>Opaque controller-issued identifier (int)</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>cookie_mask</td>
<td>Mask used to restrict the cookie bits (int)</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>table_id</td>
<td>Table ID to put the flow in (int)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>idle_timeout</td>
<td>Idle time before discarding (seconds) (int)</td>
<td>30</td>
<td>0</td>
</tr>
<tr>
<td>hard_timeout</td>
<td>Max time before discarding (seconds) (int)</td>
<td>30</td>
<td>0</td>
</tr>
<tr>
<td>priority</td>
<td>Priority level of flow entry (int)</td>
<td>11111</td>
<td>0</td>
</tr>
<tr>
<td>buffer_id</td>
<td>Buffered packet to apply to, or OFP_NO_BUFFER (int)</td>
<td>1</td>
<td>OFP_NO_BUFFER</td>
</tr>
<tr>
<td>flags</td>
<td>Bitmap of OFPFF_* flags (int)</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>match</td>
<td>Fields to match (dict)</td>
<td>{&quot;in_port&quot;:1}</td>
<td>{} #wild-carded</td>
</tr>
<tr>
<td>actions</td>
<td>Instruction set (list of dict)</td>
<td>[{&quot;type&quot;:&quot;OUTPUT&quot;,&quot;port&quot;:2}]</td>
<td>[] #DROP</td>
</tr>
</tbody>
</table>

Example of use:

```
$ curl -X POST -d '{
   "dpid": 1,
   "cookie": 1,
   "cookie_mask": 1,
   "table_id": 0,
   "idle_timeout": 30,
   "hard_timeout": 30,
   "priority": 11111,
   "flags": 1,
   "match":{
      "in_port":1
   },
   "actions":{
      {
         "type":"OUTPUT",
         "port": 2
      }
   }
}' http://localhost:8080/stats/flowentry/modify
```

**Modify flow entry strictly**

Modify flow entry strictly matching wildcards and priority

**Usage:**

```
<table>
<thead>
<tr>
<th>Method</th>
<th>POST</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
<td>/stats/flowentry/modify_strict</td>
</tr>
</tbody>
</table>
```

Request message body:
At- 
tribute   | Description                                      | Example     | Default               |
----------|--------------------------------------------------|-------------|-----------------------|
  dpid     | Datapath ID (int)                                | 1           | (Mandatory)           |
  cookie   | Opaque controller-issued identifier (int)        | 1           | 0                     |
  cookie_mask | Mask used to restrict the cookie bits (int)   | 1           | 0                     |
  table_id | Table ID to put the flow in (int)               | 0           | 0                     |
  idle_timeout | Idle time before discarding (seconds) (int) | 30          | 0                     |
  hard_timeout | Max time before discarding (seconds) (int) | 30          | 0                     |
  priority | Priority level of flow entry (int)              | 11111       | 0                     |
  buffer_id | Buffered packet to apply to, or OFP_NO_BUFFER (int) | 1          | OFP_NO_BUFFER         |
  flags    | Bitmap of OFPFF_* flags (int)                   | 1           | 0                     |
  match    | Fields to match (dict)                          | {}          | #wild-carded          |
  actions  | Instruction set (list of dict)                  | []          | #DROP                 |

Example of use:

```
$ curl -X POST -d '{
    "dpid": 1,
    "cookie": 1,
    "cookie_mask": 1,
    "table_id": 0,
    "idle_timeout": 30,
    "hard_timeout": 30,
    "priority": 11111,
    "flags": 1,
    "match":{
        "in_port":1
    },
    "actions":[
        {
            "type":"OUTPUT",
            "port": 2
        }
    ]
}' http://localhost:8080/stats/flowentry/modify_strict
```

Delete all matching flow entries

Delete all matching flow entries of the switch.

Usage:

<table>
<thead>
<tr>
<th>Method</th>
<th>POST URI</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST</td>
<td>/stats/flowentry/delete</td>
</tr>
</tbody>
</table>
Request message body:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Example</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>dpid</td>
<td>Datapath ID (int)</td>
<td>1</td>
<td>(Mandatory)</td>
</tr>
<tr>
<td>cookie</td>
<td>Opaque controller-issued identifier (int)</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>cookie_mask</td>
<td>Mask used to restrict the cookie bits (int)</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>table_id</td>
<td>Table ID to put the flow in (int)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>idle_timeout</td>
<td>Idle time before discarding (seconds) (int)</td>
<td>30</td>
<td>0</td>
</tr>
<tr>
<td>hard_timeout</td>
<td>Max time before discarding (seconds) (int)</td>
<td>30</td>
<td>0</td>
</tr>
<tr>
<td>priority</td>
<td>Priority level of flow entry (int)</td>
<td>11111</td>
<td>0</td>
</tr>
<tr>
<td>buffer_id</td>
<td>Buffered packet to apply to, or OFP_NO_BUFFER (int)</td>
<td>1</td>
<td>OFP_NO_BUFFER</td>
</tr>
<tr>
<td>out_port</td>
<td>Output port (int)</td>
<td>1</td>
<td>OFPP_ANY</td>
</tr>
<tr>
<td>out_group</td>
<td>Output group (int)</td>
<td>1</td>
<td>OFPG_ANY</td>
</tr>
<tr>
<td>flags</td>
<td>Bitmap of OFPFF_* flags (int)</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>match</td>
<td>Fields to match (dict)</td>
<td>{}</td>
<td>#wild-carded</td>
</tr>
<tr>
<td>actions</td>
<td>Instruction set (list of dict)</td>
<td>[]</td>
<td>#DROP</td>
</tr>
</tbody>
</table>

Example of use:

```
$ curl -X POST -d '{
   "dpid": 1,
   "cookie": 1,
   "cookie_mask": 1,
   "table_id": 0,
   "idle_timeout": 30,
   "hard_timeout": 30,
   "priority": 11111,
   "flags": 1,
   "match":{
       "in_port":1
   },
   "actions":{
   }
}' http://localhost:8080/stats/flowentry/delete
```

Delete flow entry strictly

Delete flow entry strictly matching wildcards and priority.

Usage:
Method | POST
---|---
URI | /stats/flowentry/delete_strict

Request message body:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Example</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>dpid</td>
<td>Datapath ID (int)</td>
<td>1</td>
<td>(Mandatory)</td>
</tr>
<tr>
<td>cookie</td>
<td>Opaque controller-issued identifier (int)</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>cookie_mask</td>
<td>Mask used to restrict the cookie bits (int)</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>table_id</td>
<td>Table ID to put the flow in (int)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>idle_timeout</td>
<td>Idle time before discarding (seconds) (int)</td>
<td>30</td>
<td>0</td>
</tr>
<tr>
<td>hard_timeout</td>
<td>Max time before discarding (seconds) (int)</td>
<td>30</td>
<td>0</td>
</tr>
<tr>
<td>priority</td>
<td>Priority level of flow entry (int)</td>
<td>11111</td>
<td>0</td>
</tr>
<tr>
<td>buffer_id</td>
<td>Buffered packet to apply to, or OFP_NO_BUFFER (int)</td>
<td>1</td>
<td>OFP_NO_BUFFER</td>
</tr>
<tr>
<td>out_port</td>
<td>Output port (int)</td>
<td>1</td>
<td>OFPP_ANY</td>
</tr>
<tr>
<td>out_group</td>
<td>Output group (int)</td>
<td>1</td>
<td>OFPG_ANY</td>
</tr>
<tr>
<td>flags</td>
<td>Bitmap of OFPFF_ flags (int)</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>match</td>
<td>Fields to match (dict)</td>
<td>{}</td>
<td>#wildcarded</td>
</tr>
<tr>
<td>actions</td>
<td>Instruction set (list of dict)</td>
<td>[]</td>
<td>#DROP</td>
</tr>
</tbody>
</table>

Example of use:

```bash
$ curl -X POST -d '{
  "dpid": 1,
  "cookie": 1,
  "cookie_mask": 1,
  "table_id": 0,
  "idle_timeout": 30,
  "hard_timeout": 30,
  "priority": 11111,
  "flags": 1,
  "match":{
    "in_port":1
  },
  "actions":[
    {
      "type":"OUTPUT",
      "port":2
    }
  ]
}' http://localhost:8080/stats/flowentry/delete_strict
```
Delete all flow entries

Delete all flow entries of the switch which specified with Datapath ID in URI.

Usage:

<table>
<thead>
<tr>
<th>Method</th>
<th>DELETE</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
<td>/stats/flowentry/clear/&lt;dpid&gt;</td>
</tr>
</tbody>
</table>

Example of use:

```
$ curl -X DELETE http://localhost:8080/stats/flowentry/clear/1
```

Add a group entry

Add a group entry to the switch.

Usage:

<table>
<thead>
<tr>
<th>Method</th>
<th>POST</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
<td>/stats/groupentry/add</td>
</tr>
</tbody>
</table>

Request message body:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Example</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>dpid</td>
<td>Datapath ID (int)</td>
<td>1</td>
<td>(Mandatory)</td>
</tr>
<tr>
<td>type</td>
<td>One of OFPGT_* (string)</td>
<td>&quot;ALL&quot;</td>
<td>&quot;ALL&quot;</td>
</tr>
<tr>
<td>group_id</td>
<td>Group ID (int)</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>buckets</td>
<td>struct ofp_bucket</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>weight</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>watch_port</td>
<td>4294967295</td>
<td>OFPP_ANY</td>
</tr>
<tr>
<td></td>
<td>watch_group</td>
<td>4294967295</td>
<td>OFPG_ANY</td>
</tr>
<tr>
<td></td>
<td>actions</td>
<td></td>
<td>[]</td>
</tr>
</tbody>
</table>

Example of use:

```
$ curl -X POST -d '{
    "dpid": 1,
    "type": "ALL",
    "group_id": 1,
    "buckets": [
        {
            "actions": [
                {
                    "type": "OUTPUT",
                    "port": 1
                }
            ]
        ]
    ]
}'
```

(continues on next page)
Modify a group entry

Modify a group entry to the switch.

Usage:

<table>
<thead>
<tr>
<th>Method</th>
<th>POST</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
<td>/stats/groupentry/modify</td>
</tr>
</tbody>
</table>

Request message body:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Example</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>dpid</td>
<td>Datapath ID (int)</td>
<td>1</td>
<td>(Mandatory)</td>
</tr>
<tr>
<td>type</td>
<td>One of OFPGT_* (string)</td>
<td>&quot;ALL&quot;</td>
<td>&quot;ALL&quot;</td>
</tr>
<tr>
<td>group_id</td>
<td>Group ID (int)</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>buckets</td>
<td>struct ofp_bucket</td>
<td></td>
<td></td>
</tr>
<tr>
<td>– weight</td>
<td>Relative weight of bucket</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>– watch_port</td>
<td>Port whose state affects</td>
<td>4294967295</td>
<td>OFPP_ANY</td>
</tr>
<tr>
<td>– watch_group</td>
<td>whether this bucket is live</td>
<td>4294967295</td>
<td>OFPG_ANY</td>
</tr>
<tr>
<td>– actions</td>
<td>0 or more actions associated</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>with the bucket (list of</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>dict)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example of use:

```
$ curl -X POST -d '{
    "dpid": 1,
    "type": "ALL",
    "group_id": 1,
    "buckets": [
        {
            "actions": [
                {
                    "type": "OUTPUT",
                    "port": 1
                }
            ]
        }
    ]
}' http://localhost:8080/stats/groupentry/add
```

Note: To confirm group entry registration, please see Get group description stats.
Delete a group entry

Delete a group entry to the switch.

Usage:

Request message body:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Example</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>dpid</td>
<td>Datapath ID (int)</td>
<td>1</td>
<td>(Mandatory)</td>
</tr>
<tr>
<td>group_id</td>
<td>Group ID (int)</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Example of use:

```
$ curl -X POST -d '{
  "dpid": 1,
  "group_id": 1
}' http://localhost:8080/stats/groupentry/delete
```

Modify the behavior of the port

Modify the behavior of the physical port.

Usage:

Request message body:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Example</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>dpid</td>
<td>Datapath ID (int)</td>
<td>1</td>
<td>(Mandatory)</td>
</tr>
<tr>
<td>port_no</td>
<td>Port number (int)</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>config</td>
<td>Bitmap of OFPPC_* flags (int)</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>mask</td>
<td>Bitmap of OFPPC_* flags to be changed (int)</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Example of use:

```
$ curl -X POST -d '{
  "dpid": 1,
  "port_no": 1,
  "config": 1,
  "mask": 1
}' http://localhost:8080/stats/portdesc/modify
```
Note: To confirm port description, please see Get ports description.

Add a meter entry

Add a meter entry to the switch.

Usage:

<table>
<thead>
<tr>
<th>Method</th>
<th>POST</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
<td>/stats/meterentry/add</td>
</tr>
</tbody>
</table>

Request message body:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Example</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>dpid</td>
<td>Datapath ID (int)</td>
<td>1</td>
<td>(Mandatory)</td>
</tr>
<tr>
<td>flags</td>
<td>Bitmap of OFPMF_* flags (list)</td>
<td>&quot;KBPS&quot;</td>
<td>[] #Empty</td>
</tr>
<tr>
<td>meter_id</td>
<td>Meter ID (int)</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>bands</td>
<td>struct ofp_meter_band_header</td>
<td></td>
<td></td>
</tr>
<tr>
<td>– type</td>
<td>One of OFPMBT_* (string)</td>
<td>&quot;DROP&quot;</td>
<td>None</td>
</tr>
<tr>
<td>– rate</td>
<td>Rate for this band (int)</td>
<td>1000</td>
<td>None</td>
</tr>
<tr>
<td>– burst_size</td>
<td>Size of bursts (int)</td>
<td>100</td>
<td>None</td>
</tr>
</tbody>
</table>

Example of use:

```bash
$ curl -X POST -d '{
    "dpid": 1,
    "flags": "KBPS",
    "meter_id": 1,
    "bands": [
        {
            "type": "DROP",
            "rate": 1000
        }
    ]
}' http://localhost:8080/stats/meterentry/add
```

Note: To confirm meter entry registration, please see Get meter config stats.

Modify a meter entry

Modify a meter entry to the switch.

Usage:

<table>
<thead>
<tr>
<th>Method</th>
<th>POST</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
<td>/stats/meterentry/modify</td>
</tr>
</tbody>
</table>

Request message body:
### Attribute

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Example</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>dpid</td>
<td>Datapath ID (int)</td>
<td>1</td>
<td>(Mandatory)</td>
</tr>
<tr>
<td>flags</td>
<td>Bitmap of OFPMF_* flags (list)</td>
<td>&quot;KBPS&quot;</td>
<td>[] #Empty</td>
</tr>
<tr>
<td>meter_id</td>
<td>Meter ID (int)</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>bands</td>
<td>struct ofp_meter_band_header</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>– type One of OFPMBT_* (string)</td>
<td>&quot;DROP&quot;</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>– rate Rate for this band (int)</td>
<td>1000</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>– burst_size Size of bursts (int)</td>
<td>100</td>
<td>None</td>
</tr>
</tbody>
</table>

Example of use:

```shell
$ curl -X POST -d '{
  "dpid": 1,
  "meter_id": 1,
  "flags": "KBPS",
  "bands": [
    {
      "type": "DROP",
      "rate": 1000
    }
  ]
}' http://localhost:8080/stats/meterentry/modify
```

### Delete a meter entry

Delete a meter entry to the switch.

Usage:

<table>
<thead>
<tr>
<th>Method</th>
<th>POST</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
<td>/stats/meterentry/delete</td>
</tr>
</tbody>
</table>

Request message body:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Example</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>dpid</td>
<td>Datapath ID (int)</td>
<td>1</td>
<td>(Mandatory)</td>
</tr>
<tr>
<td>meter_id</td>
<td>Meter ID (int)</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Example of use:

```shell
$ curl -X POST -d '{
  "dpid": 1,
  "meter_id": 1
}' http://localhost:8080/stats/meterentry/delete
```

### Modify role

Modify the role of the switch.

Usage:
Request message body:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Example</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>dpid</td>
<td>Datapath ID (int)</td>
<td>1</td>
<td>(Mandatory)</td>
</tr>
<tr>
<td>role</td>
<td>One of OFPCR_ROLE_*(string)</td>
<td>&quot;MASTER&quot;</td>
<td>OFPCR_ROLE_EQUAL</td>
</tr>
</tbody>
</table>

Example of use:

```
$ curl -X POST -d '{
  "dpid": 1,
  "role": "MASTER"
}' http://localhost:8080/stats/role
```

### 6.2.3 Support for experimenter multipart

**Send a experimenter message**

Send a experimenter message to the switch which specified with Datapath ID in URI.

Usage:

<table>
<thead>
<tr>
<th>Method</th>
<th>POST</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
<td>/stats/experimenter/&lt;dpid&gt;</td>
</tr>
</tbody>
</table>

Request message body:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Example</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>dpid</td>
<td>Datapath ID (int)</td>
<td>1</td>
<td>(Mandatory)</td>
</tr>
<tr>
<td>experimenter</td>
<td>Experimenter ID (int)</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>exp_type</td>
<td>Experimenter defined (int)</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>data_type</td>
<td>Data format type (&quot;ascii&quot; or &quot;base64&quot;)</td>
<td>&quot;ascii&quot;</td>
<td>&quot;ascii&quot;</td>
</tr>
<tr>
<td>data</td>
<td>Data to send (string)</td>
<td>&quot;data&quot;</td>
<td>&quot;&quot;  #Empty</td>
</tr>
</tbody>
</table>

Example of use:

```
$ curl -X POST -d '{
  "dpid": 1,
  "experimenter": 1,
  "exp_type": 1,
  "data_type": "ascii",
  "data": "data"
}' http://localhost:8080/stats/experimenter/1
```

### 6.2.4 Reference: Description of Match and Actions

**Description of Match on request messages**

List of Match fields (OpenFlow1.0):
### Match field Description Example

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>in_port</td>
<td>Input switch port (int)</td>
<td><code>{ &quot;in_port&quot;: 7 }</code></td>
</tr>
<tr>
<td>dl_src</td>
<td>Ethernet source address (string)</td>
<td><code>{ &quot;dl_src&quot;: &quot;aa:bb:cc:11:22:33&quot; }</code></td>
</tr>
<tr>
<td>dl_dst</td>
<td>Ethernet destination address (string)</td>
<td><code>{ &quot;dl_dst&quot;: &quot;aa:bb:cc:11:22:33&quot; }</code></td>
</tr>
<tr>
<td>dl_vlan</td>
<td>Input VLAN id (int)</td>
<td><code>{ &quot;dl_vlan&quot;: 5 }</code></td>
</tr>
<tr>
<td>dl_vlan_pcp</td>
<td>Input VLAN priority (int)</td>
<td><code>{ &quot;dl_vlan_pcp&quot;: 3, &quot;dl_vlan&quot;: 3 }</code></td>
</tr>
<tr>
<td>dl_type</td>
<td>Ethernet frame type (int)</td>
<td><code>{ &quot;dl_type&quot;: 123 }</code></td>
</tr>
<tr>
<td>nw_tos</td>
<td>IP ToS (int)</td>
<td><code>{ &quot;nw_tos&quot;: 16, &quot;dl_type&quot;: 2048 }</code></td>
</tr>
<tr>
<td>nw_proto</td>
<td>IP protocol or lower 8 bits of ARP opcode (int)</td>
<td><code>{ &quot;nw_proto&quot;: 5, &quot;dl_type&quot;: 2048 }</code></td>
</tr>
<tr>
<td>nw_src</td>
<td>IPv4 source address (string)</td>
<td><code>{ &quot;nw_src&quot;: &quot;192.168.0.1&quot;, &quot;dl_type&quot;: 2048 }</code></td>
</tr>
<tr>
<td>nw_dst</td>
<td>IPv4 destination address (string)</td>
<td><code>{ &quot;nw_dst&quot;: &quot;192.168.0.1/24&quot;, &quot;dl_type&quot;: 2048 }</code></td>
</tr>
<tr>
<td>tp_src</td>
<td>TCP/UDP source port (int)</td>
<td><code>{ &quot;tp_src&quot;: 1, &quot;nw_proto&quot;: 6, &quot;dl_type&quot;: 2048 }</code></td>
</tr>
<tr>
<td>tp_dst</td>
<td>TCP/UDP destination port (int)</td>
<td><code>{ &quot;tp_dst&quot;: 2, &quot;nw_proto&quot;: 6, &quot;dl_type&quot;: 2048 }</code></td>
</tr>
</tbody>
</table>

**Note:** IPv4 address field can be described as IP Prefix like as follows.

IPv4 address:

```
"192.168.0.1"
"192.168.0.2/24"
```

List of Match fields (OpenFlow1.2 or later):

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>in_port</td>
<td>Switch input port (int)</td>
<td><code>{ &quot;in_port&quot;: 7 }</code></td>
</tr>
<tr>
<td>in_phy_port</td>
<td>Switch physical input port (int)</td>
<td><code>{ &quot;in_phy_port&quot;: 5, &quot;in_port&quot;: 3 }</code></td>
</tr>
<tr>
<td>metadata</td>
<td>Metadata passed between tables (int or string)</td>
<td><code>{ &quot;metadata&quot;: 12345 }</code> or <code>{ &quot;metadata&quot;: 0x1212 }</code></td>
</tr>
<tr>
<td>eth_dst</td>
<td>Ethernet destination address (string)</td>
<td><code>{ &quot;eth_dst&quot;: &quot;aa:bb:cc:11:22:33/00:00:00:00:ff:ff&quot; }</code></td>
</tr>
<tr>
<td>eth_src</td>
<td>Ethernet source address (string)</td>
<td><code>{ &quot;eth_src&quot;: &quot;aa:bb:cc:11:22:33&quot; }</code></td>
</tr>
<tr>
<td>eth_type</td>
<td>Ethernet frame type (int)</td>
<td><code>{ &quot;eth_type&quot;: 2048 }</code></td>
</tr>
<tr>
<td>vlan_vid</td>
<td>VLAN id (int or string)</td>
<td><code>{ &quot;vlan_vid&quot;: 3 }</code></td>
</tr>
<tr>
<td>vlan_pcp</td>
<td>VLAN priority (int)</td>
<td><code>{ &quot;vlan_pcp&quot;: 3, &quot;vlan_vid&quot;: 3 }</code></td>
</tr>
<tr>
<td>ip_dscp</td>
<td>IP DSCP (6 bits in ToS field) (int)</td>
<td><code>{ &quot;ip_dscp&quot;: 3, &quot;eth_type&quot;: 2048 }</code></td>
</tr>
<tr>
<td>ip_ecn</td>
<td>IP ECN (2 bits in ToS field) (int)</td>
<td><code>{ &quot;ip_ecn&quot;: 0, &quot;eth_type&quot;: 34525 }</code></td>
</tr>
<tr>
<td>ip_proto</td>
<td>IP protocol (int)</td>
<td><code>{ &quot;ip_proto&quot;: 5, &quot;eth_type&quot;: 2048 }</code></td>
</tr>
<tr>
<td>ipv4_src</td>
<td>IPv4 source address (string)</td>
<td><code>{ &quot;ipv4_src&quot;: &quot;192.168.0.1&quot;, &quot;eth_type&quot;: 2048 }</code></td>
</tr>
<tr>
<td>ipv4_dst</td>
<td>IPv4 destination address (string)</td>
<td><code>{ &quot;ipv4_dst&quot;: &quot;192.168.10.10/255.255.255.0&quot;, &quot;eth_type&quot;: 2048 }</code></td>
</tr>
<tr>
<td>tcp_src</td>
<td>TCP source port (int)</td>
<td><code>{ &quot;tcp_src&quot;: 3, &quot;ip_proto&quot;: 6, &quot;eth_type&quot;: 2048 }</code></td>
</tr>
<tr>
<td>tcp_dst</td>
<td>TCP destination port (int)</td>
<td><code>{ &quot;tcp_dst&quot;: 5, &quot;ip_proto&quot;: 6, &quot;eth_type&quot;: 2048 }</code></td>
</tr>
<tr>
<td>udp_src</td>
<td>UDP source port (int)</td>
<td><code>{ &quot;udp_src&quot;: 2, &quot;ip_proto&quot;: 17, &quot;eth_type&quot;: 2048 }</code></td>
</tr>
<tr>
<td>udp_dst</td>
<td>UDP destination port (int)</td>
<td><code>{ &quot;udp_dst&quot;: 6, &quot;ip_proto&quot;: 17, &quot;eth_type&quot;: 2048 }</code></td>
</tr>
<tr>
<td>sctp_src</td>
<td>SCTP source port (int)</td>
<td><code>{ &quot;sctp_src&quot;: 99, &quot;ip_proto&quot;: 132, &quot;eth_type&quot;: 2048 }</code></td>
</tr>
<tr>
<td>sctp_dst</td>
<td>SCTP destination port (int)</td>
<td><code>{ &quot;sctp_dst&quot;: 99, &quot;ip_proto&quot;: 132, &quot;eth_type&quot;: 2048 }</code></td>
</tr>
<tr>
<td>Match field</td>
<td>Description</td>
<td>Example</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------------------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>icmpv4_type</td>
<td>ICMP type (int)</td>
<td>{&quot;icmpv4_type&quot;: 5, &quot;ip_proto&quot;: 1, &quot;eth_type&quot;: 2054}</td>
</tr>
<tr>
<td>icmpv4_code</td>
<td>ICMP code (int)</td>
<td>{&quot;icmpv4_code&quot;: 6, &quot;ip_proto&quot;: 1, &quot;eth_type&quot;: 2054}</td>
</tr>
<tr>
<td>arp_op</td>
<td>ARP opcode (int)</td>
<td>{&quot;arp_op&quot;: 3, &quot;eth_type&quot;: 2054}</td>
</tr>
<tr>
<td>arp_spa</td>
<td>ARP source IPv4 address (string)</td>
<td>{&quot;arp_spa&quot;: &quot;192.168.0.11&quot;, &quot;eth_type&quot;: 2054}</td>
</tr>
<tr>
<td>arp_tpa</td>
<td>ARP target IPv4 address (string)</td>
<td>{&quot;arp_tpa&quot;: &quot;192.168.0.44/24&quot;, &quot;eth_type&quot;: 2054}</td>
</tr>
<tr>
<td>arp_sha</td>
<td>ARP source hardware address (string)</td>
<td>{&quot;arp_sha&quot;: &quot;aa:bb:cc:11:22:33&quot;, &quot;eth_type&quot;: 2054}</td>
</tr>
<tr>
<td>arp_tha</td>
<td>ARP target hardware address (string)</td>
<td>{&quot;arp_tha&quot;: &quot;aa:bb:cc:11:22:33/00:00:00:00:ff:ff&quot;, &quot;eth_type&quot;: 2054}</td>
</tr>
<tr>
<td>ipv6_src</td>
<td>IPv6 source address (string)</td>
<td>{&quot;ipv6_src&quot;: &quot;2001::aaaaa:bbbbb:cccc:1111&quot;, &quot;eth_type&quot;: 34525}</td>
</tr>
<tr>
<td>ipv6_dst</td>
<td>IPv6 destination address (string)</td>
<td>{&quot;ipv6_dst&quot;: &quot;2001::ffff:cccc:bbbb:1111/64&quot;, &quot;eth_type&quot;: 34525}</td>
</tr>
<tr>
<td>ipv6_flabel</td>
<td>IPv6 Flow Label (int)</td>
<td>{&quot;ipv6_flabel&quot;: 2, &quot;eth_type&quot;: 34525}</td>
</tr>
<tr>
<td>ipv6_nd_target</td>
<td>Target address for Neighbor Discovery (string)</td>
<td>{&quot;ipv6_nd_target&quot;: &quot;2001::ffff:cccc:bbbb:1111&quot;, &quot;eth_type&quot;: 34525}</td>
</tr>
<tr>
<td>ipv6_nd_sll</td>
<td>Source link-layer for Neighbor Discovery (string)</td>
<td>{&quot;ipv6_nd_sll&quot;: &quot;aa:bb:cc:11:22:33&quot;, &quot;icmpv4_type&quot;: 5, &quot;ip_proto&quot;: 1}</td>
</tr>
<tr>
<td>ipv6_nd_tll</td>
<td>Target link-layer for Neighbor Discovery (string)</td>
<td>{&quot;ipv6_nd_tll&quot;: &quot;aa:bb:cc:11:22:33&quot;, &quot;icmpv6_type&quot;: 5}</td>
</tr>
<tr>
<td>mpls_label</td>
<td>MPLS label (int)</td>
<td>{&quot;mpls_label&quot;: 3, &quot;eth_type&quot;: 34888}</td>
</tr>
<tr>
<td>mpls_tc</td>
<td>MPLS Traffic Class (int)</td>
<td>{&quot;mpls_tc&quot;: 2, &quot;eth_type&quot;: 34888}</td>
</tr>
<tr>
<td>mpls_bos</td>
<td>MPLS BoS bit (int) (Openflow1.3+)</td>
<td>{&quot;mpls_bos&quot;: 1, &quot;eth_type&quot;: 34888}</td>
</tr>
<tr>
<td>pbb_isid</td>
<td>PBB I-SID (int or string) (Openflow1.3+)</td>
<td>{&quot;pbb_isid&quot;: 5, &quot;eth_type&quot;: 35047} or {&quot;pbb_isid&quot;: &quot;0x05/0xff&quot;, &quot;eth_type&quot;: 35047}</td>
</tr>
<tr>
<td>tunnel_id</td>
<td>Logical Port Metadata (int or string) (Openflow1.3+)</td>
<td>{&quot;tunnel_id&quot;: 7} or {&quot;tunnel_id&quot;: &quot;0x07/0xff&quot;, &quot;eth_type&quot;: 34525}</td>
</tr>
<tr>
<td>ipv6_exthdr</td>
<td>IPv6 Extension Header pseudo-field (int or string) (Openflow1.3+)</td>
<td>{&quot;ipv6_exthdr&quot;: 3, &quot;eth_type&quot;: 34888}</td>
</tr>
<tr>
<td>pbb_uca</td>
<td>PBB UCA hander field(int) (Openflow1.4+)</td>
<td>{&quot;pbb_uca&quot;: 1, &quot;eth_type&quot;: 35047}</td>
</tr>
<tr>
<td>tcp_flags</td>
<td>TCP flags(int) (Openflow1.5+)</td>
<td>{&quot;tcp_flags&quot;: 2, &quot;ip_proto&quot;: 6, &quot;eth_type&quot;: 2054}</td>
</tr>
<tr>
<td>actset_output</td>
<td>Output port from action set metadata(int) (Openflow1.5+)</td>
<td>{&quot;actset_output&quot;: 3}</td>
</tr>
<tr>
<td>packet_type</td>
<td>Packet type value(int) (Openflow1.5+)</td>
<td>{&quot;packet_type&quot;: [1, 2048]}</td>
</tr>
</tbody>
</table>

**Note:** Some field can be described with mask like as follows.

**Ethernet address:**

```
"aa:bb:cc:11:22:33"
"aa:bb:cc:11:22:33/00:00:00:00:ff:ff"
```

**IPv4 address:**

```
"192.168.0.11"
"192.168.0.44/24"
"192.168.10.10/255.255.255.0"
```

**IPv6 address:**

```
"2001::ffff:cccc:bbbb:1111"
"2001::ffff:cccc:bbbb:2222/64"
"2001::ffff:cccc:bbbb:2222/ffff:ffff:ffff::0"
```

**Metadata:**

```
"0x1212121212121212"
"0x3434343434343434/0x010101010101010101"
```
Example of VLAN ID match field

The following is available in OpenFlow1.0 or later.

- To match only packets with VLAN tag and VLAN ID equal value 5:

  ```
  $ curl -X POST -d '{
    "dpid": 1,
    "match": {
      "dl_vlan": 5
    },
    "actions": {
      "type": "OUTPUT",
      "port": 1
    }
  }' http://localhost:8080/stats/flowentry/add
  ```

  **Note:** When "dl_vlan" field is described as decimal int value, OFPVID_PRESENT(0x1000) bit is automatically applied.

The following is available in OpenFlow1.2 or later.

- To match only packets without a VLAN tag:

  ```
  $ curl -X POST -d '{
    "dpid": 1,
    "match": {
      "dl_vlan": "0x0000"  # Describe OFPVID_NONE(0x0000)
    },
    "actions": {
      "type": "OUTPUT",
      "port": 1
    }
  }' http://localhost:8080/stats/flowentry/add
  ```

- To match only packets with a VLAN tag regardless of its value:

  ```
  $ curl -X POST -d '{
    "dpid": 1,
    "match": {
      "dl_vlan": "0x1000/0x1000"  # Describe OFPVID_PRESENT(0x1000/0x1000)
    },
    "actions": {
      "type": "OUTPUT",
      "port": 1
    }
  }' http://localhost:8080/stats/flowentry/add
  ```

- To match only packets with VLAN tag and VLAN ID equal value 5:
Note: When using the descriptions for OpenFlow1.2 or later, please describe "dl_vlan" field as hexadecimal string value, and OFPVID_PRESENT(0x1000) bit is NOT automatically applied.

### Description of Actions on request messages

#### List of Actions (OpenFlow1.0):

<table>
<thead>
<tr>
<th>Actions</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUTPUT</td>
<td>Output packet from &quot;port&quot;</td>
<td>{'type': 'OUTPUT', 'port': 3}</td>
</tr>
<tr>
<td>SET_VLAN_VID</td>
<td>Set the 802.1Q VLAN ID using &quot;vlan_vid&quot;</td>
<td>{'type': 'SET_VLAN_VID', 'vlan_vid': 5}</td>
</tr>
<tr>
<td>SET_VLAN_PCP</td>
<td>Set the 802.1Q priority using &quot;vlan_pcp&quot;</td>
<td>{'type': 'SET_VLAN_PCP', 'vlan_pcp': 3}</td>
</tr>
<tr>
<td>STRIP_VLAN</td>
<td>Strip the 802.1Q header</td>
<td>{'type': 'STRIP_VLAN'}</td>
</tr>
<tr>
<td>SET_DL_SRC</td>
<td>Set ethernet source address using &quot;dl_src&quot;</td>
<td>{'type': 'SET_DL_SRC', 'dl_src': 'aa:bb:cc:11:22:33'}</td>
</tr>
<tr>
<td>SET_DL_DST</td>
<td>Set ethernet destination address using &quot;dl_dst&quot;</td>
<td>{'type': 'SET_DL_DST', 'dl_dst': 'aa:bb:cc:11:22:33'}</td>
</tr>
<tr>
<td>SET_NW_SRC</td>
<td>IP source address using &quot;nw_src&quot;</td>
<td>{'type': 'SET_NW_SRC', 'nw_src': '10.0.0.1'}</td>
</tr>
<tr>
<td>SET_NW_DST</td>
<td>IP destination address using &quot;nw_dst&quot;</td>
<td>{'type': 'SET_NW_DST', 'nw_dst': '10.0.0.1'}</td>
</tr>
<tr>
<td>SET_NW_TOS</td>
<td>Set IP ToS (DSCP field, 6 bits) using &quot;nw_tos&quot;</td>
<td>{'type': 'SET_NW_TOS', 'nw_tos': 184}</td>
</tr>
<tr>
<td>SET_TP_SRC</td>
<td>Set TCP/UDP source port using &quot;tp_src&quot;</td>
<td>{'type': 'SET_TP_SRC', 'tp_src': 8080}</td>
</tr>
<tr>
<td>SET_TP_DST</td>
<td>Set TCP/UDP destination port using &quot;tp_dst&quot;</td>
<td>{'type': 'SET_TP_DST', 'tp_dst': 8080}</td>
</tr>
<tr>
<td>ENQUEUE</td>
<td>Output to queue with &quot;queue_id&quot; attached to &quot;port&quot;</td>
<td>{'type': 'ENQUEUE', 'queue_id': 3, 'port': 1}</td>
</tr>
</tbody>
</table>

#### List of Actions (OpenFlow1.2 or later):
<table>
<thead>
<tr>
<th>Actions</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUTPUT</td>
<td>Output packet from &quot;port&quot;</td>
<td>{&quot;type&quot;: &quot;OUTPUT&quot;, &quot;port&quot;: 3}</td>
</tr>
<tr>
<td>COPY_TTL_OUT</td>
<td>Copy TTL outwards</td>
<td>{&quot;type&quot;: &quot;COPY_TTL_OUT&quot;}</td>
</tr>
<tr>
<td>COPY_TTL_IN</td>
<td>Copy TTL inwards</td>
<td>{&quot;type&quot;: &quot;COPY_TTL_IN&quot;}</td>
</tr>
<tr>
<td>SET_MPLS TTL</td>
<td>Set MPLS TTL using &quot;mpls_ttl&quot;</td>
<td>{&quot;type&quot;: &quot;SET_MPLS_TTL&quot;, &quot;mpls_ttl&quot;: 64}</td>
</tr>
<tr>
<td>DEC_MPLS TTL</td>
<td>Decrement MPLS TTL</td>
<td>{&quot;type&quot;: &quot;DEC_MPLS_TTL&quot;}</td>
</tr>
<tr>
<td>PUSH_VLAN</td>
<td>Push a new VLAN tag with &quot;ethertype&quot;</td>
<td>{&quot;type&quot;: &quot;PUSH_VLAN&quot;, &quot;ethertype&quot;: 33024}</td>
</tr>
<tr>
<td>POP_VLAN</td>
<td>Pop the outer VLAN tag</td>
<td>{&quot;type&quot;: &quot;POP_VLAN&quot;}</td>
</tr>
<tr>
<td>PUSH_MPLS</td>
<td>Push a new MPLS tag with &quot;ethertype&quot;</td>
<td>{&quot;type&quot;: &quot;PUSH_MPLS&quot;, &quot;ethertype&quot;: 34887}</td>
</tr>
<tr>
<td>POP_MPLS</td>
<td>Pop the outer MPLS tag with &quot;ethertype&quot;</td>
<td>{&quot;type&quot;: &quot;POP_MPLS&quot;, &quot;ethertype&quot;: 2054}</td>
</tr>
<tr>
<td>SET_QUEUE</td>
<td>Set queue id using &quot;queue_id&quot; when outputting to a port</td>
<td>{&quot;type&quot;: &quot;SET_QUEUE&quot;, &quot;queue_id&quot;: 7}</td>
</tr>
<tr>
<td>GROUP</td>
<td>Apply group identified by &quot;group_id&quot;</td>
<td>{&quot;type&quot;: &quot;GROUP&quot;, &quot;group_id&quot;: 5}</td>
</tr>
<tr>
<td>SET_NW_TTL</td>
<td>Set IP TTL using &quot;nw_ttl&quot;</td>
<td>{&quot;type&quot;: &quot;SET_NW_TTL&quot;, &quot;nw_ttl&quot;: 64}</td>
</tr>
<tr>
<td>DEC_NW_TTL</td>
<td>Decrement IP TTL</td>
<td>{&quot;type&quot;: &quot;DEC_NW_TTL&quot;}</td>
</tr>
<tr>
<td>SET_FIELD</td>
<td>Set a &quot;field&quot; using &quot;value&quot; (The set of keywords available for &quot;field&quot; is the same as match field)</td>
<td>See Example of set-field action</td>
</tr>
<tr>
<td>PUSH_PBB</td>
<td>Push a new PBB service tag with &quot;ethertype&quot; (Openflow1.3+)</td>
<td>{&quot;type&quot;: &quot;PUSH_PBB&quot;, &quot;ethertype&quot;: 35047}</td>
</tr>
<tr>
<td>POP_PBB</td>
<td>Pop the outer PBB service tag (Openflow1.3+)</td>
<td>{&quot;type&quot;: &quot;POP_PBB&quot;}</td>
</tr>
<tr>
<td>COPY_FIELD</td>
<td>Copy value between header and register (Openflow1.5+)</td>
<td>{&quot;type&quot;: &quot;COPY_FIELD&quot;, &quot;n_bits&quot;: 32, &quot;src_offset&quot;: 1, &quot;dst_offset&quot;: 2, &quot;src_oxm_id&quot;: &quot;eth_src&quot;, &quot;dst_oxm_id&quot;: &quot;eth_dst&quot;}</td>
</tr>
<tr>
<td>METER</td>
<td>Apply meter identified by &quot;meter_id&quot; (Openflow1.5+)</td>
<td>{&quot;type&quot;: &quot;METER&quot;, &quot;meter_id&quot;: 3}</td>
</tr>
<tr>
<td>EXPERIMENTER</td>
<td>Extensible action for the experimenter (Set &quot;base64&quot; or &quot;ascii&quot; to &quot;data_type&quot; field)</td>
<td>{&quot;type&quot;: &quot;EXPERIMENTER&quot;, &quot;experiment&quot;: 101, &quot;data&quot;: &quot;AACEQFBgc=&quot;, &quot;data_type&quot;: &quot;base64&quot;}</td>
</tr>
<tr>
<td>GOTO_TABLE</td>
<td>Setup the next table identified by &quot;table_id&quot; (Instruction)</td>
<td>{&quot;type&quot;: &quot;GOTO_TABLE&quot;, &quot;table_id&quot;: 8}</td>
</tr>
<tr>
<td>WRITE_METADATA</td>
<td>Setup the metadata field using &quot;metadata&quot; and &quot;metadata_mask&quot;</td>
<td>{&quot;type&quot;: &quot;WRITE_METADATA&quot;, &quot;metadata&quot;: 0x3, &quot;metadata_mask&quot;: 0x3}</td>
</tr>
<tr>
<td>METER</td>
<td>(Instruction) Apply meter identified by &quot;meter_id&quot; (deprecated in Openflow1.5)</td>
<td>{&quot;type&quot;: &quot;METER&quot;, &quot;meter_id&quot;: 3}</td>
</tr>
<tr>
<td>WRITE_ACTIONS</td>
<td>Write the action(s) onto the datapath action set</td>
<td>{&quot;type&quot;: &quot;WRITE_ACTIONS&quot;, actions&quot;: [{&quot;type&quot;: &quot;POP_VLAN&quot;}, {&quot;type&quot;: &quot;OUTPUT&quot;, &quot;port&quot;: 2}]}</td>
</tr>
<tr>
<td>CLEAR_ACTIONS</td>
<td>Clears all actions from the datapath action set</td>
<td>{&quot;type&quot;: &quot;CLEAR_ACTIONS&quot;}</td>
</tr>
</tbody>
</table>
Example of set-field action

To set VLAN ID to non-VLAN-tagged frame:

```sh
$ curl -X POST -d '{
  "dpid": 1,
  "match":{
    "dl_type": "0x8000"
  },
  "actions":[
    {
      "type": "PUSH_VLAN", # Push a new VLAN tag if a input frame is non-VLAN-tagged
      "ethertype": 33024 # Ethertype 0x8100(=33024): IEEE 802.1Q
    },
    {
      "type": "SET_FIELD",
      "field": "vlan_vid", # Set VLAN ID
      "value": 4102 # Describe sum of vlan_id(e.g. 6) | OFPVID_PRESENT(0x1000=4096)
    },
    {
      "type": "OUTPUT",
      "port": 2
    }
  ]
}' http://localhost:8080/stats/flowentry/add
```

6.3 ryu.app.rest_vtep

This sample application performs as VTEP for EVPN VXLAN and constructs a Single Subnet per EVI corresponding to the VLAN Based service in [RFC7432].

**Note:** This app will invoke OVSDB request to the switches. Please set the manager address before calling the API of this app.

```sh
$ sudo ovs-vsctl set-manager ptcp:6640
$ sudo ovs-vsctl show
  ...(snip)
Manager "ptcp:6640"
  ...(snip)
```

6.3.1 Usage Example

**Environment**

This example supposes the following environment:

```plain
Host A (172.17.0.1)          Host B (172.17.0.2)
+-----------------------------+-----------------------------+
(continues on next page)
### Configuration steps

1. Creates a new BGPSpeaker instance on each host.

   **On Host A:**
   ```
   (Host A)$ curl -X POST -d '{
   "dpid": 1,
   "as_number": 65000,
   "router_id": "172.17.0.1"
   }' http://localhost:8080/vtep/speakers | python -m json.tool
   ```

   **On Host B:**
   ```
   (Host B)$ curl -X POST -d '{
   "dpid": 1,
   "as_number": 65000,
   "router_id": "172.17.0.2"
   }' http://localhost:8080/vtep/speakers | python -m json.tool
   ```

2. Registers the neighbor for the speakers on each host.

   **On Host A:**
   ```
   (Host A)$ curl -X POST -d '{
   "address": "172.17.0.2",
   "remote_as": 65000
   }' http://localhost:8080/vtep/neighbors | python -m json.tool
   ```

   **On Host B:**
   ```
   (Host B)$ curl -X POST -d '{
   "address": "172.17.0.1",
   "remote_as": 65000
   }' http://localhost:8080/vtep/neighbors | python -m json.tool
   ```

3. Defines a new VXLAN network(VNI=10) on the Host A/B.

   **On Host A:**
   ```
   (Host A)$ curl -X POST -d '{
   "vni": 10
   }' http://localhost:8080/vtep/networks | python -m json.tool
   ```
On Host B:

```
(Host B)$ curl -X POST -d '{
"vni": 10
}' http://localhost:8080/vtep/networks | python -m json.tool
```

4. Registers the clients to the VXLAN network.

For "s1h1"(ip="10.0.0.11", mac="aa:bb:cc:00:00:11") on Host A:

```
(Host A)$ curl -X POST -d '{
"port": "s1-eth1",
"mac": "aa:bb:cc:00:00:11",
"ip": "10.0.0.11"
}' http://localhost:8080/vtep/networks/10/clients | python -m json.tool
```

For "s2h1"(ip="10.0.0.21", mac="aa:bb:cc:00:00:21") on Host B:

```
(Host B)$ curl -X POST -d '{
"port": "s2-eth1",
"mac": "aa:bb:cc:00:00:21",
"ip": "10.0.0.21"
}' http://localhost:8080/vtep/networks/10/clients | python -m json.tool
```

**Testing**

If BGP (EVPN) connection between Ryu1 and Ryu2 has been established, pings between the client s1h1 and s2h1 should work.

```
(s1h1)$ ping 10.0.0.21
```

**Troubleshooting**

If connectivity between s1h1 and s2h1 isn’t working, please check the followings.

1. Make sure that Host A and Host B have full network connectivity.

```
(Host A)$ ping 172.17.0.2
```

2. Make sure that BGP(EVPN) connection has been established.

```
(Host A)$ curl -X GET http://localhost:8080/vtep/neighbors | python -m json.tool
...

```

```

```

```
```
```
3. Make sure that BGP(EVPN) routes have been advertised.

```plaintext
(Host A)$ curl -X GET http://localhost:8080/vtep/networks |
python -m json.tool
...
{
  "10": {
    "EvpnNetwork": {
      "clients": {
        "aa:bb:cc:00:00:11": {
          "EvpnClient": {
            "ip": "10.0.0.11",
            "mac": "aa:bb:cc:00:00:11",
            "next_hop": "172.17.0.1",
            "port": 1
          }
        },
        "aa:bb:cc:00:00:21": { # route for "s2h1" on Host B
          "EvpnClient": {
            "ip": "10.0.0.21",
            "mac": "aa:bb:cc:00:00:21",
            "next_hop": "172.17.0.2",
            "port": 3
          }
        }
      },
      "ethernet_tag_id": 0,
      "route_dist": "65000:10",
      "vni": 10
    }
  }
}
```

4. Make sure that the IPv6 is enabled on your environment. Some Ryu BGP features require the IPv6 connectivity to bind sockets. Mininet seems to disable IPv6 on its installation.

For example:

```plaintext
$ sysctl net.ipv6.conf.all.disable_ipv6	net.ipv6.conf.all.disable_ipv6 = 0  # should NOT be enabled
$ grep GRUB_CMDLINE_LINUX_DEFAULT /etc/default/grub
# please remove "ipv6.disable=1" and reboot
GRUB_CMDLINE_LINUX_DEFAULT="ipv6.disable=0" quiet splash
```

5. Make sure that your switch using the OpenFlow version 1.3. This application supports only the OpenFlow version 1.3.

For example:

```plaintext
$ ovs-vsctl get Bridge s1 protocols
["OpenFlow1.3"]
```

**Note:** At the time of this writing, we use the the following version of Ryu, Open vSwitch and Mininet.
6.3.2 REST API

```python
class ryu.app.rest_vtep.RestVtepController(req, link, data, **config)

    add_speaker(req, **kwargs)
    Creates a new BGPSpeaker instance.
    Usage:

    Method | URI
    POST   | /vtep/speakers

    Request parameters:

    Attribute   | Description
    dpid        | ID of Datapath binding to speaker. (e.g. 1)
    as_number   | AS number. (e.g. 65000)
    router_id   | Router ID. (e.g. "172.17.0.1")

    Example:

    $ curl -X POST -d '{
        "dpid": 1,
        "as_number": 65000,
        "router_id": "172.17.0.1"
    }' http://localhost:8080/vtep/speakers | python -m json.tool

    {
        "172.17.0.1": {
            "EvpnSpeaker": {
                "as_number": 65000,
                "dpid": 1,
                "neighbors": {},
                "router_id": "172.17.0.1"
            }
        }
    }
```

```python
    get_speakers(_, **kwargs)
    Gets the info of BGPSpeaker instance.
    Usage:
```
Method | URI
---|---
GET | /vtep/speakers

Example:
```
$ curl -X GET http://localhost:8080/vtep/speakers |
    python -m json.tool
```

```
{
    "172.17.0.1": {
        "EvpnSpeaker": {
            "as_number": 65000,
            "dpid": 1,
            "neighbors": {
                "172.17.0.2": {
                    "EvpnNeighbor": {
                        "address": "172.17.0.2",
                        "remote_as": 65000,
                        "state": "up"
                    }
                }
            },
            "router_id": "172.17.0.1"
        }
    }
}
```

del_speaker(_, **kwargs)

Shutdows BGPSpeaker instance.

Usage:
```
Method | URI
---|---
DELETE | /vtep/speakers
```

Example:
```
$ curl -X DELETE http://localhost:8080/vtep/speakers |
    python -m json.tool
```

```
{
    "172.17.0.1": {
        "EvpnSpeaker": {
            "as_number": 65000,
            "dpid": 1,
            "neighbors": {},
            "router_id": "172.17.0.1"
        }
    }
}
```

add_neighbor(req, **kwargs)

Registers a new neighbor to the speaker.

Usage:
<table>
<thead>
<tr>
<th>Method</th>
<th>URI</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST</td>
<td>/vtep/neighbors</td>
</tr>
</tbody>
</table>

Request parameters:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>address</td>
<td>IP address of neighbor. (e.g. “172.17.0.2”)</td>
</tr>
<tr>
<td>remote_as</td>
<td>AS number of neighbor. (e.g. 65000)</td>
</tr>
</tbody>
</table>

Example:

```bash
$ curl -X POST -d '{
  "address": "172.17.0.2",
  "remote_as": 65000
}' http://localhost:8080/vtep/neighbors |
python -m json.tool

{"172.17.0.2": {
  "EvpnNeighbor": {
    "address": "172.17.0.2",
    "remote_as": 65000,
    "state": "down"
  }
}}
```

**get_neighbors(**kwargs**)

Gets a list of all neighbors.

Usage:

<table>
<thead>
<tr>
<th>Method</th>
<th>URI</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>/vtep/neighbors</td>
</tr>
</tbody>
</table>

Example:

```bash
$ curl -X GET http://localhost:8080/vtep/neighbors |
python -m json.tool

{"172.17.0.2": {
  "EvpnNeighbor": {
    "address": "172.17.0.2",
    "remote_as": 65000,
    "state": "up"
  }
}}
```

**get_neighbor**(.__**kwargs**__)

Gets the neighbor for the specified address.

Usage:
### Method

<table>
<thead>
<tr>
<th>Method</th>
<th>URI</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>/vtep/neighbors/{address}</td>
</tr>
</tbody>
</table>

Request parameters:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>address</td>
<td>IP address of neighbor. (e.g. &quot;172.17.0.2&quot;)</td>
</tr>
</tbody>
</table>

Example:

```bash
dir $ curl -X GET http://localhost:8080/vtep/neighbors/172.17.0.2 |
  python -m json.tool

dict {
    "172.17.0.2": {
        "EvpnNeighbor": {
            "address": "172.17.0.2",
            "remote_as": 65000,
            "state": "up"
        }
    }
}
```

#### del_neighbor(\_\_, **kwargs)

Unregister the specified neighbor from the speaker.

Usage:

<table>
<thead>
<tr>
<th>Method</th>
<th>URI</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELETE</td>
<td>/vtep/speaker/neighbors/{address}</td>
</tr>
</tbody>
</table>

Request parameters:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>address</td>
<td>IP address of neighbor. (e.g. &quot;172.17.0.2&quot;)</td>
</tr>
</tbody>
</table>

Example:

```bash
dir $ curl -X DELETE http://localhost:8080/vtep/speaker/neighbors/172.17.0.2 |
  python -m json.tool

dict {
    "172.17.0.2": {
        "EvpnNeighbor": {
            "address": "172.17.0.2",
            "remote_as": 65000,
            "state": "up"
        }
    }
}
```

#### add_network(req, **kwargs)

Defines a new network.
Usage:

<table>
<thead>
<tr>
<th>Method</th>
<th>URI</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST</td>
<td>/vtep/networks</td>
</tr>
</tbody>
</table>

Request parameters:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vni</td>
<td>Virtual Network Identifier (e.g. 10)</td>
</tr>
</tbody>
</table>

Example:

$ curl -X POST -d '{
  "vni": 10
}' http://localhost:8080/vtep/networks | python -m json.tool

```
{
  "10": {
    "EvpnNetwork": {
      "clients": {},
      "ethernet_tag_id": 0,
      "route_dist": "65000:10",
      "vni": 10
    }
  }
}
```

### get_networks(

Gets a list of all networks.

Usage:

<table>
<thead>
<tr>
<th>Method</th>
<th>URI</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>/vtep/networks</td>
</tr>
</tbody>
</table>

Example:

$ curl -X GET http://localhost:8080/vtep/networks | python -m json.tool

```
{
  "10": {
    "EvpnNetwork": {
      "clients": {
        "aa:bb:cc:dd:ee:ff": {
          "EvpnClient": {
            "ip": "10.0.0.1",
            "mac": "aa:bb:cc:dd:ee:ff",
            "next_hop": "172.17.0.1",
            "port": 1
          }
        }
      },
      "ethernet_tag_id": 0,
      "route_dist": "65000:10",
      "vni": 10
    }
  }
}
```
get_network(_, **kwargs)

Gets the network for the specified VNI.

Usage:

<table>
<thead>
<tr>
<th>Method</th>
<th>URI</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>/vtep/networks/{vni}</td>
</tr>
</tbody>
</table>

Request parameters:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vni</td>
<td>Virtual Network Identifier. (e.g. 10)</td>
</tr>
</tbody>
</table>

Example:

```
$ curl -X GET http://localhost:8080/vtep/networks/10 | python -m json.tool
```

```
{
   "10": {
      "EvpnNetwork": {
         "clients": {
            "aa:bb:cc:dd:ee:ff": {
               "EvpnClient": {
                  "ip": "10.0.0.1",
                  "mac": "aa:bb:cc:dd:ee:ff",
                  "next_hop": "172.17.0.1",
                  "port": 1
               }
            }
         },
         "ethernet_tag_id": 0,
         "route_dist": "65000:10",
         "vni": 10
      }
   }
}
```

del_network(_, **kwargs)

Deletes the network for the specified VNI.

Usage:

<table>
<thead>
<tr>
<th>Method</th>
<th>URI</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELETE</td>
<td>/vtep/networks/{vni}</td>
</tr>
</tbody>
</table>

Request parameters:
### Attribute Description

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vni</td>
<td>Virtual Network Identifier. (e.g. 10)</td>
</tr>
</tbody>
</table>

**Example:**

```bash
$ curl -X DELETE http://localhost:8080/vtep/networks/10 | python -m json.tool
```

```json
{
   "10": {
      "EvpnNetwork": {
         "ethernet_tag_id": 10,
         "clients": [
            {
               "EvpnClient": {
                  "ip": "10.0.0.11",
                  "mac": "e2:b1:0c:ba:42:ed",
                  "port": 1
               }
            }
         ],
         "route_dist": "65000:100",
         "vni": 10
      }
   }
}
```

### add_client (req, **kwargs)

Registers a new client to the specified network.

**Usage:**

<table>
<thead>
<tr>
<th>Method</th>
<th>URI</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST</td>
<td>/vtep/networks/{vni}/clients</td>
</tr>
</tbody>
</table>

**Request parameters:**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vni</td>
<td>Virtual Network Identifier. (e.g. 10)</td>
</tr>
<tr>
<td>port</td>
<td>Port number to connect client. For convenience, port name can be specified and automatically translated to port number. (e.g. &quot;s1-eth1&quot; or 1)</td>
</tr>
<tr>
<td>mac</td>
<td>Client MAC address to register. (e.g. &quot;aa:bb:cc:dd:ee:ff&quot;)</td>
</tr>
<tr>
<td>ip</td>
<td>Client IP address. (e.g. &quot;10.0.0.1&quot;)</td>
</tr>
</tbody>
</table>

**Example:**

```bash
$ curl -X POST -d '{
   "port": "s1-eth1",
   "mac": "aa:bb:cc:dd:ee:ff",
   "ip": "10.0.0.1"
}' http://localhost:8080/vtep/networks/10/clients | python -m json.tool
```
{ "10": {
   "EvpnClient": {
      "ip": "10.0.0.1",
      "mac": "aa:bb:cc:dd:ee:ff",
      "next_hop": "172.17.0.1",
      "port": 1
   }
}
}

def del_client(_, **kwargs)
    Registers a new client to the specified network.

Usage:

<table>
<thead>
<tr>
<th>Method</th>
<th>URI</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELETE</td>
<td>/vtep/networks/{vni}/clients/{mac}</td>
</tr>
</tbody>
</table>

Request parameters:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vni</td>
<td>Virtual Network Identifier. (e.g. 10)</td>
</tr>
<tr>
<td>mac</td>
<td>Client MAC address to register.</td>
</tr>
</tbody>
</table>

Example:

```
$ curl -X DELETE http://localhost:8080/vtep/networks/10/clients/
   →aa:bb:cc:dd:ee:ff |
python -m json.tool
```

{ "10": {
   "EvpnClient": {
      "ip": "10.0.0.1",
      "mac": "aa:bb:cc:dd:ee:ff",
      "next_hop": "172.17.0.1",
      "port": 1
   }
}

6.4 ryu.services.protocols.bgp.application

This module provides a convenient application for using Ryu BGPSpeaker and for writing your BGP application.

It reads a configuration file which includes settings for neighbors, routes and some others. Please refer to `ryu/services/protocols/bgp/bgp_sample_conf.py` for the sample configuration.

Usage Example:

```
$ ryu-manager ryu/services/protocols/bgp/application.py \
    --bgp-app-config-file ryu/services/protocols/bgp/bgp_sample_conf.py
```

Chapter 6. Built-in Ryu applications
6.4.1 SSH Console

You can also use the SSH console and see the RIB and do some operations from this console. The SSH port and username/password can be configured by the configuration file. You can check the help by hitting '?' key in this interface.

Example:

```bash
$ ssh localhost -p 4990
Hello, this is Ryu BGP speaker (version 4.19).
bgpd> # Hit '? key
  clear - allows to reset BGP connections
  help - show this help
  quit - exit this session
  set - set runtime settings
  show - shows runtime state information
bgpd>
bgpd> show rib all
Status codes: * valid, > best
Origin codes: i - IGP, e - EGP, ? - incomplete
               Network Labels Next Hop Reason Metric LocPrf Path
           --> 10.10.1.0/24 None 0.0.0.0 Only Path i
bgpd>
```

6.4.2 Integration with Other Applications

`ryu.services.protocols.bgp.application.RyuBGPSpeaker` will notifies the following events to other Ryu applications.

- EventBestPathChanged
- EventAdjRibInChanged
- EventPeerDown
- EventPeerUp

To catch these events, specify @set_ev_cls() decorator to the event handlers in the Ryu applications.

Example Application:

```python
# my_bgp_app.py
from ryu.base import app_manager
from ryu.controller.handler import set_ev_cls
from ryu.services.protocols.bgp import application

class MyBGPApppp(app_manager.RyuApp):
    _CONTEXTS = {
        'ryubgpspeaker': application.RyuBGPSpeaker,
    }

    def __init__(self, *args, **kwargs):
        super(MyBGPAppp, self).__init__(*args, **kwargs)
        
(continues on next page)```
# Stores "ryu.services.protocols.bgp.application.RyuBGPSpeaker"
# instance in order to call the APIs of
# "ryu.services.protocols.bgp.bgpspeaker.BGPSpeaker" via
# "self.app.speaker".
# Please note at this time, "BGPSpeaker" is NOT instantiated yet.
self.app = kwargs['ryubgpspeaker']

@set_ev_cls(bgp_application.EventBestPathChanged)
def _best_path_changed_handler(self, ev):
    self.logger.info('Best path changed: is_withdraw=%s, path=%s', ev.is_withdraw, ev.path)

Usage Example:

$ ryu-manager my_bgp_app.py \
    --bgp-app-config-file ryu/services/protocols/bgp/bgp_sample_conf.py

Note: For the APIs for ryu.services.protocols.bgp.bgpspeaker.BGPSpeaker, please refer to **BGP speaker library API Reference**.

## 6.4.3 API Reference

### exception

**ryu.services.protocols.bgp.application.ApplicationException**(desc=None)

Specific Base exception related to *BGPSpeaker*.

### class

**ryu.services.protocols.bgp.application.EventAdjRibInChanged**(path, is_withdraw, peer_ip, peer_as)

Event called when any adj-RIB-in path is changed due to UPDATE messages or remote peer’s down.

This event is the wrapper for adj_rib_in_change_handler of bgpspeaker.BGPSpeaker.

*path* attribute contains an instance of info_base.base.Path subclasses.

If *is_withdraw* attribute is True, *path* attribute has the information of the withdraw route.

*peer_ip* is the peer’s IP address who sent this path.

*peer_as* is the peer’s AS number who sent this path.

### class

**ryu.services.protocols.bgp.application.EventBestPathChanged**(path, is_withdraw)

Event called when any best remote path is changed due to UPDATE messages or remote peer’s down.

This event is the wrapper for best_path_change_handler of bgpspeaker.BGPSpeaker.

*path* attribute contains an instance of info_base.base.Path subclasses.

If *is_withdraw* attribute is True, *path* attribute has the information of the withdraw route.

### class

**ryu.services.protocols.bgp.application.EventPeerDown**(remote_ip, remote_as)

Event called when the session to the remote peer goes down.

This event is the wrapper for peer_down_handler of bgpspeaker.BGPSpeaker.
remote_ip attribute is the IP address of the remote peer.

remote_as attribute is the AS number of the remote peer.

class ryu.services.protocols.bgp.application.EventPeerUp(remote_ip, remote_as)
Event called when the session to the remote peer goes up.
This event is the wrapper for peer_up_handler of bgpspeaker.BGPSpeaker.

remote_ip attribute is the IP address of the remote peer.

remote_as attribute is the AS number of the remote peer.

class ryu.services.protocols.bgp.application.RyuBGPSpeaker(*args, **kwargs)
Base application for implementing BGP applications.

    start()
    Hook that is called after startup initialization is done.

ryu.services.protocols.bgp.application.load_config(config_file)
Validates the given file for use as the settings file for BGPSpeaker and loads the configuration from the given file as a module instance.

ryu.services.protocols.bgp.application.validate_rpc_host(ip)
Validates the given ip for use as RPC server address.
CHAPTER 7

Indices and tables

- genindex
- modindex
- search
ryu.app.cbench, 9
ryu.app.ofctl.api, 513
ryu.app.ofctl.exception, 514
ryu.app.rest_vtep, 567
ryu.app.simple_switch, 9
ryu.base.app_manager, 7
ryu.controller.controller, 7
ryu.controller.dpset, 7
ryu.controller.ofp_event, 7
ryu.controller.ofp_handler, 7
ryu.lib.netconf, 9
ryu.lib.of_config, 9
ryu.lib.ovs, 9
ryu.lib.ovs.bridge, 135
ryu.lib.ovs.vsctl, 134
ryu.lib.packet, 9
ryu.lib.packet.arp, 20
ryu.lib.packet.bfd, 21
ryu.lib.packet.bgp, 26
ryu.lib.packet.bmp, 41
ryu.lib.packet.bpu, 46
ryu.lib.packet.cfm, 51
ryu.lib.packet.dhcp, 57
ryu.lib.packet.dhcp6, 58
ryu.lib.packet.ethernet, 60
ryu.lib.packet.geneve, 61
ryu.lib.packet.gre, 62
ryu.lib.packet.icmp, 63
ryu.lib.packet.icmpv6, 65
ryu.lib.packet.igmp, 69
ryu.lib.packet.ipv4, 73
ryu.lib.packet.ipv6, 74
ryu.lib.packet.llc, 77
ryu.lib.packet.lldp, 79
ryu.lib.packet.mpls, 82
ryu.lib.packet.openflow, 83
ryu.lib.packet.ospf, 84
ryu.lib.packet.packet, 18
ryu.lib.packet.packet_base, 19
ryu.lib.packet.pbb, 86
ryu.lib.packet.sctp, 87
ryu.lib.packet.slow, 98
ryu.lib.packet.stream_parser, 19
ryu.lib.packet.tcp, 102
ryu.lib.packet.udp, 103
ryu.lib.packet.vlan, 104
ryu.lib.packet.rrp, 105
ryu.lib.packet.vxlan, 108
ryu.lib.packet.zebra, 109
ryu.lib.xflow, 10
ryu.ofproto.nicira_ext, 493
ryu.ofproto.ofproto_v1_0, 8
ryu.ofproto.ofproto_v1_0_parser, 8
ryu.ofproto.ofproto_v1_2, 8
ryu.ofproto.ofproto_v1_2_parser, 8
ryu.ofproto.ofproto_v1_3, 8
ryu.ofproto.ofproto_v1_3_parser, 8
ryu.ofproto.ofproto_v1_4, 9
ryu.ofproto.ofproto_v1_4_parser, 9
ryu.ofproto.ofproto_v1_5, 9
ryu.ofproto.ofproto_v1_5_parser, 9
ryu.services.protocols.bgp.application, 578
ryu.topology, 9
Symbols

_CONTEXTS (ryu.base.app_manager.RyuApp attribute), 495
_EVENTS (ryu.base.app_manager.RyuApp attribute), 495
_TYPE (ryu.ofproto.ofproto_parser.MsgBase attribute), 138

A
add_bond() (ryu.lib.ovs.bridge.OVSBridge method), 135
add_client() (ryu.app.rest_vtep.RestVtepController method), 577
add_db_attribute() (ryu.lib.ovs.bridge.OVSBridge method), 135
add_gre_port() (ryu.lib.ovs.bridge.OVSBridge method), 135
add_neighbor() (ryu.app.rest_vtep.RestVtepController method), 572
add_network() (ryu.app.rest_vtep.RestVtepController method), 574
add_protocol() (ryu.lib.packet.packet.Packet method), 18
add_speaker() (ryu.app.rest_vtep.RestVtepController method), 571
add_tunnel_port() (ryu.lib.ovs.bridge.OVSBridge method), 135
add_vxlan_port() (ryu.lib.ovs.bridge.OVSBridge method), 136
AdminReset, 26
AdminShutdown, 26
ApplicationException, 580
arp (class in ryu.lib.packet.arp), 20
arp_ip() (in module ryu.lib.packet.arp), 21
ASPathFilter (class in ryu.services.protocols.bgp.info_base.base), 129
AttrFlagError, 26
attribute_map_get() (ryu.services.protocols.bgp.bgpspeaker.BGPSpeaker method), 120
attribute_map_set() (ryu.services.protocols.bgp.bgpspeaker.BGPSpeaker method), 120
AttributeMap (class in ryu.services.protocols.bgp.info_base.base), 129
AttrLenError, 26
auth (class in ryu.lib.packet.ipv6), 74
authenticate() (ryu.lib.packet.bfd.bfd method), 25
authenticate() (ryu.lib.packet.bfd.KeyedMD5 method), 22
authenticate() (ryu.lib.packet.bfd.KeyedSHA1 method), 23
authenticate() (ryu.lib.packet.bfd.SimplePassword method), 24
AuthFailure, 26

B
BadBgpId, 31
BadLen, 31
BadMsg, 31
BadNotification, 31
BadPeerAs, 31
bfd (class in ryu.lib.packet.bfd), 25
BFDAuth (class in ryu.lib.packet.bfd), 22
BGPEvpnEsiLabelExtendedCommunity (class in ryu.lib.packet.bgp), 26
BGPEvpnEsImportRTExtendedCommunity (class in ryu.lib.packet.bgp), 26
BGPEvpnMac MobilityExtendedCommunity (class in ryu.lib.packet.bgp), 26
BgpExc, 31
BGPFowSpecRedirectCommunity (class in ryu.lib.packet.bgp), 27
BGPFowSpecTPIDActionCommunity (class in ryu.lib.packet.bgp), 27
BGPFowSpecTrafficActionCommunity (class in ryu.lib.packet.bgp), 27
BGPFlowSpecTrafficMarkingCommunity (class in ryu.lib.packet.bgp), 27
BGPFlowSpecTrafficRateCommunity (class in ryu.lib.packet.bgp), 28
BGPFlowSpecVlanActionCommunity (class in ryu.lib.packet.bgp), 27
BGPKeepAlive (class in ryu.lib.packet.bgp), 28
BGPMessage (class in ryu.lib.packet.bgp), 28
BGPNotification (class in ryu.lib.packet.bgp), 28
BGPOpen (class in ryu.lib.packet.bgp), 29
BGPPathAttributePmsiTunnel (class in ryu.lib.packet.bgp), 30
BGPRouteRefresh (class in ryu.lib.packet.bgp), 30
BGPUpdate (class in ryu.services.protocols.bgp.bgpspeaker), 119
bmp_server_add() (ryu.services.protocols.bgp.bgpspeaker.BGPSpeaker method), 120
bmp_server_del() (ryu.services.protocols.bgp.bgpspeaker.BGPSpeaker method), 120
BMPInitiation (class in ryu.lib.packet.bpm), 41
BMPMessage (class in ryu.lib.packet.bpm), 42
BMPPeerDownNotification (class in ryu.lib.packet.bpm), 42
BMPPeerMessage (class in ryu.lib.packet.bpm), 43
BMPPeerUpNotification (class in ryu.lib.packet.bpm), 44
BMPRouteMonitoring (class in ryu.lib.packet.bpm), 44
BMPStatisticsReport (class in ryu.lib.packet.bpm), 45
BMPTermination (class in ryu.lib.packet.bpm), 46
bpdu (class in ryu.lib.packet.bpdu), 51

C

cause_cookie_while_shutdown (class in ryu.lib.packet.sctp), 87
cause_invalid_param (class in ryu.lib.packet.sctp), 87
cause_invalid_stream_id (class in ryu.lib.packet.sctp), 87
cause_missing_param (class in ryu.lib.packet.sctp), 87
cause_no_userdata (class in ryu.lib.packet.sctp), 88
cause_out_of_resource (class in ryu.lib.packet.sctp), 88
cause_protocol_violation (class in ryu.lib.packet.sctp), 88
cause_restart_with_new_addr (class in ryu.lib.packet.sctp), 88
cause_stale_cookie (class in ryu.lib.packet.sctp), 89
cause_unrecognized_chunk (class in ryu.lib.packet.sctp), 89
cause_unrecognized_param (class in ryu.lib.packet.sctp), 89
cause_unresolvable_addr (class in ryu.lib.packet.sctp), 90
cause_user_initiated_abort (class in ryu.lib.packet.sctp), 90
cc_message (class in ryu.lib.packet.cfm), 51
cfm (class in ryu.lib.packet.cfm), 52
ChassisID (class in ryu.lib.packetlldp), 80
chunk_abort (class in ryu.lib.packet.sctp), 90
chunk_cookie_ack (class in ryu.lib.packet.sctp), 90
chunk_cookie_echo (class in ryu.lib.packet.sctp), 91
chunk_cwr (class in ryu.lib.packet.sctp), 91
chunk_data (class in ryu.lib.packet.sctp), 91
chunk_ecn_echo (class in ryu.lib.packet.sctp), 92
chunk_error (class in ryu.lib.packet.sctp), 92
chunk_output (class in ryu.lib.packet.sctp), 92
chunkremoveAttr (class in ryu.lib.packet.sctp), 92
chunk_heartbeat_ack (class in ryu.lib.packet.sctp), 92
chunk_init (class in ryu.lib.packet.sctp), 93
chunk_init_ack (class in ryu.lib.packet.sctp), 93
chunk_sack (class in ryu.lib.packet.sctp), 94
chunk_shutdown (class in ryu.lib.packet.sctp), 94
chunk_shutdown_ack (class in ryu.lib.packet.sctp), 94
chunk_shutdown_complete (class in ryu.lib.packet.sctp), 95
clear_db_attribute() (ryu.lib.ovs.bridge.OVSBridge method), 136
clone() (ryu.services.protocols.bgp.info_base.base.ASPathFilter method), 129
clone() (ryu.services.protocols.bgp.info_base.base.AttributeMap method), 130
clone() (ryu.services.protocols.bgp.info_base.base.PrefixFilter method), 128
close() (ryu.base.app_manager.RyuApp method), 495
CODE (ryu.lib.packet.bgp.BgpExc attribute), 31
CollisionResolution, 31
ConfigurationBPDUs (class in ryu.lib.packet.bpdu), 48
ConnRejected, 32
context_iteritems() (ryu.base.app_manager.RyuApp class method), 495
ControlFormatI (class in ryu.lib.packet.illc), 78
ControlFormatS (class in ryu.lib.packet.illc), 78
ControlFormatU (class in ryu.lib.packet.illc), 78
create() (ryu.lib.packet.vrrp.vrrpv2 static method), 107
create() (ryu.lib.packet.vrrp.vrrpv3 static method), 107
create_packet() (ryu.lib.packet.vrrp.vrrpv method), 106
flowspec_prefix_del()
    (ryu.services.protocols.bgp.bgspeaker.BGPSpeaker
     method), 123
FlowSpecComponentUnknown (class in ryu.lib.packet.bgp), 33
FlowSpecDestinationMac (class in ryu.lib.packet.bgp), 34
FlowSpecDestPort (class in ryu.lib.packet.bgp), 33
FlowSpecDestPrefix (class in ryu.lib.packet.bgp), 33
FlowSpecDSCP (class in ryu.lib.packet.bgp), 33
FlowSpecEtherType (class in ryu.lib.packet.bgp), 33
FlowSpecFragment (class in ryu.lib.packet.bgp), 34
FlowSpecICMPCode (class in ryu.lib.packet.bgp), 37
FlowSpecICMPType (class in ryu.lib.packet.bgp), 37
FlowSpecInnerVLANCos (class in ryu.lib.packet.bgp), 37
FlowSpecInnerVLANID (class in ryu.lib.packet.bgp), 37
FlowSpecIPv4Protocol (class in ryu.lib.packet.bgp), 34
FlowSpecIPv4NLRI (class in ryu.lib.packet.bgp), 34
FlowSpecIPv6DestPrefix (class in ryu.lib.packet.bgp), 36
FlowSpecIPv6FlowLabel (class in ryu.lib.packet.bgp), 36
FlowSpecIPv6Fragment (class in ryu.lib.packet.bgp), 36
FlowSpecIPv6NLRI (class in ryu.lib.packet.bgp), 36
FlowSpecIPv6SrcPrefix (class in ryu.lib.packet.bgp), 37
FlowSpecL2VPNNLRI (class in ryu.lib.packet.bgp), 37
FlowSpecLLCControl (class in ryu.lib.packet.bgp), 38
FlowSpecLLCDSAP (class in ryu.lib.packet.bgp), 38
FlowSpecLLCSSAP (class in ryu.lib.packet.bgp), 38
FlowSpecNextHeader (class in ryu.lib.packet.bgp), 38
FlowSpecPacketLen (class in ryu.lib.packet.bgp), 38
FlowSpecPort (class in ryu.lib.packet.bgp), 38
FlowSpecSNAP (class in ryu.lib.packet.bgp), 38
FlowSpecSourceMac (class in ryu.lib.packet.bgp), 38
FlowSpecSrcPort (class in ryu.lib.packet.bgp), 38
FlowSpecTCPFlags (class in ryu.lib.packet.bgp), 39
FlowSpecVLANCos (class in ryu.lib.packet.bgp), 39
FlowSpecVLANID (class in ryu.lib.packet.bgp), 39
FlowSpecVPNv4NLRI (class in ryu.lib.packet.bgp), 39
FlowSpecVPNv6NLRI (class in ryu.lib.packet.bgp), 39
from_jsondict () (ryu.lib.packet.ipv6 Packet class method), 74
from_jsondict () (ryu.lib.packet.packet.Packet class method), 18
from_jsondict () (ryu.ofproto.ofproto_parser.MsgBase
     method), 139
from_user () (ryu.lib.packet.bgp.FlowSpecIPv4NLRI
     class method), 34
from_user () (ryu.lib.packet.bgp.FlowSpecIPv6NLRI
     class method), 36
from_user () (ryu.lib.packet.bgp.FlowSpecL2VPNNLRI
     class method), 38
from_user () (ryu.lib.packet.bgp.FlowSpecVPNv4NLRI
     class method), 39
from_user () (ryu.lib.packet.bgp.FlowSpecVPNv6NLRI
     class method), 39

geneve (class in ryu.lib.packet.geneve), 61
get () (ryu.controller.dpset.DPSet method), 496
get_all () (ryu.controller.dpset.DPSet method), 496
get_controller () (ryu.lib.ovs.bridge.OVSBridge
     method), 137
get_datapath () (in module ryu.app.ofctl.api), 513
get_datapath_id () (ryu.lib.ovs.bridge.OVSBridge
     method), 137
get_db_attribute () (ryu.lib.ovs.bridge.OVSBridge
     method), 137
get_neighbor () (ryu.app.rest_vtep.RestVtepController
     method), 573
get_neighbors () (ryu.app.rest_vtep.RestVtepController
     method), 573
get_network () (ryu.app.rest_vtep.RestVtepController
     method), 576
get_networks () (ryu.app.rest_vtep.RestVtepController
     method), 576
get_ofport () (ryu.lib.ovs.bridge.OVSBridge
     method), 137
get_packet_type () (ryu.lib.packet.ethernet.ethernet class method), 61
get_packet_type () (ryu.lib.packet.packet_base.PacketBase
     class method), 19
get_packet_type () (ryu.lib.packet.udp.udp static
     method), 104
get_packet_type () (ryu.lib.packet.vlan.svlan class
     method), 104
get_packet_type () (ryu.lib.packet.vlan.vlan class
     method), 105
get_port () (ryu.controller.dpset.DPSet method), 496
get_port_name_list () (ryu.lib.ovs.bridge.OVSBridge
     method), 137
get_ports () (ryu.controller.dpset.DPSet
     method), 496
get_protocol () (ryu.lib.packet.packet.Packet
     method), 18
neighbors_get() (ryu.services.protocols.bgp.bgpspeaker.BGPSpeaker method), 125

NextHopBlackhole (class in ryu.lib.packet.zebra), 109

NextHopIFIndex (class in ryu.lib.packet.zebra), 109

NextHopIFName (class in ryu.lib.packet.zebra), 109

NextHopIPv4 (class in ryu.lib.packet.zebra), 109

NextHopIPv4IFIndex (class in ryu.lib.packet.zebra), 109

NextHopIPv4IFName (class in ryu.lib.packet.zebra), 109

NextHopIPv6 (class in ryu.lib.packet.zebra), 109

NextHopIPv6IFIndex (class in ryu.lib.packet.zebra), 109

NextHopIPv6IFName (class in ryu.lib.packet.zebra), 109

NotSync, 40

nvgre() (in module ryu.lib.packet.gre), 63

NXActionBundle (class in ryu.ofproto.ofproto_v1_3_parser), 488

NXActionBundleLoad (class in ryu.ofproto.ofproto_v1_3_parser), 489

NXActionConjunction (class in ryu.ofproto.ofproto_v1_3_parser), 487

NXActionController (class in ryu.ofproto.ofproto_v1_3_parser), 483

NXActionController2 (class in ryu.ofproto.ofproto_v1_3_parser), 483

NXActionCT (class in ryu.ofproto.ofproto_v1_3_parser), 489

NXActionDecMplsTtl (class in ryu.ofproto.ofproto_v1_3_parser), 487

NXActionDecNshTtl (class in ryu.ofproto.ofproto_v1_3_parser), 492

NXActionDecTtl (class in ryu.ofproto.ofproto_v1_3_parser), 474

NXActionDecTtlCntIds (class in ryu.ofproto.ofproto_v1_3_parser), 484

NXActionExit (class in ryu.ofproto.ofproto_v1_3_parser), 483

NXActionFinTimeout (class in ryu.ofproto.ofproto_v1_3_parser), 486

NXActionLearn (class in ryu.ofproto.ofproto_v1_3_parser), 481

NXActionMultipath (class in ryu.ofproto.ofproto_v1_3_parser), 487

NXActionNAT (class in ryu.ofproto.ofproto_v1_3_parser), 490

NXActionNote (class in ryu.ofproto.ofproto_v1_3_parser), 478

NXActionOutputReg (class in ryu.ofproto.ofproto_v1_3_parser), 480

NXActionOutputReg2 (class in ryu.ofproto.ofproto_v1_3_parser), 481

NXActionPopMpls (class in ryu.ofproto.ofproto_v1_3_parser), 491

NXActionPopQueue (class in ryu.ofproto.ofproto_v1_3_parser), 474

NXActionPushMpls (class in ryu.ofproto.ofproto_v1_0_parser), 474

NXActionRegLoad (class in ryu.ofproto.ofproto_v1_3_parser), 477

NXActionRegLoad2 (class in ryu.ofproto.ofproto_v1_3_parser), 477

NXActionRegMove (class in ryu.ofproto.ofproto_v1_3_parser), 477

NXActionResubmit (class in ryu.ofproto.ofproto_v1_3_parser), 479

NXActionResubmitTable (class in ryu.ofproto.ofproto_v1_3_parser), 479

NXActionSample (class in ryu.ofproto.ofproto_v1_3_parser), 485

NXActionSample2 (class in ryu.ofproto.ofproto_v1_3_parser), 486

NXActionSetMplsLabel (class in ryu.ofproto.ofproto_v1_0_parser), 476

NXActionSetMplsTc (class in ryu.ofproto.ofproto_v1_0_parser), 476

NXActionSetMplsTtl (class in ryu.ofproto.ofproto_v1_0_parser), 475

NXActionSetQueue (class in ryu.ofproto.ofproto_v1_0_parser), 473

NXActionStackPop (class in ryu.ofproto.ofproto_v1_3_parser), 475

NXActionStackPush (class in ryu.ofproto.ofproto_v1_3_parser), 478

NXActionStackPop (class in ryu.ofproto.ofproto_v1_3_parser), 484

NXActionStackPush (class in ryu.ofproto.ofproto_v1_3_parser), 484

NXFlowSpecLoad (class in ryu.ofproto.ofproto_v1_3_parser), 492

NXFlowSpecMatch (class in ryu.ofproto.ofproto_v1_3_parser), 492

NXFlowSpecOutput (class in ryu.ofproto.ofproto_v1_3_parser), 492

O

OFPError, 514

ofp_msg_from_jsondict() (in module ryu.ofproto.ofproto_parser), 140

OFF_VERSIONS (ryu.base.app_manager.RyuApp attribute), 495

OFPAction (class in ryu.ofproto.ofproto_v1_0_parser), 161
OFPFlowRemoved (class ryu.ofproto.ofproto_v1_0_parser), 156
OFPFlowRemoved (class ryu.ofproto.ofproto_v1_2_parser), 200
OFPFlowRemoved (class ryu.ofproto.ofproto_v1_3_parser), 264
OFPFlowRemoved (class ryu.ofproto.ofproto_v1_4_parser), 357
OFPFlowRemoved (class ryu.ofproto.ofproto_v1_5_parser), 450
OFPFlowStats (class ryu.ofproto.ofproto_v1_2_parser), 177
OFPFlowStatsReply (class ryu.ofproto.ofproto_v1_0_parser), 148
OFPFlowStatsReply (class ryu.ofproto.ofproto_v1_3_parser), 229
OFPFlowStatsReply (class ryu.ofproto.ofproto_v1_4_parser), 300
OFPFlowStatsReply (class ryu.ofproto.ofproto_v1_5_parser), 397
OFPFlowStatsRequest (class ryu.ofproto.ofproto_v1_0_parser), 148
OFPFlowStatsRequest (class ryu.ofproto.ofproto_v1_2_parser), 176
OFPFlowStatsRequest (class ryu.ofproto.ofproto_v1_3_parser), 228
OFPFlowStatsRequest (class ryu.ofproto.ofproto_v1_4_parser), 299
OFPFlowStatsRequest (class ryu.ofproto.ofproto_v1_5_parser), 396
OFPGetAsyncReply (class ryu.ofproto.ofproto_v1_3_parser), 260
OFPGetAsyncReply (class ryu.ofproto.ofproto_v1_4_parser), 346
OFPGetAsyncReply (class ryu.ofproto.ofproto_v1_5_parser), 446
OFPGetAsyncRequest (class ryu.ofproto.ofproto_v1_3_parser), 260
OFPGetAsyncRequest (class ryu.ofproto.ofproto_v1_4_parser), 346
OFPGetAsyncRequest (class ryu.ofproto.ofproto_v1_5_parser), 445
OFPGetConfigReply (class ryu.ofproto.ofproto_v1_0_parser), 143
OFPGetConfigReply (class ryu.ofproto.ofproto_v1_2_parser), 166
OFPGetConfigReply (class ryu.ofproto.ofproto_v1_3_parser), 214
OFPGetConfigReply (class ryu.ofproto.ofproto_v1_4_parser), 279
OFPGetConfigReply (class ryu.ofproto.ofproto_v1_5_parser), 378
OFPGetConfigRequest (class ryu.ofproto.ofproto_v1_0_parser), 142
in OFPGetConfigRequest (class ryu.ofproto.ofproto_v1_2_parser), 166
in OFPGetConfigRequest (class ryu.ofproto.ofproto_v1_3_parser), 214
in OFPGetConfigRequest (class ryu.ofproto.ofproto_v1_4_parser), 279
in OFPGetConfigRequest (class ryu.ofproto.ofproto_v1_5_parser), 377
in OFGroupDescStats (class ryu.ofproto.ofproto_v1_2_parser), 190
in OFGroupDescStatsReply (class ryu.ofproto.ofproto_v1_3_parser), 245
in OFGroupDescStatsReply (class ryu.ofproto.ofproto_v1_4_parser), 326
in OFGroupDescStatsReply (class ryu.ofproto.ofproto_v1_5_parser), 413
in OFGroupDescStatsRequest (class ryu.ofproto.ofproto_v1_2_parser), 189
in OFGroupDescStatsRequest (class ryu.ofproto.ofproto_v1_3_parser), 245
in OFGroupDescStatsRequest (class ryu.ofproto.ofproto_v1_4_parser), 325
in OFGroupDescStatsRequest (class ryu.ofproto.ofproto_v1_5_parser), 413
in OFGroupFeaturesStats (class ryu.ofproto.ofproto_v1_2_parser), 191
in OFGroupFeaturesStatsReply (class ryu.ofproto.ofproto_v1_3_parser), 247
in OFGroupFeaturesStatsReply (class ryu.ofproto.ofproto_v1_4_parser), 327
in OFGroupFeaturesStatsReply (class ryu.ofproto.ofproto_v1_5_parser), 415
in OFGroupMod (class ryu.ofproto.ofproto_v1_2_parser), 171
in OFGroupMod (class ryu.ofproto.ofproto_v1_3_parser), 223
in OFGroupMod (class ryu.ofproto.ofproto_v1_4_parser), 294
in OFGroupMod (class ryu.ofproto.ofproto_v1_5_parser), 387
in OFGroupStats (class ryu.ofproto.ofproto_v1_2_parser), 189
in OFGroupStatsReply (class ryu.ofproto.ofproto_v1_3_parser), 245
in OFGroupStatsReply (class ryu.ofproto.ofproto_v1_4_parser), 324
OFPPortStats (class in ryu.ofproto.ofproto_v1_0_parser, 155)
OFPPacketIn (class ryu.ofproto.ofproto_v1_2_parser, 198)
OFPPacketIn (class ryu.ofproto.ofproto_v1_3_parser, 262)
OFPPacketIn (class ryu.ofproto.ofproto_v1_4_parser, 349)
OFPPacketIn (class ryu.ofproto.ofproto_v1_5_parser, 448)
OFPPacketOut (class ryu.ofproto.ofproto_v1_0_parser, 153)
OFPPacketOut (class ryu.ofproto.ofproto_v1_2_parser, 194)
OFPPacketOut (class ryu.ofproto.ofproto_v1_3_parser, 255)
OFPPacketOut (class ryu.ofproto.ofproto_v1_4_parser, 337)
OFPPacketOut (class ryu.ofproto.ofproto_v1_5_parser, 434)
OFPPhyPort (class ryu.ofproto.ofproto_v1_0_parser, 159)
OFPPort (class in ryu.ofproto.ofproto_v1_2_parser, 206)
OFPPort (class in ryu.ofproto.ofproto_v1_3_parser, 271)
OFPPort (class in ryu.ofproto.ofproto_v1_4_parser, 369)
OFPPort (class in ryu.ofproto.ofproto_v1_5_parser, 464)
OFPPortDescStatsReply (class ryu.ofproto.ofproto_v1_3_parser, 241)
OFPPortDescStatsReply (class ryu.ofproto.ofproto_v1_4_parser, 316)
OFPPortDescStatsReply (class ryu.ofproto.ofproto_v1_5_parser, 404)
OFPPortDescStatsRequest (class ryu.ofproto.ofproto_v1_3_parser, 241)
OFPPortDescStatsRequest (class ryu.ofproto.ofproto_v1_4_parser, 316)
OFPPortDescStatsRequest (class ryu.ofproto.ofproto_v1_5_parser, 403)
OFPPortMod (class ryu.ofproto.ofproto_v1_0_parser, 145)
OFPPortMod (class ryu.ofproto.ofproto_v1_2_parser, 173)
OFPPortMod (class ryu.ofproto.ofproto_v1_3_parser, 224)
OFPPortMod (class ryu.ofproto.ofproto_v1_4_parser, 295)
OFPPortMod (class ryu.ofproto.ofproto_v1_5_parser, 390)
OFPPortStats (class ryu.ofproto.ofproto_v1_2_parser, 185)
in OFPPortStatsReply (class ryu.ofproto.ofproto_v1_0_parser, 151)
in OFPPortStatsReply (class ryu.ofproto.ofproto_v1_3_parser, 239)
in OFPPortStatsReply (class ryu.ofproto.ofproto_v1_4_parser, 313)
in OFPPortStatsReply (class ryu.ofproto.ofproto_v1_5_parser, 401)
in OFPPortStatsRequest (class ryu.ofproto.ofproto_v1_0_parser, 151)
in OFPPortStatsRequest (class ryu.ofproto.ofproto_v1_2_parser, 184)
in OFPPortStatsRequest (class ryu.ofproto.ofproto_v1_3_parser, 239)
in OFPPortStatsRequest (class ryu.ofproto.ofproto_v1_4_parser, 313)
in OFPPortStatsRequest (class ryu.ofproto.ofproto_v1_5_parser, 401)
in OFPPortStatus (class ryu.ofproto.ofproto_v1_0_parser, 157)
in OFPPortStatus (class ryu.ofproto.ofproto_v1_2_parser, 202)
in OFPPortStatus (class ryu.ofproto.ofproto_v1_3_parser, 266)
in OFPPortStatus (class ryu.ofproto.ofproto_v1_4_parser, 359)
in OFPPortStatus (class ryu.ofproto.ofproto_v1_5_parser, 452)
in OFPPQueueDescStatsReply (class ryu.ofproto.ofproto_v1_4_parser, 322)
in OFPPQueueDescStatsReply (class ryu.ofproto.ofproto_v1_5_parser, 409)
in OFPPQueueDescStatsRequest (class ryu.ofproto.ofproto_v1_4_parser, 321)
in OFPPQueueDescStatsRequest (class ryu.ofproto.ofproto_v1_5_parser, 409)
in OFPPQueueGetConfigReply (class ryu.ofproto.ofproto_v1_0_parser, 147)
in OFPPQueueGetConfigReply (class ryu.ofproto.ofproto_v1_2_parser, 193)
in OFPPQueueGetConfigReply (class ryu.ofproto.ofproto_v1_3_parser, 253)
in OFPPQueueGetConfigRequest (class ryu.ofproto.ofproto_v1_3_parser, 253)
in OFPPQueueGetConfigRequest (class ryu.ofproto.ofproto_v1_0_parser, 147)
in OFPPQueueStats (class ryu.ofproto.ofproto_v1_2_parser, 187)
in OFPPQueueStatsReply (class ryu.ofproto.ofproto_v1_0_parser, 153)
in OFPPQueueStatsReply (class ryu.ofproto.ofproto_v1_3_parser, 243)
OFPTableStatsRequest (class ryu.ofproto.ofproto_v1_3_parser), 237
OFPTableStatsRequest (class ryu.ofproto.ofproto_v1_4_parser), 309
OFPTableStatsRequest (class ryu.ofproto.ofproto_v1_5_parser), 422
OFPTableStatus (class ryu.ofproto.ofproto_v1_4_parser), 362
OFPTableStatus (class ryu.ofproto.ofproto_v1_5_parser), 456
OFPUnparseableMsg (class ryu.lib.packet.openflow), 83
OFPVendor (class in ryu.ofproto.ofproto_v1_0_parser), 159
OFPVendorStatsReply (class ryu.ofproto.ofproto_v1_0_parser), 153
OFPVendorStatsRequest (class ryu.ofproto.ofproto_v1_0_parser), 153
cos_nbits() (in module ryu.ofproto.nicira_ext), 492
openflow (class in ryu.lib.packet.openflow), 83
opt_header (class in ryu.lib.packet.ipv6), 76
OptAttrError, 40
option (class in ryu.lib.packet.dhcp), 58
Option (class in ryu.lib.packet.bmp), 61
option (class in ryu.lib.packet.bmp.data), 76
OptionDataUnknown (class in ryu.lib.packet.bmp.data), 61
options (class in ryu.lib.packet.dhcp), 58
options (class in ryu.lib.packet.bmp), 60
organization_specific_tlv (class in ryu.lib.packet.cfm), 55
OrganizationallySpecific (class in ryu.lib.packet.lldp), 80
ospf (in module ryu.lib.packet ospf), 86
OSPFDBDesc (class in ryu.lib.packet ospf), 84
OSPFHello (class in ryu.lib.packet ospf), 84
OSPFLSAck (class in ryu.lib.packet ospf), 84
OSPFLSReq (class in ryu.lib.packet ospf), 85
OSPFLSUpd (class in ryu.lib.packet ospf), 85
OSPFMessage (class in ryu.lib.packet ospf), 85
OtherConfChange, 40
out_filter_get() (ryu.services.protocols bgp.BGPSpeaker.BGPError method), 125
out_filter_set() (ryu.services.protocols bgp.BGPSpeaker.BGPError method), 126
OutOfResource, 40
OVSBridge (class in ryu.lib.ovs.bridge), 135
OVSBridgeNotFound, 138

P
pack() (ryu.lib.packet.bfd.bfd method), 26
Packet (class in ryu.lib.packet.packet), 18
PacketBase (class in ryu.lib.packet.packet_base), 19

in param_cookie_preserve (class in ryu.lib.packet.sctp), 95
in param_ecn (class in ryu.lib.packet.sctp), 95
in param_ecn (class in ryu.lib.packet.sctp), 95
in param_heartbeat (class in ryu.lib.packet.sctp), 95
in param_host_addr (class in ryu.lib.packet.sctp), 96
in param_ipv4 (class in ryu.lib.packet.sctp), 96
in param_ipv6 (class in ryu.lib.packet.sctp), 96
in param_state_cookie (class in ryu.lib.packet.sctp), 97
in param_supported_addr (class in ryu.lib.packet.sctp), 97
in param_unrecognized_param (class in ryu.lib.packet.sctp), 97

parse() (ryu.lib.packet.stream_parser.StreamParser method), 19
parser() (ryu.lib.packet.arp.arp class method), 20
parser() (ryu.lib.packet.bfd.bfd class method), 26
parser() (ryu.lib.packet.bgp.BGPKeepAlive class method), 28
parser() (ryu.lib.packet.bgp.BGPMessage class method), 28
parser() (ryu.lib.packet.bgp.BGPNotification class method), 29
parser() (ryu.lib.packet.bgp.BGPRouteRefresh class method), 30
parser() (ryu.lib.packet.bgp.BGPUpdate class method), 31
parser() (ryu.lib.packet.bmp.BMPInitiation class method), 42
parser() (ryu.lib.packet.bmp.BMPMessage class method), 42
parser() (ryu.lib.packet.bmp.BMPPeerDownNotification class method), 43
parser() (ryu.lib.packet.bmp.BMPPeerMessage class method), 43
parser() (ryu.lib.packet.bmp.BMPPeerUpNotification class method), 44
parser() (ryu.lib.packet.bmp.BMPRouteMonitoring class method), 45
parser() (ryu.lib.packet.bmp.BMPServicesReport class method), 45
parser() (ryu.lib.packet.bmp.BMPTermination class method), 46
parser() (ryu.lib.packet.bmp.bpdus.bpu class method), 51
parser() (ryu.lib.packet.bmp.bpdus.ConfigurationBPDUs class method), 49
parser() (ryu.lib.packet.bmp.bpdus.RstBPDUs class method), 50
parser() (ryu.lib.packet.bmp.topology.ChangeNotificationBPDUs class method), 50
parser() (ryu.lib.packet.cfm.cfm class method), 52
parser() (ryu.lib.packet.dhcp.dhcp class method), 57
Ryu

Reader (class in ryu.lib.mrtlib), 130
Reader (class in ryu.lib.pcaplib), 116
register_packet_type()
    (ryu.lib.packet.packet_base.PacketBase class method), 20
RegisteredNexthop (class in ryu.lib.packet.zebra), 109
remove_db_attribute()
    (ryu.lib.ovs.bridge.OVSBridge method), 138
    reply_egress_tlv (class in ryu.lib.packet.cfm), 55
    reply_ingress_tlv (class in ryu.lib.packet.cfm), 56
    reply_to_request()
    (ryu.base.app_manager.RyuApp method), 495
    RestVtepController (class in ryu.app.rest_vtep), 571
    rib_get() (ryu.services.protocols.bgp.bgpspeaker.BGPSpeaker method), 126
    RouteTargetMembershipNLRI (class in ryu.lib.packet.bgp), 41
    routing (class in ryu.lib.packet.ipv6), 76
    routing_type3 (class in ryu.lib.packet.ipv6), 76
    RoutingLoop, 41
    RstBDUs (class in ryu.lib.packet.bpu), 49
    run_command() (ryu.lib.ovs.bridge.OVSBridge method), 138
    run_command() (ryu.lib.ovs.vscit.VSCIt method), 134
    ryu.app.cbench (module), 9
    ryu.app.ofctl.api (module), 513
    ryu.app.ofctl.exception (module), 514
    ryu.app.rest_vtep (module), 567
    ryu.app.simple_switch (module), 9
    ryu.base.app_manager (module), 7
    ryu.controller.controller (module), 7
    ryu.controller.dpset (module), 7
    ryu.controller.ofp_event (module), 7
    ryu.controller.ofp_handler (module), 7
    ryu.lib.netconf (module), 9
    ryu.lib.of_conf (module), 9
    ryu.lib.ovs (module), 9

Index 601
serialize() (ryu.lib.packet.lldp.lldp method), 79
serialize() (ryu.lib.packet.lldp.lldp method), 82
serialize() (ryu.lib.packet.mpls.mpls method), 83
serialize() (ryu.lib.packet.openflow.openflow method), 84
serialize() (ryu.lib.packet.ospf.OSPFFrame method), 86
serialize() (ryu.lib.packet.packet.Packet method), 88
serialize() (ryu.lib.packet.packet_base.PacketBase method), 90
serialize() (ryu.lib.packet.pbb.itag method), 95
serialize() (ryu.lib.packet.sctp.sctp method), 98
serialize() (ryu.lib.packet.slow.lacp method), 100
serialize() (ryu.lib.packet.tcp.tcp method), 102
serialize() (ryu.lib.packet.udp.udp method), 104
serialize() (ryu.lib.packet.vrrp.vrrp method), 106
serialize() (ryu.lib.packet.vxlan.vxlan method), 108
serialize() (ryu.lib.packet.zebra.ZebraMessage method), 110
serialize_hdr() (ryu.lib.packet.bgp.BFDAuth method), 112
set_controller() (ryu.lib.ovs.bridge.OVSBridge method), 116
set_db_attribute() (ryu.lib.ovs.bridge.OVSBridge method), 118
set_ev_cls() (in module ryu.controller.handler), 120
set_qos() (ryu.lib.ovs.bridge.OVSBridge method), 122
shutdown() (ryu.services.protocols.bgp.bgpSpeaker.BGPMessage method), 126
SimplePassword (class in ryu.lib.packet.bgp.BgpExc attribute), 128
slow() (class in ryu.lib.packet.slow), 130
start() (class in ryu.base.app_manager.RyuApp method), 132
start() (ryu.services.protocols.bgp.bgpSpeaker.BGPMessage method), 134
StreamParser (class in ryu.lib.packet.bgp), 136
StreamParser (class in ryu.lib.packet.stream_parser), 138
StreamParser.TooSmallException, 139
SUB_CODE (ryu.lib.packet.bgp.BgpExc attribute), 143
svlan (class in ryu.lib.packet.vlan), 145
SystemCapabilities (class in ryu.lib.packet.lldp), 147
SystemDescription (class in ryu.lib.packet.lldp), 149
SystemName (class in ryu.lib.packet.lldp), 151
T
tcp (class in ryu.lib.packet.tcp), 153
TimeExceeded (class in ryu.lib.packet.icmp), 155
to_jsondict() (ryu.ofproto.ofproto_parser.MsgBase method), 157
TopologyChangeNotificationBPDUs (class in ryu.lib.packet.udp), 159
try_parse() (ryu.lib.packet.bgp.StreamParser method), 161
try_parse() (ryu.lib.packet.stream_parser.StreamParser method), 163
TTL (class in ryu.lib.packet.lldp), 165
U
tcp (class in ryu.lib.packet.tcp), 167
UDP (class in ryu.lib.packet.udp), 169
UnacceptableHoldTime, 171
UnexpectedMultiReply, 173
UnRegWellKnowAttr, 175
UnsupportedOptParam, 177
UnsupportedVersion, 179
V
valid_ovsdb_addr() (in module ryu.ovs.ovslib), 181
validate_rpc_host() (in module ryu.services.protocols.bgp.application), 183
vnl_from_bin() (in module ryu.lib.packet.vlan), 185
vnl_to_bin() (in module ryu.lib.packet.vlan), 187
vrf_add() (ryu.services.protocols.bgp.bgpSpeaker.BGPMessage method), 189
vrf_del() (ryu.services.protocols.bgp.bgpSpeaker.BGPMessage method), 191
vrs_get() (ryu.services.protocols.bgp.bgpSpeaker.BGPMessage method), 193
VRF (class in ryu.lib.packet.vrrp), 195
vrrpv2 (class in ryu.lib.packet.vrrp), 197
vrrpv3 (class in ryu.lib.packet.vrrp), 199
VSCtlCommand (class in ryu.lib.packet.zebra), 201
VXLAN (class in ryu.lib.packet.vxlan), 203
W
Writer (class in ryu.lib.mrlib), 205
Writer (class in ryu.lib.pcaplib), 207
Z
zebra (module in ryu.lib.packet.zebra), 209
ZebraBfdClientRegister (class in ryu.lib.packet.zebra), 211
ZebraBfdDestinationRegister (class in ryu.lib.packet.zebra), 213
ZebraBfdDestinationRegister (class in ryu.lib.packet.zebra), 215
ZebraBfdDestinationReply (class in ryu.lib.packet.zebra), 217
ZebraBfdDestinationUpdate (class in ryu.lib.packet.zebra), 219
ZebraHello (class in ryu.lib.packet.zebra), 221
Index

ryu Documentation, Release 4.34
<table>
<thead>
<tr>
<th>Function</th>
<th>Class</th>
<th>Module</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZebraImportCheckUpdate</td>
<td>(class in ryu.lib.packet.zebra)</td>
<td>ryu.lib.packet.zebra</td>
<td>111</td>
</tr>
<tr>
<td>ZebraImportRouteRegister</td>
<td>(class in ryu.lib.packet.zebra)</td>
<td>ryu.lib.packet.zebra</td>
<td>112</td>
</tr>
<tr>
<td>ZebraImportRouteUnregister</td>
<td>(class in ryu.lib.packet.zebra)</td>
<td>ryu.lib.packet.zebra</td>
<td>112</td>
</tr>
<tr>
<td>ZebraInterfaceAdd</td>
<td>(class in ryu.lib.packet.zebra)</td>
<td>ryu.lib.packet.zebra</td>
<td>112</td>
</tr>
<tr>
<td>ZebraInterfaceAddressAdd</td>
<td>(class in ryu.lib.packet.zebra)</td>
<td>ryu.lib.packet.zebra</td>
<td>112</td>
</tr>
<tr>
<td>ZebraInterfaceAddressDelete</td>
<td>(class in ryu.lib.packet.zebra)</td>
<td>ryu.lib.packet.zebra</td>
<td>112</td>
</tr>
<tr>
<td>ZebraInterfaceBfdDestinationUpdate</td>
<td>(class in ryu.lib.packet.zebra)</td>
<td>ryu.lib.packet.zebra</td>
<td>112</td>
</tr>
<tr>
<td>ZebraInterfaceDelete</td>
<td>(class in ryu.lib.packet.zebra)</td>
<td>ryu.lib.packet.zebra</td>
<td>112</td>
</tr>
<tr>
<td>ZebraInterfaceDisableRadv</td>
<td>(class in ryu.lib.packet.zebra)</td>
<td>ryu.lib.packet.zebra</td>
<td>112</td>
</tr>
<tr>
<td>ZebraInterfaceDown</td>
<td>(class in ryu.lib.packet.zebra)</td>
<td>ryu.lib.packet.zebra</td>
<td>112</td>
</tr>
<tr>
<td>ZebraInterfaceEnableRadv</td>
<td>(class in ryu.lib.packet.zebra)</td>
<td>ryu.lib.packet.zebra</td>
<td>112</td>
</tr>
<tr>
<td>ZebraInterfaceLinkParams</td>
<td>(class in ryu.lib.packet.zebra)</td>
<td>ryu.lib.packet.zebra</td>
<td>112</td>
</tr>
<tr>
<td>ZebraInterfaceNbrAddressAdd</td>
<td>(class in ryu.lib.packet.zebra)</td>
<td>ryu.lib.packet.zebra</td>
<td>112</td>
</tr>
<tr>
<td>ZebraInterfaceNbrAddressDelete</td>
<td>(class in ryu.lib.packet.zebra)</td>
<td>ryu.lib.packet.zebra</td>
<td>113</td>
</tr>
<tr>
<td>ZebraInterfaceUp</td>
<td>(class in ryu.lib.packet.zebra)</td>
<td>ryu.lib.packet.zebra</td>
<td>113</td>
</tr>
<tr>
<td>ZebraIPv4ImportLookup</td>
<td>(class in ryu.lib.packet.zebra)</td>
<td>ryu.lib.packet.zebra</td>
<td>110</td>
</tr>
<tr>
<td>ZebraIPv4NexthopAdd</td>
<td>(class in ryu.lib.packet.zebra)</td>
<td>ryu.lib.packet.zebra</td>
<td>110</td>
</tr>
<tr>
<td>ZebraIPv4NexthopDelete</td>
<td>(class in ryu.lib.packet.zebra)</td>
<td>ryu.lib.packet.zebra</td>
<td>110</td>
</tr>
<tr>
<td>ZebraIPv4NextHopLookup</td>
<td>(class in ryu.lib.packet.zebra)</td>
<td>ryu.lib.packet.zebra</td>
<td>110</td>
</tr>
<tr>
<td>ZebraIPv4NextHopLookupMRib</td>
<td>(class in ryu.lib.packet.zebra)</td>
<td>ryu.lib.packet.zebra</td>
<td>110</td>
</tr>
<tr>
<td>ZebraIPv4RouteAdd</td>
<td>(class in ryu.lib.packet.zebra)</td>
<td>ryu.lib.packet.zebra</td>
<td>110</td>
</tr>
<tr>
<td>ZebraIPv4RouteDelete</td>
<td>(class in ryu.lib.packet.zebra)</td>
<td>ryu.lib.packet.zebra</td>
<td>110</td>
</tr>
<tr>
<td>ZebraIPv4RouteIPv6NextHopAdd</td>
<td>(class in ryu.lib.packet.zebra)</td>
<td>ryu.lib.packet.zebra</td>
<td>110</td>
</tr>
<tr>
<td>ZebraIPv6ImportLookup</td>
<td>(class in ryu.lib.packet.zebra)</td>
<td>ryu.lib.packet.zebra</td>
<td>110</td>
</tr>
<tr>
<td>ZebraIPv6NexthopAdd</td>
<td>(class in ryu.lib.packet.zebra)</td>
<td>ryu.lib.packet.zebra</td>
<td>110</td>
</tr>
<tr>
<td>ZebraIPv6NexthopDelete</td>
<td>(class in ryu.lib.packet.zebra)</td>
<td>ryu.lib.packet.zebra</td>
<td>110</td>
</tr>
<tr>
<td>ZebraMessage</td>
<td>(class in ryu.lib.packet.zebra)</td>
<td>ryu.lib.packet.zebra</td>
<td>113</td>
</tr>
<tr>
<td>ZebraMplsLabelsAdd</td>
<td>(class in ryu.lib.packet.zebra)</td>
<td>ryu.lib.packet.zebra</td>
<td>114</td>
</tr>
<tr>
<td>ZebraMplsLabelsDelete</td>
<td>(class in ryu.lib.packet.zebra)</td>
<td>ryu.lib.packet.zebra</td>
<td>114</td>
</tr>
<tr>
<td>ZebraNexthopRegister</td>
<td>(class in ryu.lib.packet.zebra)</td>
<td>ryu.lib.packet.zebra</td>
<td>114</td>
</tr>
<tr>
<td>ZebraNexthopUnregister</td>
<td>(class in ryu.lib.packet.zebra)</td>
<td>ryu.lib.packet.zebra</td>
<td>114</td>
</tr>
<tr>
<td>ZebraRedistributeAdd</td>
<td>(class in ryu.lib.packet.zebra)</td>
<td>ryu.lib.packet.zebra</td>
<td>114</td>
</tr>
<tr>
<td>ZebraRedistributeDefaultAdd</td>
<td>(class in ryu.lib.packet.zebra)</td>
<td>ryu.lib.packet.zebra</td>
<td>114</td>
</tr>
<tr>
<td>ZebraRedistributeDelete</td>
<td>(class in ryu.lib.packet.zebra)</td>
<td>ryu.lib.packet.zebra</td>
<td>114</td>
</tr>
<tr>
<td>ZebraRedistributeIPv4Add</td>
<td>(class in ryu.lib.packet.zebra)</td>
<td>ryu.lib.packet.zebra</td>
<td>114</td>
</tr>
<tr>
<td>ZebraRedistributeIPv6Add</td>
<td>(class in ryu.lib.packet.zebra)</td>
<td>ryu.lib.packet.zebra</td>
<td>115</td>
</tr>
<tr>
<td>ZebraRedistributeIPv6Delete</td>
<td>(class in ryu.lib.packet.zebra)</td>
<td>ryu.lib.packet.zebra</td>
<td>115</td>
</tr>
<tr>
<td>ZebraRouterIDAdd</td>
<td>(class in ryu.lib.packet.zebra)</td>
<td>ryu.lib.packet.zebra</td>
<td>115</td>
</tr>
<tr>
<td>ZebraRouterIDDelete</td>
<td>(class in ryu.lib.packet.zebra)</td>
<td>ryu.lib.packet.zebra</td>
<td>115</td>
</tr>
<tr>
<td>ZebraRouterIDUpdate</td>
<td>(class in ryu.lib.packet.zebra)</td>
<td>ryu.lib.packet.zebra</td>
<td>115</td>
</tr>
<tr>
<td>ZebraUnknownMessage</td>
<td>(class in ryu.lib.packet.zebra)</td>
<td>ryu.lib.packet.zebra</td>
<td>115</td>
</tr>
<tr>
<td>ZebraVrfAdd</td>
<td>(class in ryu.lib.packet.zebra)</td>
<td>ryu.lib.packet.zebra</td>
<td>115</td>
</tr>
<tr>
<td>ZebraVrfDelete</td>
<td>(class in ryu.lib.packet.zebra)</td>
<td>ryu.lib.packet.zebra</td>
<td>115</td>
</tr>
<tr>
<td>ZebraVrfUnregister</td>
<td>(class in ryu.lib.packet.zebra)</td>
<td>ryu.lib.packet.zebra</td>
<td>115</td>
</tr>
</tbody>
</table>