ryu Documentation

Release 4.13

ryu development team

May 22, 2020

Contents

Contents:

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 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:

```
% pip install ryu
```

If you prefer to install Ryu from the source code:

```
% git clone git://github.com/osrg/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:

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

1.3 Optional Requirements

Some functionalities of ryu requires 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 the functionalities, please install requirements:

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

Please refer to tools/optional-requires for details.

1.4 Support

Ryu Official site is http://osrg.github.io/ryu/.

If you have any questions, suggestions, and patches, the mailing list is available at ryu-devel ML. The ML archive at Gmane is also available.

CHAPTER 2

Writing Your Ryu Application

2.1 The First Application

2.1.1 Whetting Your Appetite

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

Writing Ryu application is easy. It's just Python scripts.

2.1.2 Start Writing

We show a Ryu application that make OpenFlow switches work as a dumb layer 2 switch.

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

```
from ryu.base import app_manager

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

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

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

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

All you have to do is defining needs a new subclass of RyuApp to run your Python script as a Ryu application.

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

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

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

The first argument of the decorator indicates an event that makes function called. As you expect easily, every time Ryu gets a packet_in message, this function is called.

The second argument indicates the state of the switch. Probably, you want to ignore packet_in messages before the negotiation between Ryu and the switch finishes. Using 'MAIN_DISPATCHER' as the second argument means this function is called only after the negotiation completes.

Next let's look at the first half of the 'packet_in_handler' function.

- ev.msg is an object that represents a packet_in data structure.
- msg.dp is an object that represents a datapath (switch).
- dp.ofproto and dp.ofproto_parser are objects that represent the OpenFlow protocol that 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 need a switch to send out of all the ports so OFPP_FLOOD constant is used.
- OFPPacketOut class is used to build a packet_out message.
- If you call Datapath class's send_msg method with a OpenFlow message class object, Ryu builds and send the on-wire data format to the switch.

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

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

2.2.3 OpenFlow controller

ryu.controller.controller

ryu.controller.dpset

ryu.controller.ofp_event

ryu.controller.ofp_handler

2.2.4 OpenFlow wire protocol encoder and decoder

ryu.ofproto.ofproto_v1_0

OpenFlow 1.0 definitions.

ryu.ofproto.ofproto_v1_0_parser

ryu.ofproto.ofproto_v1_2

ryu.ofproto.ofproto_v1_2_parser

ryu.ofproto.ofproto_v1_3

ryu.ofproto.ofproto_v1_3_parser

ryu.ofproto.ofproto_v1_4

ryu.ofproto.ofproto_v1_4_parser

ryu.ofproto.ofproto_v1_5

ryu.ofproto.ofproto_v1_5_parser

2.2.5 Ryu applications

ryu.app.cbench

ryu.app.simple_switch

ryu.topology

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

2.2.6 Libraries

ryu.lib.packet

ryu.lib.ovs

ovsdb interaction library.

ryu.lib.of_config

OF-Config implementation.

ryu.lib.netconf

NETCONF definitions used by ryu/lib/of_config.

ryu.lib.xflow

An implementation of sFlow and NetFlow.

2.2.7 Third party libraries

ryu.contrib.ovs

Open vSwitch python binding. Used by ryu.lib.ovs.

ryu.contrib.oslo.config

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

ryu.contrib.ncclient

Python library for NETCONF client. Used by ryu.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 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 keep draining the receive queue by dequeueing an event and calling the appropritate event handler for the event type. Because the event handler is called in the context of the event processing thread, it should be careful for blocking. I.e. 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 machinary 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 subclass of the following class.

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.

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

2.3.8 ryu.controller.controller.Datapath

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

2.3.13 ryu.controller.ofp_event.EventOFPPortStateChange

2.3.14 ryu.controller.dpset.EventDP

2.3.15 ryu.controller.dpset.EventPortAdd

2.3.16 ryu.controller.dpset.EventPortDelete

2.3.17 ryu.controller.dpset.EventPortModify

2.3.18 ryu.controller.network.EventNetworkPort

2.3.19 ryu.controller.network.EventNetworkDel

2.3.20 ryu.controller.network.EventMacAddress

2.3.21 ryu.controller.tunnels.EventTunnelKeyAdd

2.3.22 ryu.controller.tunnels.EventTunnelKeyDel

2.3.23 ryu.controller.tunnels.EventTunnelPort

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:74:e7', '192.0.2.1', 'fe80::a60:6eff:fe7f:74e7'.

Parsing Packet

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

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

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.

Protocol Header classes

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.

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.

Attribute	Description
label	Label Value
exp	Experimental Use
bsb	Bottom of Stack
ttl	Time To Live

classmethod parser(*buf*)
Decode a protocol header.

This method is used only when decoding a packet.

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

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

serialize(*payload*, *prev*)
Encode a protocol header.

This method is used only when encoding a packet.

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

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

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

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.

Attribute	Description	Example
hwtype	ar$hrd	
proto	ar$pro	
hlen	ar$hln	
plen	ar$pln	
opcode	ar$op	
src_mac	ar$sha	'08:60:6e:7f:74:e7'
src_ip	ar$spa	'192.0.2.1'
dst_mac	ar$tha	'00:00:00:00:00:00'
dst_ip	ar$tpa	'192.0.2.2'

classmethod parser(*buf*)
Decode a protocol header.

This method is used only when decoding a packet.

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

- An object to describe the decoded header.

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

serialize(*payload*, *prev*)
Encode a protocol header.

This method is used only when encoding a packet.

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

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

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

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.

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

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.

Attribute	Description
data_len	data length
data	Internet Header + leading octets of original datagram

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.

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

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.

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

class ryu.lib.packet.icmp.**icmp**(*type_=8*, *code=0*, *csum=0*, *data=None*)
ICMP (RFC 792) header encoder/decoder class.

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

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

classmethod parser(*buf*)
Decode a protocol header.

This method is used only when decoding a packet.

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

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

serialize(*payload*, *prev*)
Encode a protocol header.

This method is used only when encoding a packet.

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

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

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

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

Parameters **buf** – binary representation of VNI.

Returns VNI integer.

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

Parameters **vni** – integer of VNI

Returns binary representation of VNI.

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.

Attribute	Description
vni	VXLAN Network Identifier

classmethod parser(*buf*)
Decode a protocol header.

This method is used only when decoding a packet.

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

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

serialize(*payload*, *prev*)
Encode a protocol header.

This method is used only when encoding a packet.

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

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

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

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.

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

classmethod parser(*buf*)
Decode a protocol header.

This method is used only when decoding a packet.

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

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

serialize(*payload=None*, *prev=None*)
Encode a protocol header.

This method is used only when encoding a packet.

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

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

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

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.

Attribute	Description
src_port	Source Port
dst_port	Destination Port
total_length	Length (0 means automatically-calculate when encoding)
csum	Checksum (0 means automatically-calculate when encoding)

static get_packet_type(*src_port*, *dst_port*)
Per-protocol dict-like get method.

Provided for convenience of protocol implementers. Internal use only.

classmethod parser(*buf*)
Decode a protocol header.

This method is used only when decoding a packet.

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

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

serialize(*payload*, *prev*)
Encode a protocol header.

This method is used only when encoding a packet.

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

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

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

DHCP packet parser/serializer

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

classmethod parser(*buf*)

Decode a protocol header.

This method is used only when decoding a packet.

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

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

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

Encode a protocol header.

This method is used only when encoding a packet.

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

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

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

class 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.

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

class 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.

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

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.

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

classmethod parser(*buf*)

Decode a protocol header.

This method is used only when decoding a packet.

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

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

serialize(*payload=None*, *prev=None*)
Encode a protocol header.

This method is used only when encoding a packet.

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

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

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

class ryu.lib.packet.dhcp6.**options**(*option_list=None*, *options_len=0*)
DHCP (RFC 3315) options encoder/decoder class.

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

class ryu.lib.packet.dhcp6.**option**(*code*, *data*, *length=0*)
DHCP (RFC 3315) options encoder/decoder class.

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

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

The format of DHCP options is:

0 1 2 3 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 +-+ | option-code | option-len | +-+ | option-data | | (option-len octets) | +-+

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

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.

http://www.ietf.org/rfc/rfc1112.txt

http://www.ietf.org/rfc/rfc2236.txt

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

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

classmethod parser(*buf*)
Decode a protocol header.

This method is used only when decoding a packet.

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

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

serialize(*payload*, *prev*)
Encode a protocol header.

This method is used only when encoding a packet.

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

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

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

class 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.

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

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

Attribute	Description
msgtype	a message type for v3.
maxresp	max response time in unit of 1/10 second.
csum	a check sum value. 0 means automatically-calculate when encoding.
address	a group address value.
s_flg	when set to 1, routers suppress the timer process.
qrv	robustness variable for a querier.
qqic	an interval time for a querier in unit of seconds.
num	a number of the multicast servers.
srcs	a list of IPv4 addresses of the multicast servers.

classmethod parser(*buf*)
Decode a protocol header.

This method is used only when decoding a packet.

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

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

serialize(*payload*, *prev*)
Encode a protocol header.

This method is used only when encoding a packet.

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

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

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

class ryu.lib.packet.igmp.**igmpv3_report**(*msgtype=34*, *csum=0*, *record_num=0*, *records=None*)
Internet Group Management Protocol(IGMP, RFC 3376) Membership Report message encoder/decoder class.

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

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

Attribute	Description
msgtype	a message type for v3.
csum	a check sum value. 0 means automatically-calculate when encoding.
record_num	a number of the group records.
records	a list of ryu.lib.packet.igmp.igmpv3_report_group. None if no records.

classmethod parser(*buf*)
Decode a protocol header.

This method is used only when decoding a packet.

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

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

serialize(*payload*, *prev*)
Encode a protocol header.

This method is used only when encoding a packet.

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

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

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

class ryu.lib.packet.igmp.**igmpv3_report_group**(*type_=0*, *aux_len=0*, *num=0*, *address='0.0.0.0'*, *srcs=None*, *aux=None*)
Internet Group Management Protocol(IGMP, RFC 3376) Membership Report Group Record message encoder/decoder class.

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

This is used with 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.

Attribute	Description
type_	a group record type for v3.
aux_len	the length of the auxiliary data.
num	a number of the multicast servers.
address	a group address value.
srcs	a list of IPv4 addresses of the multicast servers.
aux	the auxiliary data.

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.

class ryu.lib.pcaplib.**Reader**(*file_obj*)
PCAP file reader

Argument	Description
file_obj	File object which reading PCAP file in binary mode

Example of usage:

```
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.

class ryu.lib.pcaplib.**Writer**(*file_obj*, *snaplen=65535*, *network=1*)
PCAP file writer

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

Example of usage:

```
...
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 NETCONFIG and OFConfig

XML schema files for NETCONF and OFConfig are stolen from LINC whose licence is Apache 2.0. It supports only part of OFConfig so that its schema files are (intentionally) limited such that operation attributes are allowed only in several limited places. Once our library is tested with other OFConfig switches, the schema files should be updated to allow operation attribute in more places.

References

- NETCONF ietf,
- NETCONF ietf wiki,
- OF-Config spec,
- ncclient,
- ncclient repo,
- LINC git repo

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) + '.0/24'
        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

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.

Writing MRT file

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

2.4.8 OVSDB Manager library

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.

Example

The following logs all new OVSDB connections and allows creating a port on a bridge.

```
import uuid

from ryu.base import app_manager
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
        self.logger.info('New OVSDB connection from system id %s',
                         systemd_id)

    def create_port(self, systemd_id, bridge_name, name):
        new_iface_uuid = uuid.uuid4()
        new_port_uuid = uuid.uuid4()

        def _create_port(tables, insert):
            bridge = ovsdb.row_by_name(self, system_id, bridge_name)

            iface = insert(tables['Interface'], new_iface_uuid)
            iface.name = name
            iface.type = 'internal'
```

(continues on next page)

(continued from previous page)

```
            port = insert(tables['Port'], new_port_uuid)
            port.name = name
            port.interfaces = [iface]

            brdige.ports = bridfe.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 reply.insert_uuid[new_port_uuid]
```

2.5 OpenFlow protocol API Reference

2.5.1 OpenFlow version independent classes and functions

Base class for OpenFlow messages

Functions

2.5.2 OpenFlow v1.0 Messages and Structures

Controller-to-Switch Messages

Handshake

Switch Configuration

Modify State Messages

Queue Configuration Messages

Read State Messages

Send Packet Message

Barrier Message

Asynchronous Messages

Packet-In Message

Flow Removed Message

Port Status Message

Error Message

Symmetric Messages

Hello

Echo Request

Echo Reply

Vendor

Port Structures

Flow Match Structure

Action Structures

2.5.3 OpenFlow v1.2 Messages and Structures

Controller-to-Switch Messages

Handshake

Switch Configuration

Flow Table Configuration

Modify State Messages

Read State Messages

Queue Configuration Messages

Packet-Out Message

Barrier Message

Role Request Message

Asynchronous Messages

Packet-In Message

Flow Removed Message

Port Status Message

Error Message

Symmetric Messages

Hello

Echo Request

Echo Reply

Experimenter

Port Structures

Flow Match Structure

Flow Instruction Structures

Action Structures

2.5.4 OpenFlow v1.3 Messages and Structures

Controller-to-Switch Messages

Handshake

Switch Configuration

Flow Table Configuration

Modify State Messages

Multipart Messages

Queue Configuration Messages

Packet-Out Message

Barrier Message

Role Request Message

Set Asynchronous Configuration Message

Asynchronous Messages

Packet-In Message

Flow Removed Message

Port Status Message

Error Message

Symmetric Messages

Hello

Echo Request

Echo Reply

Experimenter

Port Structures

Flow Match Structure

Flow Instruction Structures

Action Structures

2.5.5 OpenFlow v1.4 Messages and Structures

Controller-to-Switch Messages

Handshake

Switch Configuration

Modify State Messages

Multipart Messages

Packet-Out Message

Barrier Message

Role Request Message

Bundle Messages

Set Asynchronous Configuration Message

Asynchronous Messages

Packet-In Message

Flow Removed Message

Port Status Message

Controller Role Status Message

Table Status Message

Request Forward Message

Symmetric Messages

Hello

Echo Request

Echo Reply

Error Message

Experimenter

Port Structures

Flow Match Structure

Flow Instruction Structures

Action Structures

2.5.6 OpenFlow v1.5 Messages and Structures

Controller-to-Switch Messages

Handshake

Switch Configuration

Modify State Messages

Multipart Messages

Controller Status Structure

2.6 Nicira Extension Structures

2.6.1 Nicira Extension Actions Structures

The followings shows the supported NXAction classes only in OpenFlow1.0

The followings shows the supported NXAction classes in OpenFlow1.0 or later

ryu.ofproto.nicira_ext.**ofs_nbits**(*start*, *end*)

The utility method for ofs_nbits

This method is used in the class to set the ofs_nbits.

This method converts start/end bits into ofs_nbits required to specify the bit range of OXM/NXM fields.

ofs_nbits can be calculated as following:

```
ofs_nbits = (start << 6) + (end - start)
```

The parameter start/end means the OXM/NXM field of ovs-ofctl command.

field[*start*..*end*]

Attribute	Description
start	Start bit for OXM/NXM field
end	End bit for OXM/NXM field

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.

Argument	Value	Description
in_port_nxm	Integer 16bit	OpenFlow port number.
eth_dst_nxm	MAC address	Ethernet destination address.
eth_src_nxm	MAC address	Ethernet source address.
eth_type_nxm	Integer 16bit	Ethernet type. Needed to support Nicira extensions that require the eth_type to be set. (i.e. tcp_
vlan_tci	Integer 16bit	VLAN TCI. Basically same as vlan_vid plus vlan_pcp.
nw_tos	Integer 8bit	IP ToS or IPv6 traffic class field dscp. Requires setting fields: eth_type_nxm = [0x0800 (IPv4)
ip_proto_nxm	Integer 8bit	IP protocol. Needed to support Nicira extensions that require the ip_proto to be set. (i.e. tcp_fl
ipv4_src_nxm	IPv4 address	IPv4 source address. Requires setting fields: eth_type_nxm = 0x0800 (IPv4)
ipv4_dst_nxm	IPv4 address	IPv4 destination address. Requires setting fields: eth_type_nxm = 0x0800 (IPv4)
tcp_src_nxm	Integer 16bit	TCP source port. Requires setting fields: eth_type_nxm = [0x0800 (IPv4)|0x86dd (IPv6)] and
tcp_dst_nxm	Integer 16bit	TCP destination port. Requires setting fields: eth_type_nxm = [0x0800 (IPv4)|0x86dd (IPv6)]
udp_src_nxm	Integer 16bit	UDP source port. Requires setting fields: eth_type_nxm = [0x0800 (IPv4)|0x86dd (IPv6)] and
udp_dst_nxm	Integer 16bit	UDP destination port. eth_type_nxm = [0x0800 (IPv4)|0x86dd (IPv6)] and ip_proto_nxm = 17
icmpv4_type_nxm	Integer 8bit	Type matches the ICMP type and code matches the ICMP code. Requires setting fields: eth_ty

Table 1 – continued from previous pa

Argument	Value	Description
icmpv4_code_nxm	Integer 8bit	Type matches the ICMP type and code matches the ICMP code. Requires setting fields: eth_ty
arp_op_nxm	Integer 16bit	Only ARP opcodes between 1 and 255 should be specified for matching. Requires setting field
arp_spa_nxm	IPv4 address	An address may be specified as an IP address or host name. Requires setting fields: eth_type_r
arp_tpa_nxm	IPv4 address	An address may be specified as an IP address or host name. Requires setting fields: eth_type_r
tunnel_id_nxm	Integer 64bit	Tunnel identifier.
arp_sha_nxm	MAC address	An address is specified as 6 pairs of hexadecimal digits delimited by colons. Requires setting f
arp_tha_nxm	MAC address	An address is specified as 6 pairs of hexadecimal digits delimited by colons. Requires setting f
ipv6_src_nxm	IPv6 address	IPv6 source address. Requires setting fields: eth_type_nxm = 0x86dd (IPv6)
ipv6_dst_nxm	IPv6 address	IPv6 destination address. Requires setting fields: eth_type_nxm = 0x86dd (IPv6)
icmpv6_type_nxm	Integer 8bit	Type matches the ICMP type and code matches the ICMP code. Requires setting fields: eth_ty
icmpv6_code_nxm	Integer 8bit	Type matches the ICMP type and code matches the ICMP code. Requires setting fields: eth_ty
nd_target	IPv6 address	The target address ipv6. Requires setting fields: eth_type_nxm = 0x86dd (IPv6) and ip_proto_
nd_sll	MAC address	The source link-layer address option. Requires setting fields: eth_type_nxm = 0x86dd (IPv6) a
nd_tll	MAC address	The target link-layer address option. Requires setting fields: eth_type_nxm = 0x86dd (IPv6) ar
ip_frag	Integer 8bit	frag_type specifies what kind of IP fragments or non-fragments to match. Requires setting fiel
ipv6_label	Integer 32bit	Matches IPv6 flow label. Requires setting fields: eth_type_nxm = 0x86dd (IPv6)
ip_ecn_nxm	Integer 8bit	Matches ecn bits in IP ToS or IPv6 traffic class fields. Requires setting fields: eth_type_nxm =
nw_ttl	Integer 8bit	IP TTL or IPv6 hop limit value ttl. Requires setting fields: eth_type_nxm = [0x0800 (IPv4)|0x
mpls_ttl	Integer 8bit	The TTL of the outer MPLS label stack entry of a packet. Requires setting fields: eth_type_nx
tun_ipv4_src	IPv4 address	Tunnel IPv4 source address. Requires setting fields: eth_type_nxm = 0x0800 (IPv4)
tun_ipv4_dst	IPv4 address	Tunnel IPv4 destination address. Requires setting fields: eth_type_nxm = 0x0800 (IPv4)
pkt_mark	Integer 32bit	Packet metadata mark.
tcp_flags_nxm	Integer 16bit	TCP Flags. Requires setting fields: eth_type_nxm = [0x0800 (IP)|0x86dd (IPv6)] and ip_proto
conj_id	Integer 32bit	Conjunction ID used only with the conjunction action
tun_gbp_id	Integer 16bit	The group policy identifier in the VXLAN header.
tun_gbp_flags	Integer 8bit	The group policy flags in the VXLAN header.
tun_flags	Integer 16bit	Flags indicating various aspects of the tunnel encapsulation.
ct_state	Integer 32bit	Conntrack state.
ct_zone	Integer 16bit	Conntrack zone.
ct_mark	Integer 32bit	Conntrack mark.
ct_label	Integer 128bit	Conntrack label.
tun_ipv6_src	IPv6 address	Tunnel IPv6 source address. Requires setting fields: eth_type_nxm = 0x86dd (IPv6)
tun_ipv6_dst	IPv6 address	Tunnel IPv6 destination address. Requires setting fields: eth_type_nxm = 0x86dd (IPv6)
_recirc_id	Integer 32bit	ID for recirculation.
_dp_hash	Integer 32bit	Flow hash computed in Datapath.
reg<idx>	Integer 32bit	Packet register. <idx> is register number 0-15.
xxreg<idx>	Integer 128bit	Packet extended-extended register. <idx> is register number 0-3.

Note: Setting the TCP flags via the nicira extensions. This is required when using OVS version < 2.4. When using the nxm fields, you need to use any nxm prereq fields as well or you will receive a OFPBMC_BAD_PREREQ error

Example:

```
# WILL NOT work
flag = tcp.TCP_ACK
match = parser.OFPMatch(
    tcp_flags_nxm=(flag, flag),
    ip_proto=inet.IPPROTO_TCP,
    eth_type=eth_type)
```

(continues on next page)

(continued from previous page)

```
# Works
flag = tcp.TCP_ACK
match = parser.OFPMatch(
    tcp_flags_nxm=(flag, flag),
    ip_proto_nxm=inet.IPPROTO_TCP,
    eth_type_nxm=eth_type)
```

2.7 Ryu API Reference

CHAPTER 3

Configuration

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
```

ctl-privkey.pem and ctl-cert.pem are generated in the current directory.

Create a switch private key and certificate:

```
% ovs-pki req+sign sc switch
```

sc-privkey.pem and sc-cert.pem are generated in the current directory.

3.1.2 Testing TLS Connection

Configuring ovs-vswitchd to use CA files using the ovs-vsctl "set-ssl" command, e.g.:

```
% ovs-vsctl set-ssl /etc/openvswitch/sc-privkey.pem \
  /etc/openvswitch/sc-cert.pem \
  /usr/local/var/lib/openvswitch/pki/controllerca/cacert.pem
% ovs-vsctl add-br br0
% ovs-vsctl set-controller br0 ssl:127.0.0.1:6633
```

Substitute the correct file names, if they differ from the ones used above. You should use absolute file names.

Run Ryu with CA files:

```
% 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:

```
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> a
ddress:('127.0.0.1', 61302)
hello ev <ryu.controller.ofp_event.EventOFPHello object at 0x1047806d0>
move onto config mode
switch features ev version: 0x1 msg_type 0x6 xid 0xb0bb34e5 port OFPPhyPort(port
_no=65534, hw_addr='\x16\xdc\xa2\xe2}K', name='br0\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00', config=0, state=0, curr=0, advertised=0, supported=0, p
eer=0)
move onto main mode
```

3.2 Topology Viewer

ryu.app.gui_topology.gui_topology provides topology visualization.

This depends on following ryu applications.

ryu.app.rest_topology	Get node and link data.
ryu.app.ws_topology	Being notified change of link up/down.
ryu.app.ofctl_rest	Get flows of datapaths.

3.2.1 Usage

Run mininet (or join your real environment):

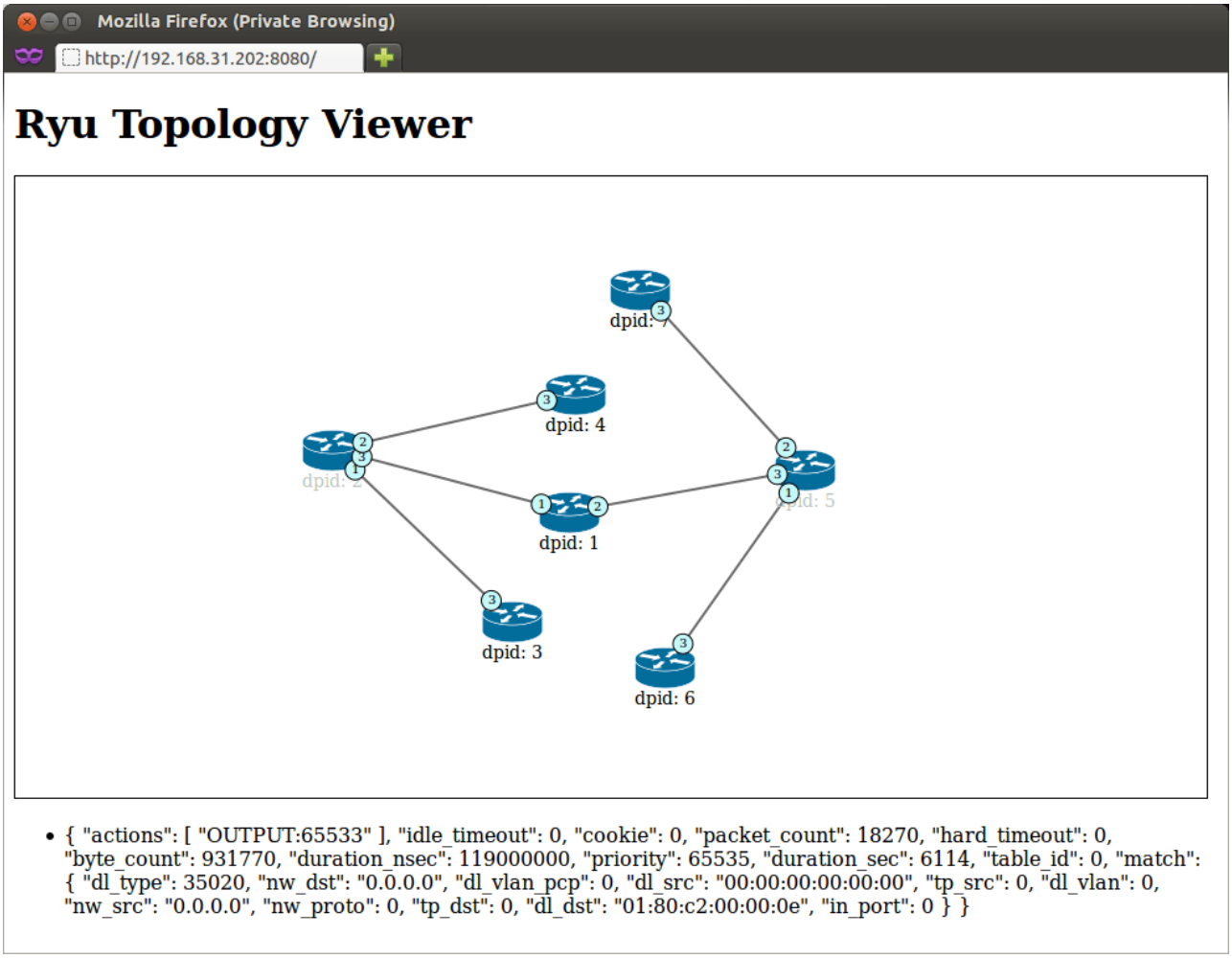
```
$ sudo mn --controller remote --topo tree,depth=3
```

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



- { "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_pcp": 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:c2:00:00:0e", "in_port": 0 } }

CHAPTER 4

Tests

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
```

(continues on next page)

(continued from previous page)

```
# 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.

```
               ^ veth5
               |
               V veth5-br1
       ------------------------
       |Linux Brirge vrrp-br1|
       ------------------------
veth3-br1^             ^ veth4-br1
         |             |
    veth3V             V veth4
    ----------        ----------
    |netns   |        |netns   |
    |gateway1|        |gateway2|
    |ryu-vrrp|        |ryu-vrrp|
    ----------        ----------
    veth1^             ^ veth2
         |             |
veth1-br0V             V veth2-br0
       ------------------------
```

(continues on next page)

(continued from previous page)

```
       |Linux Brirge vrrp-br0|
       -----------------------
               ^ veth0-br0
               |
               V veth0
```

Here's the helper executable, ryu-vrrp:

```
#!/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)

(continued from previous 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)
```

(continues on next page)

(continued from previous page)

```
    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 http://osrg.github.io/ryu/

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:

```
# apt-get install git-core bridge-utils libpcap0.8 libpcap-dev libcap2-bin uml-
↪utilities
```

On RedHat/CentOS:

```
# yum install git sudo bridge-utils libpcap libpcap-devel libcap tunctl
```

Note that on RedHat/CentOS 5.x you need a newer version of libpcap:

```
# yum erase libpcap libpcap-devel
# yum install flex byacc
# wget http://www.tcpdump.org/release/libpcap-1.2.1.tar.gz
# tar xzf libpcap-1.2.1.tar.gz
# cd libpcap-1.2.1
# ./configure
# make && make install
```

get LINC repo and built

Clone LINC repo:

```
% git clone git://github.com/FlowForwarding/LINC-Switch.git
```

Then compile everything:

```
% cd LINC-Switch
% make
```

Note: At the time of this writing, test_of_config fails due to a bug of LINC. You can try this test with LINC which is built by the following methods.

```
% cd LINC-Switch
% make
% cd deps/of_config
% git reset --hard f772af4b765984381ad024ca8e5b5b8c54362638
% cd ../..
% make offline
```

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.

```
[{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)

(continued from previous page)

```
                {controllers,[{"Switch0-Default-Controller","127.0.0.1",6633,tcp}]},
                {controllers_listener,{"127.0.0.1",9998,tcp}},
                {queues_status,enabled},
                {ports,[{port,1,{queues,[]}},{port,2,{queues,[991,992]}}]}]}
                ,
            {switch,7,
                [{backend,linc_us3},
                 {controllers,[{"Switch7-Controller","127.0.0.1",6633,tcp}]},
                 {controllers_listener,disabled},
                 {queues_status,enabled},
                 {ports,[{port,4,{queues,[]}},{port,3,{queues,[993,994]}}]}]}]}
        ]}]},
{enetconf,
    [{capabilities,
         [{base,{1,0}},
          {base,{1,1}},
          {startup,{1,0}},
          {'writable-running',{1,0}}]},
     {callback_module,linc_ofconfig},
     {sshd_ip,{127,0,0,1}},
     {sshd_port,1830},
     {sshd_user_passwords,[{"linc","linc"}]}]},
{lager,
    [{handlers,
         [{lager_console_backend,debug},
          {lager_file_backend,
              [{"log/error.log",error,10485760,"$D0",5},
               {"log/console.log",info,10485760,"$D0",5}]}]}]},
{sasl,
    [{sasl_error_logger,{file,"log/sasl-error.log"}},
     {errlog_type,error},
     {error_logger_mf_dir,"log/sasl"},
     {error_logger_mf_maxbytes,10485760},
     {error_logger_mf_maxfiles,5}]},
{sync,[{excluded_modules,[procket]}]}].
```

4.2.4 setup for LINC

As the above sys.config requires some network interface, create them:

```
# ip link add linc-port type veth peer name linc-port-peer
# ip link set linc-port up
# ip link add linc-port2 type veth peer name linc-port-peer2
# ip link set linc-port2 up
# ip link add linc-port3 type veth peer name linc-port-peer3
# ip link set linc-port3 up
# ip link add linc-port4 type veth peer name linc-port-peer4
# ip link set linc-port4 up
```

After stopping LINC, those created interfaces can be deleted:

```
# ip link delete linc-port
# ip link delete linc-port2
# ip link delete linc-port3
# ip link delete linc-port4
```

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
```

CHAPTER 5

Snort Intergration

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----+
             |       |
+-------+   +----------+   +-------+
| HostA |---| OFSwitch |---| HostB |
+-------+   +----------+   +-------+
```

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------+    |
                 |             |
+-------+   +----------+   +-----------+
| HostA |---| OFSwitch |---| LAN (*CP) |
+-------+   +----------+   +-----------+
                 |             |
            +----------+   +----------+
```

(continues on next page)

(continued from previous page)

```
            |  HostB    |    |    Ryu     |
            +-----------+    +------------+
```

***CP: Control Plane**

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 referto 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`

```
alert icmp any any -> any any (msg:"Pinging...";sid:1000004;)
alert tcp any any -> any 80 (msg:"Port 80 is accessing"; sid:1000003;)
```

- Add the custom rules in `/etc/snort/snort.conf`

```
include $RULE_PATH/Myrules.rules
```

Configure NIC as a promiscuous mode.

```
$ sudo ifconfig eth1 promisc
```

5.4 Usage

[Option 1]

1. Modify the `simple_switch_snort.py`:

```
socket_config = {'unixsock': True}
# True: Unix Domain Socket Server [Option1]
# False: Network Socket Server [Option2]
```

2. Run Ryu with sample application:

```
$ 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:

```
$ sudo -i
$ snort -i eth1 -A unsock -l /tmp -c /etc/snort/snort.conf
```

4. Send an ICMP packet from HostA (192.168.8.40) to HostB (192.168.8.50):

```
$ ping 192.168.8.50
```

5. You can see the result under next section.

[Option 2]

1. Modify the `simple_switch_snort.py`:

```
socket_config = {'unixsock': False}
# True: Unix Domain Socket Server [Option1]
# False: Network Socket Server [Option2]
```

2. Run 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

5. Send an ICMP packet from HostA (192.168.8.40) to HostB (192.168.8.50):

```
$ ping 192.168.8.50
```

6. You can see the alert message below:

```
alertmsg: Pinging...
icmp(code=0,csum=19725,data=echo(data=array('B', [97, 98, 99, 100, 101, 102, 103,␣
↪104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119,␣
↪97, 98, 99, 100, 101, 102, 103, 104, 105]),id=1,seq=78),type=8)

ipv4(csum=42562,dst='192.168.8.50',flags=0,header_length=5,identification=724,
↪offset=0,option=None,proto=1,src='192.168.8.40',tos=0,total_length=60,ttl=128,
↪version=4)

ethernet(dst='00:23:54:5a:05:14',ethertype=2048,src='00:23:54:6c:1d:17')


alertmsg: Pinging...
icmp(code=0,csum=21773,data=echo(data=array('B', [97, 98, 99, 100, 101, 102, 103,␣
↪104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119,␣
↪97, 98, 99, 100, 101, 102, 103, 104, 105]),id=1,seq=78),type=0)

ipv4(csum=52095,dst='192.168.8.40',flags=0,header_length=5,identification=7575,
↪offset=0,option=None,proto=1,src='192.168.8.50',tos=0,total_length=60,ttl=64,
↪version=4)
```

(continues on next page)

(continued from previous page)

CHAPTER 6

Built-in Ryu applications

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:

```
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

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*)
OFPErrorMsg is received.

exception ryu.app.ofctl.exception.**UnexpectedMultiReply**(*result*)
Two or more replies are received for reply_muiti=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

6.2.1 Retrieve the switch stats

Get all switches

Get the list of all switches which connected to the controller.

Usage:

Method	GET
URI	/stats/switches

Response message body:

Attribute	Description	Example
dpid	Datapath ID	1

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:

Method	GET
URI	/stats/desc/<dpid>

Response message body:

Attribute	Description	Example
dpid	Datapath ID	“1”
mfr_desc	Manufacturer description	“Nicira, Inc.”,
hw_desc	Hardware description	“Open vSwitch”,
sw_desc	Software description	“2.3.90”,
serial_num	Serial number	“None”,
dp_desc	Human readable description of datapath	“None”

Example of use:

```
$ 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:

Method	GET
URI	/stats/flow/<dpid>

Response message body(OpenFlow1.3 or earlier):

Attribute	Description	Example
dpid	Datapath ID	"1"
length	Length of this entry	88
table_id	Table ID	0
duration_sec	Time flow has been alive in seconds	2
duration_nsec	Time flow has been alive in nanoseconds beyond duration_sec	6.76e+08
priority	Priority of the entry	11111
idle_timeout	Number of seconds idle before expiration	0
hard_timeout	Number of seconds before expiration	0
flags	Bitmap of OFPFF_* flags	1
cookie	Opaque controller-issued identifier	1
packet_count	Number of packets in flow	0
byte_count	Number of bytes in flow	0
match	Fields to match	{"in_port": 1}
actions	Instruction set	["OUTPUT:2"]

Response message body(OpenFlow1.4 or later):

Attribute	Description	Example
dpid	Datapath ID	"1"
length	Length of this entry	88
table_id	Table ID	0
duration_sec	Time flow has been alive in seconds	2
duration_nsec	Time flow has been alive in nanoseconds beyond duration_sec	6.76e+08
priority	Priority of the entry	11111
idle_timeout	Number of seconds idle before expiration	0
hard_timeout	Number of seconds before expiration	0
flags	Bitmap of OFPFF_* flags	1
cookie	Opaque controller-issued identifier	1
packet_count	Number of packets in flow	0
byte_count	Number of bytes in flow	0
importance	Eviction precedence	0
match	Fields to match	{"eth_type": 2054}
instructions	struct ofp_instruction_header	[{"type":GOTO_TABLE", "table_id":1}]

Example of use:

```
$ curl -X GET http://localhost:8080/stats/flow/1
```

Response (OpenFlow1.3 or earlier):

```
{
  "1": [
```

(continues on next page)

(continued from previous page)

```
    {
      "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:

Attribute	Description	Example	Default
table_id	Table ID (int)	0	OFPTT_ALL
out_port	Require matching entries to include this as an output port (int)	2	OFPP_ANY
out_group	Require matching entries to include this as an output group (int)	1	OFPG_ANY
cookie	Require matching entries to contain this cookie value (int)	1	0
cookie_mask	Mask used to restrict the cookie bits that must match (int)	1	0
match	Fields to match (dict)	{"in_port": 1}	{} #wildcarded
priority	Priority of the entry (int) (See Note)	11111	#wildcarded

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,
```

(continues on next page)

(continued from previous page)

```
      "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 aggregate flow stats

Get aggregate flow stats of the switch which specified with Datapath ID in URI.

Usage:

Method	GET
URI	/stats/aggregateflow/<dpid>

Response message body:

Attribute	Description	Example
dpid	Datapath ID	"1"
packet_count	Number of packets in flows	18
byte_count	Number of bytes in flows	756
flow_count	Number of flows	3

Example of use:

```
$ curl -X GET http://localhost:8080/stats/aggregateflow/1
```

```
{
  "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:

Method	POST
URI	/stats/aggregateflow/<dpid>

Request message body:

Attribute	Description	Example	Default
table_id	Table ID (int)	0	OFPTT_ALL
out_port	Require matching entries to include this as an output port (int)	2	OFPP_ANY
out_group	Require matching entries to include this as an output group (int)	1	OFPG_ANY
cookie	Require matching entries to contain this cookie value (int)	1	0
cookie_mask	Mask used to restrict the cookie bits that must match (int)	1	0
match	Fields to match (dict)	{"in_port": 1}	{} #wildcarded

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
```

```
{
  "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:

Method	GET
URI	/stats/table/<dpid>

Response message body(OpenFlow1.0):

Attribute	Description	Example
dpid	Datapath ID	"1"
table_id	Table ID	0
name	Name of Table	"classifier"
max_entries	Max number of entries supported	1e+06
wildcards	Bitmap of OFPFW_* wildcards that are supported by the table	["IN_PORT","DL_VLAN"]
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

Response message body(OpenFlow1.2):

Attribute	Description	Example
dpid	Datapath ID	"1"
table_id	Table ID	0
name	Name of Table	"classifier"
match	Bitmap of (1 << OFPXMT_*) that indicate the fields the table can match on	["OFB_IN_PORT","OFB_METADATA"]
wildcards	Bitmap of (1 << OFPXMT_*) wildcards that are supported by the table	["OFB_IN_PORT","OFB_METADATA"]
write_actions	Bitmap of OFPAT_* that are supported by the table with OFPIT_WRITE_ACTIONS	["OUTPUT","SET_MPLS_TTL"]
apply_actions	Bitmap of OFPAT_* that are supported by the table with OFPIT_APPLY_ACTIONS	["OUTPUT","SET_MPLS_TTL"]
write_setfields	Bitmap of (1 << OFPXMT_*) header fields that can be set with OFPIT_WRITE_ACTIONS	["OFB_IN_PORT","OFB_METADATA"]
apply_setfields	Bitmap of (1 << OFPXMT_*) header fields that can be set with OFPIT_APPLY_ACTIONS	["OFB_IN_PORT","OFB_METADATA"]
metadata_match	Bits of metadata table can match	18446744073709552000
metadata_write	Bits of metadata table can write	18446744073709552000
instructions	Bitmap of OFPIT_* values supported	["GOTO_TABLE","WRITE_METADATA"]
config	Bitmap of OFPTC_* values	[]
max_entries	Max number of entries supported	1e+06
active_count	Number of active entries	0
lookup_count	Number of packets looked up in table	0
matched_count	Number of packets that hit table	8

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:

```
$ curl -X GET http://localhost:8080/stats/table/1
```

Response (OpenFlow1.0):

```
{
  "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):

```
{
  "1": [
    {
      "apply_setfields": [
       "OFB_IN_PORT",
       "OFB_METADATA"
      ],
      "match": [
       "OFB_IN_PORT",
       "OFB_METADATA"
      ],
```

(continues on next page)

(continued from previous page)

```
      "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"
      ],
```

(continues on next page)

(continued from previous page)

```
      "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:

Method	GET
URI	/stats/tablefeatures/<dpid>

Response message body:

Attribute	Description	Example
dpid	Datapath ID	"1"
table_id	Table ID	0
name	Name of Table	"table_0"
metadata_match	Bits of metadata table can match	18446744073709552000
metadata_write	Bits of metadata table can write	18446744073709552000
config	Bitmap of OFPTC_* values	0
max_entries	Max number of entries supported	4096
properties	struct ofp_table_feature_prop_header	[{"type": "INSTRUCTIONS","instruction_ids": [...]},...]

Example of use:

```
$ curl -X GET http://localhost:8080/stats/tablefeatures/1
```

```
{
  "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
           },
```

(continues on next page)

(continued from previous page)

```
            ...
          ]
        },
        ...
      ],
      "name": "table_1"
    },
    ...
  ]
}
```

Get ports stats

Get ports stats of the switch which specified with Datapath ID in URI.

Usage:

Method	GET
URI	/stats/port/<dpid>[/<port>]

Note: Specification of port number is optional.

Response message body(OpenFlow1.3 or earlier):

Attribute	Description	Example
dpid	Datapath ID	"1"
port_no	Port number	1
rx_packets	Number of received packets	9
tx_packets	Number of transmitted packets	6
rx_bytes	Number of received bytes	738
tx_bytes	Number of transmitted bytes	252
rx_dropped	Number of packets dropped by RX	0
tx_dropped	Number of packets dropped by TX	0
rx_errors	Number of receive errors	0
tx_errors	Number of transmit errors	0
rx_frame_err	Number of frame alignment errors	0
rx_over_err	Number of packets with RX overrun	0
rx_crc_err	Number of CRC errors	0
collisions	Number of collisions	0
duration_sec	Time port has been alive in seconds	12
duration_nsec	Time port has been alive in nanoseconds beyond duration_sec	9.76e+08

Response message body(OpenFlow1.4 or later):

Attribute	Description	Example
dpid	Datapath ID	"1"
port_no	Port number	1
rx_packets	Number of received packets	9
tx_packets	Number of transmitted packets	6
rx_bytes	Number of received bytes	738
tx_bytes	Number of transmitted bytes	252
rx_dropped	Number of packets dropped by RX	0
tx_dropped	Number of packets dropped by TX	0
rx_errors	Number of receive errors	0
tx_errors	Number of transmit errors	0
duration_sec	Time port has been alive in seconds	12
duration_nsec	Time port has been alive in nanoseconds beyond duration_sec	9.76e+08
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):

```
{
   "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,
       "duration_nsec": 12,
       "duration_sec": 9.76e+08,
       "properties": [
         {
           "rx_frame_err": 0,
           "rx_over_err": 0,
           "rx_crc_err": 0,
           "collisions": 0,
           "type": "ETHERNET"
         },
         {
           "bias_current": 300,
           "flags": 3,
           "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": "OPTICAL"
         },
         {
           "data": [],
           "exp_type": 0,
           "experimenter": 101,
           "type": "EXPERIMENTER"
         },
         {
           :

           :
         }
       ]
     }
   ]
 }
```


Get ports description

Get ports description of the switch which specified with Datapath ID in URI.

Usage(OpenFlow1.4 or earlier):

Method	GET
URI	/stats/portdesc/<dpid>

Usage(OpenFlow1.5 or later):

Method	GET
URI	/stats/portdesc/<dpid>/[<port>]

Note: Specification of port number is optional.

Response message body(OpenFlow1.3 or earlier):

Attribute	Description	Example
dpid	Datapath ID	"1"
port_no	Port number	1
hw_addr	Ethernet hardware address	"0a:b6:d0:0c:e1:d7"
name	Name of port	"s1-eth1"
config	Bitmap of OFPPC_* flags	0
state	Bitmap of OFPPS_* flags	0
curr	Current features	2112
advertised	Features being advertised by the port	0
supported	Features supported by the port	0
peer	Features advertised by peer	0
curr_speed	Current port bitrate in kbps	1e+07
max_speed	Max port bitrate in kbps	0

Response message body(OpenFlow1.4 or later):

Attribute	Description	Example
dpid	Datapath ID	"1"
port_no	Port number	1
hw_addr	Ethernet hardware address	"0a:b6:d0:0c:e1:d7"
name	Name of port	"s1-eth1"
config	Bitmap of OFPPC_* flags	0
state	Bitmap of OFPPS_* flags	0
length	Length of this entry	168
properties	struct ofp_port_desc_prop_header	[{"length": 32, "curr": 10248,...},...]

Example of use:

```
$ curl -X GET http://localhost:8080/stats/portdesc/1
```

Response (OpenFlow1.3 or earlier):

```
{
  "1": [
    {
      "port_no": 1,
      "hw_addr": "0a:b6:d0:0c:e1:d7",
```

(continues on next page)

(continued from previous page)

```
      "name": "s1-eth1",
      "config": 0,
      "state": 0,
      "curr": 2112,
      "advertised": 0,
      "supported": 0,
      "peer": 0,
      "curr_speed": 1e+07,
      "max_speed": 0
    },
    {
      :
      :
    }
  ]
}
```

Response (OpenFlow1.4 or later):

```
{
   "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)

(continued from previous page)

```
            "type": "EXPERIMENTER"
          },
          {
            :

            :
          }
        ]
      }
    ]
}
```

Get queues stats

Get queues stats of the switch which specified with Datapath ID in URI.

Usage:

Method	GET
URI	/stats/queue/<dpid>[/<port>[/<queue_id>]]

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):

Attribute	Description	Example
dpid	Datapath ID	"1"
port_no	Port number	1
queue_id	Queue ID	0
tx_bytes	Number of transmitted bytes	0
tx_packets	Number of transmitted packets	0
tx_errors	Number of packets dropped due to overrun	0
duration_sec	Time queue has been alive in seconds	4294963425
duration_nsec	Time queue has been alive in nanoseconds beyond duration_sec	3912967296

Response message body(OpenFlow1.4 or later):

Attribute	Description	Example
dpid	Datapath ID	"1"
port_no	Port number	1
queue_id	Queue ID	0
tx_bytes	Number of transmitted bytes	0
tx_packets	Number of transmitted packets	0
tx_errors	Number of packets dropped due to overrun	0
duration_sec	Time queue has been alive in seconds	4294963425
duration_nsec	Time queue has been alive in nanoseconds beyond duration_sec	3912967296
length	Length of this entry	104
properties	struct ofp_queue_stats_prop_header	[{"type": 65535,"length": 12,...},...]

Example of use:

```
$ 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,
```

(continues on next page)

(continued from previous 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": 4294963425,
      "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:

Method	GET
URI	/stats/queueconfig/<dpid>/[<port>]

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:

Attribute	Description	Example
dpid	Datapath ID	"1"
port	Port which was queried	1
queues	struct ofp_packet_queue	
– queue_id	ID for the specific queue	2
– port	Port this queue is attached to	0
– properties	struct ofp_queue_prop_header properties	[{"property": "MIN_RATE","rate": 80}]

Example of use:

```
$ curl -X GET http://localhost:8080/stats/queueconfig/1/1
```


```
{
  "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:

Method	GET
URI	/stats/queuedesc/<dpid>[/<port>/[<queue_id>]]

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:

Attribute	Description	Example
dpid	Datapath ID	"1"
len	Length in bytes of this queue desc	88
port_no	Port which was queried	1
queue_id	Queue ID	1
properties	struct ofp_queue_desc_prop_header	[{"length": 8, . . . },. . .]

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)

(continued from previous page)

```
            "experimenter": 101,
            "data": [1],
            "type": "EXPERIMENTER"
          },
          {
            :

            :
          }
        ]
      }
    ]
}
```

Get groups stats

Get groups stats of the switch which specified with Datapath ID in URI.

Usage:

Method	GET
URI	/stats/group/<dpid>[/<group_id>]

Note: Specification of group id is optional.

Response message body:

Attribute	Description	Example
dpid	Datapath ID	"1"
length	Length of this entry	56
group_id	Group ID	1
ref_count	Number of flows or groups that directly forward to this group	1
packet_count	Number of packets processed by group	0
byte_count	Number of bytes processed by group	0
duration_sec	Time group has been alive in seconds	161
duration_nsec	Time group has been alive in nanoseconds beyond duration_sec	3.03e+08
bucket_stats	struct ofp_bucket_counter	
– packet_count	Number of packets processed by bucket	0
– byte_count	Number of bytes processed by bucket	0

Example of use:

```
$ curl -X GET http://localhost:8080/stats/group/1
```

```
{
  "1": [
    {
      "length": 56,
      "group_id": 1,
      "ref_count": 1,
      "packet_count": 0,
      "byte_count": 0,
      "duration_sec": 161,
      "duration_nsec": 3.03e+08,
      "bucket_stats": [
        {
          "packet_count": 0,
          "byte_count": 0
        }
      ]
    }
  ]
}
```


Get group description stats

Get group description stats of the switch which specified with Datapath ID in URI.

Usage(Openflow1.4 or earlier):

Method	GET
URI	/stats/groupdesc/<dpid>

Usage(Openflow1.5 or later):

Method	GET
URI	/stats/groupdesc/<dpid>/[<group_id>]

Note: Specification of group id is optional.

Response message body(Openflow1.3 or earlier):

Attribute	Description	Example
dpid	Datapath ID	"1"
type	One of OFPGT_*	"ALL"
group_id	Group ID	1
buckets	struct ofp_bucket	
– weight	Relative weight of bucket (Only defined for select groups)	0
– watch_port	Port whose state affects whether this bucket is live (Only required for fast failover groups)	4294967295
– watch_group	Group whose state affects whether this bucket is live (Only required for fast failover groups)	4294967295
– actions	0 or more actions associated with the bucket	["OUTPUT:1"]

Response message body(Openflow1.4 or later):

Attribute	Description	Example
dpid	Datapath ID	"1"
type	One of OFPGT_*	"ALL"
group_id	Group ID	1
length	Length of this entry	40
buckets	struct ofp_bucket	
– weight	Relative weight of bucket (Only defined for select groups)	0
– watch_port	Port whose state affects whether this bucket is live (Only required for fast failover groups)	4294967295
– watch_group	Group whose state affects whether this bucket is live (Only required for fast failover groups)	4294967295
– len	Length the bucket in bytes, including this header and any adding to make it 64-bit aligned.	32
– actions	0 or more actions associated with the bucket	[{"OUTPUT:1", "max_len": 65535,...}]

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,
```

(continues on next page)

(continued from previous page)

```
        "buckets": [
          {
            "weight": 1,
            "watch_port": 1,
            "watch_group": 1,
            "len": 32,
            "actions": [
              {
                  "type": "OUTPUT",
                  "max_len": 65535,
                  "port": 2
              }
            ]
          }
        ]
      }
    ]
}
```

Get group features stats

Get group features stats of the switch which specified with Datapath ID in URI.

Usage:

Method	GET
URI	/stats/groupfeatures/<dpid>

Response message body:

Attribute	Description	Example
dpid	Datapath ID	"1"
types	Bitmap of (1 << OFPGT_*) values supported	[]
capabilities	Bitmap of OFPGFC_* capability supported	["SELECT_WEIGHT","SELECT_LIVENESS","CHAINING"]
max_groups	Maximum number of groups for each type	[{"ALL": 4294967040},...]
actions	Bitmaps of (1 << OFPAT_*) values supported	[{"ALL": ["OUTPUT",...]},...]

Example of use:

```
$ curl -X GET http://localhost:8080/stats/groupfeatures/1
```

```
{
  "1": [
    {
      "types": [],
      "capabilities": [
```

(continues on next page)

(continued from previous page)

```
        "SELECT_WEIGHT",
        "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:

Attribute	Description	Example
dpid	Datapath ID	"1"
meter_id	Meter ID	1
len	Length in bytes of this stats	56
flow_count	Number of flows bound to meter	0
packet_in_count	Number of packets in input	0
byte_in_count	Number of bytes in input	0
duration_sec	Time meter has been alive in seconds	37
duration_nsec	Time meter has been alive in nanoseconds beyond duration_sec	988000
band_stats	struct ofp_meter_band_stats	
– packet_band_count	Number of packets in band	0
– byte_band_count	Number of bytes in band	0

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(Openflow1.4 or earlier):

Method	GET
URI	/stats/meterconfig/<dpid>[/<meter_id>]

Usage(Openflow1.5 or later):

Method	GET
URI	/stats/meterdesc/<dpid>[/<meter_id>]

> **Note:** Specification of meter id is optional.

Response message body:

Attribute	Description	Example
dpid	Datapath ID	"1"
flags	All OFPMC_* that apply	"KBPS"
meter_id	Meter ID	1
bands	struct ofp_meter_band_header	
– type	One of OFPMBT_*	"DROP"
– rate	Rate for this band	1000
– burst_size	Size of bursts	0

Example of use:

```
$ curl -X GET http://localhost:8080/stats/meterconfig/1
```

```
{
  "1": [
    {
      "flags": [
        "KBPS"
      ],
      "meter_id": 1,
      "bands": [
        {
          "type": "DROP",
          "rate": 1000,
          "burst_size": 0
        }
      ]
    }
```

(continues on next page)

(continued from previous page)

```
    ]
}
```

Get meter features stats

Get meter features stats of the switch which specified with Datapath ID in URI.

Usage:

Method	GET
URI	/stats/meterfeatures/<dpid>

Response message body:

Attribute	Description	Example
dpid	Datapath ID	"1"
max_meter	Maximum number of meters	256
band_types	Bitmaps of (1 << OFPMBT_*) values supported	["DROP"]
capabilities	Bitmaps of "ofp_meter_flags"	["KBPS", "BURST", "STATS"]
max_bands	Maximum bands per meters	16
max_color	Maximum color value	8

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(Openflow1.4 or earlier):

Attribute	Description	Example
dpid	Datapath ID	1
role	One of OFPCR_ROLE_*	“EQUAL”
generation_id	Master Election Generation Id	0

Response message body(Openflow1.5 or later):

Attribute	Description	Example
dpid	Datapath ID	1
role	One of OFPCR_ROLE_*	“EQUAL”
short_id	ID number for the controller	0
generation_id	Master Election Generation Id	0

Example of use:

```
$ curl -X GET http://localhost:8080/stats/role/1
```

Response (Openflow1.4 or earlier):

```
{
    "1": [
        {
            "generation_id": 0,
            "role": "EQUAL"
        }
    ]
}
```

Response (Openflow1.5 or later):

```
{
    "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:

Method	POST
URI	/stats/flowentry/add

Request message body(Openflow1.3 or earlier):

Attribute	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)	{“in_port”:1}	{} #wild-carded
actions	Instruction set (list of dict)	[{“type”:”OUTPUT”, “port”:2}]	[] #DROP

Request message body(Openflow1.4 or later):

Attribute	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)	{“in_port”:1}	{} #wild-carded
instructions	Instruction set (list of dict)	[{“type”:”METER”, “meter_id”:2}]	[] #DROP

Note: For description of match and actions, please see *Reference: Description of Match and Actions*.

Example of use(Openflow1.3 or earlier):

```
$ 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,
```

(continues on next page)

(continued from previous page)

```
    "priority": 44444,
    "match":{
        "in_port":1
    },
    "actions":[
        {
            "type":"METER",
            "meter_id": 1
        }
    ]
 }' http://localhost:8080/stats/flowentry/add
```

Example of use(Openflow1.4 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,
```

(continues on next page)

(continued from previous 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:

Method	POST
URI	/stats/flowentry/modify

Request message body:

Attribute	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)	{“in_port”:1}	{} #wild-carded
actions	Instruction set (list of dict)	[{“type”:”OUTPUT”, “port”:2}]	[] #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
```

Modify flow entry strictly

Modify flow entry strictly matching wildcards and priority

Usage:

Method	POST
URI	/stats/flowentry/modify_strict

Request message body:

Attribute	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)	{“in_port”:1}	{} #wildcarded
actions	Instruction set (list of dict)	[{“type”:”OUTPUT”, “port”:2}]	[] #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:

Method	POST
URI	/stats/flowentry/delete

Request message body:

Attribute	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
out_port	Output port (int)	1	OFPP_ANY
out_group	Output group (int)	1	OFPG_ANY
flags	Bitmap of OFPFF_* flags (int)	1	0
match	Fields to match (dict)	{“in_port”:1}	{} #wildcarded
actions	Instruction set (list of dict)	[{“type”:”OUTPUT”, “port”:2}]	[] #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/delete
```

Delete flow entry strictly

Delete flow entry strictly matching wildcards and priority.

Usage:

Method	POST
URI	/stats/flowentry/delete_strict

Request message body:

Attribute	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
out_port	Output port (int)	1	OFPP_ANY
out_group	Output group (int)	1	OFPG_ANY
flags	Bitmap of OFPFF_* flags (int)	1	0
match	Fields to match (dict)	{“in_port”:1}	{} #wild-carded
actions	Instruction set (list of dict)	[{“type”:”OUTPUT”, “port”:2}]	[] #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/delete_strict
```

Delete all flow entries

Delete all flow entries of the switch which specified with Datapath ID in URI.

Usage:

Method	DELETE
URI	/stats/flowentry/clear/<dpid>

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:

Method	POST
URI	/stats/groupentry/add

Request message body:

Attribute	Description	Example	Default
dpid	Datapath ID (int)	1	(Mandatory)
type	One of OFPGT_* (string)	"ALL"	"ALL"
group_id	Group ID (int)	1	0
buckets	struct ofp_bucket		
– weight	Relative weight of bucket (Only defined for select groups)	0	0
– watch_port	Port whose state affects whether this bucket is live (Only required for fast failover groups)	4294967295	OFPP_ANY
– watch_group	Group whose state affects whether this bucket is live (Only required for fast failover groups)	4294967295	OFPG_ANY
– actions	0 or more actions associated with the bucket (list of dict)	[{"type": "OUTPUT", "port": 1}]	[] #DROP

Example of use:

```
$ curl -X POST -d '{
    "dpid": 1,
    "type": "ALL",
    "group_id": 1,
    "buckets": [
        {
            "actions": [
                {
                    "type": "OUTPUT",
```

(continues on next page)

(continued from previous page)

```
                    "port": 1
                }
            ]
        }
    ]
}' http://localhost:8080/stats/groupentry/add
```

Note: To confirm group entry registration, please see *Get group description stats*.

Modify a group entry

Modify a group entry to the switch.

Usage:

Method	POST
URI	/stats/groupentry/modify

Request message body:

Attribute	Description	Example	Default
dpid	Datapath ID (int)	1	(Mandatory)
type	One of OFPGT_* (string)	"ALL"	"ALL"
group_id	Group ID (int)	1	0
buckets	struct ofp_bucket		
– weight	Relative weight of bucket (Only defined for select groups)	0	0
– watch_port	Port whose state affects whether this bucket is live (Only required for fast failover groups)	4294967295	OFPP_ANY
– watch_group	Group whose state affects whether this bucket is live (Only required for fast failover groups)	4294967295	OFPG_ANY
– actions	0 or more actions associated with the bucket (list of dict)	[{"type": "OUTPUT", "port": 1}]	[] #DROP

Example of use:

```
$ curl -X POST -d '{
    "dpid": 1,
    "type": "ALL",
    "group_id": 1,
    "buckets": [
        {
            "actions": [
                {
                    "type": "OUTPUT",
                    "port": 1
                }
```

(continues on next page)

(continued from previous page)

```
            ]
        }
    ]
 }' http://localhost:8080/stats/groupentry/modify
```

Delete a group entry

Delete a group entry to the switch.

Usage:

Method	POST
URI	/stats/groupentry/delete

Request message body:

Attribute	Description	Example	Default
dpid	Datapath ID (int)	1	(Mandatory)
group_id	Group ID (int)	1	0

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:

Method	POST
URI	/stats/portdesc/modify

Request message body:

Attribute	Description	Example	Default
dpid	Datapath ID (int)	1	(Mandatory)
port_no	Port number (int)	1	0
config	Bitmap of OFPPC_* flags (int)	1	0
mask	Bitmap of OFPPC_* flags to be changed (int)	1	0

Example of use:

```
$ curl -X POST -d '{
    "dpid": 1,
    "port_no": 1,
    "config": 1,
```

(continues on next page)

(continued from previous page)

```
    "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:

Method	POST
URI	/stats/meterentry/add

Request message body:

Attribute	Description	Example	Default
dpid	Datapath ID (int)	1	(Mandatory)
flags	Bitmap of OFPMF_* flags (list)	[“KBPS”]	[] #Empty
meter_id	Meter ID (int)	1	0
bands	struct ofp_meter_band_header		
– type	One of OFPMBT_* (string)	“DROP”	None
– rate	Rate for this band (int)	1000	None
– burst_size	Size of bursts (int)	100	None

Example of use:

```
$ 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:

Method	POST
URI	/stats/meterentry/modify

Request message body:

Attribute	Description	Example	Default
dpid	Datapath ID (int)	1	(Mandatory)
flags	Bitmap of OFPMF_* flags (list)	[“KBPS”]	[] #Empty
meter_id	Meter ID (int)	1	0
bands	struct ofp_meter_band_header		
– type	One of OFPMBT_* (string)	“DROP”	None
– rate	Rate for this band (int)	1000	None
– burst_size	Size of bursts (int)	100	None

Example of use:

```
$ 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:

Method	POST
URI	/stats/meterentry/delete

Request message body:

Attribute	Description	Example	Default
dpid	Datapath ID (int)	1	(Mandatory)
meter_id	Meter ID (int)	1	0

Example of use:

```
$ curl -X POST -d '{
    "dpid": 1,
    "meter_id": 1
 }' http://localhost:8080/stats/meterentry/delete
```

Modify role

modify the role of the switch.

Usage:

Method	POST
URI	/stats/role

Request message body:

Attribute	Description	Example	Default
dpid	Datapath ID (int)	1	(Mandatory)
role	One of OFPCR_ROLE_*(string)	“MASTER”	OFPCR_ROLE_EQUAL

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:

Method	POST
URI	/stats/experimenter/<dpid>

Request message body:

Attribute	Description	Example	Default
dpid	Datapath ID (int)	1	(Mandatory)
experimenter	Experimenter ID (int)	1	0
exp_type	Experimenter defined (int)	1	0
data_type	Data format type (“ascii” or “base64”)	“ascii”	“ascii”
data	Data to send (string)	“data”	“” #Empty

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
in_port	Input switch port (int)	{“in_port”: 7}
dl_src	Ethernet source address (string)	{“dl_src”: “aa:bb:cc:11:22:33”}
dl_dst	Ethernet destination address (string)	{“dl_dst”: “aa:bb:cc:11:22:33”}
dl_vlan	Input VLAN id (int)	{“dl_vlan”: 5}
dl_vlan_pcp	Input VLAN priority (int)	{“dl_vlan_pcp”: 3, “dl_vlan”: 3}
dl_type	Ethernet frame type (int)	{“dl_type”: 123}
nw_tos	IP ToS (int)	{“nw_tos”: 16, “dl_type”: 2048}
nw_proto	IP protocol or lower 8 bits of ARP opcode (int)	{“nw_proto”: 5, “dl_type”: 2048}
nw_src	IPv4 source address (string)	{“nw_src”: “192.168.0.1”, “dl_type”: 2048}
nw_dst	IPv4 destination address (string)	{“nw_dst”: “192.168.0.1/24”, “dl_type”: 2048}
tp_src	TCP/UDP source port (int)	{“tp_src”: 1, “nw_proto”: 6, “dl_type”: 2048}
tp_dst	TCP/UDP destination port (int)	{“tp_dst”: 2, “nw_proto”: 6, “dl_type”: 2048}

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):

Match field	Description	Example
in_port	Switch input port (int)	{“in_port”: 7}
in_phy_port	Switch physical input port (int)	{“in_phy_port”: 5, “in_port”: 3}
metadata	Metadata passed between tables (int or string)	{“metadata”: 12345} or {“metadata”: “0x121
eth_dst	Ethernet destination address (string)	{“eth_dst”: “aa:bb:cc:11:22:33/00:00:00:00:ff
eth_src	Ethernet source address (string)	{“eth_src”: “aa:bb:cc:11:22:33”}
eth_type	Ethernet frame type (int)	{“eth_type”: 2048}
vlan_vid	VLAN id (int or string)	See *Example of VLAN ID match field*
vlan_pcp	VLAN priority (int)	{“vlan_pcp”: 3, “vlan_vid”: 3}
ip_dscp	IP DSCP (6 bits in ToS field) (int)	{“ip_dscp”: 3, “eth_type”: 2048}
ip_ecn	IP ECN (2 bits in ToS field) (int)	{“ip_ecn”: 0, “eth_type”: 34525}
ip_proto	IP protocol (int)	{“ip_proto”: 5, “eth_type”: 34525}
ipv4_src	IPv4 source address (string)	{“ipv4_src”: “192.168.0.1”, “eth_type”: 2048
ipv4_dst	IPv4 destination address (string)	{“ipv4_dst”: “192.168.10.10/255.255.255.0”,
tcp_src	TCP source port (int)	{“tcp_src”: 3, “ip_proto”: 6, “eth_type”: 2048

Table 1 – continued from previous page

Match field	Description	Example
tcp_dst	TCP destination port (int)	{“tcp_dst”: 5, “ip_proto”: 6, “eth_type”: 2048
udp_src	UDP source port (int)	{“udp_src”: 2, “ip_proto”: 17, “eth_type”: 20
udp_dst	UDP destination port (int)	{“udp_dst”: 6, “ip_proto”: 17, “eth_type”: 20
sctp_src	SCTP source port (int)	{“sctp_src”: 99, “ip_proto”: 132, “eth_type”:
sctp_dst	SCTP destination port (int)	{“sctp_dst”: 99, “ip_proto”: 132, “eth_type”:
icmpv4_type	ICMP type (int)	{“icmpv4_type”: 5, “ip_proto”: 1, “eth_type”
icmpv4_code	ICMP code (int)	{“icmpv4_code”: 6, “ip_proto”: 1, “eth_type’
arp_op	ARP opcode (int)	{“arp_op”: 3, “eth_type”: 2054}
arp_spa	ARP source IPv4 address (string)	{“arp_spa”: “192.168.0.11”, “eth_type”: 2054
arp_tpa	ARP target IPv4 address (string)	{“arp_tpa”: “192.168.0.44/24”, “eth_type”: 2
arp_sha	ARP source hardware address (string)	{“arp_sha”: “aa:bb:cc:11:22:33”, “eth_type”:
arp_tha	ARP target hardware address (string)	{“arp_tha”: “aa:bb:cc:11:22:33/00:00:00:00:f
ipv6_src	IPv6 source address (string)	{“ipv6_src”: “2001::aaaa:bbbb:cccc:1111”, “
ipv6_dst	IPv6 destination address (string)	{“ipv6_dst”: “2001::ffff:cccc:bbbb:1111/64”,
ipv6_flabel	IPv6 Flow Label (int)	{“ipv6_flabel”: 2, “eth_type”: 34525}
icmpv6_type	ICMPv6 type (int)	{“icmpv6_type”: 3, “ip_proto”: 58, “eth_type
icmpv6_code	ICMPv6 code (int)	{“icmpv6_code”: 4, “ip_proto”: 58, “eth_type
ipv6_nd_target	Target address for Neighbor Discovery (string)	{“ipv6_nd_target”: “2001::ffff:cccc:bbbb:111
ipv6_nd_sll	Source link-layer for Neighbor Discovery (string)	{“ipv6_nd_sll”: “aa:bb:cc:11:22:33”, “icmpv6
ipv6_nd_tll	Target link-layer for Neighbor Discovery (string)	{“ipv6_nd_tll”: “aa:bb:cc:11:22:33”, “icmpv6
mpls_label	MPLS label (int)	{“mpls_label”: 3, “eth_type”: 34888}
mpls_tc	MPLS Traffic Class (int)	{“mpls_tc”: 2, “eth_type”: 34888}
mpls_bos	MPLS BoS bit (int) (Openflow1.3+)	{“mpls_bos”: 1, “eth_type”: 34888}
pbb_isid	PBB I-SID (int or string) (Openflow1.3+)	{“pbb_isid”: 5, “eth_type”: 35047} or{“pbb_
tunnel_id	Logical Port Metadata (int or string) (Openflow1.3+)	{“tunnel_id”: 7} or {“tunnel_id”: “0x07/0xff”
ipv6_exthdr	IPv6 Extension Header pseudo-field (int or string) (Openflow1.3+)	{“ipv6_exthdr”: 3, “eth_type”: 34525} or {“ip
pbb_uca	PBB UCA hander field(int) (Openflow1.4+)	{“pbb_uca”: 1, “eth_type”: 35047}
tcp_flags	TCP flags(int) (Openflow1.5+)	{“tcp_flags”: 2, “ip_proto”: 6, “eth_type”: 20
actset_output	Output port from action set metadata(int) (Openflow1.5+)	{“actset_output”: 3}
packet_type	Packet type value(int) (Openflow1.5+)	{“packet_type”: [1, 2048]}

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:ffff::0"
```

Metadata:

```
"0x1212121212121212"
"0x3434343434343434/0x01010101010101010"
```

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":[
        {
```

(continues on next page)

(continued from previous page)

```
            "type":"OUTPUT",
            "port": 1
        }
    ]
 }' http://localhost:8080/stats/flowentry/add
```

- To match only packets with VLAN tag and VLAN ID equal value 5:

```
$ curl -X POST -d '{
    "dpid": 1,
    "match":{
        "dl_vlan": "0x1005"   # Describe sum of VLAN-ID(e.g. 5) | OFPVID_
↪PRESENT(0x1000)
    },
    "actions":[
        {
            "type":"OUTPUT",
            "port": 1
        }
    ]
 }' http://localhost:8080/stats/flowentry/add
```

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):

Actions	Description	Example
OUTPUT	Output packet from “port”	{“type”: “OUTPUT”, “port”: 3}
SET_VLAN_VID	Set the 802.1Q VLAN ID using “vlan_vid”	{“type”: “SET_VLAN_VID”, “vlan_vid”: 5}
SET_VLAN_PCP	Set the 802.1Q priority using “vlan_pcp”	{“type”: “SET_VLAN_PCP”, “vlan_pcp”: 3}
STRIP_VLAN	Strip the 802.1Q header	{“type”: “STRIP_VLAN”}
SET_DL_SRC	Set ethernet source address using “dl_src”	{“type”: “SET_DL_SRC”, “dl_src”: “aa:bb:cc:11:22:33”}
SET_DL_DST	Set ethernet destination address using “dl_dst”	{“type”: “SET_DL_DST”, “dl_dst”: “aa:bb:cc:11:22:33”}
SET_NW_SRC	IP source address using “nw_src”	{“type”: “SET_NW_SRC”, “nw_src”: “10.0.0.1”}
SET_NW_DST	IP destination address using “nw_dst”	{“type”: “SET_NW_DST”, “nw_dst”: “10.0.0.1”}
SET_NW_TOS	Set IP ToS (DSCP field, 6 bits) using “nw_tos”	{“type”: “SET_NW_TOS”, “nw_tos”: 184}
SET_TP_SRC	Set TCP/UDP source port using “tp_src”	{“type”: “SET_TP_SRC”, “tp_src”: 8080}
SET_TP_DST	Set TCP/UDP destination port using “tp_dst”	{“type”: “SET_TP_DST”, “tp_dst”: 8080}
ENQUEUE	Output to queue with “queue_id” attached to “port”	{“type”: “ENQUEUE”, “queue_id”: 3, “port”: 1}

List of Actions (OpenFlow1.2 or later):

Actions	Description	Example
OUTPUT	Output packet from “port”	{“type”: “OUTPUT”, “port”: 3}
COPY_TTL_OUT	Copy TTL outwards	{“type”: “COPY_TTL_OUT”}
COPY_TTL_IN	Copy TTL inwards	{“type”: “COPY_TTL_IN”}
SET_MPLS_TTL	Set MPLS TTL using “mpls_ttl”	{“type”: “SET_MPLS_TTL”, “mpls_ttl”: 64}
DEC_MPLS_TTL	Decrement MPLS TTL	{“type”: “DEC_MPLS_TTL”}
PUSH_VLAN	Push a new VLAN tag with “ethertype”	{“type”: “PUSH_VLAN”, “ethertype”: 33024}
POP_VLAN	Pop the outer VLAN tag	{“type”: “POP_VLAN”}
PUSH_MPLS	Push a new MPLS tag with “ethertype”	{“type”: “PUSH_MPLS”, “ethertype”: 34887}
POP_MPLS	Pop the outer MPLS tag with “ethertype”	{“type”: “POP_MPLS”, “ethertype”: 2054}
SET_QUEUE	Set queue id using “queue_id” when outputting to a port	{“type”: “SET_QUEUE”, “queue_id”: 7}
GROUP	Apply group identified by “group_id”	{“type”: “GROUP”, “group_id”: 5}
SET_NW_TTL	Set IP TTL using “nw_ttl”	{“type”: “SET_NW_TTL”, “nw_ttl”: 64}
DEC_NW_TTL	Decrement IP TTL	{“type”: “DEC_NW_TTL”}
SET_FIELD	Set a “field” using “value” (The set of keywords available for “field” is the same as match field)	See *Example of set-field action*
PUSH_PBB	Push a new PBB service tag with “ethertype” (Openflow1.3+)	{“type”: “PUSH_PBB”, “ethertype”: 35047}
POP_PBB	Pop the outer PBB service tag (Openflow1.3+)	{“type”: “POP_PBB”}
COPY_FIELD	Copy value between header and register (Openflow1.5+)	{“type”: “COPY_FIELD”, “n_bits”: 32, “src_offset”: 1, “dst_offset”: 2, “src_oxm_id”: “eth_src”, “dst_oxm_id”: “eth_dst”}
METER	Apply meter identified by “meter_id” (Openflow1.5+)	{“type”: “METER”, “meter_id”: 3}
EXPERIMENTER	Extensible action for the experimenter (Set “base64” or “ascii” to “data_type” field)	{“type”: “EXPERIMENTER”, “experimenter”: 101, “data”: “AAECAwQFBgc=”, “data_type”: “base64”}
GOTO_TABLE	(Instruction) Setup the next table identified by “table_id”	{“type”: “GOTO_TABLE”, “table_id”: 8}
WRITE_METADATA	(Instruction) Setup the metadata field using “metadata” and “metadata_mask”	{“type”: “WRITE_METADATA”, “metadata”: 0x3, “metadata_mask”: 0x3}
METER	(Instruction) Apply meter identified by “meter_id” (deprecated in Openflow1.5)	{“type”: “METER”, “meter_id”: 3}
WRITE_ACTIONS	(Instruction) Write the action(s) onto the datapath action set	{“type”: “WRITE_ACTIONS”, actions”:[{“type”:”POP_VLAN”,},{ “type”:”OUTPUT”, “port”: 2}]}
CLEAR_ACTIONS	(Instruction) Clears all actions from the datapath action set	{“type”: “CLEAR_ACTIONS”}

Example of set-field action

To set VLAN ID to non-VLAN-tagged frame:

```
$ 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␣
↪VLAN-tagged frame
        },
        {
            "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

6.3.1 REST API

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

r

Index

S

T

U

V

W