自学习交换机

```python
#自学习交换机的部署
#原包里对应文件example_switch_13.py
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet


class ExampleSwitch13(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super(ExampleSwitch13, self).__init__(*args, **kwargs)
        # initialize mac address table.
        self.mac_to_port = {}

    @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
    def switch_features_handler(self, ev):
        datapath = ev.msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        # install the table-miss flow entry.
        match = parser.OFPMatch()
        actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
                                          ofproto.OFPCML_NO_BUFFER)]
        self.add_flow(datapath, 0, match, actions)

    def add_flow(self, datapath, priority, match, actions):
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        # construct flow_mod message and send it.
        inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
                                             actions)]
        mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
                                match=match, instructions=inst)
        datapath.send_msg(mod)

    @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
    def _packet_in_handler(self, ev):
        msg = ev.msg
        datapath = msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        # get Datapath ID to identify OpenFlow switches.
```

```
        dpid = datapath.id
        self.mac_to_port.setdefault(dpid, {})

        # analyse the received packets using the packet library.
        pkt = packet.Packet(msg.data)
        eth_pkt = pkt.get_protocol(ethernet.ethernet)
        dst = eth_pkt.dst
        src = eth_pkt.src

        # get the received port number from packet_in message.
        in_port = msg.match['in_port']

        self.logger.info("packet in %s %s %s %s", dpid, src, dst, in_port)

        # learn a mac address to avoid FLOOD next time.
        self.mac_to_port[dpid][src] = in_port

        # if the destination mac address is already learned,
        # decide which port to output the packet, otherwise FLOOD.
        if dst in self.mac_to_port[dpid]:
            out_port = self.mac_to_port[dpid][dst]
        else:
            out_port = ofproto.OFPP_FLOOD

        # construct action list.
        actions = [parser.OFPActionOutput(out_port)]

        # install a flow to avoid packet_in next time.
        if out_port != ofproto.OFPP_FLOOD:
            match = parser.OFPMatch(in_port=in_port, eth_dst=dst)
            self.add_flow(datapath, 1, match, actions)

        # construct packet_out message and send it.
        out = parser.OFPPacketOut(datapath=datapath,
                                  buffer_id=ofproto.OFP_NO_BUFFER,
                                  in_port=in_port, actions=actions,
                                  data=msg.data)
        datapath.send_msg(out)
```

结合之前openflow的学习，下面详解代码：

1.1 Flow-Mod消息

Flow-Mod消息用于流表操作，包括添加、删除、修改流表项。该消息由控制器下发给交换机，从而指导交换机对数据包的处理

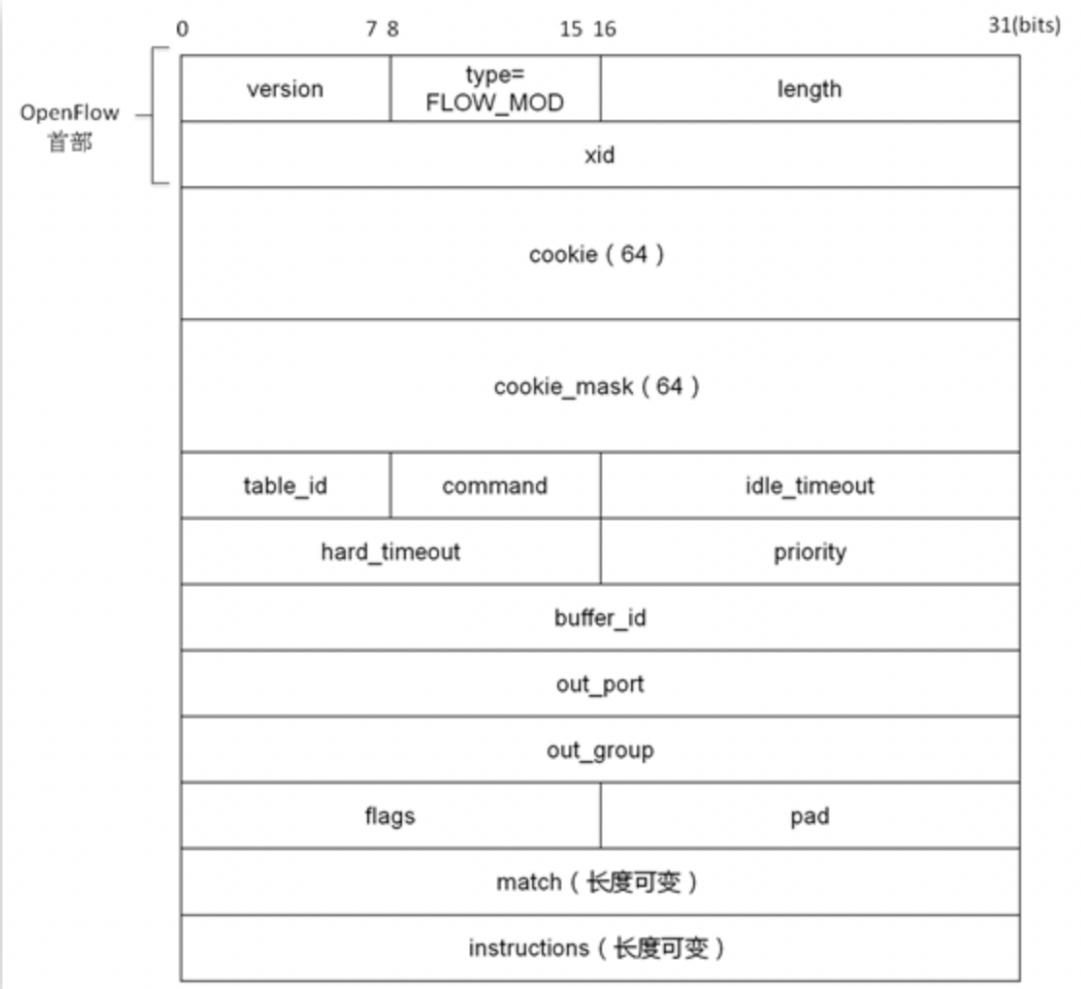
```python
def add_flow(self, datapath, priority, match, actions):
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser

    # construct flow_mod message and send it.
    inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
                                         actions)]
    mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
                            match=match, instructions=inst)
    datapath.send_msg(mod)
```

动作（Action）Action是指对数据包的具体处理动作，可分为两类

> 一类是定义数据包的转发，另一类是修改数据包包头字段

指令名称	OpenFlow交换机是否支持	功能
Apply-Actions	可选	立即执行动作列表（Action List）中的动作，且不改变动作集（Action Set）
Clear-Actions	必须	清空动作集
Write-Actions	必须	添加一条动作到动作集或修改动作集中的动作
Goto-Table	必须	跳转到指定流表
Write-Metadata	可选	更新元数据，在多个流表之间传递信息
Stat-Trigger	可选	若流的某个统计信息超过设定阈值，生成一个事件通知控制器

其在OpenFlow1.3中的消息格式如下图所示。



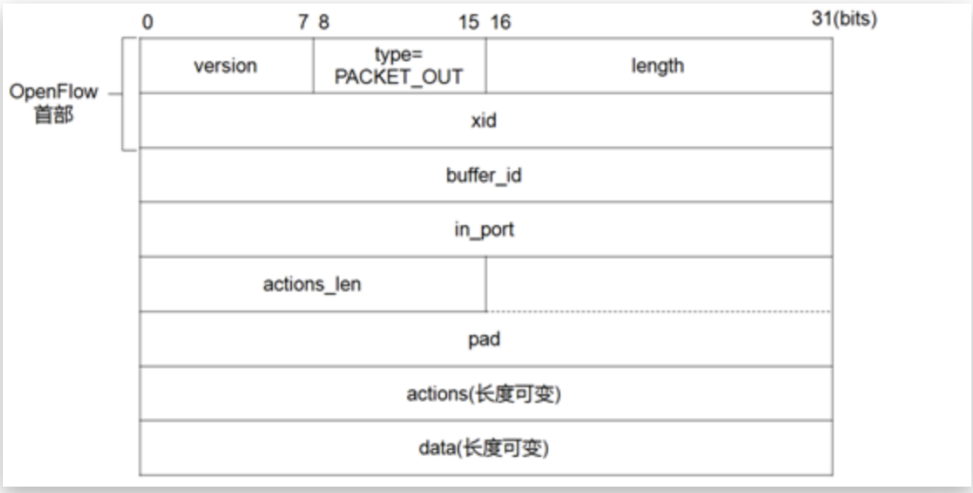
- command：
  - ADD:添加流表项
  - MODIFY：根据匹配域，修改所有匹配的流表项，可能有多条流表项被修改
  - MODIFY_STRICT：根据匹配域以及优先级，修改特定的流表项，只有一条流表项被修改
  - DELETE：根据匹配域，删除所有匹配的流表项，可能有多条流表项被删除
  - DELETE_STRICT：根据匹配域以及优先级，删除特定的流表项，只有一条流表项被删除

1.2 Packet-Out消息

```
    # construct packet_out message and send it.
    out = parser.OFPPacketOut(datapath=datapath,
                              buffer_id=ofproto.OFP_NO_BUFFER,
                              in_port=in_port, actions=actions,
                              data=msg.data)
    datapath.send_msg(out)
```

Packet-Out消息用于指定交换机将数据包从指定端口转发出去。触发该消息的情况有两种：1.转发Packet-In消息携带的数据包 2.转发控制器主动构造的数据包（如用于链路发现的LLDP报文）。此外，由于该消息能够携带自定义数据包，控制器通过在Packet-Out消息中封装探测包并下发至指定交换机，就能够发起主动测量任务，配合Packet-In消息可实现对网络拓扑与链路时延的测量，详情参考基于OpenFlow消息的网络测量方法。

其在OpenFlow1.3中的消息格式如下图所示。



- buffer_id：若该字段为-1，表明交换机未缓存数据包，Packet-Out消息携带控制器创建的数据包发送至交换机；否则，Packet-Out消息表示交换机需要将本地缓存的数据包按照Packet-Out中的actions进行处理。

Packet-Out 讯息相对应的类别是 `OFPPacketOut`。

OFPPacketOut 建构子的参数如下所示。

datapath

指定 OpenFlow 交换器对应的 Datapath 类别实体。

buffer_id

指定 OpenFlow 交换器上封包对应的缓冲区。如果不想使用缓冲区，则指定为 `OFP_NO_BUFFER`。

in_port

指定接收封包的端口号。如果不想使用的话就指定为 `OFPP_CONTROLLER`。

actions

指定 actions list。

data

设定封包的 binary data。主要用在 buffer_id 为 `OFP_NO_BUFFER` 的情况。如果使用了 OpenFlow 交换器的缓冲区则可以省略。

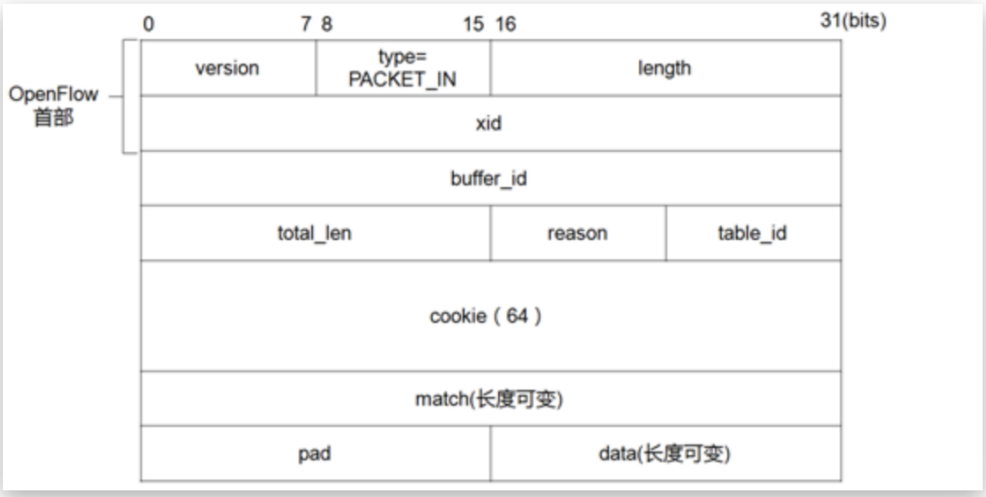
交换器的实作时，在 Packet-In 讯息中指定 buffer_id。若是 Packet-In 讯息中 buffer_id 被设定为无效时。Packet-In 的封包必须指定 data 以便传送。

1.3 Packet-In消息

```
@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
    msg = ev.msg
    datapath = msg.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser
```

Packet-In消息用于将OpenFlow交换机上指定数据包交给控制器处理，一般流表匹配中出现Table-Miss时或流表项显示指定将数据包交给控制器时，触发该消息。此外，它还能用于主动测量时回收探测包，从而结合Packet-Out消息实现对网络拓扑与链路时延的测量，详情参考基于OpenFlow消息的网络测量方法。

其在OpenFlow1.3中的消息格式如下图所示。



- buffer_id：若该字段为-1，表明交换机未缓存数据包，Packet-In消息需要携带完整数据包发送至控制器；否则，表明数据包已在交换机上缓存，Packet-In消息只携带部分数据包上传至控制器。