```
import struct
from ryu.lib import addrconv
from ryu.lib import lacplib
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER
from ryu.controller.handler import MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3
from ryu.lib import lacplib
from ryu.lib.dpid import str_to_dpid
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet


class SimpleSwitchLacp13(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]
    _CONTEXTS = {'lacplib': lacplib.LacpLib}

    def __init__(self, *args, **kwargs):
        super(SimpleSwitchLacp13,self).__init__(*args, **kwargs)
        self.mac_to_port = {}
        self._lacp =kwargs['lacplib']
        self._lacp.add(
            dpid=str_to_dpid('0000000000000001'),ports=[1,2])

    @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 table-miss flow entry
        # We specify NO BUFFER to max_len of the output action due to
        # OVS bug. At this moment, if we specify a lesser number e.g.,
        # 128, OVS will send Packet-In with invalid buffer_id and
        # truncated packet data. In that case, we cannot output packets
        # correctly.

        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
```

```
46          inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,actions)]
47          mod = parser.OFPFlowMod(datapath=datapath, priority=priority, match=match,
instructions=inst)
48          datapath.send_msg(mod)
49      def del_flow(self, datapath, match):
50          ofproto = datapath.ofproto
51          parser = datapath.ofproto_parser
52          mod = parser.OFPFlowMod(datapath=datapath,
53                  command=ofproto.OFPFC_DELETE, out_port=ofproto.OFPP_ANY,
54                  out_group=ofproto.OFPG_ANY, match=match)
55          datapath.send_msg(mod)
56      @set_ev_cls(lacplib.EventPacketIn, MAIN_DISPATCHER)
57      def _packet_in_handler(self, ev):
58          msg = ev.msg
59          datapath = msg.datapath
60          ofproto = datapath.ofproto
61          parser = datapath.ofproto_parser
62          in_port = msg.match['in_port']
63
64          pkt = packet.Packet(msg.data)
65          eth = pkt.get_protocols(ethernet.ethernet)[0]
66
67          dst = eth.dst
68          src = eth.src
69
70          dpid = datapath.id self.mac_to_port.setdefault(dpid, {})
71
72          self.logger.info("packet in %s %s %s %s", dpid, src, dst, in_port)
73
74          # learn a mac address to avoid FLOOD next time.
75          self.mac_to_port[dpid][src] = in_port
76
77          if dst in self.mac_to_port[dpid]:
78              out_port = self.mac_to_port[dpid][dst]
79          else:
80              out_port = ofproto.OFPP_FLOOD
81
82          actions = [parser.OFPActionOutput(out_port)]
83
84          # install a flow to avoid packet_in next time
85          if out_port != ofproto.OFPP_FLOOD:
86              match = parser.OFPMatch(in_port=in_port, eth_dst=dst)
87              self.add_flow(datapath, 1, match, actions)
88
89          data = None
90          if msg.buffer_id == ofproto.OFP_NO_BUFFER:
```

```
91            data = msg.data
92
93        out = parser.OFPPacketOut(datapath=datapath, buffer_id=msg.buffer_id,
94                                  in_port=in_port, actions=actions, data=data)
95        datapath.send_msg(out)
96
97    @set_ev_cls(lacplib.EventSlaveStateChanged, MAIN_DISPATCHER)
98    def _slave_state_changed_handler(self, ev):
99        datapath = ev.datapath
100       dpid = datapath.id
101       port_no = ev.port
102       enabled = ev.enabled
103       self.logger.info("slave state changed port: %d enabled: %s",
104                        port_no, enabled)
105       if dpid in self.mac_to_port:
106           for mac in self.mac_to_port[dpid]:
107               match = datapath.ofproto_parser.OFPMatch(eth_dst=mac)
108               self.del_flow(datapath, match)
109           del self.mac_to_port[dpid]
110       self.mac_to_port.setdefault(dpid, {})
```