Ryu的应用开发（二）Learning Switch

一：自学习交换机（二层MAC交换机）的编程思路

明确问题
设计解决方案
确定具体技术方案
部署实施
验证方案
优化

（一）明确问题

如何实现软件定义的自学习交换机?

（二）设计解决方案

通过控制器来实现自学习交换算法，然后指导数据平面实现交换机操作

（三）确定具体的技术方案

控制器选用Ryu,数据平面通过Mininet模拟

（四）部署实施

在控制器上编程开发交换机应用，创建实验网络为验证方案做准备

（五）验证方案

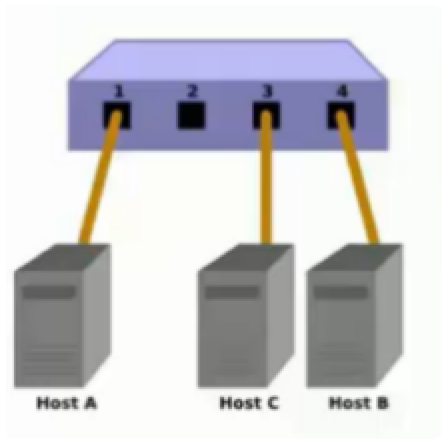
运行程序，调试程序，验证程序

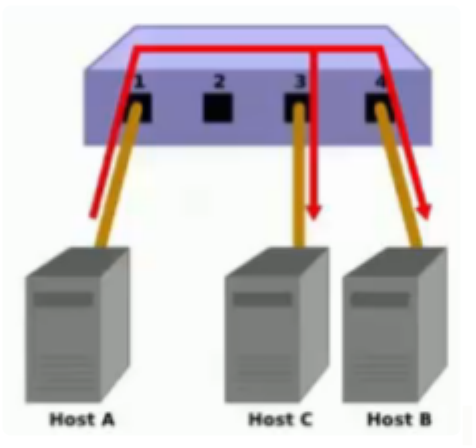
（六）优化

验证成功后，优化程序

二：自学习交换机原理

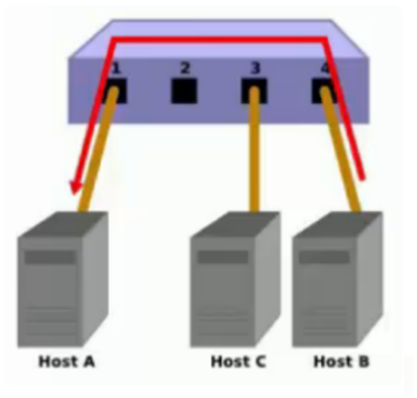
（一）普通交换机实现





交换机MAC地址表记录了统一网段中的各个主机对应**交换机的端口**和**主机的MAC地址**

当主机A要和主机 B 通信时，初始交换机M A C 表是空的，会先记录主机A的MAC地址和对应的交换机端口，然后查找交换机MAC中是否有目标M A C 地址，没有找到，会向其他所有端口泛洪查找

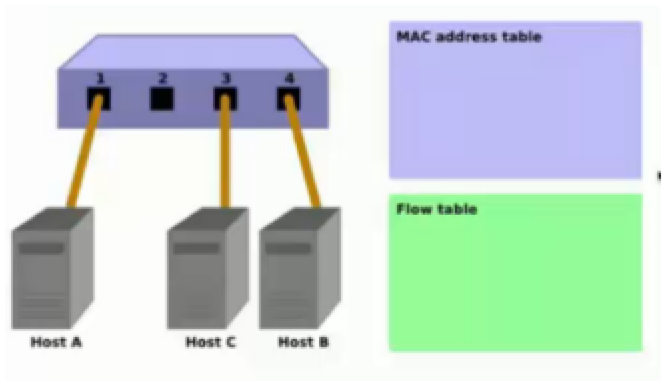


泛洪，通知其他主机。主机C接收到数据包，发现不是自己的，则不处理，丢弃数据包。当主机B接收后，发现是找自己的，则可以进行消息通信。交换机先进行MAC学习，记录主机B的M A C 信息，再进行查表转发，单播发送给主机 A

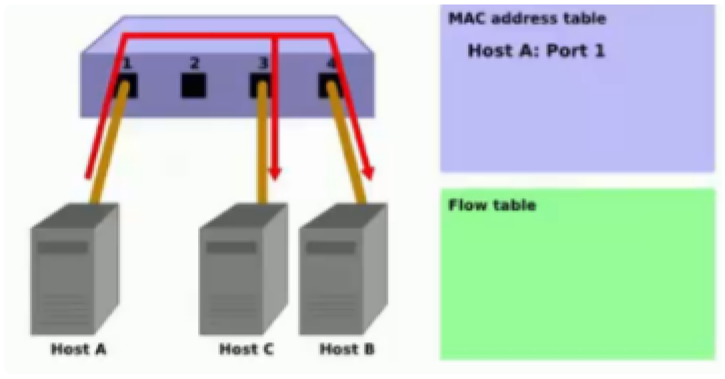
（二）SDN中交换机实现

S D N 中交换机不存储MAC表，（datapath)只存在流表。其地址学习操作由控制器(控制器中包含MAC 地址表)实现，之后控制器下发流表项给交换机

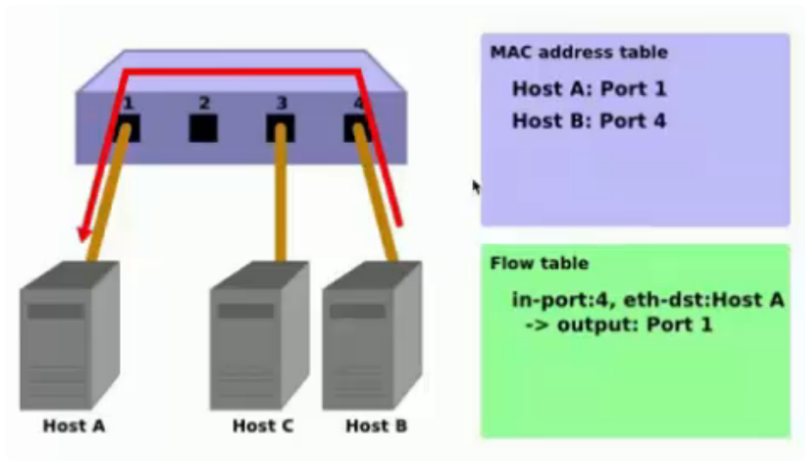
1.主机A向主机B发送信息，流表中只存在默认流表，告诉交换机将数据包发送给控制器。



2.控制器先进行MAC地址学习，记录主机A的MAC地址和其对应交换机端口，然后查询MAC地址表，查找主机Ｂ信息。没有则下发流表项告诉交换机先泛洪试试



3.泛洪后，主机Ｃ接收后丢弃数据包，不处理。主机Ｂ发现是寻找自己的，则进行消息回送，由于交换机流表中没有处理主机Ｂ到主机Ａ的信息的流表项，所以只能向控制器发送数据包。控制器先学习主机Ｂ的ＭＡＣ地址和对应交换机端口，之后查询ＭＡＣ地址表，找到主机A的MAC信息，下发流表项，告诉交换机如何处理主机B–>主机A的消息



4.注意：这里交换机的流表项中只存在主机B–>主机A的流表项处理方案，不存在主机Ａ－>主机Ｂ的处理流表项（但是控制器ＭＡＣ地址表中是存在主机Ｂ的信息），所以会在下一次数据传送中，控制器下发响应的流表项。但是其实可以实现（在３中一次下发两个流表项）

三：代码实现

（一）全部代码

Python 复制代码

```python
from ryu.base import app_manager
from ryu.ofproto import ofproto_v1_3
from ryu.controller import ofp_event
from ryu.controller.handler import set_ev_cls
from ryu.controller.handler import CONFIG_DISPATCHER,MAIN_DISPATCHER
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet

class SelfLearnSwitch(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]    #set openflow protocol version while we support

    def __init__(self,*args,**kwargs):
        super(SelfLearnSwitch,self).__init__(*args,**kwargs)
        #set a data construction to save MAC Address Table
        self.Mac_Port_Table={}

    @set_ev_cls(ofp_event.EventOFPSwitchFeatures)
    def switch_features_handler(self,ev):
        '''
        manage the initial link, from switch to controller
        '''
        #first parse event to get datapath and openflow protocol
        msg = ev.msg
        datapath = msg.datapath
        ofproto = datapath.ofproto
        ofp_parser = datapath.ofproto_parser

        self.logger.info("datapath: %s link to controller",datapath.id)

        #secondly set match and action
        match = ofp_parser.OFPMatch()    #all data message match successful
        actions = [ofp_parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,ofproto.OFPCML_NO_BUFFER)]    #set receive port and buffer for switch

        #add flow and send it to switch in add_flow
        self.add_flow(datapath,0,match,actions,"default flow entry")

    def add_flow(self,datapath,priority,match,actions,extra_info):
        """
        add flow entry to switch
        """

        #get open flow protocol infomation
        ofproto = datapath.ofproto
        ofp_parser = datapath.ofproto_parser

        #set instruction infomation from openflow protocol 1.3
        inst = [ofp_parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,actions)]

        #set flow entry mod
        mod = ofp_parser.OFPFlowMod(datapath=datapath,priority=priority,match=match,instructions=inst)

        print("send "+extra_info)
        #send flow entry to switch
        datapath.send_msg(mod)

    @set_ev_cls(ofp_event.EventOFPPacketIn,MAIN_DISPATCHER)
    def packet_in_handler(self,ev):
        '''
        manage infomation from switch
        '''

        #first parser openflow protocol
        msg = ev.msg
        datapath = msg.datapath
        ofproto = datapath.ofproto
        ofp_parser = datapath.ofproto_parser

        #get datapath id from datapath, and save dpid into MAC table (default)
        dpid = datapath.id
        self.Mac_Port_Table.setdefault(dpid, {})

        #analysize packet, get ethernet data, get host MAC info
        pkt = packet.Packet(msg.data)
        eth_pkt = pkt.get_protocol(ethernet.ethernet)
        dst = eth_pkt.dst
        src = eth_pkt.src

        #get switch port where host packet send in
        in_port = msg.match['in_port']

        self.logger.info("Controller %s get packet, Mac address from: %s send to: %s , send from datapath: %s,in port is: %s"
                            ,dpid,src,dst,dpid,in_port)

        #save src data into dictionary---MAC address table
        self.Mac_Port_Table[dpid][src] = in_port
        #query MAC address table to get destinction host`s port from current datapath
        #---first: find port to send packet
        #---second: not find port,so send packet by flood
        if dst in self.Mac_Port_Table[dpid]:
            Out_Port = self.Mac_Port_Table[dpid][dst]
        else:
            Out_Port = ofproto.OFPP_FLOOD

        #set match-action from above status
        actions = [ofp_parser.OFPActionOutput(Out_Port)]

        #add a new flow entry to switch by add_flow
        if Out_Port != ofproto.OFPP_FLOOD:    #if Out_port == ofproto.OFPP_FLOOD ---> flow entry == default flow entry, it already exist
            match = ofp_parser.OFPMatch(in_port=in_port,eth_dst = dst)
            self.add_flow(datapath, 1, match, actions,"a new flow entry by specify port")
            self.logger.info("send packet to switch port: %s",Out_Port)

        #finally send the packet to datapath, to achive self_learn_switch
        Out = ofp_parser.OFPPacketOut(datapath=datapath,buffer_id=msg.buffer_id,
                                in_port=in_port,actions=actions,data=msg.data)

        datapath.send_msg(Out)
```

（二）代码讲解（一）

Python 复制代码

```python
from ryu.base import app_manager
from ryu.ofproto import ofproto_v1_3
from ryu.controller import ofp_event
from ryu.controller.handler import set_ev_cls
from ryu.controller.handler import CONFIG_DISPATCHER,MAIN_DISPATCHER
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet

class SelfLearnSwitch(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]    #set openflow protocol version while we support

    def __init__(self,*args,**kwargs):
        super(SelfLearnSwitch,self).__init__(*args,**kwargs)
        #set a data construction to save MAC Address Table
        self.Mac_Port_Table={}

    @set_ev_cls(ofp_event.EventOFPSwitchFeatures)
    def switch_features_handler(self,ev):
        '''
        manage the initial link, from switch to controller
        '''
        #first parse event to get datapath and openflow protocol
        msg = ev.msg
        datapath = msg.datapath
        ofproto = datapath.ofproto
        ofp_parser = datapath.ofproto_parser

        self.logger.info("datapath: %s link to controller",datapath.id)

        #secondly set match and action
        match = ofp_parser.OFPMatch()    #all data message match successful
        actions = [ofp_parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,ofproto.OFPCML_NO_BUFFER)]    #set receive port and buffer for switch

        #add flow and send it to switch in add_flow
        self.add_flow(datapath,0,match,actions,"default flow entry")

    def add_flow(self,datapath,priority,match,actions,extra_info):
        """
        add flow entry to switch
        """


        #get open flow protocol infomation
        ofproto = datapath.ofproto
        ofp_parser = datapath.ofproto_parser

        #set instruction infomation from openflow protocol 1.3
        inst = [ofp_parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,actions)]

        #set flow entry mod
        mod = ofp_parser.OFPFlowMod(datapath=datapath,priority=priority,match=match,instructions=inst)

        print("send "+extra_info)
        #send flow entry to switch
        datapath.send_msg(mod)
```

以上代码同SDN实验---Ryu的应用开发（一）Hub实现，实现了设备与控制器初始连接，下发默认流表项，使得默认情况下，交换机在无法匹配到流表项时，直接去找控制器。一个一个公共函数add_flow实现流表下发。注意：在__init__方法中实现了数据结构《字典》去存储MAC地址表，为下面做准备

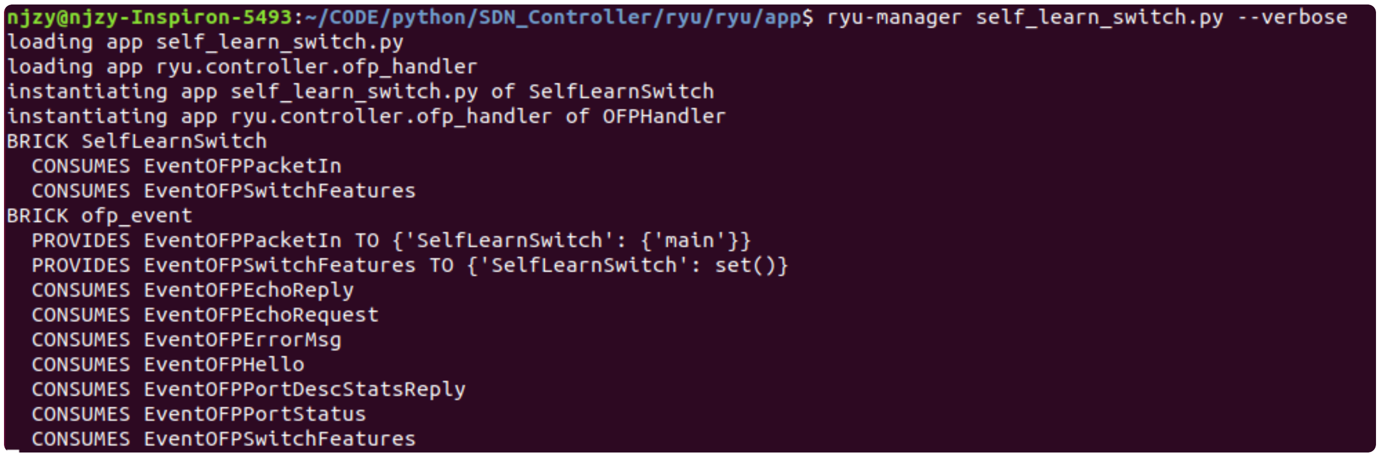
（三）代码讲解（二）

Python 复制代码

```python
 @set_ev_cls(ofp_event.EventOFPPacketIn,MAIN_DISPATCHER)
    def packet_in_handler(self,ev):
        '''
        manage infomation from switch
        '''

        #first parser openflow protocol       # 先解析OpenFlow协议信息
        msg = ev.msg
        datapath = msg.datapath
        ofproto = datapath.ofproto
        ofp_parser = datapath.ofproto_parser

        #get datapath id from datapath, and save dpid into MAC table (default)     获取datapath(虚拟交换机的id),用dpid初始化一个键值
        dpid = datapath.id
        self.Mac_Port_Table.setdefault(dpid, {})

        #analysize packet, get ethernet data, get host MAC info 分析packet数据包，因为转发的包，都是基于以太网协议的，所以需要用到以太网协议进行解析，获取源MAC和目的MAC
        pkt = packet.Packet(msg.data)
        eth_pkt = pkt.get_protocol(ethernet.ethernet)
        dst = eth_pkt.dst
        src = eth_pkt.src

        #get switch port where host packet send in    获取datapath的数据输入端口
        in_port = msg.match['in_port']

        self.logger.info("Controller %s get packet, Mac address from: %s send to: %s , send from datapath: %s,in port is: %s"
                            ,dpid,src,dst,dpid,in_port)    #打印调试信息

        #save src data into dictionary---MAC address table    将源MAC地址保存，学习，放入MAC表中
        self.Mac_Port_Table[dpid][src] = in_port

        #query MAC address table to get destinction host`s port from current datapath    查询MAC表，是否有目标MAC地址的键值
        #---first: find port to send packet    如果找到，我们则按照该端口发送
        #---second: not find port,so send packet by flood    如果没有找到，我们需要泛洪发送给下一个（或者下几个）交换机，依次查询
        if dst in self.Mac_Port_Table[dpid]:
            Out_Port = self.Mac_Port_Table[dpid][dst]
        else:
            Out_Port = ofproto.OFPP_FLOOD

        #set match-action from above status    开始设置match-actions匹配动作
        actions = [ofp_parser.OFPActionOutput(Out_Port)]

        #add a new flow entry to switch by add_flow    进行对应的流表项下发《重点》
        if Out_Port != ofproto.OFPP_FLOOD:
            match = ofp_parser.OFPMatch(in_port=in_port,eth_dst = dst)
            self.add_flow(datapath, 1, match, actions,"a new flow entry by specify port")
            self.logger.info("send packet to switch port: %s",Out_Port)

        #finally send the packet to datapath, to achive self_learn_switch    最后我们将之前交换机发送上来的数据，重新发给交换机
        Out = ofp_parser.OFPPacketOut(datapath=datapath,buffer_id=msg.buffer_id,
                                in_port=in_port,actions=actions,data=msg.data)    #我们必须加上这个data,才可以将packet数据包发送回去《重点》不然会出错××××××

        datapath.send_msg(Out)
```

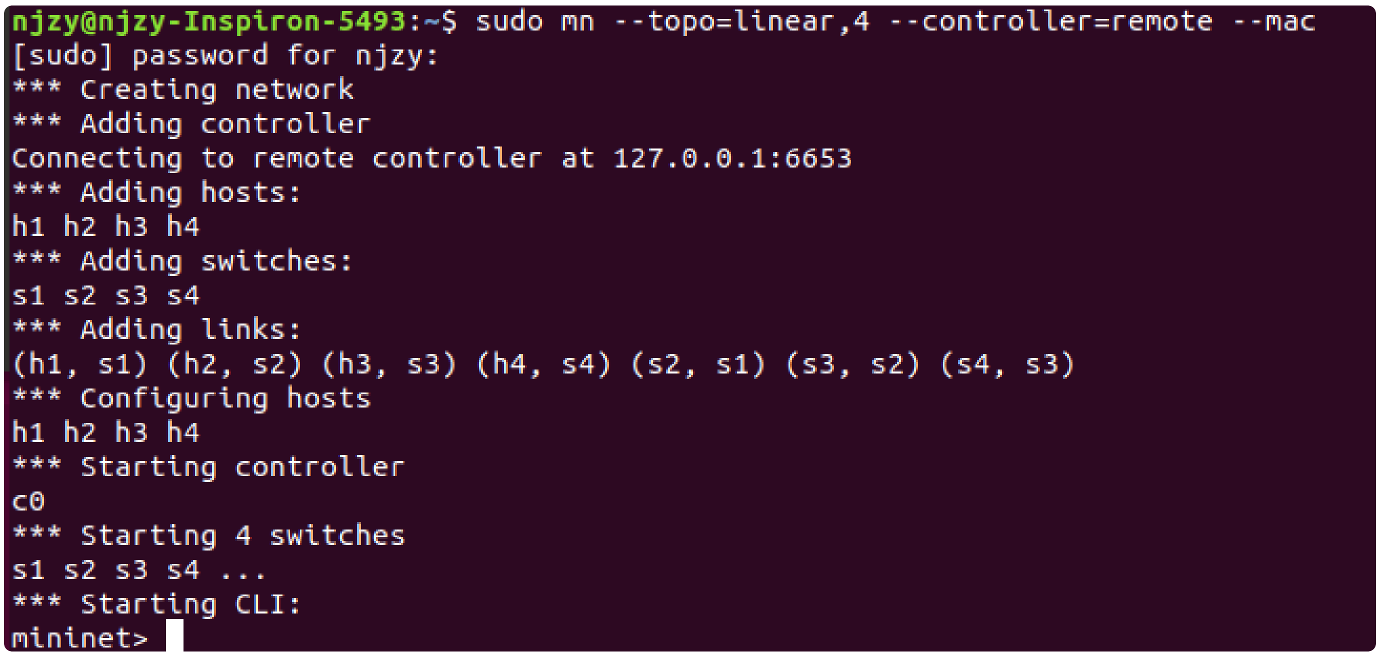
（四）实验演示

1.启动Ryu控制器



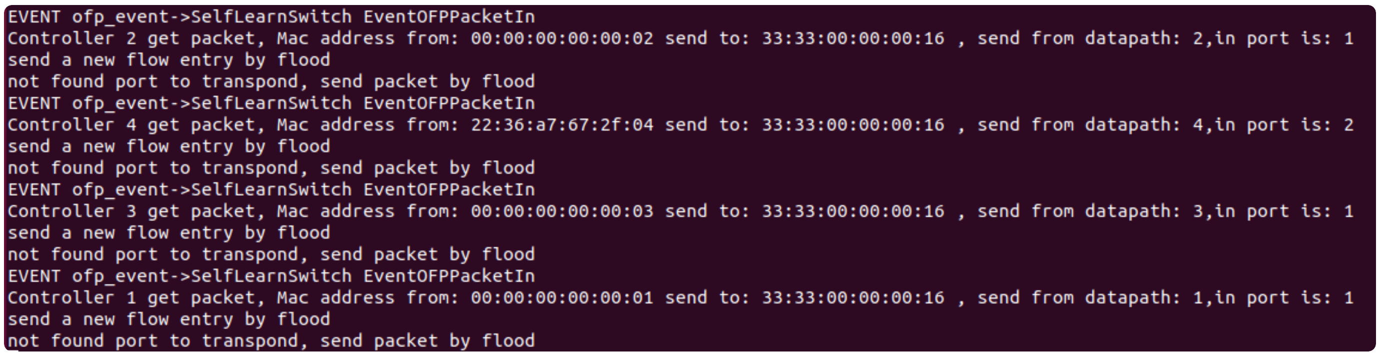
```
njzy@njzy-Inspiron-5493:~/CODE/python/SDN_Controller/ryu/ryu/app$ ryu-manager self_learn_switch.py --verbose
loading app self_learn_switch.py
loading app ryu.controller.ofp_handler
instantiating app self_learn_switch.py of SelfLearnSwitch
instantiating app ryu.controller.ofp_handler of OFPHandler
BRICK SelfLearnSwitch
  CONSUMES EventOFPPacketIn
  CONSUMES EventOFPSwitchFeatures
BRICK ofp_event
  PROVIDES EventOFPPacketIn TO {'SelfLearnSwitch': {'main'}}
  PROVIDES EventOFPSwitchFeatures TO {'SelfLearnSwitch': set()}
  CONSUMES EventOFPEchoReply
  CONSUMES EventOFPEchoRequest
  CONSUMES EventOFPErrorMsg
  CONSUMES EventOFPHello
  CONSUMES EventOFPPortDescStatsReply
  CONSUMES EventOFPPortStatus
  CONSUMES EventOFPSwitchFeatures
```

2.启动mininet



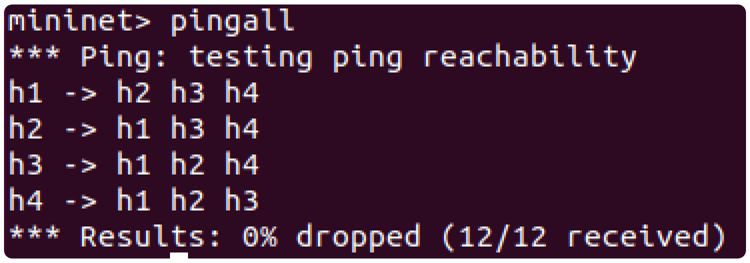
```
njzy@njzy-Inspiron-5493:~$ sudo mn --topo=linear,4 --controller=remote --mac
[sudo] password for njzy:
*** Creating network
*** Adding controller
Connecting to remote controller at 127.0.0.1:6653
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2 s3 s4
*** Adding links:
(h1, s1) (h2, s2) (h3, s3) (h4, s4) (s2, s1) (s3, s2) (s4, s3)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 4 switches
s1 s2 s3 s4 ...
*** Starting CLI:
mininet>
```

3.Ryu进行响应



```
EVENT ofp_event->SelfLearnSwitch EventOFPPacketIn
Controller 2 get packet, Mac address from: 00:00:00:00:00:02 send to: 33:33:00:00:00:16 , send from datapath: 2,in port is: 1
send a new flow entry by flood
not found port to transpond, send packet by flood
EVENT ofp_event->SelfLearnSwitch EventOFPPacketIn
Controller 4 get packet, Mac address from: 22:36:a7:67:2f:04 send to: 33:33:00:00:00:16 , send from datapath: 4,in port is: 2
send a new flow entry by flood
not found port to transpond, send packet by flood
EVENT ofp_event->SelfLearnSwitch EventOFPPacketIn
Controller 3 get packet, Mac address from: 00:00:00:00:00:03 send to: 33:33:00:00:00:16 , send from datapath: 3,in port is: 1
send a new flow entry by flood
not found port to transpond, send packet by flood
EVENT ofp_event->SelfLearnSwitch EventOFPPacketIn
Controller 1 get packet, Mac address from: 00:00:00:00:00:01 send to: 33:33:00:00:00:16 , send from datapath: 1,in port is: 1
send a new flow entry by flood
not found port to transpond, send packet by flood
```

注意：这里我一启动Mininet，就已经获取了所有的MAC信息，应该是主机接入网络后发送某些数据包，导致控制器获得了MAC表（需要使用wireshark抓包工具进行分析....后面进行补充）



```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
```

网络可达，说明实现自学习交换机

四：补充知识

（一）pkt = packet.Packet(msg.data) 一个类，在Ryu/lib/packet/模块下，用于包的解码/编码

```python
class Packet(StringifyMixin):
    """A packet decoder/encoder class.

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

    *data* is a bytearray to describe a raw datagram to decode.  data是一个未加工的报文数据，  即msg.data直接从事件的msg中获取的数据
    When decoding, a Packet object is iteratable.
    Iterated values are protocol (ethernet, ipv4, ...) headers and the payload.
    Protocol headers are instances of subclass of packet_base.PacketBase.
    The payload is a bytearray.  They are iterated in on-wire order.

    *data* should be omitted when encoding a packet.
    """

    # Ignore data field when outputting json representation.
    _base_attributes = ['data']

    def __init__(self, data=None, protocols=None, parse_cls=ethernet.ethernet):    # 协议解析，默认是按照以太网协议
        super(Packet, self).__init__()
        self.data = data
        if protocols is None:
            self.protocols = []
        else:
```

（二）eth_pkt = pkt.get_protocol(ethernet.ethernet) 返回与指定协议匹配的协议列表。从packet包中获取协议信息（协议包含我们需要的dst,src等，如三中所示）

```python
class Packet(StringifyMixin):

    def add_protocol(self, proto):
        """Register a protocol *proto* for this packet.

        This method is legal only when encoding a packet.

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

        self.protocols.append(proto)

    def get_protocols(self, protocol):
        """Returns a list of protocols that matches to the specified protocol.
        """
        if isinstance(protocol, packet_base.PacketBase):
            protocol = protocol.__class__
        assert issubclass(protocol, packet_base.PacketBase)
```

（三）eth_pkt = pkt.get_protocol(ethernet.ethernet) 一个类，也在Ryu/lib/packet/模块下，用于以太网报头编码器/解码器类。

Python 复制代码

```python
class ethernet(packet_base.PacketBase):
    """Ethernet header encoder/decoder class.

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

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

    _PACK_STR = '!6s6sH'
    _MIN_LEN = struct.calcsize(_PACK_STR)
    _MIN_PAYLOAD_LEN = 46
    _TYPE = {
        'ascii': [
            'src', 'dst'
        ]
    }

    def __init__(self, dst='ff:ff:ff:ff:ff:ff', src='00:00:00:00:00:00',
                 ethertype=ether.ETH_TYPE_IP):
        super(ethernet, self).__init__()
        self.dst = dst
        self.src = src
        self.ethertype = ethertype

    @classmethod
    def parser(cls, buf):
        dst, src, ethertype = struct.unpack_from(cls._PACK_STR, buf)
        return (cls(addrconv.mac.bin_to_text(dst),
                    addrconv.mac.bin_to_text(src), ethertype),
                ethernet.get_packet_type(ethertype),
                buf[ethernet._MIN_LEN:])

    def serialize(self, payload, prev):
        # Append padding if the payload is less than 46 bytes long
        pad_len = self._MIN_PAYLOAD_LEN - len(payload)
        if pad_len > 0:
            payload.extend(b'\x00' * pad_len)
        return struct.pack(ethernet._PACK_STR,
                           addrconv.mac.text_to_bin(self.dst),
                           addrconv.mac.text_to_bin(self.src),
                           self.ethertype)

    @classmethod
    def get_packet_type(cls, type_):
        """Override method for the ethernet IEEE802.3 Length/Type
        field (self.ethertype).

        If the value of Length/Type field is less than or equal to
        1500 decimal(05DC hexadecimal), it means Length interpretation
        and be passed to the LLC sublayer."""
        if type_ <= ether.ETH_TYPE_IEEE802_3:
            type_ = ether.ETH_TYPE_IEEE802_3
        return cls._TYPES.get(type_)
```