西安交通大学软件定义网络实验一

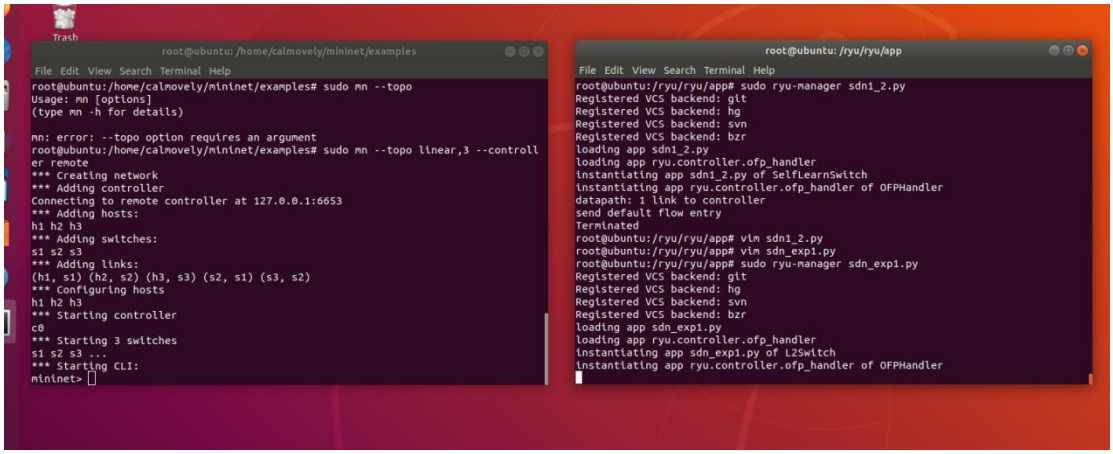
实验准备：

安装 mininet，ryu 等

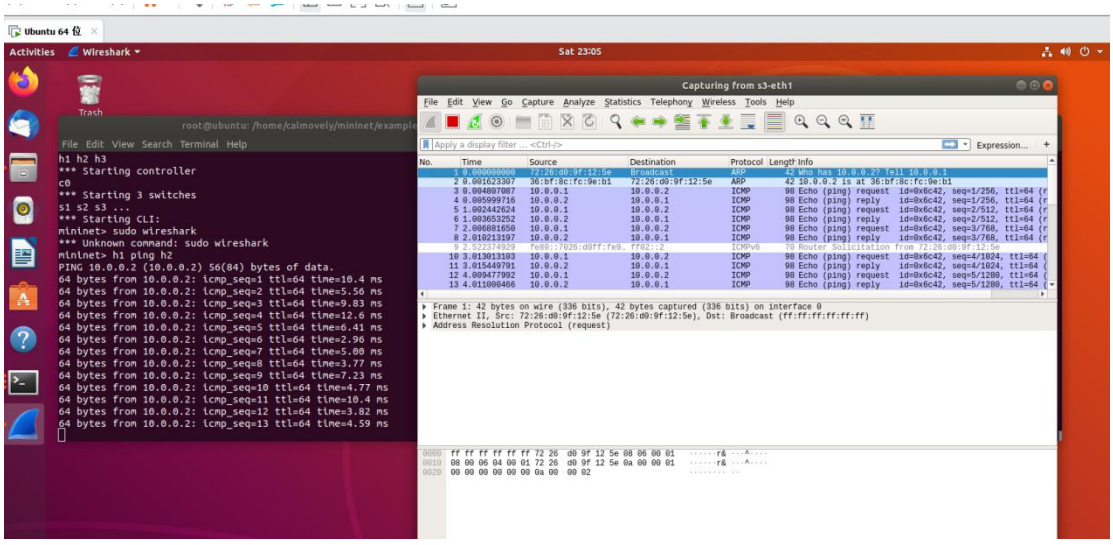
实验内容：

实验一：

先 cd 到 ryu/ryu/app 目录，将写好的 py 程序粘贴在目录下，同时开启 mininet 和 ryu，在另一个终端中执行，sudo wireshark 命令，开始抓包



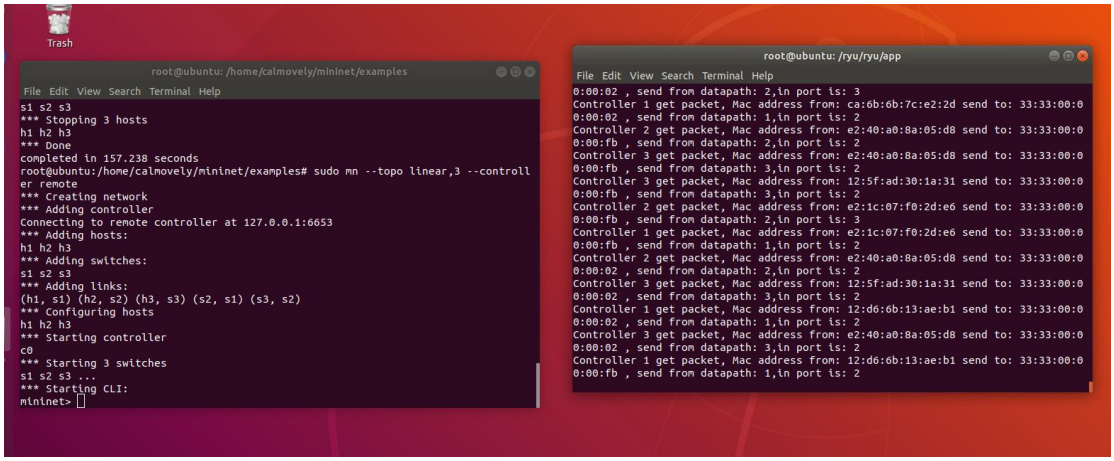
使用 h1 ping h2，并在 h3 处抓包

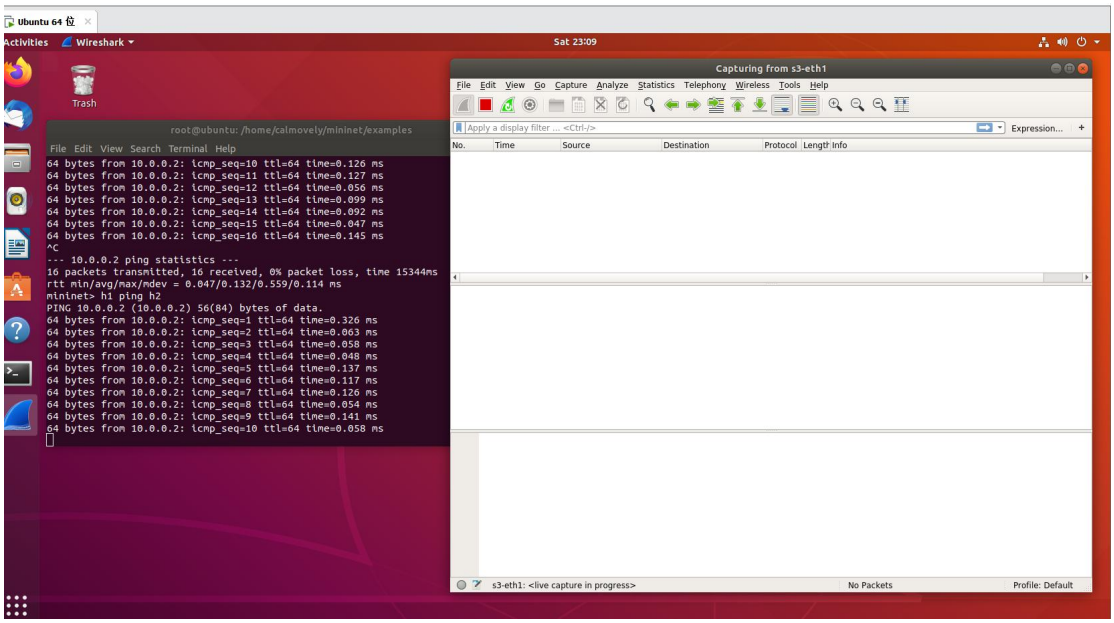


可以看到我们抓到了包。

实验二：

将另一个程序执行，重复之前的操作





可以看到，在 h1 ping h2 之后,h3 处并没有抓到包

实验成功

源代码：

实验一：

```
        FPInstructionActions(ofp.OFPIT_APPLY_ACTIONS, actions)]
                mod = parser.OFPFlowMod(datapath=dp, priority=priority,
match=match,instructions=inst)
                dp.send_msg(mod)
        # add default flow table which sends packets to the controller
            @set_ev_cls(ofp_event.EventOFPSwitchFeatures,
CONFIG_DISPATCHER)
            def switch_features_handler(self, ev):
                msg = ev.msg
                dp = msg.datapath
                ofp = dp.ofproto
                parser = dp.ofproto_parser
                match = parser.OFPMatch()
                actions =
[parser.OFPActionOutput(ofp.OFPP_CONTROLLER,ofp.OFPCML_NO_BUFFER)]
                self.add_flow(dp, 0, match, actions)
        # handle packet_in message
            @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
            def packet_in_handler(self, ev):
                msg = ev.msg
                dp = msg.datapath
                ofp = dp.ofproto
```

```
                parser = dp.ofproto_parser
                actions = [parser.OFPActionOutput(ofp.OFPP_FLOOD)]
                out = parser.OFPPacketOut(datapath=dp,
buffer_id=msg.buffer_id,in_port=msg.match['in_port'], actions=actions,
data=msg.data)
```

实验二：

```
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 enty 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)
```