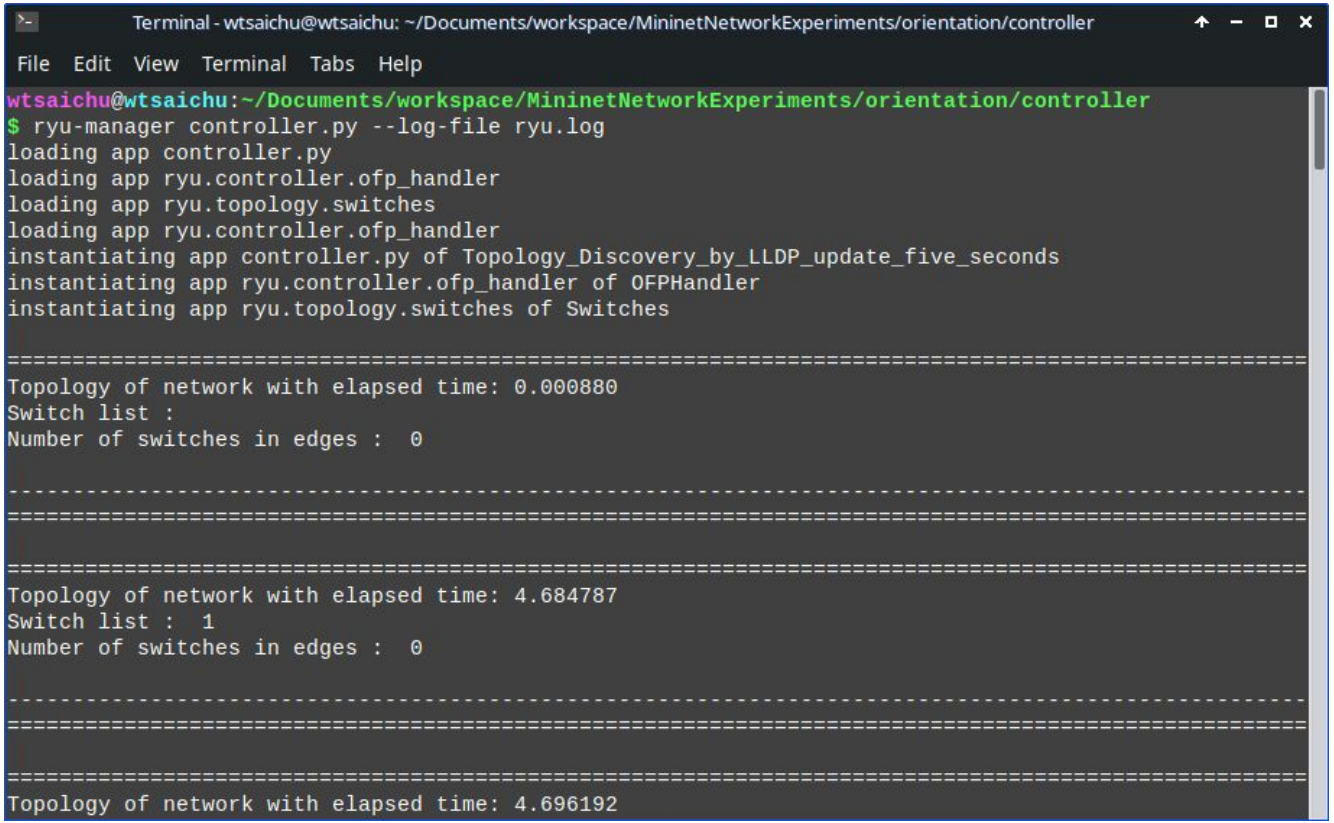
Topology discovery

Consequent

Command

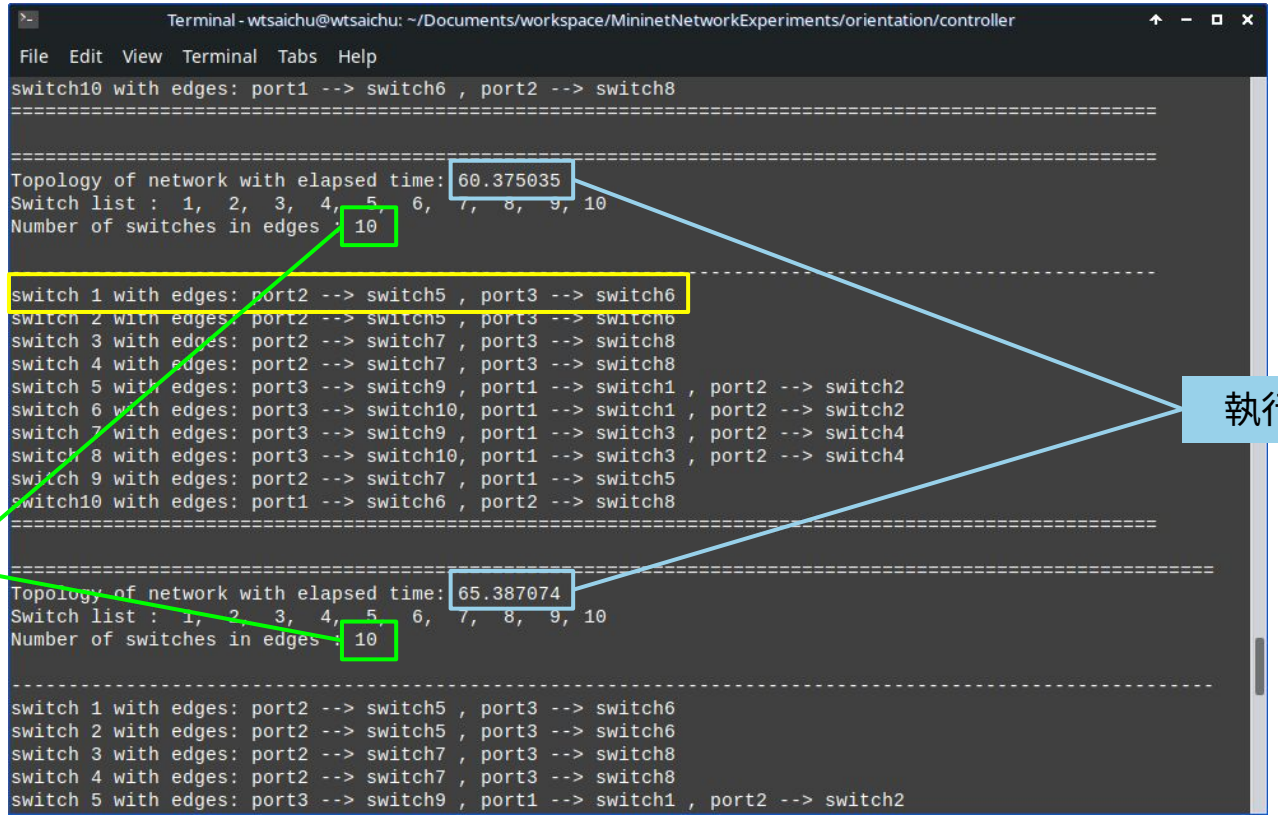

Terminal - wtsaichu@wtsaichu: ~/Documents/workspace/MininetNetworkExperiments/orientation/controller
File Edit View Terminal Tabs Help
wtsaichu@wtsaichu:~/Documents/workspace/MininetNetworkExperiments/orientation/controller
$ ryu-manager controller.py --log-file ryu.log
loading app controller.py
loading app ryu.controller.ofp_handler
loading app ryu.topology.switches
loading app ryu.controller.ofp_handler
instantiating app controller.py of Topology_Discovery_by_LLDP_update_five_seconds
instantiating app ryu.controller.ofp_handler of OFPHandler
instantiating app ryu.topology.switches of Switches
Topology of network with elapsed time: 0.000880
Switch list :
Number of switches in edges : 0
Topology of network with elapsed time: 4.684787
Switch list : 1
Number of switches in edges : 0
Topology of network with elapsed time: 4.696192

Topology

單一交換機的 port 連接到的交換機

已連接的交換機數量

執行時間

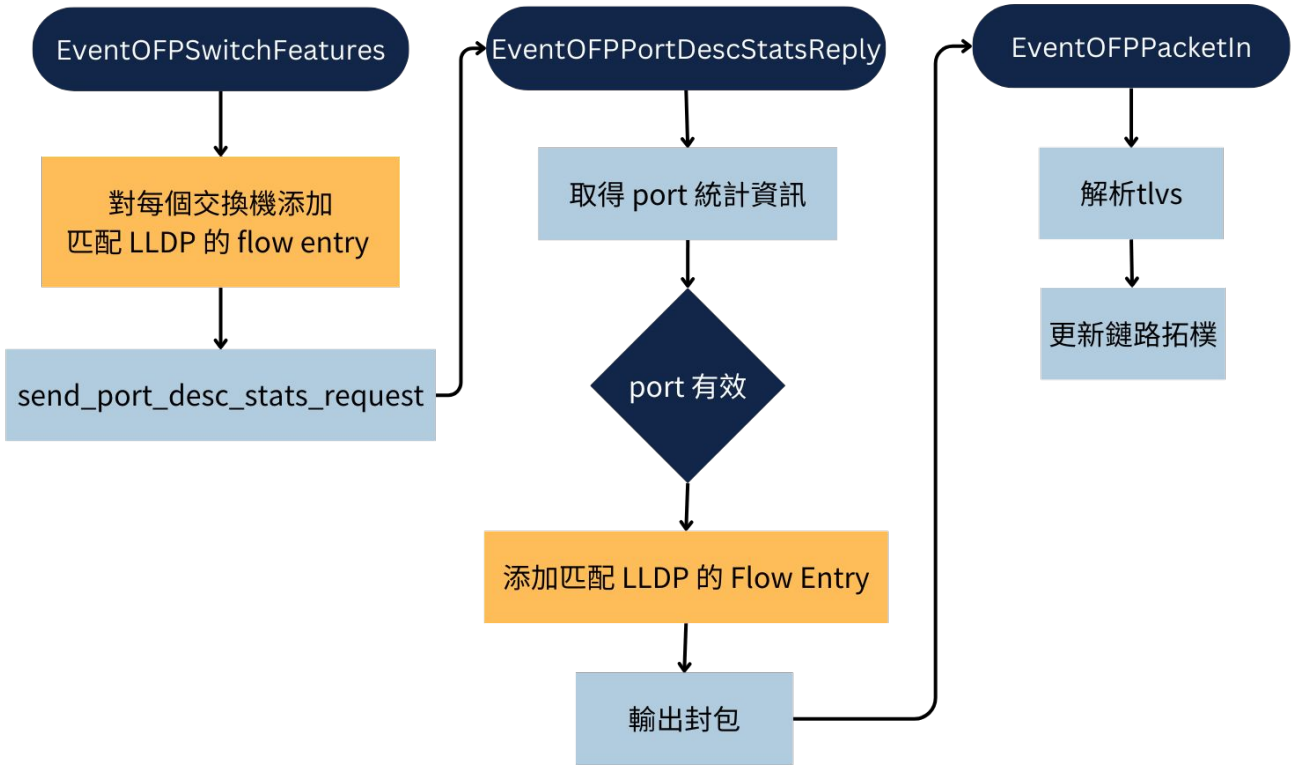


Flow table

```
root@wtsaichu:/home/wtsaichu/Documents/workspace/MininetNetworkExperiments/orientation/topology# ovs-ofctl dump-flows switch1
 cookie=0x0, duration=0.494s, table=0, n_packets=0, n_bytes=0, hard_timeout=4, priority=2,ipv6,in_port="switch1-eth1" actions=output:"switch1-eth3",CONTROLLER:6553
 cookie=0x0, duration=0.494s, table=0, n_packets=0, n_bytes=0, hard_timeout=4, priority=2,ipv6,in_port="switch1-eth2" actions=output:"switch1-eth3",CONTROLLER:6553
 cookie=0x0, duration=0.494s, table=0, n_packets=0, n_bytes=0, hard_timeout=4, priority=2,ipv6,in_port="switch1-eth3" actions=output:"switch1-eth2",CONTROLLER:6553
 cookie=0x0, duration=101.392s, table=0, n_packets=89, n_bytes=5350, priority=0 actions=CONTROLLER:65535
root@wtsaichu:/home/wtsaichu/Documents/workspace/MininetNetworkExperiments/orientation/topology# ovs-ofctl dump-flows switch5
 cookie=0x0, duration=3.506s, table=0, n_packets=0, n_bytes=0, hard_timeout=4, priority=2,ipv6,in_port="switch5-eth1" actions=output:"switch5-eth3",CONTROLLER:6553
 cookie=0x0, duration=3.506s, table=0, n_packets=0, n_bytes=0, hard_timeout=4, priority=2,ipv6,in_port="switch5-eth2" actions=output:"switch5-eth3",CONTROLLER:6553
 cookie=0x0, duration=3.506s, table=0, n_packets=0, n_bytes=0, hard_timeout=4, priority=2,ipv6,in_port="switch5-eth3" actions=output:"switch5-eth2",CONTROLLER:6553
 cookie=0x0, duration=104.349s, table=0, n_packets=5313, n_bytes=800052, priority=0 actions=CONTROLLER:65535
root@wtsaichu:/home/wtsaichu/Documents/workspace/MininetNetworkExperiments/orientation/topology# ovs-ofctl dump-flows switch9
 cookie=0x0, duration=0.709s, table=0, n_packets=0, n_bytes=0, hard_timeout=4, priority=2,ipv6,in_port="switch9-eth1" actions=output:"switch9-eth2",CONTROLLER:6553
 cookie=0x0, duration=0.709s, table=0, n_packets=0, n_bytes=0, hard_timeout=4, priority=2,ipv6,in_port="switch9-eth2" actions=output:"switch9-eth1",CONTROLLER:6553
 cookie=0x0, duration=106.500s, table=0, n_packets=5274, n_bytes=798138, priority=0 actions=CONTROLLER:65535
root@wtsaichu:/home/wtsaichu/Documents/workspace/MininetNetworkExperiments/orientation/topology#
```

Thinking

Flow diagram



EventOFPSwitchFeatures key code

```
1  self.send_port_desc_stats_request(datapath) # 發送一個 send_port_desc_stats_request 給各個switch
2  self.logger.info("Datapath {:2d} send an OFPMP_PORT_STATS Request.".format(datapath.id))    # 顯示添加完成的 log
3  self.show_topology_lldp()   # 顯示拓樸 (LLDP)
```

EventOFPPortDescStatsReply key code

```python
# OFPPortDescStatsRequest 的響應, 統計 port 的資訊
@set_ev_cls(ofp_event.EventOFPPortDescStatsReply, MAIN_DISPATCHER)
def port_desc_stats_reply_handler(self, event):
    # 取得訊息
    datapath = event.msg.datapath    # 數據平面的交換機 (datapath) 結構
    ofproto = datapath.ofproto    # OpenFlow 協議相關訊息
    ofp_parser = datapath.ofproto_parser    # 創建和解析 OpenFlow message

    ports = {}  # 交換機上的 port

    # 遍歷 event 中收到的每個 port 的統計訊息
    for statistic in event.msg.body:
        if statistic.port_no <= ofproto.OFPP_MAX:    # 如果 port_no(port number) 小於或等於 OFPP_MAX (最大的 port number ) -> 表示該 port 有效且不是 reserved port
            ports.update( {statistic.port_no : statistic.hw_addr} )   # 添加有效的 port 訊息  port number : MAC 地址
    self.ports_details.update({ datapath.id : ports} )    # 更新該交換機的 port 統計資訊

    ports_string = ""
    for key in self.ports_details[datapath.id].keys():
        ports_string += " {} : {:2s}, ".format(key, self.ports_details[datapath.id][key])
    self.logger.info("Switch{:2d} with ports infromation : {}".format(datapath.id, ports_string))

    # 遍歷 ports 的每個 port, 並且為該 port 發送 LLDP 封包
    for port_number in ports.keys():
        ingress_port = int(port_number)   # 輸入 port 為 port 的 port number
        match = ofp_parser.OFPMatch(eth_type = 34525 , in_port = ingress_port) # 如果封包匹配 in_port = ingress_port 且 為 LLDP 類型

        for other_port_number in ports.keys():    # 遍歷其他非 ingress_port 的 port
            if(other_port_number != ingress_port):    # 如果是其他 port
                out_port = other_port_number    # 轉發 port 為 other_port
                self.send_lldp_packet(datapath, other_port_number, ports[other_port_number], 1)  # 發送 LLDP 封包

                actions = [ofp_parser.OFPActionOutput(out_port),
                           ofp_parser.OFPActionOutput(ofproto.OFPP_CONTROLLER, ofproto.OFPCML_NO_BUFFER)]    # 進行轉發封包的 action : 轉發到 output_port 以及控制器
                insturctions = [ofp_parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS, actions)] # 立即執行該動作
                flow_add_modification_message = ofp_parser.OFPFlowMod(  # 添加 flow entry message , 屬於Controller-to-switch Messages
                    datapath = datapath,    # 交換機
                    match = match,  # 匹配項目
                    cookie = 0, # Cookie 為 0
                    command = ofproto.OFPFC_ADD,    # 0, /* New flow. */ 標示消息類型為 OFPFC_ADD
                    idle_timeout = 0,   # 不限制匹配過期時間 (永久存在)
                    hard_timeout = 4,   # 不限制硬性過期時間 (永久存在)
                    priority = 2,   # 優先級為 2, 為了覆蓋掉預設的 LLDP 封包轉發動作
                    instructions = insturctions # 執行的動作
                )
                datapath.send_msg(flow_add_modification_message)    # 發送往交換機
                self.logger.info("Switch{:2d} add a flow entry with match field : eth_type = 34525 , in_port = {}".format(datapath.id, ingress_port))
```

發送 LLDP 封包

EventOFPPortDescStatsReply key code

發送 LLDP 封包

```python
# 遍歷 ports 的每個 port, 並且為該 port 發送 LLDP 封包
for port_number in ports.keys():
    ingress_port = int(port_number)   # 輸入 port 為 port 的 port number
    match = ofp_parser.OFPMatch(eth_type = 34525 , in_port = ingress_port) # 如果封包匹配 in_port = ingress_port 且 為 LLDP 類型

    for other_port_number in ports.keys():    # 遍歷其他非 ingress_port 的 port
        if(other_port_number != ingress_port):    # 如果是其他 port
            out_port = other_port_number    # 轉發 port 為 other_port
            self.send_lldp_packet(datapath, other_port_number, ports[other_port_number], 1)  # 發送 LLDP 封包

            actions = [ofp_parser.OFPActionOutput(out_port),
                       ofp_parser.OFPActionOutput(ofproto.OFPP_CONTROLLER, ofproto.OFPCML_NO_BUFFER)]    # 進行轉發封包的 action : 轉發到 output_port 以及控制器
            insturctions = [ofp_parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS, actions)] # 立即執行該動作
            flow_add_modification_message = ofp_parser.OFPFlowMod(  # 添加 flow entry message , 屬於Controller-to-switch Messages
                datapath = datapath,    # 交換機
                match = match,  # 匹配項目
                cookie = 0, # Cookie 為 0
                command = ofproto.OFPFC_ADD,    # 0, /* New flow. */ 標示消息類型為 OFPFC_ADD
                idle_timeout = 0,   # 不限制匹配過期時間 (永久存在)
                hard_timeout = 4,   # 不限制硬性過期時間 (永久存在)
                priority = 2,   # 優先級為 2, 為了覆蓋掉預設的 LLDP 封包轉發動作
                instructions = insturctions # 執行的動作
            )
            datapath.send_msg(flow_add_modification_message)    # 發送往交換機
```

LLDP 轉發規則

EventOFPPacketIn key code

```python
# 響應封包進入控制器的事件
@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def packet_in_handler(self, event):
    message = event.msg # message of event
    datapath = event.msg.datapath    # 數據平面的交換機 (datapath) 結構

    package = packet.Packet(data = message.data)  # 取得封包
    datapath_id = datapath.id   # 來源的交換機
    ingress_port  = message.match['in_port']    # 輸入的 port

    package_ethernet = package.get_protocol(ethernet.ethernet)  # ethernet frame

    # 過濾協議為 LLDP 的封包
    if(package_ethernet.ethertype == ether_types.ETH_TYPE_LLDP):
        package_LLDP = package.get_protocol(lldp.lldp)    # 取得 LLDP 封包
        lldp_datapathid = package_LLDP.tlvs[0].chassis_id.decode()  # 連接到的目標交換機 ID
        lldp_ingress_port = package_LLDP.tlvs[1].port_id.decode()   # 連接到的目標交換機 port

        origin_graph = self.switch_graph[datapath_id]   # 未更新的圖[本交換機]
        origin_graph.update({lldp_datapathid : 1})  # 更新圖[本交換機], 權重為 1
        self.switch_graph.update({datapath_id : origin_graph})  # 更新圖到全圖

        origin_switch_port = self.switch_ports[datapath_id] # 未更新的交換機與 port 的連接關係[本交換機]
        origin_switch_port.update({ingress_port : lldp_datapathid}) # 更新交換機與 port 的連接關係[本交換機], port_number : connected_switch
        self.switch_ports.update({datapath_id : origin_switch_port})    # 更新交換機與 port 的連接關係[整個拓樸]

        # 這個解註解會打印出交換機相連的邊以及 port
        # print("switch{:2d} : {} <---> switch{:2s} : {}".format(datapath_id, ingress_port, lldp_datapathid, lldp_ingress_port))
```

針對 LLDP 包的處理

EventOFPPacketIn key code

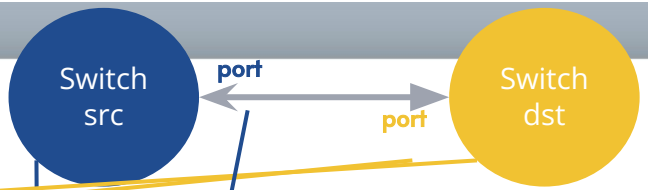
```python
# 過濾協議為 LLDP 的封包
if(package_ethernet.ethertype == ether_types.ETH_TYPE_LLDP):
    package_LLDP = package.get_protocol(lldp.lldp)     # 取得 LLDP 封包
    lldp_datapathid = package_LLDP.tlvs[0].chassis_id.decode()  # 連接到的目標交換機 ID
    lldp_ingress_port = package_LLDP.tlvs[1].port_id.decode()   # 連接到的目標交換機 port

    origin_graph = self.switch_graph[datapath_id]   # 未更新的圖[本交換機]
    origin_graph.update({lldp_datapathid : 1})  # 更新圖[本交換機], 權重為 1
    self.switch_graph.update({datapath_id : origin_graph})  # 更新圖到全圖

    origin_switch_port = self.switch_ports[datapath_id] # 未更新的交換機與 port 的連接關係[本交換機]
    origin_switch_port.update({ingress_port : lldp_datapathid}) # 更新交換機與 port 的連接關係[本交換機], port_number : connected_switch
    self.switch_ports.update({datapath_id : origin_switch_port})    # 更新交換機與 port 的連接關係[整個拓樸]

    # 這個解註解會打印出交換機相連的邊以及 port
    # print("switch{:2d} : {} <---> switch{:2s} : {}".format(datapath_id, ingress_port, lldp_datapathid, lldp_ingress_port))

```



Refresh topology every 5s

```python
# 監控線程執行的函數，每 5 秒執行一次
def every_five_second_monitoring(self):
    while(True):
        try:
            for datapath_id in self.datapaths:  # 更新拓樸變化
                self.send_port_desc_stats_request(self.datapaths[datapath_id])  # 發送 EventOFPPortDescStatsRequest 觸發 EventOFPPortDescStatsReply 事件更新拓樸
            self.show_topology_lldp()    # 顯示拓樸變化
        except KeyError:
            self.logger.info("Topology discovery happens KeyError")
        hub.sleep(5)    # 停止 5 秒
```