# 网络层控制平面实验
# 流表下发

计算机网络 2021秋

罗聪
21210240093@m.fudan.edu.cn

梅昊
22210240098@m.fudan.edu.cn

# 基本介绍

**网络组件：**

- Ryu Controller：OpenFlow接口之上，可以在Ryu编写自己的控制程序，基于Python
- OpenFlow VirtualSwitch：OpenFlow接口之下，行为与硬件交换机一致，性能弱于硬件交换机
- mininet：网络模拟平台，建立虚拟的OpenFlow网络

**Ryu控制器安装：**

- 方式1：pip install ryu
- 方式2：
  git clone https://github.com/faucetsdn/ryu.git
  cd ryu; pip install .

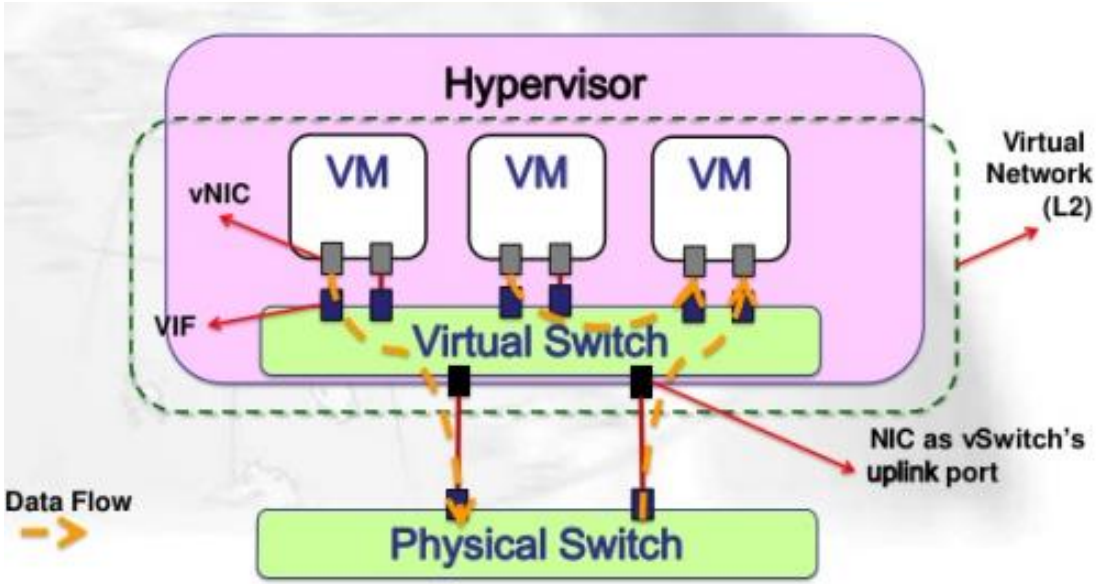
**参考资料**：

- https://ryu.readthedocs.io/en/latest/index.html

# 基本介绍

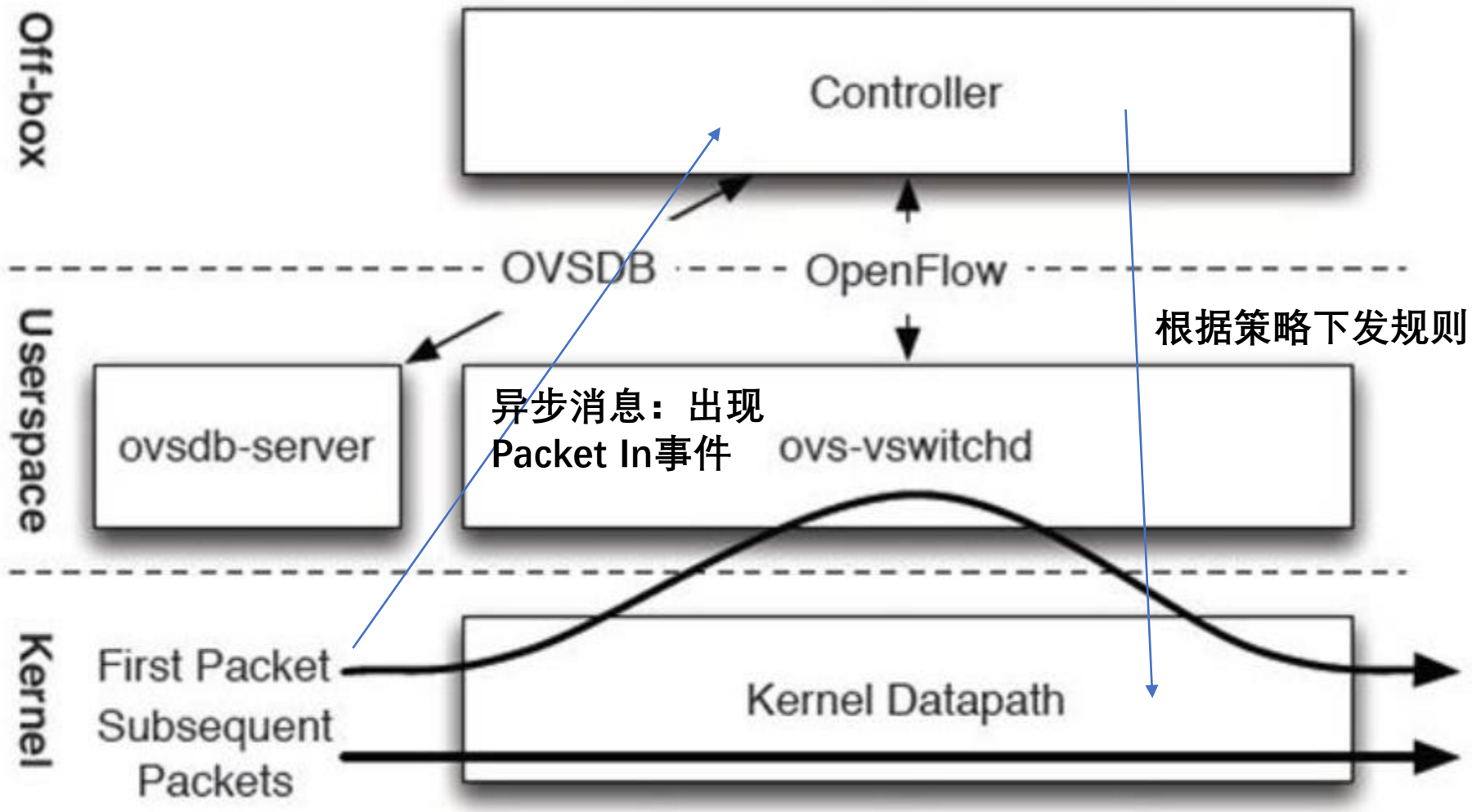
OpenFlow VirtualSwitch(OVS)
作用:

- 传递虚拟机VM之间的流量
- 以及实现VM和外界网络的通信

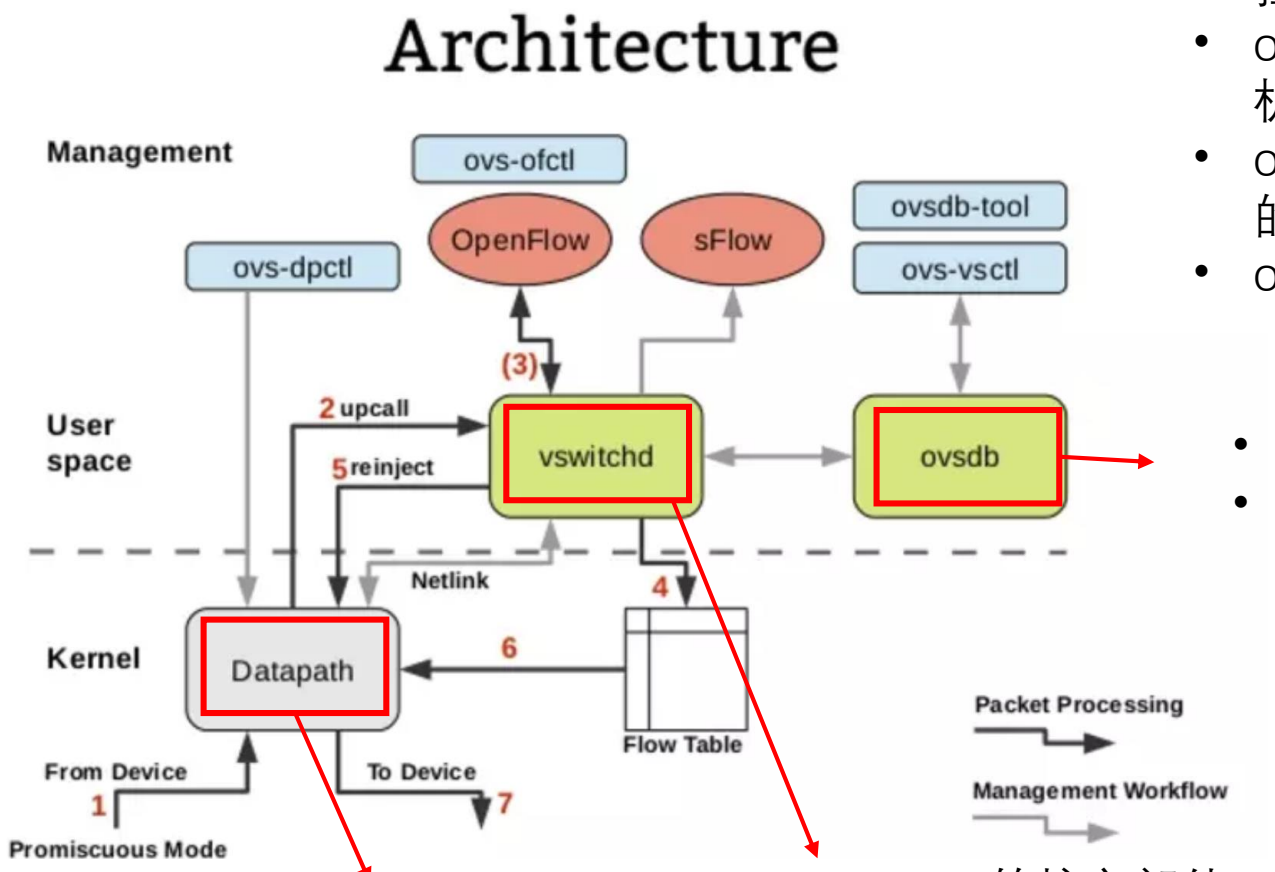



- Bridge: 代表一个以太网交换机（Switch）
- Port: 端口是收发数据包的单元，每个 Port 都隶属于一个 Bridge
- Interface: OVS与外部交换数据包的组件，一个接口就是一个网卡，OVS生成/物理网卡挂在/OS生成
- datapath: 在 OVS 中，datapath负责执行数据交换，每个交换机有一个datapath-id
- Flow table:定义了端口之间数据包的交换规则

# 基本介绍


Off-box
Controller
OVSDB
OpenFlow
Userspace
ovsdb-server
ovs-vswitchd
异步消息：出现
Packet In事件
根据策略下发规则
Kernel
First Packet
Subsequent
Packets
Kernel Datapath

# 基本介绍




命令行工具：

- ovs-dpctl：用来配置交换机内核模块，可以控制转发规则
- ovs-ofctl：用来控制OVS 作为OpenFlow 交换机工作时候的流表内容
- ovs-vsctl：主要是获取或者更改ovs-vswitchd 的配置信息
- ovsdb-tool：数据库管理工具

- 存了整个OVS 的配置信息
- ovs-vswitchd 会根据数据库中的配置信息工作

- 把从接收端口收到的数据包在流表中进行匹配，并执行匹配到的动作

- OVS的核心部件，实现交换功能
- 和上层 controller 通信遵从 OPENFLOW 协议
- 和内核模块通过netlink通信

# OVS命令操作

**命令行工具**： ovs-ofctl

- ovs-ofctl --help 显示功能及用法
- ovs-ofctl show SWITCH 查看设备信息 ovs-ofctl show s1
- ovs-ofctl dump-flows SWITCH 输出交换机内流表 ovs-ofctl dump-flows s1
- ovs-ofctl add-flow SWITCH FLOW 添加流 ovs-ofctl add-flow s1 in_port=1,actions=output:2

**命令行工具**： ovs-dpctl

- ovs-dpctl --help 显示功能及用法
- ovs-dpctl dump-flows 输出OVS的kernel flow cache表
  - 是整个OpenFlow流表的子集，存在OVS‘s flow cache，5s内失效，这使得OVS可以支持很大的流表
- ovs-dpctl show 显示包查找信息（包数/字节数/命中率）-s见各端口细节

```
root@ubuntu:/home/yuri# ovs-dpctl dump-flows
recirc_id(0),in_port(3),eth(src=00:00:00:00:00:03,dst=00:00:00:00:00:01),eth_type(0x0800),ipv4(frag=no), packets:3, bytes:294, used:0.137s, actions:2
recirc_id(0),in_port(2),eth(src=00:00:00:00:00:01,dst=ff:ff:ff:ff:ff:ff),eth_type(0x0806), packets:0, bytes:0, used:never, actions:userspace(pid=2972290750,controller(reason=7,dont_send=0,continuation=0,recirc_id=50,rule_cookie=0,controller_id=0,max_len=65535))
recirc_id(0),in_port(2),eth(src=00:00:00:00:00:01,dst=00:00:00:00:00:03),eth_type(0x0800),ipv4(frag=no), packets:3, bytes:294, used:0.137s, actions:3
root@ubuntu:/home/yuri# ovs-ofctl dump-flows s1
 cookie=0x0, duration=406.472s, table=0, n_packets=5, n_bytes=434, in_port="s1-eth2",dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01 actions=output:"s1-eth1"
 cookie=0x0, duration=406.470s, table=0, n_packets=4, n_bytes=336, in_port="s1-eth1",dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02 actions=output:"s1-eth2"
 cookie=0x0, duration=25.828s, table=0, n_packets=12, n_bytes=1120, in_port="s1-eth3",dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:01 actions=output:"s1-eth1"
 cookie=0x0, duration=25.827s, table=0, n_packets=11, n_bytes=1022, in_port="s1-eth1",dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:03 actions=output:"s1-eth3"
```

# Ryu使用指南

**运行指令**：ryu-manager yourapp.py
/path/ryu/ryu/app 中有一些例子可供参考

**应用举例**:
Step 1：Openflow控制器端：ryu-manager ./ryu/ryu/app/simple_switch.py --verbose
Step 2：Mininet生成网络：sudo mn --topo single,3 --mac --switch ovsk --controller remote

- mininet 创建三个虚拟主机，给定IP地址
- 在内核中创建一个三端口的Openflow软件交换机
- 通过虚拟的以太网线缆连接虚拟主机与软件交换机
- 为每个主机设置MAC地址
- 连接软件交换机与控制器

注：simple_switchl2/l3/l4/l5.py用不同的openflow协议版本实现同样的功能

# Ryu使用案例分析

**应用举例**:

Step 1：Openflow控制器端：ryu-manager ./ryu/ryu/app/simple_switch.py --verbose

Step 2：Mininet生成网络：sudo mn --topo single,3 --mac --switch ovsk --controller remote

Step 3：ovs-ofctl观察流表

```
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=7.78 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.123 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.039 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.043 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.035 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.041 ms
^C
--- 10.0.0.2 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5090ms
rtt min/avg/max/mdev = 0.035/1.344/7.785/2.880 ms
```

```
EVENT ofp_event->SimpleSwitch EventOFPPacketIn
packet in 1 00:00:00:00:00:01 33:33:00:00:00:02 1
EVENT ofp_event->SimpleSwitch EventOFPPacketIn
packet in 1 00:00:00:00:00:02 33:33:00:00:00:02 2
```

① 流表起初为空

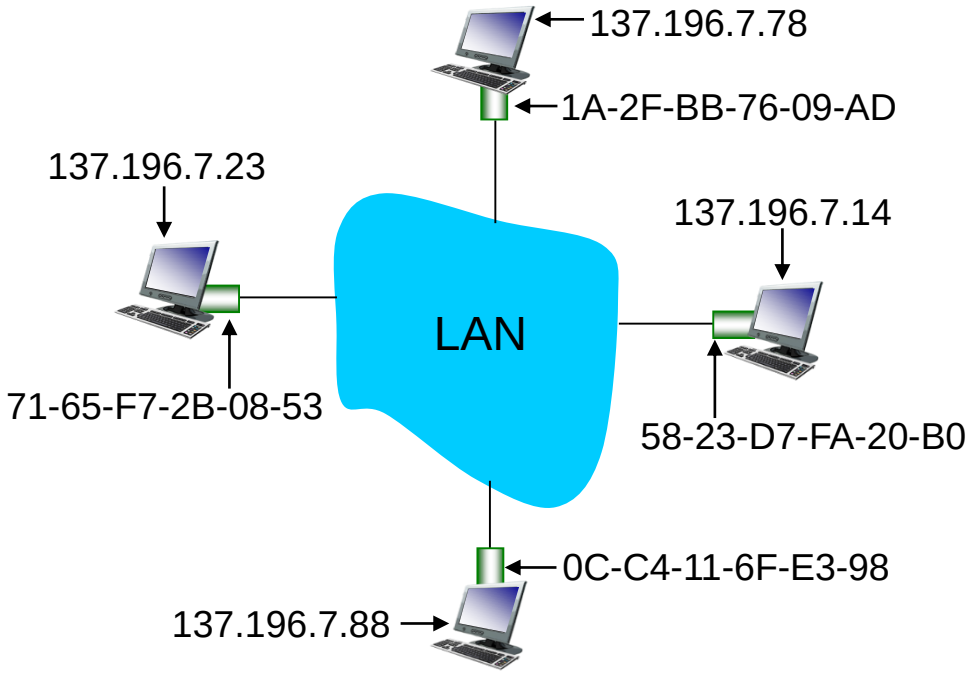
② 触发控制器下发规则

③ 基于流表的数据层转发更快

```
root@ubuntu:/home/yuri# ovs-ofctl dump-flows s1
root@ubuntu:/home/yuri# ovs-ofctl dump-flows s1
 cookie=0x0, duration=173.140s, table=0, n_packets=7, n_bytes=630, in_port="s1-eth2",dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01 actions=output:"s1-eth1"
 cookie=0x0, duration=173.138s, table=0, n_packets=6, n_bytes=532, in_port="s1-eth1",dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02 actions=output:"s1-eth2"
```

# ARP: Address Resolution Protocol

**Question:**已知IP地址，如何确定接口的MAC地址?




- 在LAN上的每个IP节点都有一个ARP表
- ARP表： 包括一些LAN节点IP/MAC地址的映射

< IP address; MAC address; TTL>

- TTL时间是指地址映射失效的时间
- 典型是20min

# ARP协议：在同一个LAN

- A要发送帧给B(B的IP地址已知)，但B的MAC地址不在A的ARP表中
- A广播包含B的IP地址的ARP查询包
  - Dest MAC address =FF-FF-FF-FF-FF-FF
  - LAN上的所有节点都会收到该查询包
- B接收到ARP包，回复A自己的MAC地址
  - 帧发送给A
  - 用A的MAC地址（单播）
- A在自己的ARP表中，缓存IP-to-MAC地址映射关系，直到信息超时
  - 软状态: 靠定期刷新维持的系统状态
  - 定期刷新周期之间维护的状态信息可能和原有系统不一致
- ARP是即插即用的
  - 节点自己创建ARP的表项
  - 无需网络管理员的干预

所谓的learning switches指的是A找B泛洪一次，B再找A不需要了
在A找B的时候同时在B的ARP表中记录A的端口

# Ryu组件介绍与代码分析

以simple_switch.py为主分析

**初始化**

```
from ryu.base import app_manager
from ryu.ofproto import ofproto_v1_0


class SimpleSwitch(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_0.OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super(SimpleSwitch, self).__init__(*args,
**kwargs)
        self.mac_to_port = {}
```

ryu.base.app_manager Ryu应用程序的中央管理

- 加载Ryu应用程序
- 为Ryu应用程序提供上下文
- 在Ryu应用程序之间路由消息
- app_manager.RyuApp是所有Ryu Applications的基类，我们要实现一个控制器应用，必须继承该基类

OpenFlow wire protocol encoder and decoder

- ryu.ofproto.ofproto_v1_0　OpenFlow 1.0定义
- ryu.ofproto.ofproto_v1_0_parser　实现OpenFlow 1.0的编码器和解码器
- 一直到v1_5，类似

# Ryu组件介绍与代码分析

**controller处理收到的OpenFlow消息**

```
from ryu.controller import ofp_event
from ryu.controller.handler import
MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls


@set_ev_cls(ofp_event.EventOFPPacketIn,
MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
```

装饰器
参数1：指定触发该函数的事件
参数2：协商

**ryu.controller.controller：**OpenFlow控制器的主要组件

- **ryu.controller.ofp_event** OpenFlow事件类，描述了从已连接的交换机接收OpenFlow消息的过程
  - ofp_event.EventOFPPacketIn
  - ofp_event.EventOFPSwitchFeatures
- **ryu.controller.handler.set_ev_cls(*ev_cls*, *dispatchers=None*)**
  - Ryu的OpenFlow控制器部分自动解码从交换机接收到的OpenFlow消息，并将这些事件发送到Ryu应用程序，该应用程序使用ryu.controller.handler.set_ev_cls进行下一步处理
- **ryu.controller.handler.dispatcher** 指定了在协商的哪个阶段生成事件传给处理器
  - 'CONFIG_DISPATCHER':协商版本并发送功能请求消息
  - 'MAIN_DISPATCHER':接收交换机功能信息并发送set-config消息

# Ryu组件介绍与代码分析

OpenFlow 1.3及以后的版本需要加这部分代码

**处理未命中表项**

```
@set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
def switch_features_handler(self, ev):    # handshake阶段
        datapath = ev.msg.datapath          # OpenFlow交换机实例
        ofproto = datapath.ofproto          #协商的openflow协议版本
        parser = datapath.ofproto_parser

    # install the table-miss flow entry.
        match = parser.OFPMatch()           # 无参数意味着match任意一个包
        actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,ofproto.OFPCML_NO_BUFFER)]
        self.add_flow(datapath, 0, match, actions) # 向流表下发一条表项
```

**parser.OFPActionOutput(*port, max_len=65509, type_=None, len_=None*)**

用于指定Packet-Out and Flow Mod messages中的包转发

输出一个包到交换机的指定port，max_len是能传到控制器的最大包长

除了指定的交换机端口，还可以是特定值：

- **OFPP_CONTROLLER** Sent to the controller as a Packet-In message
- **OFPP_FLOOD** Flooded to all physical ports of the VLAN except blocked ports and receiving ports
- **OFPP_TABLE** Perform actions in the flow table on the packet

# Ryu组件介绍与代码分析

**在流表中增加表项**

```
def add_flow(self, datapath, priority, match, actions): # 各协议版本传的参数有点不同，自行参考相应版本
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser
    inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,actions)] # instruction是当包满足match时要执行的动作
    mod = parser.OFPFlowMod(datapath=datapath, priority=priority,match=match, instructions=inst) # FlowMod可以让我们向switch内写入定义的flow entry
    datapath.send_msg(mod) # 把flow entry发给交换机
```

self.add_flow(datapath, 0, match, actions) 任意匹配的优先级应该最低

**parser.OFPInstructionActions(*type_, actions=None, len_=None*)**

对动作本身维护，type指定操作类型

- OFPIT_WRITE_ACTIONS  Add an action that is specified in the current set of actions. If same type of action has been set already, it is replaced with the new action.
- OFPIT_APPLY_ACTIONS  Immediately apply the specified action without changing the action set.
- OFPIT_CLEAR_ACTIONS  Delete all actions in the current action set.

**parser.OFPFlowMod(*datapath, cookie=0, cookie_mask=0, table_id=0, command=0, idle_timeout=0, hard_timeout=0, priority=32768, buffer_id=4294967295, out_port=0, out_group=0, flags=0, match=None, instructions=None*)**

修改流表项，控制器将该消息发出，以修改流表

# Ryu组件介绍与代码分析

**controller处理收到的OpenFlow消息**

```
@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
    msg = ev.msg                #switch送来的事件ev,ev.msg 是表示packet_in数据结构的一个对象
    datapath = msg.datapath      #msg.datapath是switch Datapath的一个对象，是哪个switch发来的消息
    ofproto = datapath.ofproto   #协商的版本
    pkt = packet.Packet(msg.data)

    eth = pkt.get_protocol(ethernet.ethernet) # 获取二层包头信息
    if eth.ethertype == ether_types.ETH_TYPE_LLDP: # ignore lldp packet
        return
    dst = eth.dst               # 得到目的MAC地址
    src = eth.src               # 得到源MAC地址
    dpid = datapath.id
    self.mac_to_port.setdefault(dpid, {})
    self.logger.info("packet in %s %s %s %s", dpid, src, dst, msg.in_port)
```

**get_protocol(*protocol*)**: Returns the firstly found protocol that matches to the specified protocol.

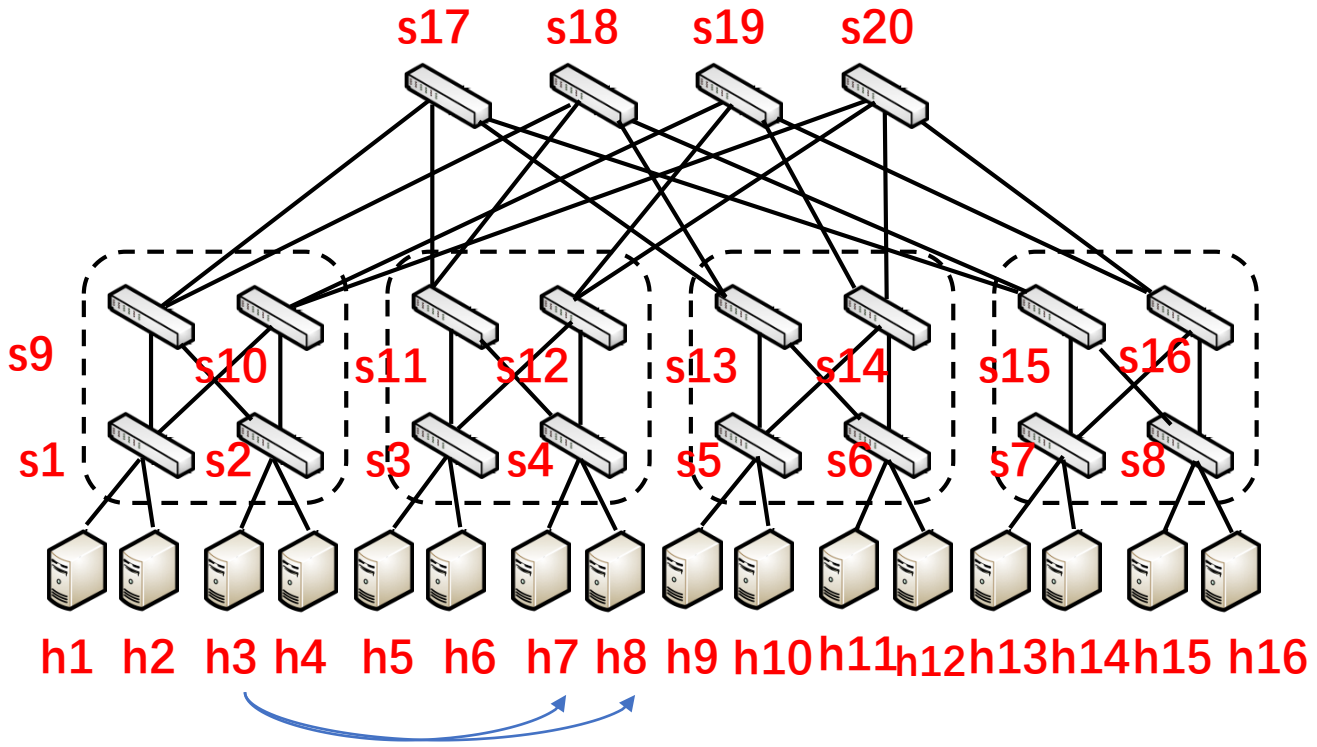
```
_ipv4 = pkt.get_protocol(ipv4.ipv4)
src_ip = _ipv4 .src
dst_ip = _ipv4 .dst
```

```
arp_pkt = pkt.get_protocol(arp.arp)
src_ip = arp_pkt.src_ip
dst_ip = arp_pkt.dst_ip
```

# Ryu组件介绍与代码分析

```
# learn a mac address to avoid FLOOD next time.
self.mac_to_port[dpid][src] = msg.in_port
if dst in self.mac_to_port[dpid]:
    out_port = self.mac_to_port[dpid][dst]
else:
    out_port = ofproto.OFPP_FLOOD
actions = [datapath.ofproto_parser.OFPActionOutput(out_port)]
# install a flow to avoid packet_in next time
if out_port != ofproto.OFPP_FLOOD:
    self.add_flow(datapath, msg.in_port, dst, src, actions)
data = None
if msg.buffer_id == ofproto.OFP_NO_BUFFER:  # 还得把包送往该去的端口
        data = msg.data
out = datapath.ofproto_parser.OFPPacketOut(
    datapath=datapath, buffer_id=msg.buffer_id, in_port=msg.in_port,
    actions=actions, data=data)
datapath.send_msg(out)
```

# 实验任务


s17
s18
s19
s20
s9
s10
s11
s12
s13
s14
s15
s16
s1
s2
s3
s4
s5
s6
s7
s8
h1
h2
h3
h4
h5
h6
h7
h8
h9
h10
h11
h12
h13
h14
h15
h16

# 实验任务

要求下发流表实现以下路由算法：

- Left Path Routing (LPR)：所有的流都从最左边的路径到达目的地，如H3到H8的路径为H3->S2->S9->S17->S11->S4->H8
- Random Selection Routing (RSR)：从可选路径中随机选择一条路径
- Least Loaded Routing (LLR): 该方案下流轮流到来，每个流应该选择最大负载最小的路径。例如H1->S1->S9->S2->H3的链路负载为(2Mbps, 2Mbps, 4Mpbs, 2Mbps)，H1->S1->S10->S2->H3的链路负载为(3Mbps, 3Mbps, 3Mbps, 3Mbps)，从H1到H3的路径应选择后者

# 实验任务

- 给定上述拓扑，并生成流，其中
  - 每台host i (1<=i<=15)发送8条流，每流大小为1Mbps
  - 其中四条流发给(i+4)%16，另外四条流发给(i+5)%16
  - 所有的MAC地址给定
- mininet启动时iperf会自动产生流
- parallel_traffic_generator.py 流同时生成
- sequential_traffic_generator.py 流间隔生成
- 每个脚本持续100s

# 实验任务分析

- 下发的流表格式：

**IP包：**

Match: eth_type, in_port, ipv4_src, ipv4_dst

Action: OFPActionOutput( outport )

**ARP包：**

Match: eth_type, in_port, arp_spa, arp_tpa

Action: OFPActionOutput( outport )

- 处理首包：

参考simple_switch的首包处理，将actions改为 OFPActionOutput(ofproto.OFPP_TABLE)

- 核心工作1：按要求计算路由
- 核心工作2：确定路由路径上每个交换机的in_port值和out_port值，以便写flow entry

注：通用代码及交换机间的连接情况已提供（ryu-manager xxx.py --verbose --observe-links）

# 实验要求

- 提交LPR.py，RSR.py，LLR.py
- 使用parallel_traffic_generator.py 测试LPR, RSR
- 使用sequential_traffic_generator.py 测试LLR.py
- 要求iperf流能正确建立
- LPR/RSR下输出H{x%16}→ H{(x+4)%16}及H{x%16} → H{(x+5)%16}的路径，其中x为学号的后两位
- LLR下打印出前10条流的路径

# 提交方式

- 1. 按照实验要求完成实验，提交实验报告与源码
- 2. 提交方式：上传elearning，命名格式：学号-姓名-实验五(2)
- ddl: 2022/12/22