

# 实验一指导书

实验目的

问题背景

实验任务一：自学习交换机

问题说明

代码框架

结果示例

实验任务二：环路广播

代码框架

结果示例

附加题

扩展资料

## 实验目的

---

通过本次实验，希望大家掌握以下内容：

- 了解**自学习交换机**的工作原理，学习用 Ryu app 编程实现自学习交换机。
- 思考环状拓扑中出现**广播风暴**的原因，复习传统网络怎样处理这一问题——生成树协议。
- **SDN 控制器**掌握网络全局信息，思考 SDN 如何借助这一优势，以多种新策略解决环路广播问题。
- 理解数控平面之间利用 Packet In、Packet Out、Flow Mod 等 OpenFlow Message 进行交互的过程。
- 学习 Flow Table 的使用，理解默认流表项的作用。

## 问题背景

---

你是一名网络工程师，擅长利用 SDN 技术解决网络设计中的难题。而你的许多同事是传统网络技术的支持者，对于 SDN 技术，他们经常发出质疑：“传统网络工作地蛮好的，为什么要把它替换掉？”

一次意外中你穿越到了 1969 年，正值互联网的前身 ARPANET 的初创时期。与历史不同的是，你将以 SDN 提供的全新方案，设计、建立、维护 ARPANET。

接下来的四次实验分别涉及**二层交换机**、**网络状态感知**、**路由策略**、**网络验证**四个主题，请跟随 ARPANET 的发展历程，用你的智慧解决纷至沓来的一个个问题，通过一次次实验逐步丰富、完善你所设计的 ARPANET。

在实验中，注意对比 SDN 和传统网络在解决同一问题时各自采用的策略，客观分析 SDN 的得与失。

最后，祝你最终收获一个令自己满意的作品！

## 实验任务一：自学习交换机

---



Dezember 1969

1969年的 ARPANET 非常简单，仅由四个结点组成。假设每个结点都对应一个交换机，每个交换机都具有一个直连主机，你的任务是实现不同主机之间的正常通信。

预备实验中的简单交换机洪泛数据包，虽然能初步实现主机间的通信，但会带来不必要的带宽消耗；并且会使通信内容泄露给第三者。因此，请你在简单交换机的基础上实现二层自学习交换机，避免数据包的洪泛。

## 问题说明

- SDN 自学习交换机的工作流程可以参考：
  - (1) 控制器为每个交换机维护一个 `mac-port` 映射表。
  - (2) 控制器收到 `packet_in` 消息后，解析其中携带的数据包。
  - (3) 控制器学习 `src_mac - in_port` 映射。
  - (4) 控制器查询 `dst_mac`，如果未学习，则洪泛数据包；如果已学习，则向指定端口转发数据包 (`packet_out`)，并向交换机下发流表项 (`flow_mod`)，指导交换机转发同类型的数据包。
- 网络拓扑为 `topo_1969_1.py`，启动方式：

```
sudo python topo_1969_1.py
```

- 可以不考虑交换机对数据包的缓存 (`no_buffer`)。

## 代码框架

以下给出代码框架，只需补充关键的若干行实现即可：

```
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import MAIN_DISPATCHER, CONFIG_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet
class Switch(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]
```

```

def __init__(self, *args, **kwargs):
    super(Switch, self).__init__(*args, **kwargs)
    # maybe you need a global data structure to save the mapping

def add_flow(self, datapath, priority, match,
actions, idle_timeout=0, hard_timeout=0):
    dp = datapath
    ofp = dp.ofproto
    parser = dp.ofproto_parser
    inst = [parser.OFPInstructionActions(ofp.OFPIT_APPLY_ACTIONS, actions)]
    mod = parser.OFPFlowMod(datapath=dp, priority=priority,
        idle_timeout=idle_timeout,
        hard_timeout=hard_timeout,
        match=match, instructions=inst)

    dp.send_msg(mod)

@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)

@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

    # the identity of switch
    dpid = dp.id
    self.mac_to_port.setdefault(dpid, {})
    # the port that receive the packet
    in_port = msg.match['in_port']
    pkt = packet.Packet(msg.data)
    eth_pkt = pkt.get_protocol(ethernet.ethernet)
    # get the mac
    dst = eth_pkt.dst
    src = eth_pkt.src
    # we can use the logger to print some useful information
    self.logger.info('packet: %s %s %s %s', dpid, src, dst, in_port)

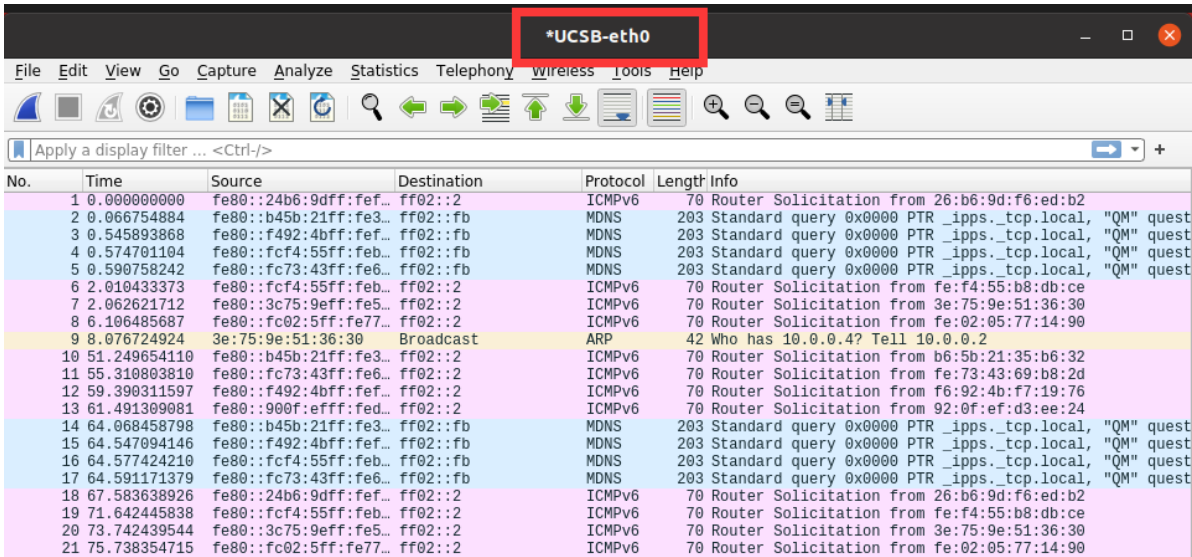
    # you need to code here to avoid the direct flooding
    # having fun
    # :)

```

## 结果示例

UCLA ping UTAH, UCSB 不再收到相关数据包:

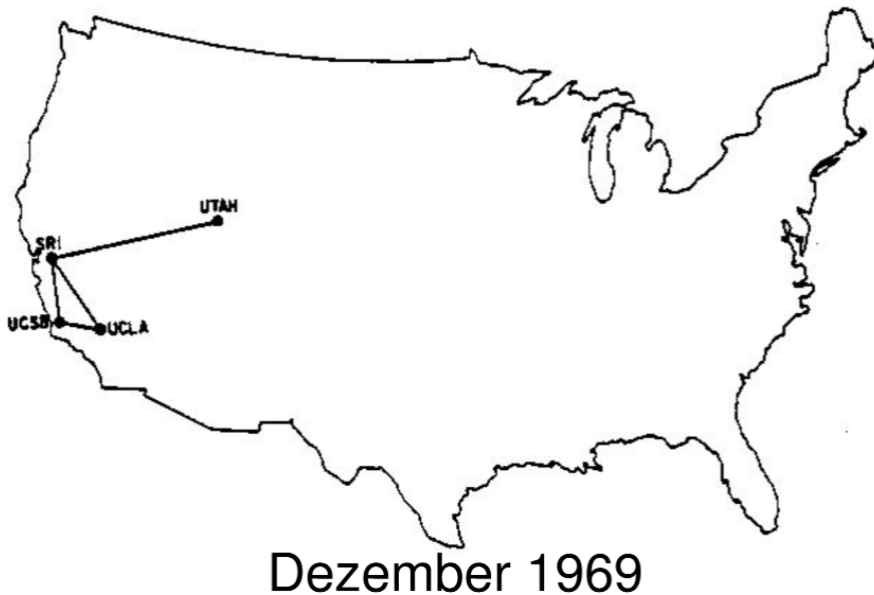
```
mininet> UCLA ping UTAH
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=283 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=128 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=127 ms
64 bytes from 10.0.0.4: icmp_seq=4 ttl=64 time=126 ms
64 bytes from 10.0.0.4: icmp_seq=5 ttl=64 time=127 ms
64 bytes from 10.0.0.4: icmp_seq=6 ttl=64 time=127 ms
64 bytes from 10.0.0.4: icmp_seq=7 ttl=64 time=128 ms
```



The image shows a Wireshark capture window titled '\*UCSB-eth0'. The interface shows a list of network packets. The table below represents the data visible in the packet list pane.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	fe80::24b6:9dff:fe...	ff02::2	ICMPv6	70	Router Solicitation from 26:b6:9d:f6:ed:b2
2	0.066754884	fe80::b45b:21ff:fe3...	ff02::fb	MDNS	203	Standard query 0x0000 PTR _ipps._tcp.local, "QM" quest
3	0.545893868	fe80::f492:4bff:fe...	ff02::fb	MDNS	203	Standard query 0x0000 PTR _ipps._tcp.local, "QM" quest
4	0.574701104	fe80::fcf4:55ff:feb...	ff02::fb	MDNS	203	Standard query 0x0000 PTR _ipps._tcp.local, "QM" quest
5	0.590758242	fe80::fc73:43ff:fe6...	ff02::fb	MDNS	203	Standard query 0x0000 PTR _ipps._tcp.local, "QM" quest
6	2.010433373	fe80::fcf4:55ff:feb...	ff02::2	ICMPv6	70	Router Solicitation from fe:f4:55:b8:db:ce
7	2.062621712	fe80::3c75:9eff:fe5...	ff02::2	ICMPv6	70	Router Solicitation from 3e:75:9e:51:36:30
8	6.106485687	fe80::fc02:5ff:fe77...	ff02::2	ICMPv6	70	Router Solicitation from fe:02:05:77:14:90
9	8.076724924	3e:75:9e:51:36:30	Broadcast	ARP	42	Who has 10.0.0.4? Tell 10.0.0.2
10	51.249654110	fe80::b45b:21ff:fe3...	ff02::2	ICMPv6	70	Router Solicitation from b6:5b:21:35:b6:32
11	55.310803810	fe80::fc73:43ff:fe6...	ff02::2	ICMPv6	70	Router Solicitation from fe:73:43:69:b8:2d
12	59.390311597	fe80::f492:4bff:fe...	ff02::2	ICMPv6	70	Router Solicitation from f6:92:4b:f7:19:76
13	61.491309081	fe80::900f:eff:fed...	ff02::2	ICMPv6	70	Router Solicitation from 92:0f:ef:d3:ee:24
14	64.068458798	fe80::b45b:21ff:fe3...	ff02::fb	MDNS	203	Standard query 0x0000 PTR _ipps._tcp.local, "QM" quest
15	64.547094146	fe80::f492:4bff:fe...	ff02::fb	MDNS	203	Standard query 0x0000 PTR _ipps._tcp.local, "QM" quest
16	64.577424210	fe80::fcf4:55ff:feb...	ff02::fb	MDNS	203	Standard query 0x0000 PTR _ipps._tcp.local, "QM" quest
17	64.591171379	fe80::fc73:43ff:fe6...	ff02::fb	MDNS	203	Standard query 0x0000 PTR _ipps._tcp.local, "QM" quest
18	67.583638926	fe80::24b6:9dff:fe...	ff02::2	ICMPv6	70	Router Solicitation from 26:b6:9d:f6:ed:b2
19	71.642445838	fe80::fcf4:55ff:feb...	ff02::2	ICMPv6	70	Router Solicitation from fe:f4:55:b8:db:ce
20	73.742439544	fe80::3c75:9eff:fe5...	ff02::2	ICMPv6	70	Router Solicitation from 3e:75:9e:51:36:30
21	75.738354715	fe80::fc02:5ff:fe77...	ff02::2	ICMPv6	70	Router Solicitation from fe:02:05:77:14:90

## 实验任务二：环路广播



UCLA 和 UCSB 通信频繁，两者间建立了一条直连链路。在新的拓扑 topo\_1969\_2.py 中运行自学习交换机，UCLA 和 UTAH 之间无法正常通信。分析流表发现，源主机虽然只发了很少的几个数据包，但流表项却匹配了上千次；Wireshark 也截取到了数目异常大的相同报文。

```

mininet> dpctl dump-flows
*** s1 ***
cookie=0x0, duration=5.472s, table=0, n_packets=1950, n_bytes=81900, priority=1,in_port="s1-eth2",dl_dst=d2:bd:58:de:55:0f actions=output:"s1-eth3"
cookie=0x0, duration=5.459s, table=0, n_packets=0, n_bytes=0, priority=1,in_port="s1-eth4",dl_dst=d2:bd:58:de:55:0f actions=output:"s1-eth3"
cookie=0x0, duration=5.870s, table=0, n_packets=3811, n_bytes=458556, priority=0 actions=CONTROLLER:65535
*** s2 ***
cookie=0x0, duration=5.479s, table=0, n_packets=1950, n_bytes=81900, priority=1,in_port="s2-eth1",dl_dst=d2:bd:58:de:55:0f actions=output:"s2-eth2"
cookie=0x0, duration=5.874s, table=0, n_packets=3806, n_bytes=458185, priority=0 actions=CONTROLLER:65535
*** s3 ***
cookie=0x0, duration=5.474s, table=0, n_packets=1947, n_bytes=81774, priority=1,in_port="s3-eth2",dl_dst=d2:bd:58:de:55:0f actions=output:"s3-eth3"
cookie=0x0, duration=5.877s, table=0, n_packets=3813, n_bytes=458640, priority=0 actions=CONTROLLER:65535
*** s4 ***
cookie=0x0, duration=5.483s, table=0, n_packets=0, n_bytes=0, priority=1,in_port="s4-eth2",dl_dst=d2:bd:58:de:55:0f actions=output:"s4-eth2"
cookie=0x0, duration=5.469s, table=0, n_packets=1945, n_bytes=81690, priority=1,in_port="s4-eth3",dl_dst=d2:bd:58:de:55:0f actions=output:"s4-eth3"
cookie=0x0, duration=5.879s, table=0, n_packets=3817, n_bytes=459291, priority=0 actions=CONTROLLER:65535

```

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
59051	2.026122393	ce:f5:0d:e8:73:ef	7e:db:ad:0f:88:e7	ARP	42	10.0.0.4 is at ce:f5:0d:e8:73:ef
59052	2.026124266	ce:f5:0d:e8:73:ef	7e:db:ad:0f:88:e7	ARP	42	10.0.0.4 is at ce:f5:0d:e8:73:ef
59053	2.026126159	ce:f5:0d:e8:73:ef	7e:db:ad:0f:88:e7	ARP	42	10.0.0.4 is at ce:f5:0d:e8:73:ef
59054	2.026594269	ce:f5:0d:e8:73:ef	7e:db:ad:0f:88:e7	ARP	42	10.0.0.4 is at ce:f5:0d:e8:73:ef
59055	2.026598235	ce:f5:0d:e8:73:ef	7e:db:ad:0f:88:e7	ARP	42	10.0.0.4 is at ce:f5:0d:e8:73:ef
59056	2.026600469	ce:f5:0d:e8:73:ef	7e:db:ad:0f:88:e7	ARP	42	10.0.0.4 is at ce:f5:0d:e8:73:ef
59057	2.026602422	ce:f5:0d:e8:73:ef	7e:db:ad:0f:88:e7	ARP	42	10.0.0.4 is at ce:f5:0d:e8:73:ef
59058	2.026604335	ce:f5:0d:e8:73:ef	7e:db:ad:0f:88:e7	ARP	42	10.0.0.4 is at ce:f5:0d:e8:73:ef
59059	2.026606248	ce:f5:0d:e8:73:ef	7e:db:ad:0f:88:e7	ARP	42	10.0.0.4 is at ce:f5:0d:e8:73:ef
59060	2.026608101	ce:f5:0d:e8:73:ef	7e:db:ad:0f:88:e7	ARP	42	10.0.0.4 is at ce:f5:0d:e8:73:ef
59061	2.026610004	ce:f5:0d:e8:73:ef	7e:db:ad:0f:88:e7	ARP	42	10.0.0.4 is at ce:f5:0d:e8:73:ef
59062	2.026611877	ce:f5:0d:e8:73:ef	7e:db:ad:0f:88:e7	ARP	42	10.0.0.4 is at ce:f5:0d:e8:73:ef
59063	2.026613710	ce:f5:0d:e8:73:ef	7e:db:ad:0f:88:e7	ARP	42	10.0.0.4 is at ce:f5:0d:e8:73:ef
59064	2.026615583	ce:f5:0d:e8:73:ef	7e:db:ad:0f:88:e7	ARP	42	10.0.0.4 is at ce:f5:0d:e8:73:ef
59065	2.026617436	ce:f5:0d:e8:73:ef	7e:db:ad:0f:88:e7	ARP	42	10.0.0.4 is at ce:f5:0d:e8:73:ef
59066	2.026619289	ce:f5:0d:e8:73:ef	7e:db:ad:0f:88:e7	ARP	42	10.0.0.4 is at ce:f5:0d:e8:73:ef
59067	2.026621182	ce:f5:0d:e8:73:ef	7e:db:ad:0f:88:e7	ARP	42	10.0.0.4 is at ce:f5:0d:e8:73:ef
59068	2.026623075	ce:f5:0d:e8:73:ef	7e:db:ad:0f:88:e7	ARP	42	10.0.0.4 is at ce:f5:0d:e8:73:ef
59069	2.027093219	ce:f5:0d:e8:73:ef	7e:db:ad:0f:88:e7	ARP	42	10.0.0.4 is at ce:f5:0d:e8:73:ef
59070	2.027096925	ce:f5:0d:e8:73:ef	7e:db:ad:0f:88:e7	ARP	42	10.0.0.4 is at ce:f5:0d:e8:73:ef
59071	2.027098838	ce:f5:0d:e8:73:ef	7e:db:ad:0f:88:e7	ARP	42	10.0.0.4 is at ce:f5:0d:e8:73:ef
59072	2.027100711	ce:f5:0d:e8:73:ef	7e:db:ad:0f:88:e7	ARP	42	10.0.0.4 is at ce:f5:0d:e8:73:ef
59073	2.027102594	ce:f5:0d:e8:73:ef	7e:db:ad:0f:88:e7	ARP	42	10.0.0.4 is at ce:f5:0d:e8:73:ef
59074	2.027104537	ce:f5:0d:e8:73:ef	7e:db:ad:0f:88:e7	ARP	42	10.0.0.4 is at ce:f5:0d:e8:73:ef
59075	2.027106410	ce:f5:0d:e8:73:ef	7e:db:ad:0f:88:e7	ARP	42	10.0.0.4 is at ce:f5:0d:e8:73:ef
59076	2.027108293	ce:f5:0d:e8:73:ef	7e:db:ad:0f:88:e7	ARP	42	10.0.0.4 is at ce:f5:0d:e8:73:ef
59077	2.027110156	ce:f5:0d:e8:73:ef	7e:db:ad:0f:88:e7	ARP	42	10.0.0.4 is at ce:f5:0d:e8:73:ef
59078	2.027112039	ce:f5:0d:e8:73:ef	7e:db:ad:0f:88:e7	ARP	42	10.0.0.4 is at ce:f5:0d:e8:73:ef
59079	2.027113912	ce:f5:0d:e8:73:ef	7e:db:ad:0f:88:e7	ARP	42	10.0.0.4 is at ce:f5:0d:e8:73:ef
59080	2.027115785	ce:f5:0d:e8:73:ef	7e:db:ad:0f:88:e7	ARP	42	10.0.0.4 is at ce:f5:0d:e8:73:ef
59081	2.027117678	ce:f5:0d:e8:73:ef	7e:db:ad:0f:88:e7	ARP	42	10.0.0.4 is at ce:f5:0d:e8:73:ef
59082	2.027119531	ce:f5:0d:e8:73:ef	7e:db:ad:0f:88:e7	ARP	42	10.0.0.4 is at ce:f5:0d:e8:73:ef
59083	2.027121414	ce:f5:0d:e8:73:ef	7e:db:ad:0f:88:e7	ARP	42	10.0.0.4 is at ce:f5:0d:e8:73:ef
59084	2.027597206	ce:f5:0d:e8:73:ef	7e:db:ad:0f:88:e7	ARP	42	10.0.0.4 is at ce:f5:0d:e8:73:ef

这实际上是 ARP 广播数据包在环状拓扑中洪泛导致的，传统网络利用生成树协议解决这一问题。在 SDN 中，不必局限于生成树协议，可以通过多种新的策略解决这一问题。以下给出一种解决思路，请在自学交换机的基础上完善代码，解决问题：

当序号为 `dpid` 的交换机从 `in_port` 第一次收到某个 `src_mac` 主机发出，询问 `dst_ip` 的广播 ARP Request 数据包时，控制器记录一个映射 `(dpid, src_mac, dst_ip) -> in_port`。下一次该交换机收到同一 `(src_mac, dst_ip)` 但 `in_port` 不同的 ARP Request 数据包时直接丢弃，否则洪泛。

## 代码框架

```

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

ETHERNET = ethernet.ethernet.__name__
ETHERNET_MULTICAST = "ff:ff:ff:ff:ff:ff"
ARP = arp.arp.__name__

class Switch_Dict(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

```

```

def __init__(self, *args, **kwargs):
    super(Switch_Dict, self).__init__(*args, **kwargs)
    self.sw = {} #(dpid, src_mac, dst_ip)=>in_port, you may use it in
mission 2
    # maybe you need a global data structure to save the mapping
    # just data structure in mission 1

def add_flow(self, datapath, priority, match, actions, idle_timeout=0,
hard_timeout=0):
    dp = datapath
    ofp = dp.ofproto
    parser = dp.ofproto_parser
    inst = [parser.OFPInstructionActions(ofp.OFPIT_APPLY_ACTIONS, actions)]
    mod = parser.OFPFlowMod(datapath=dp, priority=priority,
                             idle_timeout=idle_timeout,
                             hard_timeout=hard_timeout,
                             match=match, instructions=inst)

    dp.send_msg(mod)

@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)

@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

    # the identity of switch
    dpid = dp.id
    self.mac_to_port.setdefault(dpid, {})
    # the port that receive the packet
    in_port = msg.match['in_port']
    pkt = packet.Packet(msg.data)
    eth_pkt = pkt.get_protocol(ethernet.ethernet)
    if eth_pkt.ethertype == ether_types.ETH_TYPE_LLDP:
        return
    if eth_pkt.ethertype == ether_types.ETH_TYPE_IPV6:
        return
    # get the mac
    dst = eth_pkt.dst
    src = eth_pkt.src
    # get protocols
    header_list = dict((p.protocol_name, p) for p in pkt.protocols if
type(p) != str)
    if dst == ETHERNET_MULTICAST and ARP in header_list:
    # you need to code here to avoid broadcast loop to finish mission 2

```



```
# self-learning
# you need to code here to avoid the direct flooding
# having fun
# :)
# just code in mission 1
```

## 结果示例

解决 ARP 数据包在环状拓扑中的洪泛问题后，UCLA 和 UTAH 之间可以 ping 通，并且流表项的匹配次数明显减少：

```
*** s1 ***
cookie=0x0, duration=1.861s, table=0, n_packets=0, n_bytes=0, hard_timeout=5, priority=1,in_port="s1-eth2",dl_dst=26:c8:1a:08:00:11 actions=output:"s1-eth4"
cookie=0x0, duration=1.853s, table=0, n_packets=0, n_bytes=0, hard_timeout=5, priority=1,in_port="s1-eth4",dl_dst=56:ee:b7:e1:7e:0d actions=output:"s1-eth2"
cookie=0x0, duration=7.865s, table=0, n_packets=10, n_bytes=1015, priority=0 actions=CONTROLLER:65535
*** s2 ***
cookie=0x0, duration=1.871s, table=0, n_packets=0, n_bytes=0, hard_timeout=5, priority=1,in_port="s2-eth1",dl_dst=26:c8:1a:08:00:11 actions=output:"s2-eth2"
cookie=0x0, duration=1.856s, table=0, n_packets=0, n_bytes=0, hard_timeout=5, priority=1,in_port="s2-eth2",dl_dst=56:ee:b7:e1:7e:0d actions=output:"s2-eth1"
cookie=0x0, duration=7.867s, table=0, n_packets=7, n_bytes=567, priority=0 actions=CONTROLLER:65535
*** s3 ***
cookie=0x0, duration=7.870s, table=0, n_packets=4, n_bytes=490, priority=0 actions=CONTROLLER:65535
*** s4 ***
cookie=0x0, duration=1.865s, table=0, n_packets=0, n_bytes=0, hard_timeout=5, priority=1,in_port="s4-eth2",dl_dst=26:c8:1a:08:00:11 actions=output:"s4-eth1"
cookie=0x0, duration=1.864s, table=0, n_packets=0, n_bytes=0, hard_timeout=5, priority=1,in_port="s4-eth1",dl_dst=56:ee:b7:e1:7e:0d actions=output:"s4-eth2"
cookie=0x0, duration=7.873s, table=0, n_packets=8, n_bytes=770, priority=0 actions=CONTROLLER:65535
```

## 附加题

实验任务二只给出了一种参考方案，SDN 中还有多种方案可供选择，请尝试设计实现一种新的策略解决环路广播问题。

附加题主要面向学有余力的同学，请结合自身实际决定是否选做。

## 扩展资料

- SDN 论坛: [sdnlab](#)
- 关于 Mininet 的更多资料: [Mininet Doc](#), [Mininet API](#)
- 关于 Ryu APP 开发的更多资料: [Ryu Book](#)