SDN 实验报告

Lab2 自学习交换机与广播风暴

课程名称：	软件定义网络
姓名：	曾锦程
学院：	计算机学院
专业：	计算机 001 班
学号：	2203613040
指导老师：	张鹏

2023 年 3 月 31 日

西安交通大學实验报告

专业：计算机 001 班
姓名：曾锦程
学号：2203613040
日期：2023 年 3 月 31 日
地点：

课程名称：软件定义网络　指导老师：张鹏　成绩：
实验名称：Lab2 自学习交换机与广播风暴　实验类型：设计实验　同组学生姓名：

一、　实验任务一：自学习交换机

1.　背景介绍

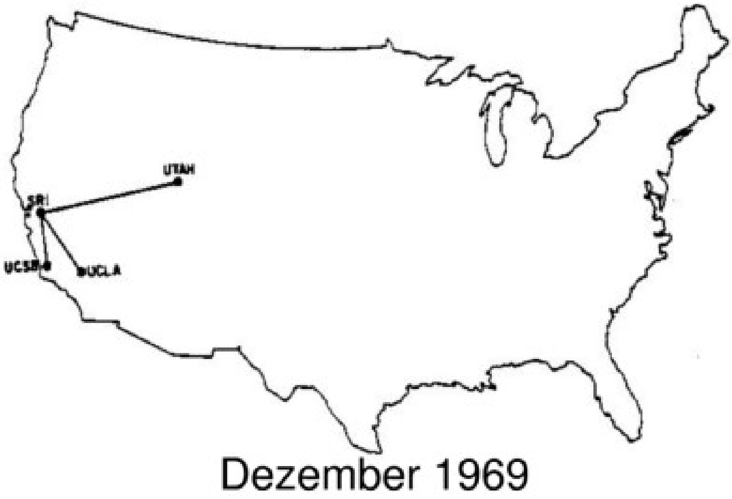


图 1: ARPANET-1

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。前文给出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。

2.　自学习交换机原理

(a) 控制器为每个交换机维护一个 **mac-port** 映射表。

(b) 控制器收到 **packet_in** 消息后，解析其中携带的数据包。

(c) 控制器学习 **src_mac - in_port** 映射。

(d) 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。

3. 关键代码设计思路

(1) 未考虑 Buffer 代码

在构造函数中添加一个全局 mac 与 port 映射表，相当于存储所有传统交换机的 mac 表。

```
self.mac_to_port = {}
```

将收到的包源 mac 地址与接收端口对应，相当于传统交换机自学习 mac 表的过程。

```
self.mac_to_port[dpid][src] = in_port
```

将收到的包与映射表匹配，如果能匹配到，则输出端口为表中端口，如果匹配不到，则洪泛数据包。

```
#if match, then send to outport
if dst in self.mac_to_port[dpid]:
    out_port = self.mac_to_port[dpid][dst]
#if not match, then flood
else:
    out_port = ofp.OFPP_FLOOD
```

控制器对交换机作出指示，如果输出端口不是洪泛，添加流表项，增加匹配次数，完成数据包的转发，如果洪泛，则直接转发。

```
#pass information
actions = [parser.OFPActionOutput(out_port)]
if out_port != ofp.OFPP_FLOOD:
    match = parser.OFPMatch(in_port=in_port, eth_dst=dst, eth_src=src)
    self.add_flow(dp, 1, match, actions)
data = msg.data
out = parser.OFPPacketOut(datapath=dp,buffer_id=msg.buffer_id,in_port=in_port,
     actions=actions, data=data)
dp.send_msg(out)
```

(2) 基于 Buffer 的代码改进

Buffer 是交换机中的概念，意思就是说数据包发入交换机，交换机之后与控制器交互，此时可以不把整个数据包发给控制器，可以通过 Bufferid 进行通信，以下是对三种消息基于 Buffer 改进的介绍：

(a) Packetin 消息：用于标记缓存在交换机中的数据报文 id，如报文被 action 上送到控制器中 maxlen 字段或者 table_miss 消息限制长度，而通过 bufferid 将报文缓存在交换机中，以便被另外两种消息来调用；

(b) Packetout 消息：用于控制器将原先 buffer 在交换机中的报文，通过 Packetout 个形式从交换机的某个物理口送出去；

(c) Flowmod 消息：如果 flowmod 中带有 bufferid，那么说明这个 flowmod 需要做两件事情，第一是正常下发一条 flow，其次是把交换机中先前 buffer 的那个数据报文，Packetout 到 table 来匹配一次下的这条 flow；注意以上两个指令都是通过这个带有 bufferid 的消息执行的，不需要控制器另外下 packet_out 消息，这种设计思路是非常巧妙的。

(d) 代码修改：

首先将 add_flow 函数修改成能够支持 Buffer 机制的函数，即支持上文介绍的 Buffer 机制的 Flowmod。

```
def add_flow(self, datapath, priority, match,
    actions,buffer_id=None,idle_timeout=0,hard_timeout=0):
    dp = datapath
    ofp = dp.ofproto
    parser = dp.ofproto_parser
    inst = [parser.OFPInstructionActions(ofp.OFPIT_APPLY_ACTIONS, actions)]
    if buffer_id:
        mod = parser.OFPFlowMod(datapath=datapath,
            priority=priority,buffer_id=buffer_id,idle_timeout=idle_timeout,
        hard_timeout=hard_timeout,match=match,instructions=inst)
    else:
        mod = parser.OFPFlowMod(datapath=datapath,
            priority=priority,idle_timeout=idle_timeout,
        hard_timeout=hard_timeout,match=match, instructions=inst)
dp.send_msg(mod)
```

在代码中加入以下逻辑：

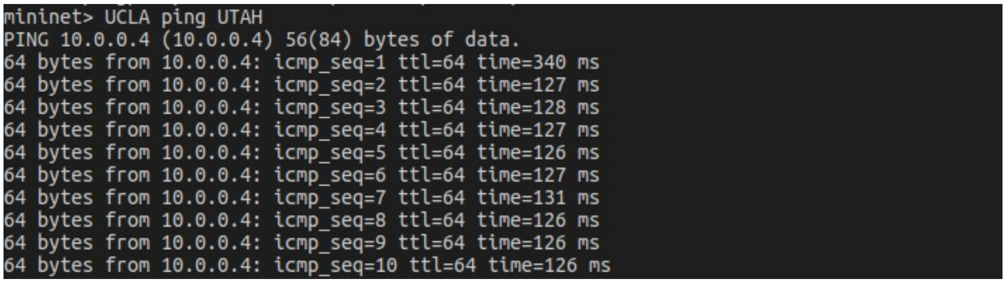
(1) 输出不是洪泛的情况下，如果支持 Buffer 机制的话，直接发出支持 Buffer 机制的 Flowmod 消息即可（此时不需要 Packetout，可以直接 return，因为原报文重新发到 table 匹配)。

(2) 如果是支持 Buffer 机制且输出洪泛的话，将数据 data 置为空，Packetout 只需要传输一个 buffer_id 即可转发消息。

```
    actions = [parser.OFPActionOutput(out_port)]
    if out_port != ofp.OFPP_FLOOD:
        match = parser.OFPMatch(in_port=in_port, eth_dst=dst, eth_src=src)
        if msg.buffer_id != ofp.OFP_NO_BUFFER:
            self.add_flow(dp, 1, match, actions, msg.buffer_id)
            return
        else:
            self.add_flow(dp, 1, match, actions)
    data = None
    if msg.buffer_id == ofp.OFP_NO_BUFFER:
        data = msg.data
    out = parser.OFPPacketOut(datapath=dp,buffer_id=msg.buffer_id,in_port=in_port,
        actions=actions, data=data)
    dp.send_msg(out)
```

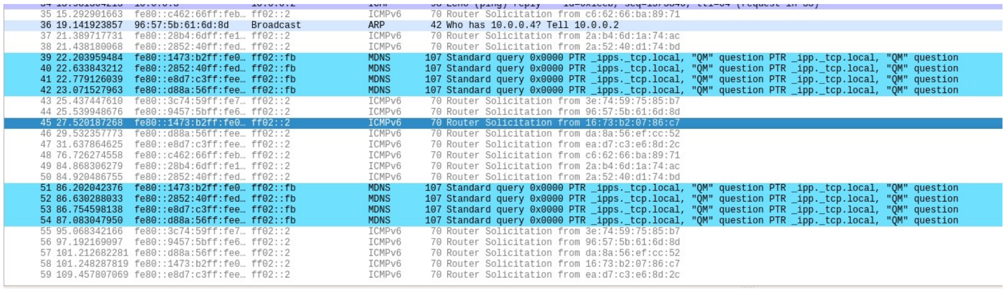
4. 运行结果与分析

可以由图 2，图 3 看出在 UCLA ping UTAH 的过程中，数据包不再发给 UCSB，实现了自学习交换机。



```
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=340 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=127 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=128 ms
64 bytes from 10.0.0.4: icmp_seq=4 ttl=64 time=127 ms
64 bytes from 10.0.0.4: icmp_seq=5 ttl=64 time=126 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=131 ms
64 bytes from 10.0.0.4: icmp_seq=8 ttl=64 time=126 ms
64 bytes from 10.0.0.4: icmp_seq=9 ttl=64 time=126 ms
64 bytes from 10.0.0.4: icmp_seq=10 ttl=64 time=126 ms
```

图 2: UCLA ping UTAH



```
35 15.292901663 fe80::c462:66ff:feb… ff02::2   ICMPv6  70 Router Solicitation from c6:62:66:ba:89:71
36 19.141923857 96:57:5b:61:6d:8d    Broadcast ARP     42 Who has 10.0.0.4? Tell 10.0.0.2
37 21.389717731 fe80::28b4:6dff:fe1… ff02::2   ICMPv6  70 Router Solicitation from 2a:b4:6d:1a:74:ac
38 21.438180068 fe80::2852:40ff:fed… ff02::2   ICMPv6  70 Router Solicitation from 2a:52:40:d1:74:bd
39 22.203959484 fe80::1473:b2ff:fe0… ff02::fb  MDNS   107 Standard query 0x0000 PTR _ipps._tcp.local, "QM" question PTR _ipp._tcp.local, "QM" question
40 22.633843212 fe80::2852:40ff:fed… ff02::fb  MDNS   107 Standard query 0x0000 PTR _ipps._tcp.local, "QM" question PTR _ipp._tcp.local, "QM" question
41 22.779126039 fe80::e8d7:c3ff:fee… ff02::fb  MDNS   107 Standard query 0x0000 PTR _ipps._tcp.local, "QM" question PTR _ipp._tcp.local, "QM" question
42 23.071527963 fe80::d88a:56ff:fee… ff02::fb  MDNS   107 Standard query 0x0000 PTR _ipps._tcp.local, "QM" question PTR _ipp._tcp.local, "QM" question
43 25.437447610 fe80::3c74:59ff:fe7… ff02::2   ICMPv6  70 Router Solicitation from 3e:74:59:75:85:b7
44 25.539948676 fe80::9457:5bff:fe6… ff02::2   ICMPv6  70 Router Solicitation from 96:57:5b:61:6d:8d
45 27.520187268 fe80::1473:b2ff:fe0… ff02::2   ICMPv6  70 Router Solicitation from 16:73:b2:07:86:c7
46 29.532357773 fe80::d88a:56ff:fee… ff02::2   ICMPv6  70 Router Solicitation from da:8a:56:ef:cc:52
47 31.637864625 fe80::e8d7:c3ff:fee… ff02::2   ICMPv6  70 Router Solicitation from ea:d7:c3:e6:8d:2c
48 76.726274558 fe80::c462:66ff:feb… ff02::2   ICMPv6  70 Router Solicitation from c6:62:66:ba:89:71
49 84.868306279 fe80::28b4:6dff:fe1… ff02::2   ICMPv6  70 Router Solicitation from 2a:b4:6d:1a:74:ac
50 84.920486755 fe80::2852:40ff:fed… ff02::2   ICMPv6  70 Router Solicitation from 2a:52:40:d1:74:bd
51 86.202042376 fe80::1473:b2ff:fe0… ff02::fb  MDNS   107 Standard query 0x0000 PTR _ipps._tcp.local, "QM" question PTR _ipp._tcp.local, "QM" question
52 86.630288033 fe80::2852:40ff:fed… ff02::fb  MDNS   107 Standard query 0x0000 PTR _ipps._tcp.local, "QM" question PTR _ipp._tcp.local, "QM" question
53 86.754598138 fe80::e8d7:c3ff:fee… ff02::fb  MDNS   107 Standard query 0x0000 PTR _ipps._tcp.local, "QM" question PTR _ipp._tcp.local, "QM" question
54 87.083047950 fe80::d88a:56ff:fee… ff02::fb  MDNS   107 Standard query 0x0000 PTR _ipps._tcp.local, "QM" question PTR _ipp._tcp.local, "QM" question
55 95.068342166 fe80::3c74:59ff:fe7… ff02::2   ICMPv6  70 Router Solicitation from 3e:74:59:75:85:b7
56 97.192169097 fe80::9457:5bff:fe6… ff02::2   ICMPv6  70 Router Solicitation from 96:57:5b:61:6d:8d
57 101.212682281 fe80::d88a:56ff:fee… ff02::2  ICMPv6  70 Router Solicitation from da:8a:56:ef:cc:52
58 101.248287819 fe80::1473:b2ff:fe0… ff02::2  ICMPv6  70 Router Solicitation from 16:73:b2:07:86:c7
59 109.457807069 fe80::e8d7:c3ff:fee… ff02::2  ICMPv6  70 Router Solicitation from ea:d7:c3:e6:8d:2c
```

图 3: UCSB Capture

二、 实验任务二：广播风暴

1. 背景介绍

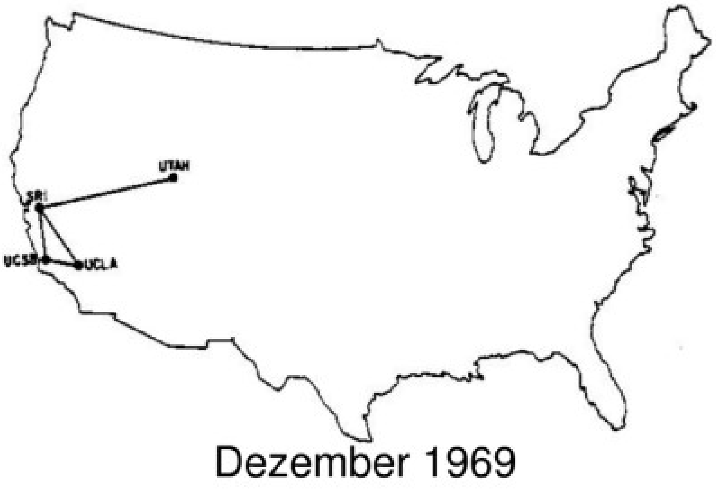


图 4: ARPANET-2

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

2. 广播风暴及其解决思路

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，请在自学习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。

3. 关键代码设计思路

在构造函数初始化一个表，维护一个 (dpid,src_mac,dst_ip)->in_port 的映射。

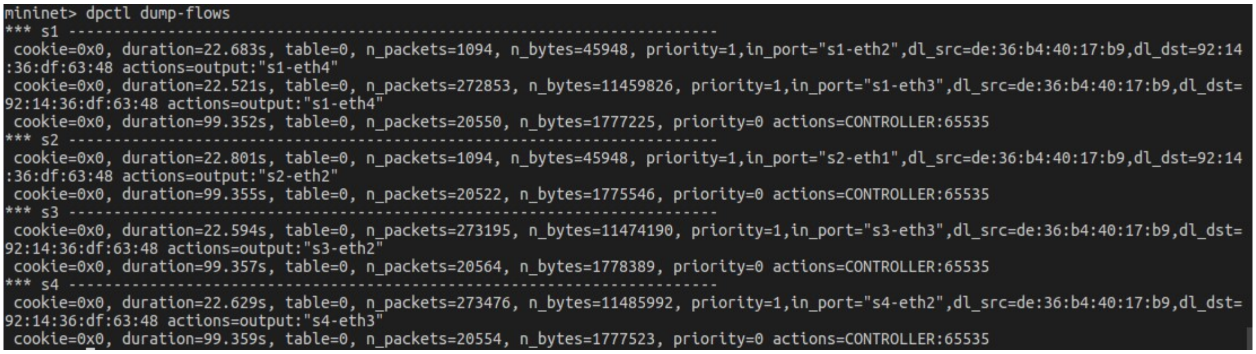
```
    self.sw = {}
```

即对于 arp 包来说，问题在于如何分辨这个包是交换机转发而来还是主机发来的，既可以先存储一个表映射，arp 包一定是主机先发起，所以先通过 sw 存储映射，表示只能从这个端口进入 arp 包，其他端口的 arp 包全部都要丢掉（因为其他端口的 arp 数据包是环路广播的结果），而对于其他交换机也是绑定端口和特定 arp 请求包的发送许可，从而实现广播风暴的避免，与生成树协议些许相似。对于符合映射的包正常转发，而不符合映射则丢弃，若没有对应映射，则进行学习，学习后正常转发。

```
if dst == ETHERNET_MULTICAST and ARP in header_list:
    arp_dst_ip = header_list[ARP].dst_ip
    if (dp.id, src, arp_dst_ip) in self.sw:
        if self.sw[(dp.id, src, arp_dst_ip)] != in_port:
            out = dp.ofproto_parser.OFPPacketOut(datapath=dp,
            buffer_id=dp.ofproto.OFP_NO_BUFFER,in_port=in_port,actions=[], data=None)
            dp.send_msg(out)
            return
        else:
            self.sw[(dp.id, src, arp_dst_ip)] = in_port
```

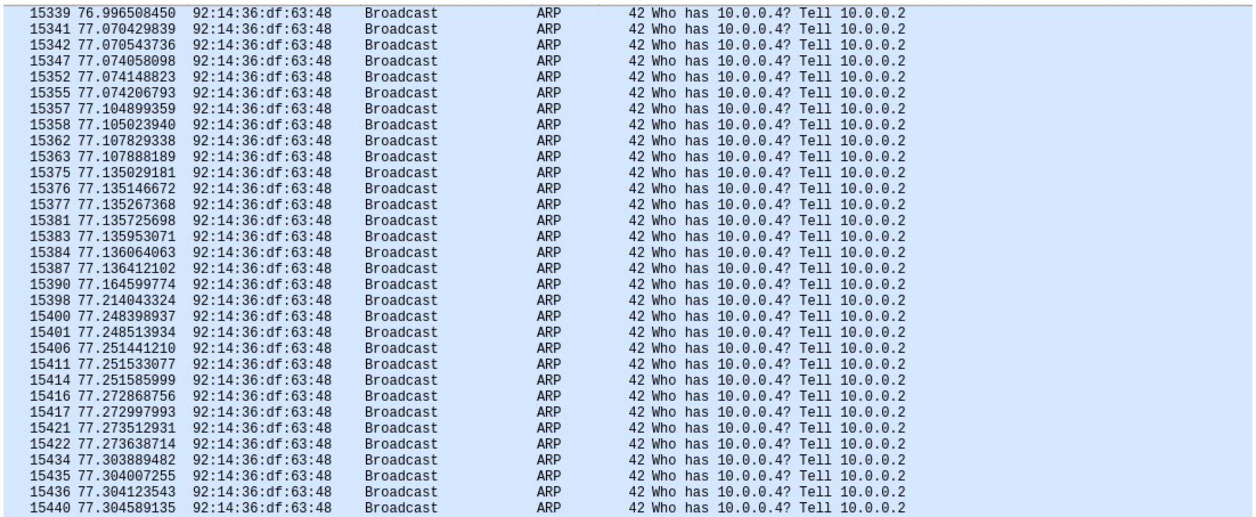
4. 运行结果与分析

使用实验 1 的自学习交换机控制器来应用实验二，发现明显的广播风暴现象，流表匹配次数巨大。



```
mininet> dpctl dump-flows
*** s1 ------------------------------------------------------------------
 cookie=0x0, duration=22.683s, table=0, n_packets=1094, n_bytes=45948, priority=1,in_port="s1-eth2",dl_src=de:36:b4:40:17:b9,dl_dst=92:14
:36:df:63:48 actions=output:"s1-eth4"
 cookie=0x0, duration=22.521s, table=0, n_packets=272853, n_bytes=11459826, priority=1,in_port="s1-eth3",dl_src=de:36:b4:40:17:b9,dl_dst=
92:14:36:df:63:48 actions=output:"s1-eth4"
 cookie=0x0, duration=99.352s, table=0, n_packets=20550, n_bytes=1777225, priority=0 actions=CONTROLLER:65535
*** s2 ------------------------------------------------------------------
 cookie=0x0, duration=22.801s, table=0, n_packets=1094, n_bytes=45948, priority=1,in_port="s2-eth1",dl_src=de:36:b4:40:17:b9,dl_dst=92:14
:36:df:63:48 actions=output:"s2-eth2"
 cookie=0x0, duration=99.355s, table=0, n_packets=20522, n_bytes=1775546, priority=0 actions=CONTROLLER:65535
*** s3 ------------------------------------------------------------------
 cookie=0x0, duration=22.594s, table=0, n_packets=273195, n_bytes=11474190, priority=1,in_port="s3-eth3",dl_src=de:36:b4:40:17:b9,dl_dst=
92:14:36:df:63:48 actions=output:"s3-eth2"
 cookie=0x0, duration=99.357s, table=0, n_packets=20564, n_bytes=1778389, priority=0 actions=CONTROLLER:65535
*** s4 ------------------------------------------------------------------
 cookie=0x0, duration=22.629s, table=0, n_packets=273476, n_bytes=11485992, priority=1,in_port="s4-eth2",dl_src=de:36:b4:40:17:b9,dl_dst=
92:14:36:df:63:48 actions=output:"s4-eth3"
 cookie=0x0, duration=99.359s, table=0, n_packets=20554, n_bytes=1777523, priority=0 actions=CONTROLLER:65535
```

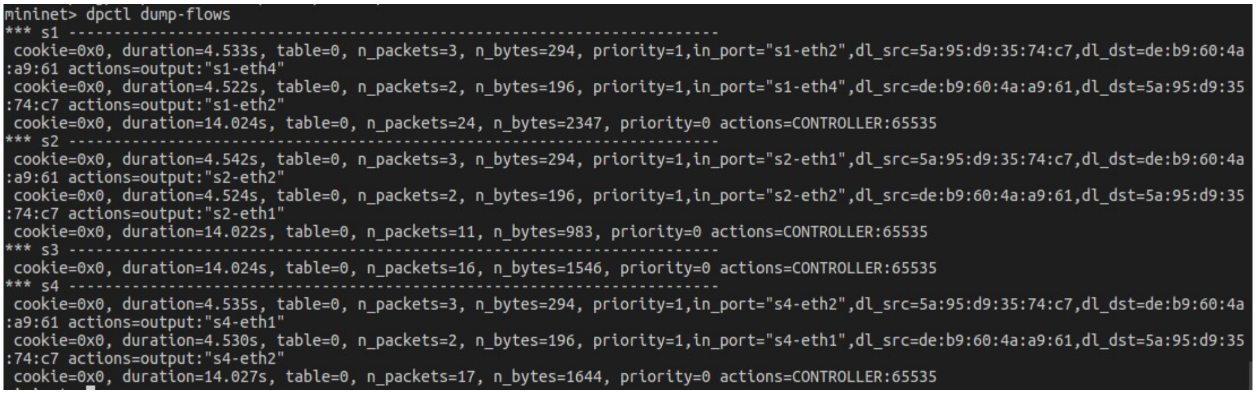
图 5: 解决前流表



```
   15339 76.996508450  92:14:36:df:63:48     Broadcast          ARP        42 Who has 10.0.0.4? Tell 10.0.0.2
   15341 77.070429839  92:14:36:df:63:48     Broadcast          ARP        42 Who has 10.0.0.4? Tell 10.0.0.2
   15342 77.070543736  92:14:36:df:63:48     Broadcast          ARP        42 Who has 10.0.0.4? Tell 10.0.0.2
   15347 77.074058098  92:14:36:df:63:48     Broadcast          ARP        42 Who has 10.0.0.4? Tell 10.0.0.2
   15352 77.074148823  92:14:36:df:63:48     Broadcast          ARP        42 Who has 10.0.0.4? Tell 10.0.0.2
   15355 77.074206793  92:14:36:df:63:48     Broadcast          ARP        42 Who has 10.0.0.4? Tell 10.0.0.2
   15357 77.104899359  92:14:36:df:63:48     Broadcast          ARP        42 Who has 10.0.0.4? Tell 10.0.0.2
   15358 77.105023940  92:14:36:df:63:48     Broadcast          ARP        42 Who has 10.0.0.4? Tell 10.0.0.2
   15362 77.107829338  92:14:36:df:63:48     Broadcast          ARP        42 Who has 10.0.0.4? Tell 10.0.0.2
   15363 77.107888189  92:14:36:df:63:48     Broadcast          ARP        42 Who has 10.0.0.4? Tell 10.0.0.2
   15375 77.135029181  92:14:36:df:63:48     Broadcast          ARP        42 Who has 10.0.0.4? Tell 10.0.0.2
   15376 77.135146672  92:14:36:df:63:48     Broadcast          ARP        42 Who has 10.0.0.4? Tell 10.0.0.2
   15377 77.135267368  92:14:36:df:63:48     Broadcast          ARP        42 Who has 10.0.0.4? Tell 10.0.0.2
   15381 77.135725698  92:14:36:df:63:48     Broadcast          ARP        42 Who has 10.0.0.4? Tell 10.0.0.2
   15383 77.135953071  92:14:36:df:63:48     Broadcast          ARP        42 Who has 10.0.0.4? Tell 10.0.0.2
   15384 77.136064063  92:14:36:df:63:48     Broadcast          ARP        42 Who has 10.0.0.4? Tell 10.0.0.2
   15387 77.136412102  92:14:36:df:63:48     Broadcast          ARP        42 Who has 10.0.0.4? Tell 10.0.0.2
   15390 77.164599774  92:14:36:df:63:48     Broadcast          ARP        42 Who has 10.0.0.4? Tell 10.0.0.2
   15398 77.214043324  92:14:36:df:63:48     Broadcast          ARP        42 Who has 10.0.0.4? Tell 10.0.0.2
   15400 77.248398937  92:14:36:df:63:48     Broadcast          ARP        42 Who has 10.0.0.4? Tell 10.0.0.2
   15401 77.248513934  92:14:36:df:63:48     Broadcast          ARP        42 Who has 10.0.0.4? Tell 10.0.0.2
   15406 77.251441210  92:14:36:df:63:48     Broadcast          ARP        42 Who has 10.0.0.4? Tell 10.0.0.2
   15411 77.251533077  92:14:36:df:63:48     Broadcast          ARP        42 Who has 10.0.0.4? Tell 10.0.0.2
   15414 77.251585999  92:14:36:df:63:48     Broadcast          ARP        42 Who has 10.0.0.4? Tell 10.0.0.2
   15416 77.272868756  92:14:36:df:63:48     Broadcast          ARP        42 Who has 10.0.0.4? Tell 10.0.0.2
   15417 77.272997993  92:14:36:df:63:48     Broadcast          ARP        42 Who has 10.0.0.4? Tell 10.0.0.2
   15421 77.273512931  92:14:36:df:63:48     Broadcast          ARP        42 Who has 10.0.0.4? Tell 10.0.0.2
   15422 77.273638714  92:14:36:df:63:48     Broadcast          ARP        42 Who has 10.0.0.4? Tell 10.0.0.2
   15434 77.303889482  92:14:36:df:63:48     Broadcast          ARP        42 Who has 10.0.0.4? Tell 10.0.0.2
   15435 77.304007255  92:14:36:df:63:48     Broadcast          ARP        42 Who has 10.0.0.4? Tell 10.0.0.2
   15436 77.304123543  92:14:36:df:63:48     Broadcast          ARP        42 Who has 10.0.0.4? Tell 10.0.0.2
   15440 77.304589135  92:14:36:df:63:48     Broadcast          ARP        42 Who has 10.0.0.4? Tell 10.0.0.2
```

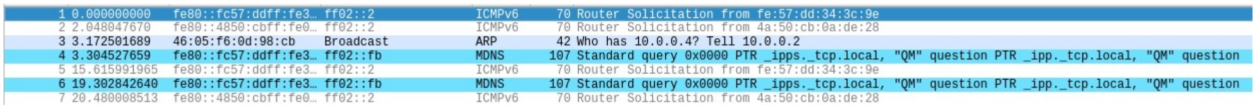
图 6: 解决前 UCSB 抓包

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，UCSB 监听也没有了广播风暴现象。



```
mininet> dpctl dump-flows
*** s1 ------------------------------------------------------------------
 cookie=0x0, duration=4.533s, table=0, n_packets=3, n_bytes=294, priority=1,in_port="s1-eth2",dl_src=5a:95:d9:35:74:c7,dl_dst=de:b9:60:4a
:a9:61 actions=output:"s1-eth4"
 cookie=0x0, duration=4.522s, table=0, n_packets=2, n_bytes=196, priority=1,in_port="s1-eth4",dl_src=de:b9:60:4a:a9:61,dl_dst=5a:95:d9:35
:74:c7 actions=output:"s1-eth2"
 cookie=0x0, duration=14.024s, table=0, n_packets=24, n_bytes=2347, priority=0 actions=CONTROLLER:65535
*** s2 ------------------------------------------------------------------
 cookie=0x0, duration=4.542s, table=0, n_packets=3, n_bytes=294, priority=1,in_port="s2-eth1",dl_src=5a:95:d9:35:74:c7,dl_dst=de:b9:60:4a
:a9:61 actions=output:"s2-eth2"
 cookie=0x0, duration=4.524s, table=0, n_packets=2, n_bytes=196, priority=1,in_port="s2-eth2",dl_src=de:b9:60:4a:a9:61,dl_dst=5a:95:d9:35
:74:c7 actions=output:"s2-eth1"
 cookie=0x0, duration=14.022s, table=0, n_packets=11, n_bytes=983, priority=0 actions=CONTROLLER:65535
*** s3 ------------------------------------------------------------------
 cookie=0x0, duration=14.024s, table=0, n_packets=16, n_bytes=1546, priority=0 actions=CONTROLLER:65535
*** s4 ------------------------------------------------------------------
 cookie=0x0, duration=4.535s, table=0, n_packets=3, n_bytes=294, priority=1,in_port="s4-eth2",dl_src=5a:95:d9:35:74:c7,dl_dst=de:b9:60:4a
:a9:61 actions=output:"s4-eth1"
 cookie=0x0, duration=4.530s, table=0, n_packets=2, n_bytes=196, priority=1,in_port="s4-eth1",dl_src=de:b9:60:4a:a9:61,dl_dst=5a:95:d9:35
:74:c7 actions=output:"s4-eth2"
 cookie=0x0, duration=14.027s, table=0, n_packets=17, n_bytes=1644, priority=0 actions=CONTROLLER:65535
```

图 7: 解决后流表



```
1 0.000000000   fe80::fc57:ddff:fe3… ff02::2       ICMPv6    70 Router Solicitation from fe:57:dd:34:3c:9e
2 2.048047670   fe80::4850:cbff:fe0… ff02::2       ICMPv6    70 Router Solicitation from 4a:50:cb:0a:de:28
3 3.172501689   46:05:f6:0d:98:cb    Broadcast     ARP       42 Who has 10.0.0.4? Tell 10.0.0.2
4 3.304527659   fe80::fc57:ddff:fe3… ff02::fb      MDNS     107 Standard query 0x0000 PTR _ipps._tcp.local, "QM" question PTR _ipp._tcp.local, "QM" question
5 15.615991965  fe80::fc57:ddff:fe3… ff02::2       ICMPv6    70 Router Solicitation from fe:57:dd:34:3c:9e
6 19.302842640  fe80::fc57:ddff:fe3… ff02::fb      MDNS     107 Standard query 0x0000 PTR _ipps._tcp.local, "QM" question PTR _ipp._tcp.local, "QM" question
7 20.480008513  fe80::4850:cbff:fe0… ff02::2       ICMPv6    70 Router Solicitation from 4a:50:cb:0a:de:28
```

图 8: 解决后 UCSB 抓包

5. 方案优缺点分析

优点: 方法简单，实现成本低。
缺点: 实际上还是堵塞了某端口，占用了交换机链路带宽，每次需要丢包。

三、　实验任务三：附加方案

1. 新的解决思路

可以让控制器学习每个交换机与主机连接的 mac 地址和端口。直接让 arp 请求和响应报文不通过交换机链路，直接通过其他交换机交付主机。这样就不会出现 arp 广播风暴, 示意图如图所示。

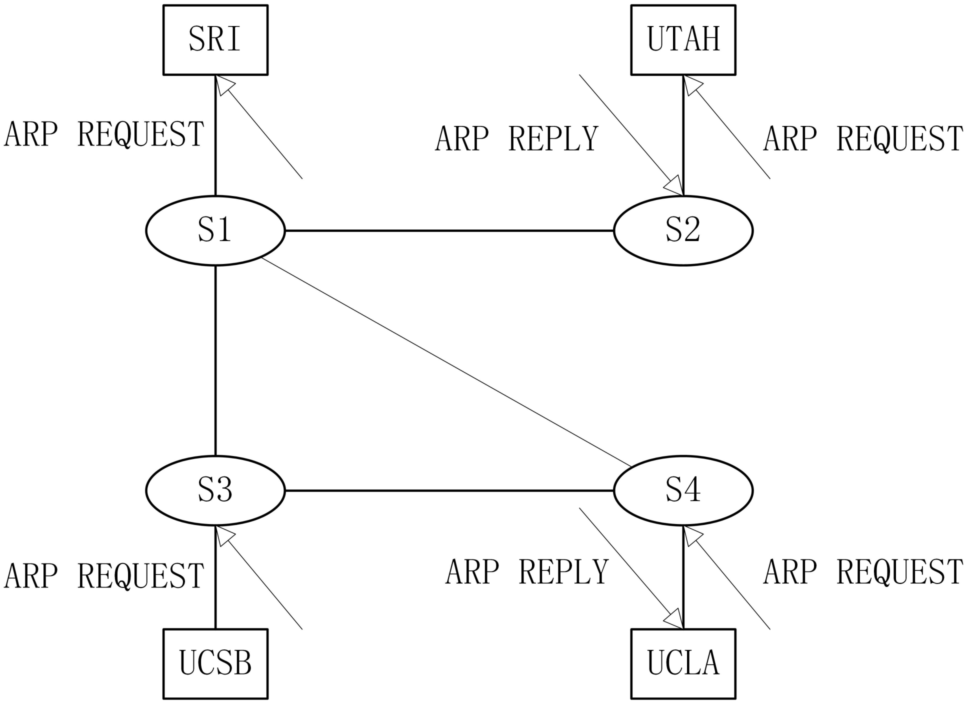


图 9: 解决后流表

2. 关键代码设计思路

在构造函数初始化一个表，维护一个每个交换机主机和交换机端口的映射。

```
self.host_mac_port = {}
```

下面是一个实现上述机制的代码，首先分析是否为 ARP 包:

(1) 如果是 ARP 包则查询 src_mac 地址是否在该交换机的表中，如果在，则说明是正常发送，则其他交换机转发到连接的所有主机，如果不在，则说明不是正常发送，这时候学习 src_mac 及其对应的交换机端口号, 并且不处理 arp 请求。

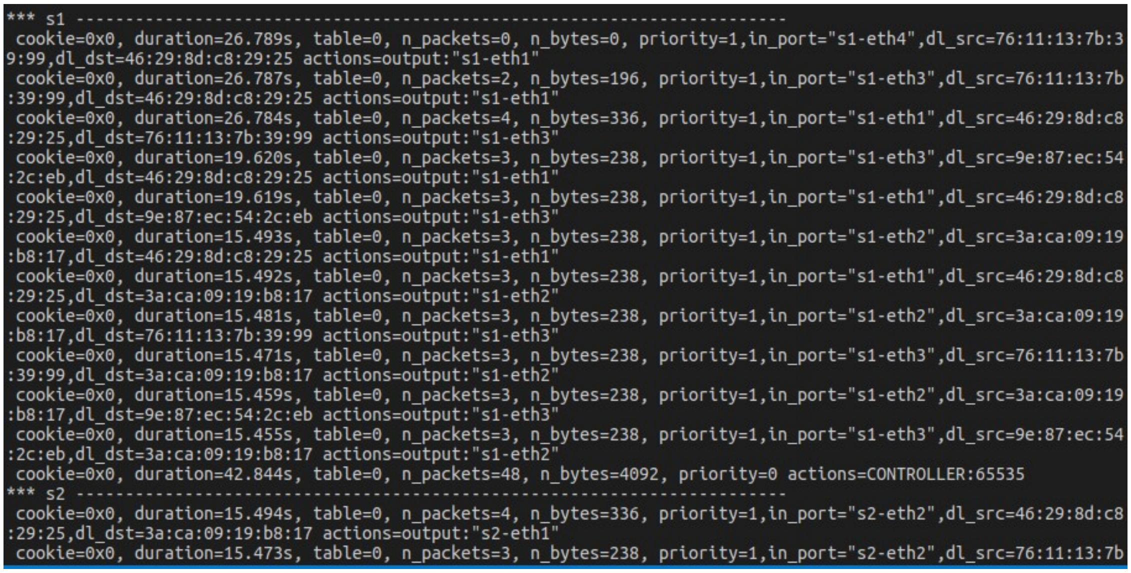
```
if ARP in header_list:
    opcode = header_list[ARP].opcode
    if opcode == arp.ARP_REQUEST:
        self.host_mac_port.setdefault(dp, {})
        if src in self.host_mac_port[dp]:
            for switch in self.host_mac_port.keys():
                ofp = switch.ofproto
                parser = switch.ofproto_parser
                for out_port in self.host_mac_port[switch].values():
                    actions = [parser.OFPActionOutput(out_port)]
                    out = parser.OFPPacketOut(datapath=switch,buffer_id=ofp.OFP_NO_BUFFER,
                    in_port=ofp.OFPP_CONTROLLER,actions=actions, data=msg.data)
                    switch.send_msg(out)
            return
        else:
            self.host_mac_port[dp][src] = in_port
            out = dp.ofproto_parser.OFPPacketOut(datapath=dp,buffer_id=dp.ofproto.OFP_NO_BUFFER
            ,in_port=in_port,actions=[], data=None)
            dp.send_msg(out)
            return
```

(2) 如果是 arp 响应报文，则查询目的 mac 地址是否在表中，如果在某个交换机的表中，则某个交换机直接将 arp 响应报文发送给该主机，实现响应。

```
    elif opcode == arp.ARP_REPLY:
        for switch in self.host_mac_port.keys():
            if dst in self.host_mac_port[switch]:
                ofp = switch.ofproto
                parser = switch.ofproto_parser
                out_port = self.host_mac_port[switch][dst]
                actions = [parser.OFPActionOutput(out_port)]
                out = parser.OFPPacketOut(datapath=switch,buffer_id=ofp.OFP_NO_BUFFER,
                in_port=ofp.OFPP_CONTROLLER,actions=actions, data=msg.data)
                switch.send_msg(out)
        return
```

3. 运行结果与分析

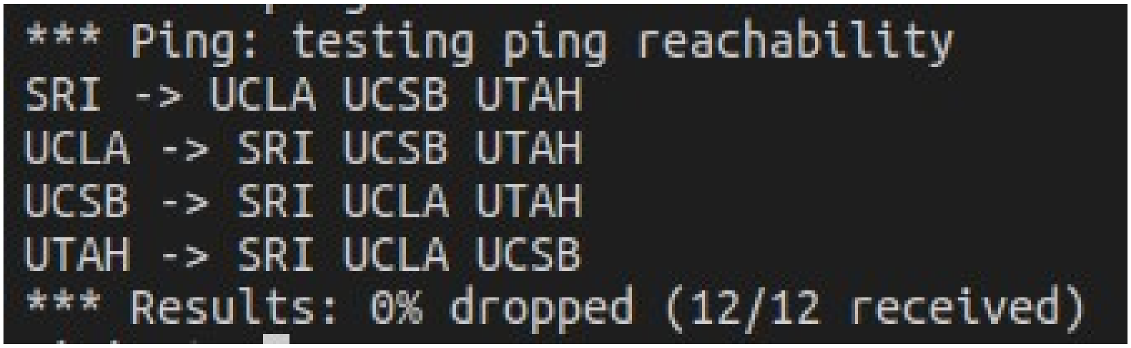
解决 ARP 数据包在环状拓扑中的洪泛问题后，UCLA 和 UTAH 之间可以 ping 通，并且流表项的匹配次数明显减少，UCSB 监听也没有了广播风暴现象。



```
*** s1 ------------------------------------------------------------------------
 cookie=0x0, duration=26.789s, table=0, n_packets=0, n_bytes=0, priority=1,in_port="s1-eth4",dl_src=76:11:13:7b:3
9:99,dl_dst=46:29:8d:c8:29:25 actions=output:"s1-eth1"
 cookie=0x0, duration=26.787s, table=0, n_packets=2, n_bytes=196, priority=1,in_port="s1-eth3",dl_src=76:11:13:7b
:39:99,dl_dst=46:29:8d:c8:29:25 actions=output:"s1-eth1"
 cookie=0x0, duration=26.784s, table=0, n_packets=4, n_bytes=336, priority=1,in_port="s1-eth1",dl_src=46:29:8d:c8
:29:25,dl_dst=76:11:13:7b:39:99 actions=output:"s1-eth3"
 cookie=0x0, duration=19.620s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s1-eth3",dl_src=9e:87:ec:54
:2c:eb,dl_dst=46:29:8d:c8:29:25 actions=output:"s1-eth1"
 cookie=0x0, duration=19.619s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s1-eth1",dl_src=46:29:8d:c8
:29:25,dl_dst=9e:87:ec:54:2c:eb actions=output:"s1-eth3"
 cookie=0x0, duration=15.493s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s1-eth2",dl_src=3a:ca:09:19
:b8:17,dl_dst=46:29:8d:c8:29:25 actions=output:"s1-eth1"
 cookie=0x0, duration=15.492s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s1-eth1",dl_src=46:29:8d:c8
:29:25,dl_dst=3a:ca:09:19:b8:17 actions=output:"s1-eth2"
 cookie=0x0, duration=15.481s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s1-eth2",dl_src=3a:ca:09:19
:b8:17,dl_dst=76:11:13:7b:39:99 actions=output:"s1-eth3"
 cookie=0x0, duration=15.471s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s1-eth3",dl_src=76:11:13:7b
:39:99,dl_dst=3a:ca:09:19:b8:17 actions=output:"s1-eth2"
 cookie=0x0, duration=15.459s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s1-eth2",dl_src=3a:ca:09:19
:b8:17,dl_dst=9e:87:ec:54:2c:eb actions=output:"s1-eth3"
 cookie=0x0, duration=15.455s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s1-eth3",dl_src=9e:87:ec:54
:2c:eb,dl_dst=3a:ca:09:19:b8:17 actions=output:"s1-eth2"
 cookie=0x0, duration=42.844s, table=0, n_packets=48, n_bytes=4092, priority=0 actions=CONTROLLER:65535
*** s2 ------------------------------------------------------------------------
 cookie=0x0, duration=15.494s, table=0, n_packets=4, n_bytes=336, priority=1,in_port="s2-eth2",dl_src=46:29:8d:c8
:29:25,dl_dst=3a:ca:09:19:b8:17 actions=output:"s2-eth1"
 cookie=0x0, duration=15.473s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s2-eth2",dl_src=76:11:13:7b
```

图 10: 流表匹配次数减少

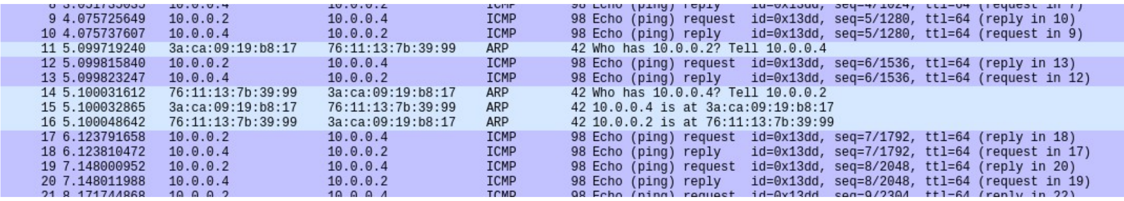
所有节点能够成功 ping 通。



```
*** Ping: testing ping reachability
SRI  -> UCLA UCSB UTAH
UCLA  -> SRI UCSB UTAH
UCSB  -> SRI UCLA UTAH
UTAH  -> SRI UCLA UCSB
*** Results: 0% dropped (12/12 received)
```

图 11: 能够成功 ping 通

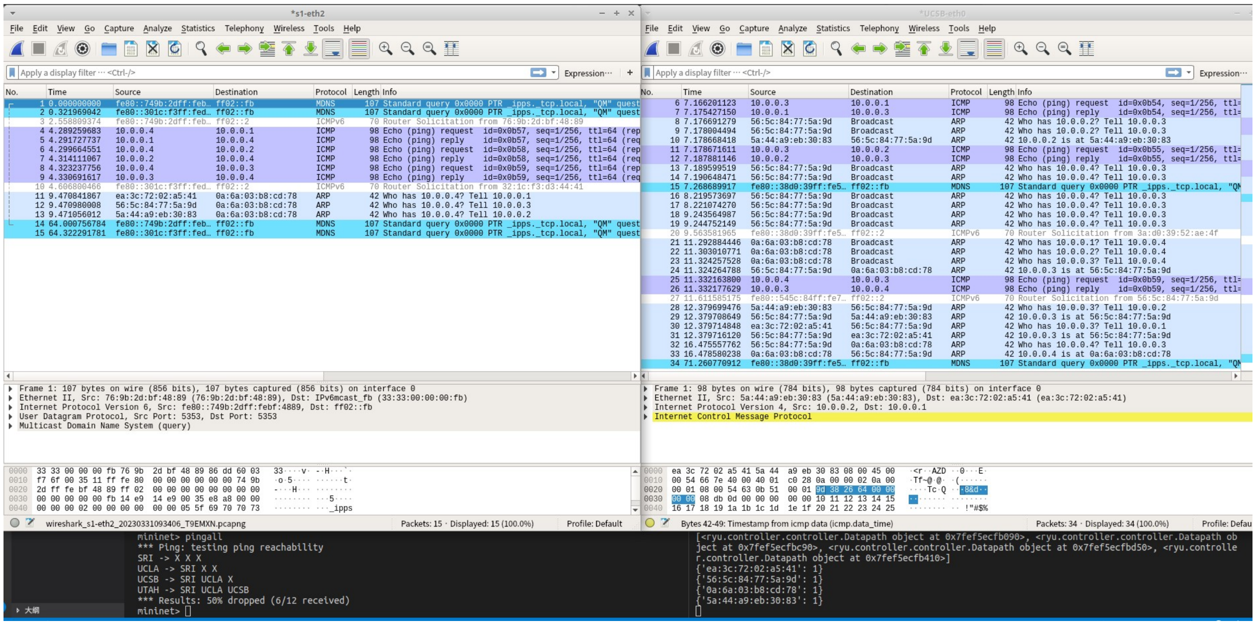
在 UTAH 中抓包，发现能够正常抓包。



```
  8 3.051735035   10.0.0.4            10.0.0.2            ICMP       98 Echo (ping) reply    id=0x13dd, seq=4/1024, ttl=64 (request in 7)
  9 4.075725649   10.0.0.2            10.0.0.4            ICMP       98 Echo (ping) request  id=0x13dd, seq=5/1280, ttl=64 (reply in 10)
 10 4.075737607   10.0.0.4            10.0.0.2            ICMP       98 Echo (ping) reply    id=0x13dd, seq=5/1280, ttl=64 (request in 9)
 11 5.099719240   3a:ca:09:19:b8:17   76:11:13:7b:39:99   ARP        42 Who has 10.0.0.2? Tell 10.0.0.4
 12 5.099815840   10.0.0.2            10.0.0.4            ICMP       98 Echo (ping) request  id=0x13dd, seq=6/1536, ttl=64 (reply in 13)
 13 5.099823247   10.0.0.4            10.0.0.2            ICMP       98 Echo (ping) reply    id=0x13dd, seq=6/1536, ttl=64 (request in 12)
 14 5.100031612   76:11:13:7b:39:99   3a:ca:09:19:b8:17   ARP        42 Who has 10.0.0.4? Tell 10.0.0.2
 15 5.100032865   3a:ca:09:19:b8:17   76:11:13:7b:39:99   ARP        42 10.0.0.4 is at 3a:ca:09:19:b8:17
 16 5.100048642   76:11:13:7b:39:99   3a:ca:09:19:b8:17   ARP        42 10.0.0.2 is at 76:11:13:7b:39:99
 17 6.123791658   10.0.0.2            10.0.0.4            ICMP       98 Echo (ping) request  id=0x13dd, seq=7/1792, ttl=64 (reply in 18)
 18 6.123810472   10.0.0.4            10.0.0.2            ICMP       98 Echo (ping) reply    id=0x13dd, seq=7/1792, ttl=64 (request in 17)
 19 7.148000952   10.0.0.2            10.0.0.4            ICMP       98 Echo (ping) request  id=0x13dd, seq=8/2048, ttl=64 (reply in 20)
 20 7.148011988   10.0.0.4            10.0.0.2            ICMP       98 Echo (ping) reply    id=0x13dd, seq=8/2048, ttl=64 (request in 19)
 21 8.171744868   10.0.0.2            10.0.0.4            ICMP       98 Echo (ping) request  id=0x13dd, seq=9/2304, ttl=64 (reply in 22)
```

图 12: UTAH 抓包

可以在下图抓包发现，链路上并没有广播 arp 包，而是直接通过其他交换机广播发给主机，很好地实现了结果。


*s1-eth2
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ··· <Ctrl-/>
Expression···
*UCSB-eth0
大纲

```
No.  Time          Source                   Destination          Protocol Length Info
  1  0.000000000   fe80::749b:2dff:feb…     ff02::fb             MDNS     107 Standard query 0x0000 PTR _ipps._tcp.local, "QM" quest
  2  0.321969042   fe80::301c:f3ff:fed…     ff02::fb             MDNS     107 Standard query 0x0000 PTR _ipps._tcp.local, "QM" quest
  3  2.558809374   fe80::749b:2dff:feb…     ff02::2              ICMPv6    70 Router Solicitation from 76:9b:2d:bf:48:89
  4  4.289259683   10.0.0.4                 10.0.0.1             ICMP      98 Echo (ping) request  id=0x0b57, seq=1/256, ttl=64 (rep
  5  4.291727737   10.0.0.1                 10.0.0.4             ICMP      98 Echo (ping) reply    id=0x0b57, seq=1/256, ttl=64 (req
  6  4.299664551   10.0.0.4                 10.0.0.2             ICMP      98 Echo (ping) request  id=0x0b58, seq=1/256, ttl=64 (rep
  7  4.314111067   10.0.0.2                 10.0.0.4             ICMP      98 Echo (ping) reply    id=0x0b58, seq=1/256, ttl=64 (req
  8  4.323237756   10.0.0.4                 10.0.0.3             ICMP      98 Echo (ping) request  id=0x0b59, seq=1/256, ttl=64 (rep
  9  4.330691617   10.0.0.3                 10.0.0.4             ICMP      98 Echo (ping) reply    id=0x0b59, seq=1/256, ttl=64 (req
 10  4.606800466   fe80::301c:f3ff:fed…     ff02::2              ICMPv6    70 Router Solicitation from 32:1c:f3:d3:44:41
 11  9.470841867   ea:3c:72:02:a5:41        0a:6a:03:b8:cd:78    ARP       42 Who has 10.0.0.4? Tell 10.0.0.1
 12  9.470980008   56:5c:84:77:5a:9d        0a:6a:03:b8:cd:78    ARP       42 Who has 10.0.0.4? Tell 10.0.0.3
 13  9.471056012   5a:44:a9:eb:30:83        0a:6a:03:b8:cd:78    ARP       42 Who has 10.0.0.4? Tell 10.0.0.2
 14 64.000756784   fe80::749b:2dff:feb…     ff02::fb             MDNS     107 Standard query 0x0000 PTR _ipps._tcp.local, "QM" quest
 15 64.322291781   fe80::301c:f3ff:fed…     ff02::fb             MDNS     107 Standard query 0x0000 PTR _ipps._tcp.local, "QM" quest

▸ Frame 1: 107 bytes on wire (856 bits), 107 bytes captured (856 bits) on interface 0
▸ Ethernet II, Src: 76:9b:2d:bf:48:89 (76:9b:2d:bf:48:89), Dst: IPv6mcast_fb (33:33:00:00:00:fb)
▸ Internet Protocol Version 6, Src: fe80::749b:2dff:febf:4889, Dst: ff02::fb
▸ User Datagram Protocol, Src Port: 5353, Dst Port: 5353
▸ Multicast Domain Name System (query)

0000  33 33 00 00 00 fb 76 9b  2d bf 48 89 86 dd 60 03   33····v· -·H···`·
0010  f7 6f 00 35 11 ff fe 80  00 00 00 00 00 00 74 9b   ·o·5···· ······t·
0020  2d ff fe bf 48 89 ff 02  00 00 00 00 00 00 00 00   -···H··· ········
0030  00 00 00 00 00 fb 14 e9  14 e9 00 35 e8 a8 00 00   ········ ···5····
0040  00 00 00 02 00 00 00 00  00 00 05 5f 69 70 70 73   ········ ···_ipps

wireshark_s1-eth2_20230331093406_T9EMXN.pcapng    Packets: 15 · Displayed: 15 (100.0%)    Profile: Default
```

```
No.  Time          Source                   Destination          Protocol Length Info
  6  7.166201123   10.0.0.3                 10.0.0.1             ICMP      98 Echo (ping) request  id=0x0b54, seq=1/256, ttl=
  7  7.175427150   10.0.0.1                 10.0.0.3             ICMP      98 Echo (ping) reply    id=0x0b54, seq=1/256, ttl=
  8  7.176691279   56:5c:84:77:5a:9d        Broadcast            ARP       42 Who has 10.0.0.2? Tell 10.0.0.3
  9  7.178004494   56:5c:84:77:5a:9d        Broadcast            ARP       42 Who has 10.0.0.2? Tell 10.0.0.3
 10  7.178668418   5a:44:a9:eb:30:83        56:5c:84:77:5a:9d    ARP       42 10.0.0.2 is at 5a:44:a9:eb:30:83
 11  7.178671611   10.0.0.3                 10.0.0.2             ICMP      98 Echo (ping) request  id=0x0b55, seq=1/256, ttl=
 12  7.187881146   10.0.0.2                 10.0.0.3             ICMP      98 Echo (ping) reply    id=0x0b55, seq=1/256, ttl=
 13  7.189599519   56:5c:84:77:5a:9d        Broadcast            ARP       42 Who has 10.0.0.4? Tell 10.0.0.3
 14  7.190648471   56:5c:84:77:5a:9d        Broadcast            ARP       42 Who has 10.0.0.4? Tell 10.0.0.3
 15  7.268689917   fe80::38d0:39ff:fe5…     ff02::fb             MDNS     107 Standard query 0x0000 PTR _ipps._tcp.local, "QM
 16  8.219573697   56:5c:84:77:5a:9d        Broadcast            ARP       42 Who has 10.0.0.4? Tell 10.0.0.3
 17  8.221074270   56:5c:84:77:5a:9d        Broadcast            ARP       42 Who has 10.0.0.4? Tell 10.0.0.3
 18  9.243564987   56:5c:84:77:5a:9d        Broadcast            ARP       42 Who has 10.0.0.4? Tell 10.0.0.3
 19  9.244752149   56:5c:84:77:5a:9d        Broadcast            ARP       42 Who has 10.0.0.4? Tell 10.0.0.3
 20  9.563581965   fe80::38d0:39ff:fe5…     ff02::2              ICMPv6    70 Router Solicitation from 3a:d0:39:52:ae:4f
 21 11.292884446   0a:6a:03:b8:cd:78        Broadcast            ARP       42 Who has 10.0.0.1? Tell 10.0.0.4
 22 11.303010771   0a:6a:03:b8:cd:78        Broadcast            ARP       42 Who has 10.0.0.2? Tell 10.0.0.4
 23 11.324257528   0a:6a:03:b8:cd:78        Broadcast            ARP       42 Who has 10.0.0.3? Tell 10.0.0.4
 24 11.324264788   56:5c:84:77:5a:9d        0a:6a:03:b8:cd:78    ARP       42 10.0.0.3 is at 56:5c:84:77:5a:9d
 25 11.332163800   10.0.0.4                 10.0.0.3             ICMP      98 Echo (ping) request  id=0x0b59, seq=1/256, ttl=
 26 11.332177629   10.0.0.3                 10.0.0.4             ICMP      98 Echo (ping) reply    id=0x0b59, seq=1/256, ttl=
 27 11.611585175   fe80::545c:84ff:fe7…     ff02::2              ICMPv6    70 Router Solicitation from 56:5c:84:77:5a:9d
 28 12.379699476   5a:44:a9:eb:30:83        56:5c:84:77:5a:9d    ARP       42 Who has 10.0.0.3? Tell 10.0.0.2
 29 12.379708649   56:5c:84:77:5a:9d        5a:44:a9:eb:30:83    ARP       42 10.0.0.3 is at 56:5c:84:77:5a:9d
 30 12.379714848   ea:3c:72:02:a5:41        56:5c:84:77:5a:9d    ARP       42 Who has 10.0.0.3? Tell 10.0.0.1
 31 12.379716120   56:5c:84:77:5a:9d        ea:3c:72:02:a5:41    ARP       42 10.0.0.3 is at 56:5c:84:77:5a:9d
 32 16.475557762   56:5c:84:77:5a:9d        0a:6a:03:b8:cd:78    ARP       42 Who has 10.0.0.4? Tell 10.0.0.3
 33 16.478580238   0a:6a:03:b8:cd:78        56:5c:84:77:5a:9d    ARP       42 10.0.0.4 is at 0a:6a:03:b8:cd:78
 34 71.260770912   fe80::38d0:39ff:fe5…     ff02::fb             MDNS     107 Standard query 0x0000 PTR _ipps._tcp.local, "QM

▸ Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0
▸ Ethernet II, Src: 5a:44:a9:eb:30:83 (5a:44:a9:eb:30:83), Dst: ea:3c:72:02:a5:41 (ea:3c:72:02:a5:41)
▸ Internet Protocol Version 4, Src: 10.0.0.2, Dst: 10.0.0.1
▸ Internet Control Message Protocol

0000  ea 3c 72 02 a5 41 5a 44  a9 eb 30 83 08 00 45 00   ·<r··AZD ··0···E·
0010  00 54 66 7e 40 00 40 01  c0 28 0a 00 00 02 0a 00   ·Tf~@·@· ·(······
0020  00 01 08 00 54 63 0b 51  00 01 9d 38 26 64 00 00   ····Tc·Q ···8&d··
0030  00 00 08 db 0d 00 00 00  00 00 10 11 12 13 14 15   ········ ········
0040  16 17 18 19 1a 1b 1c 1d  1e 1f 20 21 22 23 24 25   ········ ·· !"#$%

Bytes 42-49: Timestamp from icmp data (icmp.data_time)    Packets: 34 · Displayed: 34 (100.0%)    Profile: Defau
```

```
mininet> pingall
*** Ping: testing ping reachability
SRI -> X X X
UCLA -> SRI X X
UCSB -> SRI UCLA X
UTAH -> SRI UCLA UCSB
*** Results: 50% dropped (6/12 received)
mininet> 
```

```
[<ryu.controller.controller.Datapath object at 0x7fef5ecfb090>, <ryu.controller.controller.Datapath ob
ject at 0x7fef5ecfbc90>, <ryu.controller.controller.Datapath object at 0x7fef5ecfbd50>, <ryu.controlle
r.controller.Datapath object at 0x7fef5ecfb410>]
{'ea:3c:72:02:a5:41': 1}
{'56:5c:84:77:5a:9d': 1}
{'0a:6a:03:b8:cd:78': 1}
{'5a:44:a9:eb:30:83': 1}
```

图 13: 交换机链路上并没有广播 arp 包

4. 方案优缺点分析

优点: 直接不占用交换机链路带宽，可以充分利用交换机的带宽。

缺点: 方法复杂，不够简洁，而且需要先 ping 来获取主机与端口绑定信息。没有办法对后期带有 dst 的 arp 请求报文进行处理，仍然会走交换机链路。

四、 附录

1. Learning_Switch.py

```python
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
        self.mac_to_port = {}

    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
        # :)
        # learn src mac -> port
        self.mac_to_port[dpid][src] = in_port
        #if match, then send to outport
        if dst in self.mac_to_port[dpid]:
            out_port = self.mac_to_port[dpid][dst]
        #if not match, then flood
        else:
            out_port = ofp.OFPP_FLOOD
        #pass information
        actions = [parser.OFPActionOutput(out_port)]

        if out_port != ofp.OFPP_FLOOD:
            match = parser.OFPMatch(in_port=in_port, eth_dst=dst, eth_src=src)
            self.add_flow(dp, 1, match, actions)

        data = msg.data
        out = parser.OFPPacketOut(datapath=dp,buffer_id=msg.buffer_id,in_port=in_port,
             actions=actions, data=data)
        dp.send_msg(out)
```

2. Learning_Switch_Modified.py

```
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
        self.mac_to_port = {}

    def add_flow(self, datapath, priority, match,
         actions,buffer_id=None,idle_timeout=0,hard_timeout=0):
        dp = datapath
        ofp = dp.ofproto
        parser = dp.ofproto_parser
        inst = [parser.OFPInstructionActions(ofp.OFPIT_APPLY_ACTIONS, actions)]
        if buffer_id:
            mod = parser.OFPFlowMod(datapath=datapath,
                 priority=priority,buffer_id=buffer_id,idle_timeout=idle_timeout,
                             hard_timeout=hard_timeout,match=match,instructions=inst)
        else:
            mod = parser.OFPFlowMod(datapath=datapath,
                 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
        # :)
        # learn src mac -> port
        self.mac_to_port[dpid][src] = in_port
        #if match, then send to outport
        if dst in self.mac_to_port[dpid]:
            out_port = self.mac_to_port[dpid][dst]
        #if not match, then flood
        else:
            out_port = ofp.OFPP_FLOOD
        #pass information
        actions = [parser.OFPActionOutput(out_port)]

        if out_port != ofp.OFPP_FLOOD:
            match = parser.OFPMatch(in_port=in_port, eth_dst=dst, eth_src=src)
            if msg.buffer_id != ofp.OFP_NO_BUFFER:
                self.add_flow(dp, 1, match, actions, msg.buffer_id)
                return
            else:
                self.add_flow(dp, 1, match, actions)

        data = None
        if msg.buffer_id == ofp.OFP_NO_BUFFER:
            data = msg.data

        out = parser.OFPPacketOut(datapath=dp,buffer_id=msg.buffer_id,in_port=in_port, actions=actions, data=data)
        dp.send_msg(out)
```

3. Broadcast_Loop.py

```
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
        self.mac_to_port = {}



    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
            arp_dst_ip = header_list[ARP].dst_ip
            if (dp.id, src, arp_dst_ip) in self.sw:
                if self.sw[(dp.id, src, arp_dst_ip)] != in_port:
                    out = dp.ofproto_parser.OFPPacketOut(datapath=dp,
                    buffer_id=dp.ofproto.OFP_NO_BUFFER,
                    in_port=in_port,actions=[], data=None)
                    dp.send_msg(out)
                    return
            else:
                self.sw[(dp.id, src, arp_dst_ip)] = in_port

        # self-learning
        # you need to code here to avoid the direct flooding
        # having fun
        # :)
        # just code in mission 1
        # learn src mac -> port
        self.mac_to_port[dpid][src] = in_port
        #if match, then send to outport
        if dst in self.mac_to_port[dpid]:
            out_port = self.mac_to_port[dpid][dst]
        #if not match, then fllink_list ood
        else:
            out_port = ofp.OFPP_FLOOD
        #pass information
        actions = [parser.OFPActionOutput(out_port)]

        if out_port != ofp.OFPP_FLOOD:
            match = parser.OFPMatch(in_port=in_port, eth_dst=dst, eth_src=src)
            self.add_flow(dp, 1, match, actions)

        data = None
        if msg.buffer_id == ofp.OFP_NO_BUFFER:
            data = msg.data

        out = parser.OFPPacketOut(datapath=dp,buffer_id=msg.buffer_id,in_port=in_port, actions=actions, data=data)
        dp.send_msg(out)
```

4. Broadcast_Loop2.py

```
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.mac_to_port = {}
        self.host_mac_port = {}


    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(dp, {})
        # 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)

            self.mac_to_port[dp][src] = in_port
            if dst in self.mac_to_port[dp]:
                out_port = self.mac_to_port[dp][dst]
            else:
                out_port = ofp.OFPP_FLOOD
            actions = [parser.OFPActionOutput(out_port)]
            if out_port != ofp.OFPP_FLOOD:
                match = parser.OFPMatch(in_port=in_port, eth_dst=dst, eth_src=src)
                self.add_flow(dp, 1, match, actions)

            if ARP in header_list:
            # you need to code here to avoid broadcast loop to finish mission 2
                opcode = header_list[ARP].opcode
                if  opcode == arp.ARP_REQUEST:
                    self.host_mac_port.setdefault(dp, {})
                    if src in self.host_mac_port[dp]:
                        for switch in self.host_mac_port.keys():
                            ofp = switch.ofproto
                            parser = switch.ofproto_parser
                            for out_port in self.host_mac_port[switch].values():
                                #print(out_port)
                                actions = [parser.OFPActionOutput(out_port)]
                                out = parser.OFPPacketOut(datapath=switch,buffer_id=ofp.OFP_NO_BUFFER,
                                in_port=ofp.OFPP_CONTROLLER,actions=actions, data=msg.data)
                                switch.send_msg(out)
                        return
                else:
                    self.host_mac_port[dp][src] = in_port
                    return

                elif opcode == arp.ARP_REPLY:
                    for switch in self.host_mac_port.keys():
                        if dst in self.host_mac_port[switch]:
                            ofp = switch.ofproto
                            parser = switch.ofproto_parser
                            out_port = self.host_mac_port[switch][dst]
                            actions = [parser.OFPActionOutput(out_port)]
                            out = parser.OFPPacketOut(datapath=switch,buffer_id=ofp.OFP_NO_BUFFER,
                            in_port=ofp.OFPP_CONTROLLER,actions=actions, data=msg.data)
                            switch.send_msg(out)
                    return

        data = None
        if msg.buffer_id == ofp.OFP_NO_BUFFER:
            data = msg.data
        out = parser.OFPPacketOut(datapath=dp,buffer_id=msg.buffer_id,in_port=in_port, actions=actions, data=data)
```

```
    dp.send_msg(out)
```