**SDN-lab2实验报告**

**实验任务一：自学习交换机**

1.工作流程

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。

2.topo\_1969\_1网络拓扑结构：

**SRI(h1)**

**|**

**s1**

**/ | \**

**s2 s3 s4**

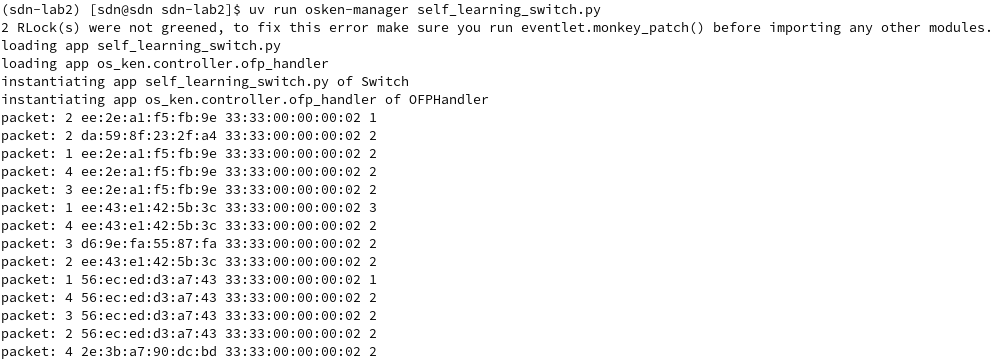
**| | |**

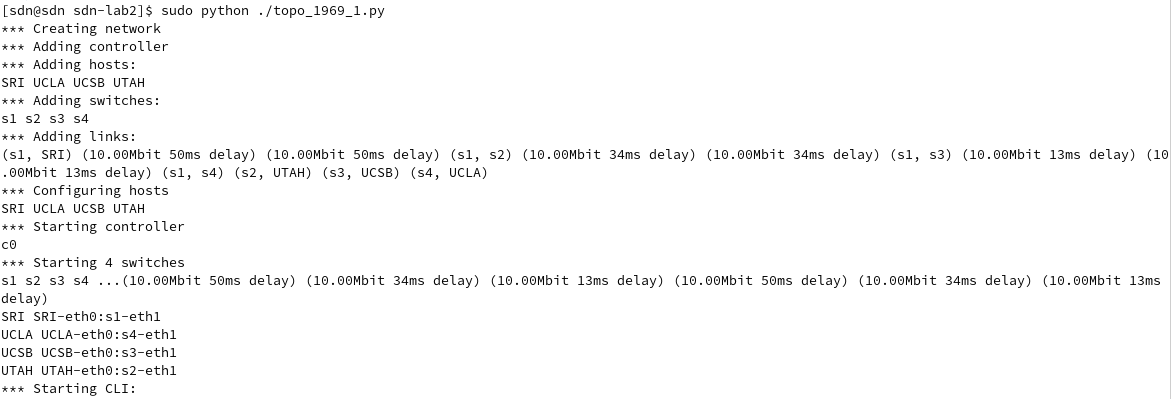
**UTAH UCSB UCLA**

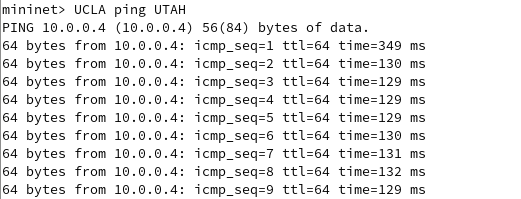
**(h2) (h3) (h4)**

3.实验步骤

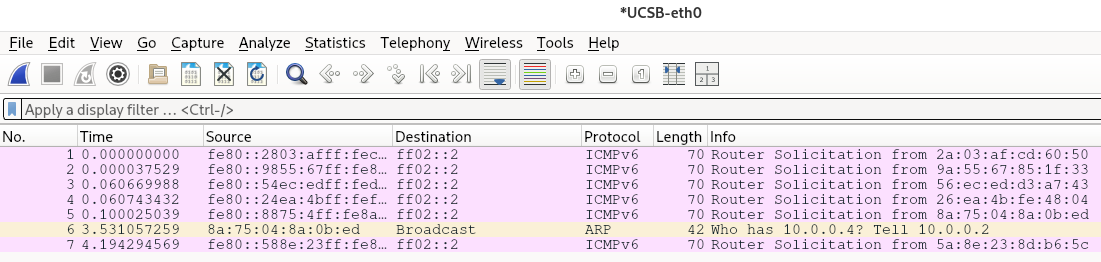
启动控制器和网络拓扑







UCLA ping UTAH可以ping通



UCSB没有收到相关数据包

初始时交换机还没学习MAC地址，UCLA要ping UTAH，先发送ARP请求报文获取UTAH的MAC地址。ARP请求报文以广播形式发送，因此UCSB可以捕获到ARP请求报文。

UTAH收到ARP请求后，会给UCLA单播回复ARP响应，交换机学习到UCLA和UTAH之间的通信路径是UCLA→s4→s1→s2→UTAH，不经过s3，UCSB不会捕获ARP响应和后续报文。

4.代码

1. from os\_ken.base import app\_manager
2. from os\_ken.controller import ofp\_event
3. from os\_ken.controller.handler import MAIN\_DISPATCHER, CONFIG\_DISPATCHER
4. from os\_ken.controller.handler import set\_ev\_cls
5. from os\_ken.ofproto import ofproto\_v1\_3
6. from os\_ken.lib.packet import packet
7. from os\_ken.lib.packet import ethernet
9. class Switch(app\_manager.OSKenApp):
10. OFP\_VERSIONS = [ofproto\_v1\_3.OFP\_VERSION]
11. def \_\_init\_\_(self, \*args, \*\*kwargs):
12. super(Switch, self).\_\_init\_\_(\*args, \*\*kwargs)
13. self.mac\_to\_port = {}  *#保存MAC地址到端口的映射关系*
15. *#添加流表项*
16. def add\_flow(self,datapath,priority,match,actions,idle\_timeout=0,hard\_timeout=0):
17. dp = datapath
18. ofp = dp.ofproto
19. parser = dp.ofproto\_parser
20. inst = [parser.OFPInstructionActions(ofp.OFPIT\_APPLY\_ACTIONS, actions)]
21. mod = parser.OFPFlowMod(datapath=dp, priority=priority,
22. idle\_timeout=idle\_timeout,
23. hard\_timeout=hard\_timeout,
24. match=match,instructions=inst)
25. dp.send\_msg(mod)
27. *#交换机与控制器建立连接后，创建一个默认流表项，将所有数据包发送到控制器*
28. @set\_ev\_cls(ofp\_event.EventOFPSwitchFeatures, CONFIG\_DISPATCHER)
29. def switch\_features\_handler(self, ev):
30. msg = ev.msg
31. dp = msg.datapath
32. ofp = dp.ofproto
33. parser = dp.ofproto\_parser
34. match = parser.OFPMatch()
35. actions = [parser.OFPActionOutput(ofp.OFPP\_CONTROLLER,ofp.OFPCML\_NO\_BUFFER)]
36. self.add\_flow(dp, 0, match, actions)
38. *#处理交换机转发来的数据包*
39. @set\_ev\_cls(ofp\_event.EventOFPPacketIn, MAIN\_DISPATCHER)
40. def packet\_in\_handler(self, ev):
41. msg = ev.msg
42. dp = msg.datapath
43. ofp = dp.ofproto
44. parser = dp.ofproto\_parser
46. dpid = dp.id  *#获取交换机ID*
47. self.mac\_to\_port.setdefault(dpid,{})
49. in\_port = msg.match['in\_port']  *#获取入端口*
50. pkt = packet.Packet(msg.data)
52. eth\_pkt = pkt.get\_protocol(ethernet.ethernet)  *#解析数据包*
53. dst = eth\_pkt.dst  *#获取源MAC和目标MAC*
54. src = eth\_pkt.src
55. self.logger.info('packet: %s %s %s %s', dpid, src, dst, in\_port)  *#记录日志信息*

58. self.mac\_to\_port[dpid][src] = in\_port  *#记录源MAC地址和它进入的端口*
59. if dst in self.mac\_to\_port[dpid]:  *#检查目标MAC是否已知*
60. out\_port = self.mac\_to\_port[dpid][dst]  *#如果知道，获取对应的出端口*
61. else:
62. out\_port = ofp.OFPP\_FLOOD  *#如果不知道，设置为泛洪*
64. actions = [parser.OFPActionOutput(out\_port)]
66. *#添加流表项和发送数据包*
67. *#如果目标MAC已知，添加一条流表项，下次相同的数据包就不需要经过控制器*
68. if out\_port != ofp.OFPP\_FLOOD:
69. match = parser.OFPMatch(in\_port = in\_port, eth\_src = src, eth\_dst = dst)
70. self.add\_flow(dp, 1, match, actions)
72. *#创建并发送OFPPacketOut消息，指示交换机如何处理这个数据包*
73. out = parser.OFPPacketOut(datapath = dp, buffer\_id = msg.buffer\_id,in\_port = in\_port,
74. actions = actions, data = msg.data)
75. dp.send\_msg(out)

**实验任务二：处理环路广播**

1.处理方法

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。

2.topo\_1969\_2拓扑结构

**SRI(h1)**

**|**

**s1**

**/ | \**

**s2 s3—s4**

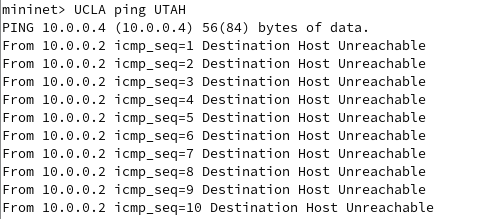
**| | |**

**UTAH UCSB UCLA**

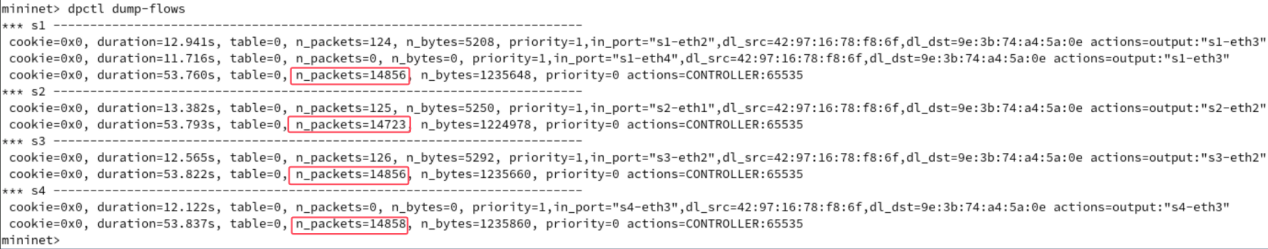
**(h2) (h3) (h4)**

可以看出s1 s3 s4形成环路

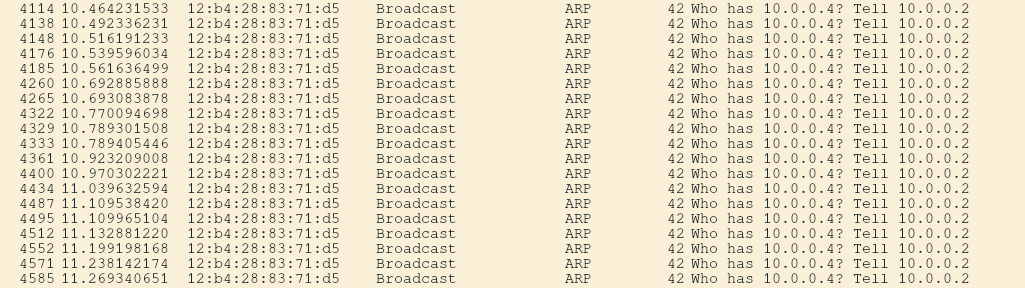
2.实验步骤



在拓扑topo\_1969\_2.py中运⾏⾃学习交换机，UCLA和UTAH之间⽆法正常通信

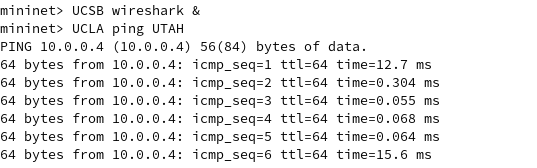


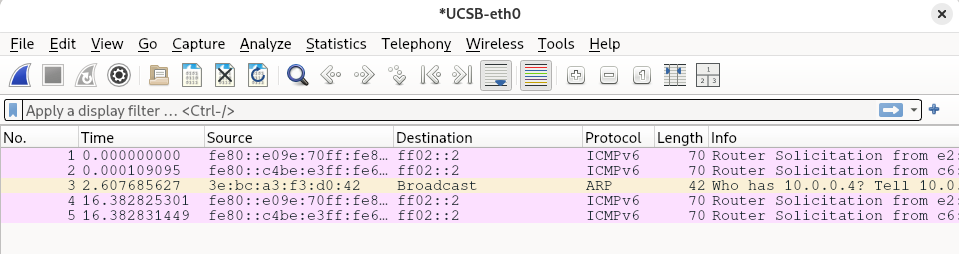
源主机虽然只发了很少的⼏个数据包，但流表项却匹配了上千次



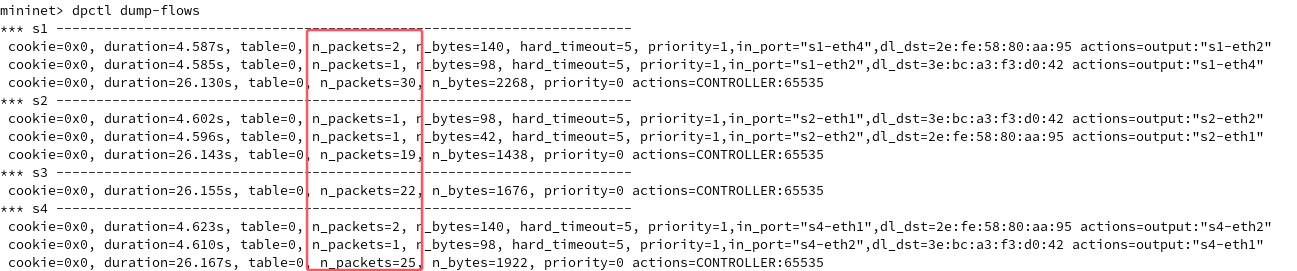
WireShark也截取到了数⽬异常⼤的相同报⽂，这是ARP⼴播报文在环状拓扑中洪泛导致的。

修改交换机代码后，UCLA ping UTAH可以ping通





UCLA和UTAH之间可以ping通



流表项的匹配次数明显减少

3.代码

1. import eventlet
2. eventlet.monkey\_patch()
3. from os\_ken.base import app\_manager
4. from os\_ken.controller import ofp\_event
5. from os\_ken.controller.handler import MAIN\_DISPATCHER, CONFIG\_DISPATCHER
6. from os\_ken.controller.handler import set\_ev\_cls
7. from os\_ken.ofproto import ofproto\_v1\_3
8. from os\_ken.lib.packet import packet
9. from os\_ken.lib.packet import ethernet
10. from os\_ken.lib.packet import arp
11. from os\_ken.lib.packet import ether\_types
12. ETHERNET = ethernet.ethernet.\_\_name\_\_
13. ETHERNET\_MULTICAST = "ff:ff:ff:ff:ff:ff"
14. ARP = arp.arp.\_\_name\_\_
15. class Switch\_Dict(app\_manager.OSKenApp):
16. OFP\_VERSIONS = [ofproto\_v1\_3.OFP\_VERSION]
17. def \_\_init\_\_(self, \*args, \*\*kwargs):
18. super(Switch\_Dict, self).\_\_init\_\_(\*args, \*\*kwargs)
19. self.sw = {}  *#用于环路检测的字典*
20. self.mac\_to\_port = {}
21. *#添加流表项*
22. def add\_flow(self, datapath, priority, match, actions, idle\_timeout=0, hard\_timeout=0):
23. dp = datapath
24. ofp = dp.ofproto
25. parser = dp.ofproto\_parser
26. inst = [parser.OFPInstructionActions(ofp.OFPIT\_APPLY\_ACTIONS, actions)]
27. mod = parser.OFPFlowMod(datapath=dp, priority=priority,
28. idle\_timeout=idle\_timeout,
29. hard\_timeout=hard\_timeout,
30. match=match, instructions=inst)
31. dp.send\_msg(mod)
32. @set\_ev\_cls(ofp\_event.EventOFPSwitchFeatures, CONFIG\_DISPATCHER)
33. def switch\_features\_handler(self, ev):
34. msg = ev.msg
35. dp = msg.datapath
36. ofp = dp.ofproto
37. parser = dp.ofproto\_parser
38. match = parser.OFPMatch()
39. actions = [parser.OFPActionOutput(ofp.OFPP\_CONTROLLER, ofp.OFPCML\_NO\_BUFFER)]
40. self.add\_flow(dp, 0, match, actions)
41. @set\_ev\_cls(ofp\_event.EventOFPPacketIn, MAIN\_DISPATCHER)
42. def packet\_in\_handler(self, ev):
43. msg = ev.msg
44. dp = msg.datapath
45. ofp = dp.ofproto
46. parser = dp.ofproto\_parser
47. dpid = dp.id
48. self.mac\_to\_port.setdefault(dpid, {})
49. in\_port = msg.match['in\_port']
50. pkt = packet.Packet(msg.data)
51. eth\_pkt = pkt.get\_protocol(ethernet.ethernet)
52. if eth\_pkt.ethertype == ether\_types.ETH\_TYPE\_LLDP:  *#丢弃LLDP协议包（防止环路）*
53. return
54. if eth\_pkt.ethertype == ether\_types.ETH\_TYPE\_IPV6:  *#丢弃IPv6协议包*
55. return
56. dst = eth\_pkt.dst
57. src = eth\_pkt.src
58. *#获取数据包中的协议头*
59. header\_list = dict((p.protocol\_name, p) for p in pkt.protocols if type(p) != str)
60. *# 收到广播的ARP请求报文时*
61. if dst == ETHERNET\_MULTICAST and ARP in header\_list:
62. *# you need to code here to avoid broadcast loop to finish mission 2*
64. arp\_dst\_ip = header\_list[ARP].dst\_ip  *#获取目的ip地址*
65. if (dpid, src, arp\_dst\_ip) in self.sw:  *#检查是否已记录该ARP请求*
66. if self.sw[(dpid, src, arp\_dst\_ip)] != in\_port:  *#如果入端口与记录的不同，说明出现环路*
67. *#丢弃数据包*
68. out = parser.OFPPacketOut(datapath=dp, buffer\_id=msg.buffer\_id, in\_port=in\_port, actions=[], data=None)
69. dp.send\_msg(out)
70. return
71. else:  *#第一次收到该ARP报文*
72. self.sw[(dpid, src, arp\_dst\_ip)] = in\_port  *#记录端口*
73. self.mac\_to\_port[dpid][src] = in\_port
74. if dst in self.mac\_to\_port[dpid]:
75. out\_port = self.mac\_to\_port[dpid][dst]
76. else:
77. out\_port = ofp.OFPP\_FLOOD
78. actions = [parser.OFPActionOutput(out\_port)]
79. *#如果是已知单播数据包，添加流表项*
80. if out\_port != ofp.OFPP\_FLOOD:
81. match = parser.OFPMatch(in\_port=in\_port, eth\_dst=dst)
82. self.add\_flow(dp, 1, match, actions, hard\_timeout=5)
84. data = None
85. if msg.buffer\_id == ofp.OFP\_NO\_BUFFER:
86. data = msg.data
88. out = parser.OFPPacketOut(datapath=dp, buffer\_id=msg.buffer\_id,
89. in\_port=in\_port, actions=actions, data=data)
90. dp.send\_msg(out)