**SDN第一次实验报告**

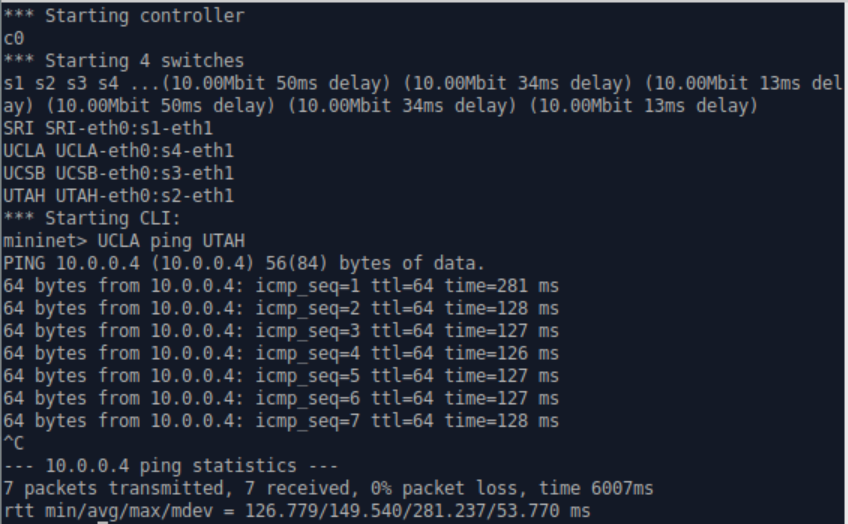
实验一：自学习交换机

实验思路：

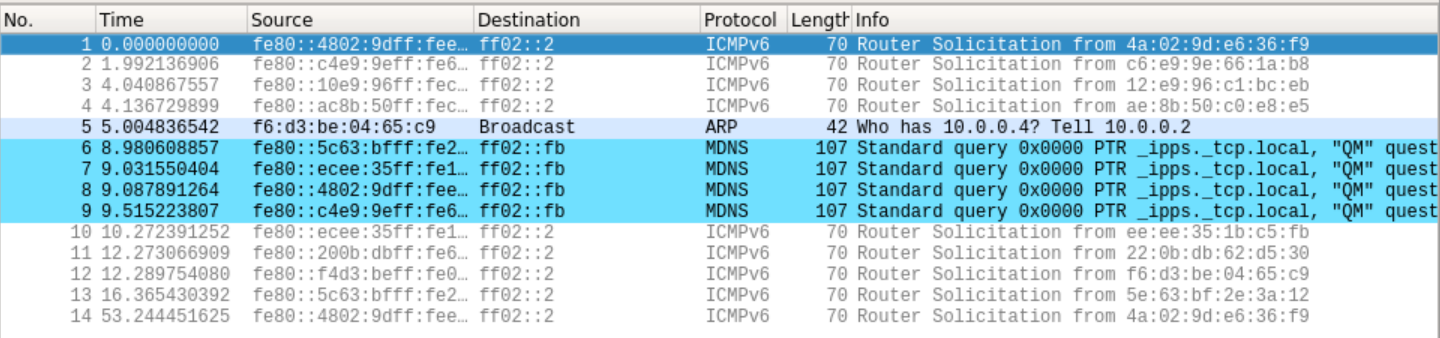
简单交换机在转发数据包时只是简单地洪泛数据包，两台主机在通信时，网络中的其它主机也会收到数据包，同时也会造成带宽的浪费。

为了解决这些问题，在控制器中设置了一个mac-port映射表，在接收到packet\_in消息后，控制器将收到数据包的mac地址和in\_port记录在映射表中，如果在表中能查到dst\_mac，则向指定的端口转发数据包，否则洪泛数据包。同时控制器向交换机下发流表，指导交换机转发类似的数据包。

实验结果：



UCLA，UCSB，UTAH，SRI之间可以相互ping通



在设置了自学习交换机后，UCLA在ping UTAH时，UCSB不再收到相关数据包

实验代码：

1. **from** ryu.base **import** app\_manager
2. **from** ryu.controller **import** ofp\_event
3. **from** ryu.controller.handler **import** MAIN\_DISPATCHER,CONFIG\_DISPATCHER
4. **from** ryu.controller.handler **import** set\_ev\_cls
5. **from** ryu.ofproto **import** ofproto\_v1\_3
7. **class** L2Switch(app\_manager.RyuApp):
8. OFP\_VERSIONS = [ofproto\_v1\_3.OFP\_VERSION]
10. **def** \_\_init\_\_(self, \*args, \*\*kwargs):
11. super(L2Switch,self).\_\_init\_\_(\*args,\*\*kwargs)
13. #add a flow table entry in switch
14. **def** add\_flow(self,datapath,priority,match,actions):
15. dp=datapath
16. ofp=dp.ofproto
17. parser=dp.ofproto\_parser
19. #construct a FlowMod message,send it to a switch to add a flow table entry
20. inst = [parser.OFPInstructionActions(ofp.OFPIT\_APPLY\_ACTIONS,actions)]
21. mod = parser.OFPFlowMod(datapath=dp,priority=priority,match=match,instructions=inst)
22. dp.send\_msg(mod)
24. #add default flow table which sends packets to the controller
25. @set\_ev\_cls(ofp\_event.EventOFPSwitchFeatures,CONFIG\_DISPATCHER)
26. **def** switch\_features\_handler(self,ev):
27. msg =ev.msg
28. dp=msg.datapath
29. ofp=dp.ofproto
30. parser=dp.ofproto\_parser
32. match=parser.OFPMatch()
33. actions=[parser.OFPActionOutput(ofp.OFPP\_CONTROLLER,ofp.OFPCML\_NO\_BUFFER)]#send packets to controller
34. self.add\_flow(dp,0,match,actions)
36. #handle packet\_in message
37. @set\_ev\_cls(ofp\_event.EventOFPPacketIn,MAIN\_DISPATCHER)
38. **def** packet\_in\_handler(self,ev):
39. msg=ev.msg
40. dp=msg.datapath
41. ofp=dp.ofproto
42. parser=dp.ofproto\_parser
43. actions=[parser.OFPActionOutput(ofp.OFPP\_FLOOD)]
44. out=parser.OFPPacketOut(
45. datapath=dp,buffer\_id=msg.buffer\_id,
46. in\_port=msg.match['in\_port'],actions=actions,data=msg.data)
47. dp.send\_msg(out)

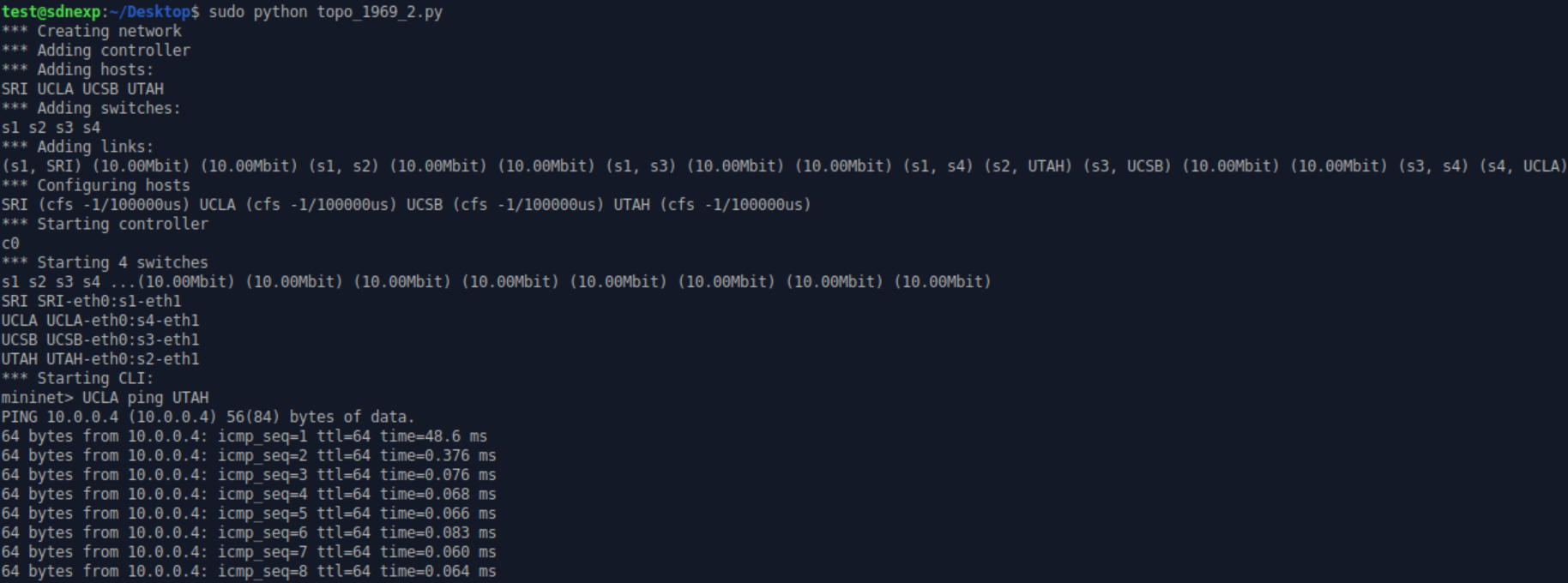
实验二：环路广播

实验思路：

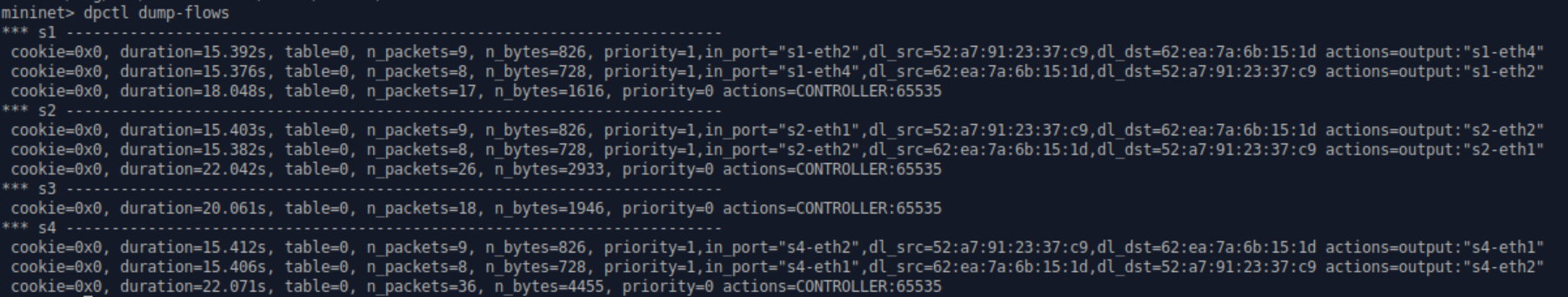
由于环状拓扑，ARP在广播洪泛时会造成广播风暴。为了解决这个问题，当序号为 dpid 的交换机从某端口第一次收到某个src\_mac主机发出，询问dst\_ip的广播ARP Request数据包时，控制器记录一个映射(dpid, src\_mac, dst\_ip)->in\_port。当下一次该交换机收到同一(src\_mac, dst\_ip) 发来的ARP Request报文但in\_port不同的 ARP Request 数据包时直接丢弃，否则洪泛。

同时使用实验一的方法避免洪泛。

实验结果：



在解决了环路广播洪泛后，UCLA可以ping通UTAH



查看流表项的匹配次数，匹配次数明显减少

实验代码：

1. **from** ryu.base **import** app\_manager
2. **from** ryu.controller **import** ofp\_event
3. **from** ryu.controller.handler **import** MAIN\_DISPATCHER, CONFIG\_DISPATCHER
4. **from** ryu.controller.handler **import** set\_ev\_cls
5. **from** ryu.ofproto **import** ofproto\_v1\_3
6. **from** ryu.lib.packet **import** packet
7. **from** ryu.lib.packet **import** ethernet
8. **from** ryu.lib.packet **import** arp
9. **from** ryu.lib.packet **import** ether\_types
11. ETHERNET = ethernet.ethernet.\_\_name\_\_
12. ETHERNET\_MULTICAST = "ff:ff:ff:ff:ff:ff"
13. ARP = arp.arp.\_\_name\_\_

16. **class** Switch\_Dict(app\_manager.RyuApp):
17. OFP\_VERSIONS = [ofproto\_v1\_3.OFP\_VERSION]
19. **def** \_\_init\_\_(self, \*args, \*\*kwargs):
20. super(Switch\_Dict, self).\_\_init\_\_(\*args, \*\*kwargs)
21. self.sw = {} #(dpid, src\_mac, dst\_ip)=>in\_port, you may use it in mission 2
22. # maybe you need a global data structure to save the mapping
23. # just data structure in mission 1
24. self.mac\_to\_port={}

27. **def** add\_flow(self, datapath, priority, match, actions, idle\_timeout=0, hard\_timeout=0):
28. dp = datapath
29. ofp = dp.ofproto
30. parser = dp.ofproto\_parser
31. inst = [parser.OFPInstructionActions(ofp.OFPIT\_APPLY\_ACTIONS, actions)]
32. mod = parser.OFPFlowMod(datapath=dp, priority=priority,
33. idle\_timeout=idle\_timeout,
34. hard\_timeout=hard\_timeout,
35. match=match, instructions=inst)
36. dp.send\_msg(mod)
38. @set\_ev\_cls(ofp\_event.EventOFPSwitchFeatures, CONFIG\_DISPATCHER)
39. **def** switch\_features\_handler(self, ev):
40. msg = ev.msg
41. dp = msg.datapath
42. ofp = dp.ofproto
43. parser = dp.ofproto\_parser
44. match = parser.OFPMatch()
45. actions = [parser.OFPActionOutput(ofp.OFPP\_CONTROLLER, ofp.OFPCML\_NO\_BUFFER)]
46. self.add\_flow(dp, 0, match, actions)
48. @set\_ev\_cls(ofp\_event.EventOFPPacketIn, MAIN\_DISPATCHER)
49. **def** packet\_in\_handler(self, ev):
50. msg = ev.msg
51. dp = msg.datapath
52. ofp = dp.ofproto
53. parser = dp.ofproto\_parser
55. # the identity of switch
56. dpid = dp.id
57. self.mac\_to\_port.setdefault(dpid, {})
58. self.sw.setdefault(dpid, {})
59. # the port that receive the packet
60. in\_port = msg.match['in\_port']
61. pkt = packet.Packet(msg.data)
62. eth\_pkt = pkt.get\_protocol(ethernet.ethernet)
64. **if** eth\_pkt.ethertype == ether\_types.ETH\_TYPE\_LLDP:
65. **return**
66. **if** eth\_pkt.ethertype == ether\_types.ETH\_TYPE\_IPV6:
67. **return**
68. # get the mac
69. dst = eth\_pkt.dst
70. src = eth\_pkt.src
71. # get protocols
72. header\_list = dict((p.protocol\_name, p) **for** p **in** pkt.protocols **if** type(p) != str)
73. **if** dst == ETHERNET\_MULTICAST **and** ARP **in** header\_list:
74. # you need to code here to avoid broadcast loop to finish mission 2
75. dst\_ip=header\_list[ARP].dst\_ip
76. **if** (dpid,src,dst\_ip) **not** **in** self.sw:
77. self.sw[(dpid,src,dst\_ip)]=in\_port
78. **elif** self.sw[(dpid,src,dst\_ip)]!=in\_port: #drop the packet
79. **return**
80. # self-learning
81. # you need to code here to avoid the direct flooding
82. # having fun
83. # :)
84. # just code in mission 1
85. self.mac\_to\_port[dpid][src]=in\_port
86. **if** dst **in** self.mac\_to\_port[dpid]:
87. out\_port=self.mac\_to\_port[dpid][dst]
88. **else**:
89. out\_port=ofp.OFPP\_FLOOD
90. actions=[parser.OFPActionOutput(out\_port)]
91. #install flow
92. **if** out\_port!=ofp.OFPP\_FLOOD:
93. match=parser.OFPMatch(in\_port=in\_port,eth\_dst=dst,eth\_src=src)
94. self.add\_flow(dp,1,match,actions)
95. data=None
96. **if** msg.buffer\_id==ofp.OFP\_NO\_BUFFER:
97. data=msg.data
98. out=parser.OFPPacketOut(datapath=dp,buffer\_id=msg.buffer\_id,in\_port=in\_port,actions=actions,data=data)
99. dp.send\_msg(out)