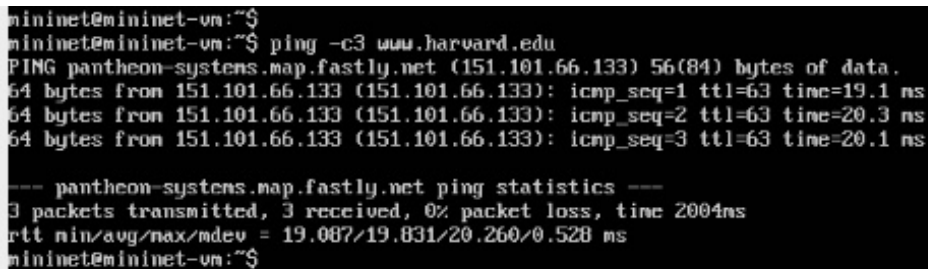


CS143 PA1

Miles Butler, Jeffrey Mayolo, Angela Wu

February 2023

1 Problem 1



```
mininet@mininet-vm:~$  
mininet@mininet-vm:~$ ping -c3 www.harvard.edu  
PING pantheon-systems.map.fastly.net (151.101.66.133) 56(84) bytes of data:  
64 bytes from 151.101.66.133 (151.101.66.133): icmp_seq=1 ttl=63 time=19.1 ns  
64 bytes from 151.101.66.133 (151.101.66.133): icmp_seq=2 ttl=63 time=20.3 ns  
64 bytes from 151.101.66.133 (151.101.66.133): icmp_seq=3 ttl=63 time=20.1 ns  
  
--- pantheon-systems.map.fastly.net ping statistics ---  
3 packets transmitted, 3 received, 0% packet loss, time 2004ms  
rtt min/avg/max/mdev = 19.087/19.831/20.260/0.528 ms  
mininet@mininet-vm:~$
```

(a) terminal output from pinging Harvard

1.1 What did you see?

I saw that the VM (mininet) sent a ping request to Harvard's servers and Harvard web servers responded to each request.

1.2 How many ICMP packets (ping requests) were sent out and replied to successfully by the web server?

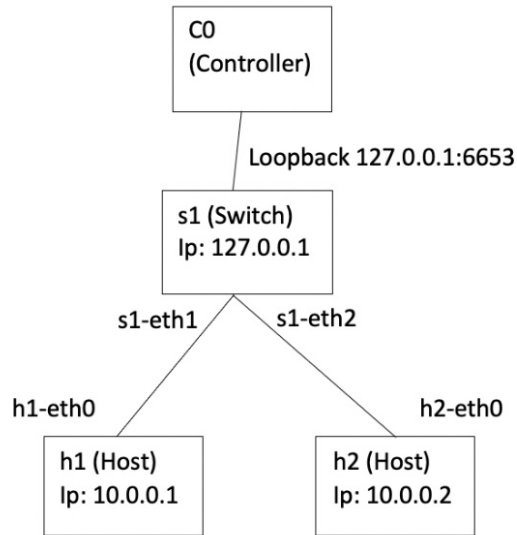
3 Packets were transmitted from the Vm to Harvard's web server and all 3 were replied to.

1.3 How long did the replies for each individual ping request take?

Each Individual ping request took about 20ms

2 Problem 2

Draw a precise diagram of the default network topology. Label each node and its network interface(s) correctly



(a) Default network topology

3 Problem 3

3.1 Does pingging between hosts work?

Pinging between hosts doesn't work as seen below

```

*** Starting CLI:
mininet> h1 ping -c3 h2
PING 10.0.0.7 (10.0.0.7) 56(84) bytes of data:
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable
From 10.0.0.1 icmp_seq=2 Destination Host Unreachable
From 10.0.0.1 icmp_seq=3 Destination Host Unreachable

--- 10.0.0.7 ping statistics ---
3 packets transmitted, 0 received, +3 errors, 100% packet loss, time 2027ms
pipe 3
mininet> _

```

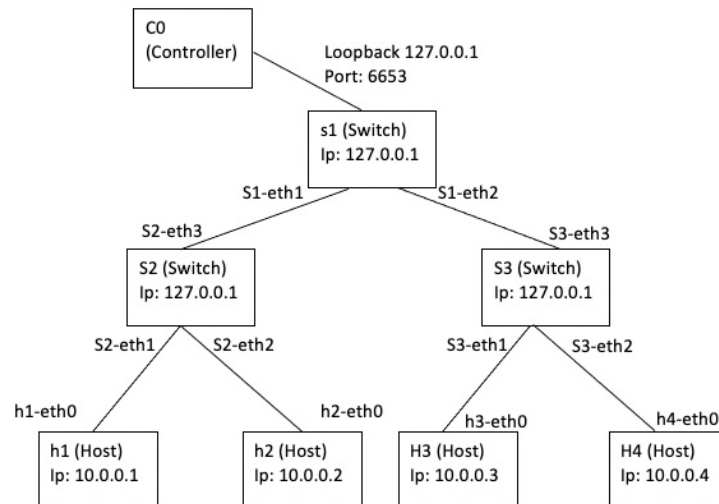
(a) terminal output from pinging hosts

3.2 Explain the difference between this topology and the reference one from Problem 2. If you were able to ping successfully, draw a diagram of the network topology. If you were unable to ping successfully, explain how you could fix the issue.

The difference between this topology and the default topology above is that within this topology each host is connected to an individual switch rather than all of the hosts being connected to a single switch. Even though all of the switches are connected to each-other, they operate at the data link layer (layer 2) of the OSI model meaning they can only forward traffic between different networks (layer 3). In order to fix this problem, we would need to either allow IP forwarding on the switch or there would need to be a router to handle inter-host communication.

4 Problem 4

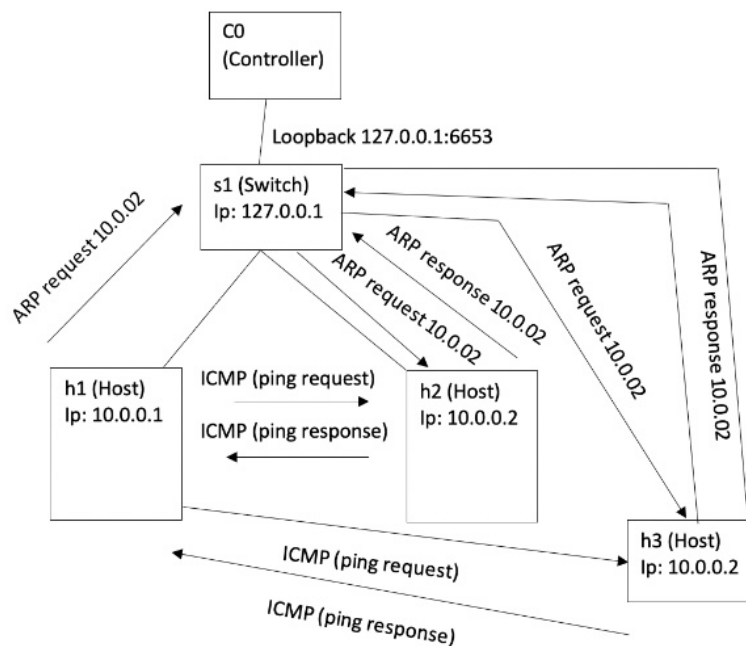
Draw a precise diagram of this network topology, labeling each node and its network interface(s) correctly with IP addresses (with same guidelines as before).



(a) Fanout network topology

5 Problem 5

5.1 You should see two types of packets in the tcpdump output on the xterms of h2 and h3. What are the two types? Draw the path of ping packets on a diagram of the network (both request and response packets).

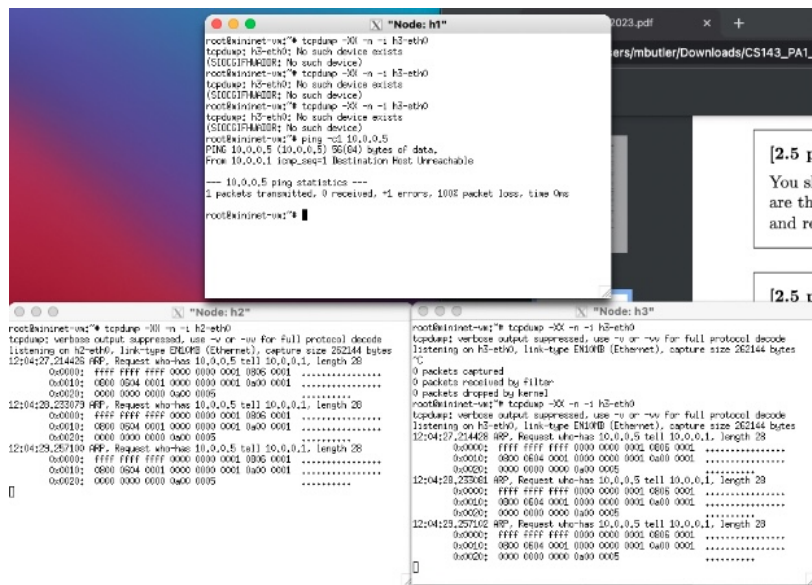


(a) packet paths

The Two types of packets are ICMP requests and responses (ping) and ARP requests.

5.2 Now, see what happens when a non-existent host fails to reply (e.g., ping -c1 10.0.0.5). How many packets and of what type do you see in the tcpdump output for h2 and h3?

I see that each host (h2 and h3) get 3 ARP requests from the h1 node (10.0.0.1) requesting the address of 10.0.0.5.



(a) non-existent host reply

5.3 What sort of speeds are you seeing? The virtual links in Mininet have a very high bandwidth, so they should not be a bottleneck/reason for slow speeds. Based on what you now know about OpenFlow and this hub, can you give a few reasons for why speeds may be slower than expected?

The iperf from the networks reports speeds of 103Gbits/s which is slightly slower than the speeds we were getting with baseline networks.

Hubs use half-duplex communication, meaning it can only transmit or receive data at any given time, leading to slower speeds compared to full-duplex communication used by modern switches. Hubs also forward all incoming data to all connected hubs causing unnecessary congestion and slowing down speeds.

The hub may not have sufficient processing power to handle a large volume of traffic, leading to slow speeds (e.g., there may not be sufficient processing power to serve packets constantly if all components are emulated in real time).

6 Problem 6

What do you see? Examine the code of the L2 learning controller at `/pox/pox/forwarding/l2_learning.py`. Explain the behavior of this controller.

```
mininet@mininet-vm:~/pox$ python pox.py log.level --DEBUG forwarding.l2_learning
POX 0.7.0 (gar) / Copyright 2011-2020 James McCauley, et al.
DEBUG:core:POX 0.7.0 (gar) going up...
DEBUG:core:Running on CPython (3.8.5/Jul 28 2020 12:59:40)
DEBUG:core:Platform is Linux-5.4.0-42-generic-x86_64-with-glibc2.29
WARNING:version:Support for Python 3 is experimental.
INFO:core:POX 0.7.0 (gar) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
INFO:openflow.of_01:[00-00-00-00-00-01 1] connected
DEBUG:forwarding.l2_learning:Connection [00-00-00-00-00-01 1]
DEBUG:forwarding.l2_learning:Port for 00:00:00:00:00:02 unknown -- flooding
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:02.2 -> 00:00:00:00:00:01.1
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:02.2 -> 00:00:00:00:00:01.1
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:01.1 -> 00:00:00:00:00:02.2
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:01.1 -> 00:00:00:00:00:02.2
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:02.2 -> 00:00:00:00:00:01.1
```

(a) behavior of L2 controller

7 Problem 7

```

1  #!/usr/bin/python
2  from mininet.topo import Topo
3  from mininet.topo import Topo
4  from mininet.link import TCLink
5  from mininet.log import setLogLevel
6  from mininet.net import Mininet
7  from mininet.node import CPULimitedHost
8  from mininet.topo import Topo
9  from mininet.util import dumpNodeConnections, irange
10
11 class CustomTopo(Topo):
12     """
13     Simple Data Center Topology
14
15     linkopts# - link parameters (where #: 1: core, 2: aggregation, 3: edge)
16     fanout - number of child switches per parent switch
17     """
18
19     def __init__(self, linkopts1, linkopts2, linkopts3, fanout=2, **opts):
20         """Initialize topology and default options"""
21         Topo.__init__(self, **opts)
22
23
24
25         """ Implement your logic here """
26         self.host_count = 0
27         self.edge_count = 0
28         self.fanout = fanout
29
30         # ADD Core SWITCH
31         core = self.addSwitch('c1')
32         for i in range(1,self.fanout+1):
33             agg = self.addSwitch('a{}'.format(i))
34             #Add link from core to agg
35             self.addLink( agg,core, **linkopts1)
36
37             #Add Edge switches
38             for i in range(1,self.fanout+1):
39                 self.edge_count += 1
40                 edge = self.addSwitch('e{}'.format(self.edge_count))
41                 self.addLink(agg, edge, **linkopts2)
42                 #Add hosts
43                 for i in range(1,self.fanout+1):
44                     self.host_count += 1
45
46 def perfTest():
47     "Create network and run simple performance test"
48     topo = mytopo(linkopts1,linkopts2,linkopts3)
49     net = Mininet(topo=topo, host=CPULimitedHost, link=TCLink)
50     net.start()
51     print("Dumping host connections")
52     dumpNodeConnections(net.hosts)
53     print("Testing network connectivity")
54     net.pingAll()
55     print("Testing bandwidth between h1 and h4")
56     h1, h4 = net.get("h1", "h4")
57     net.iperf((h1, h4))
58     net.stop()
59
60
61 linkopts1 = dict(bw=100, delay="5ms", loss=10, max_queue_size=1000, use_htb=True)
62 linkopts2 = dict(bw=50, delay="5ms", loss=10, max_queue_size=1000, use_htb=True)
63 linkopts3 = dict(bw=10, delay="5ms", loss=10, max_queue_size=1000, use_htb=True)
64 topos = { "custom": ( lambda: CustomTopo(linkopts1,linkopts2,linkopts3) ) }

```

Listing 1: Python example

8 Problem 8

```

1  from pox.core import core

```

```

2 import pox.openflow.libopenflow_01 as of
3 from pox.lib.revent import *
4 from pox.lib.util import dpidToStr
5 from pox.lib.addresses import EthAddr
6 from collections import namedtuple
7 import os
8 import csv
9 """ Add your imports here ... """
10
11
12 log = core.getLogger()
13 policyFile = f"{os.environ['HOME']}/pox/pox/misc/firewall-policies.csv"
14 ### Add global variables and data preprocessing here ###
15
16 class Firewall(EventMixin):
17     def __init__(self):
18         self.listenTo(core.openflow)
19         log.debug("Enabling Firewall Module")
20         core.openflow.addListenerByName("ConnectionUp", self._handle_ConnectionUp)
21         with open(policyFile, 'r') as f:
22             reader = csv.reader(f)
23             next(reader)
24             self.policies = list(reader)
25
26         self.policies_set = set()
27
28         for i in self.policies:
29             self.policies_set.update([((EthAddr(i[1]), EthAddr(i[2]))))]
30
31
32     def _handle_ConnectionUp(self, event):
33         dpid = dpidToStr(event.dpid)
34         #Allow all traffic
35         switch = event.connection
36         match = of.ofp_match()
37         action = of.ofp_action_output(port=of.OFPP_CONTROLLER)
38         flow_mod = of.ofp_flow_mod(match=match, actions=[action])
39         switch.send(flow_mod)
40
41
42         # Set up flow rules to block pairs of MAC addresses
43         for mac in self.policies_set:
44             match = of.ofp_match(dl_src=mac[0], dl_dst=mac[1])
45             flow_mod = of.ofp_flow_mod(match=match)
46             switch.send(flow_mod)
47
48     def launch():
49         # start Firewall module
50         core.registerNew(Firewall)

```

Listing 2: Python example

9 Problem 9

9.1 Implement the topology depicted in Figure 5 in topo.py.

```

1 from mininet.topo import Topo
2
3 class Q9Topo(Topo):
4     def _init_(self, **opts):
5         # Initialize topology and default options
6         Topo._init_(self, **opts)
7
8         # Add switches
9         s12 = self.addSwitch('s12')
10        s14 = self.addSwitch('s14')
11        s18 = self.addSwitch('s18')
12        s16 = self.addSwitch('s16')
13        s11 = self.addSwitch('s11')

```

```

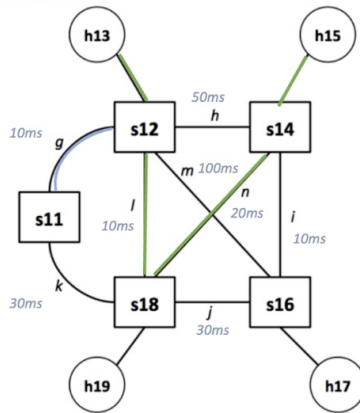
14
15 # Add hosts
16 h13 = self.addHost('h13', delay='50ms', bw=10, max_queue_size=1000, use_htb=True)
17 h15 = self.addHost('h15', delay='50ms', bw=10, max_queue_size=1000, use_htb=True)
18 h17 = self.addHost('h17', delay='50ms', bw=10, max_queue_size=1000, use_htb=True)
19 h19 = self.addHost('h19', delay='50ms', bw=10, max_queue_size=1000, use_htb=True)
20
21 # Add links
22 self.addLink(s12, s16, delay='100ms', link_name='link_m')
23 self.addLink(s14, s18, delay='20ms', link_name='link_n')
24 self.addLink(s12, s14, delay='50ms', link_name='link_h')
25 self.addLink(s14, s16, delay='10ms', link_name='link_l')
26 self.addLink(s16, s18, delay='30ms', link_name='link_j')
27 self.addLink(s18, s12, delay='10ms', link_name='link_l')
28 self.addLink(s12, s11, delay='10ms', link_name='link_g')
29 self.addLink(s11, s18, delay='30ms', link_name='link_k')
30
31 # Add host-switch links
32 self.addLink(s12, h13, bw=10, max_queue_size=1000, use_htb=True, link_name='link_s12_h13')
33 self.addLink(s14, h15, bw=10, max_queue_size=1000, use_htb=True, link_name='link_s14_h15')
34 self.addLink(s16, h17, bw=10, max_queue_size=1000, use_htb=True, link_name='link_s16_h17')
35 self.addLink(s18, h19, bw=10, max_queue_size=1000, use_htb=True, link_name='link_s18_h19')
36
37 topos = {"custom": (lambda: Q9Topo())}

```

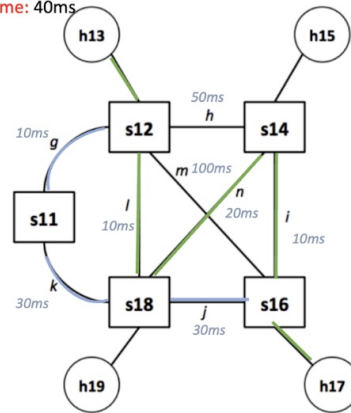
Listing 3: Python example

9.2 Factoring in the delays shown for each link, manually determine the “shortest” paths (i.e., those with the least delay) for each pair of hosts in the topology. Which nodes are in each shortest path and how much time does each path require? Note: use Dijkstra’s algorithm.

h13-h15
Route: h13→s12→s18(10ms)→s14(20ms)→h15
Total time: 30ms



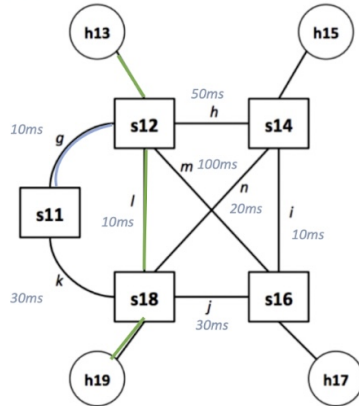
Selected path
Other paths explored via Dijkstra’s algorithm
h13-h17
Route: h13→s12→s18(10ms)→s14(20ms)→s16(10ms)→h17
Total time: 40ms



h13-h19

Route: h13→s12→s18(10ms)→h19

Total time: 10ms



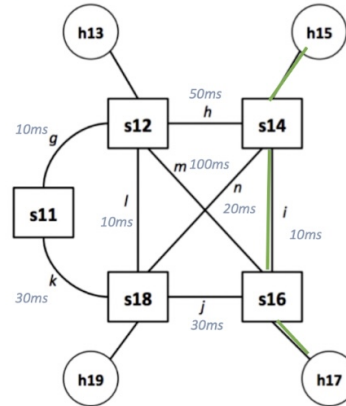
Selected path

Other paths explored via Dijkstra's algorithm

h15-h17

Route: h15→s14→s16(10ms)→h17

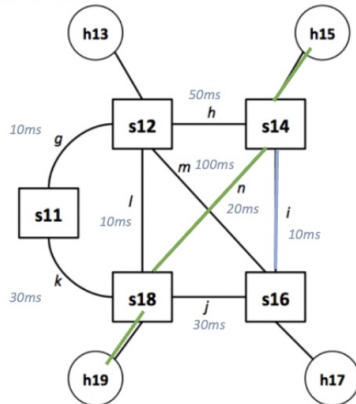
Total time: 10ms



h15-h19

Route: h15→s14→s18(20ms)→h19

Total time: 20ms



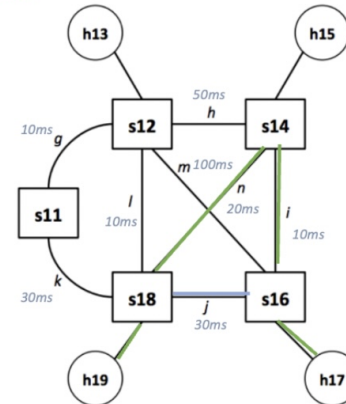
Selected path

Other paths explored via Dijkstra's algorithm

h17-h19

Route: h17→s16→s14(10ms)→s18(20ms)→h19

Total time: 30ms



(a) Quickest path for each node pair

9.3 Write pseudocode for your controller using Dijkstra's algorithm to find the shortest paths between all pairs of nodes given a topology and link delay table. You may assume that a dictionary mapping link labels to delay values exists (such as in delay.csv).

```
1 #from copy import deepcopy
2 import os
3 import csv
4 from pox.core import core
5 import pox.openflow.libopenflow_01 as of
6 from pox.lib.revent import *
7 from pox.lib.util import dpidToStr
8 from pox.lib.addresses import EthAddr
9 from pox.lib.addresses import IPAddr
10 from collections import namedtuple
11 from collections import defaultdict
12
13 log = core.getLogger()
14 delayFile = "delay.csv"
```



```

15
16 #List hosts and which switch they connect to
17 hosts = {'h13': 's12', 'h15': 's14', 'h17': 's16', 'h19': 's18'}
18
19 #Generate dictionary of nodes connected to links in csv
20 link_to_node = {"g": ("s11", "s12"), "h": ("s12", "s14"),
21                 "i": ("s14", "s16"), "j": ("s16", "s18"), "k": ("s11", "s18"),
22                 "l": ("s12", "s18"), "m": ("s12", "s16"), "n": ("s14", "s18")}
23
24 #generate mapping of connections
25 switch_ports = {'s11': {'s12': 1, 's18': 2}, 's12': {'h13': 1, 's11': 2, 's14': 3, 's18': 4, 's16':
26                 5},
27                 'h13': {'s12': 0}, 's14': {'h15': 1, 's12': 2, 's16': 3, 's18': 4}, 'h15': {'s14': 0},
28                 's16': {'h17': 1, 's14': 2, 's18': 3, 's12': 4}, 'h17': {'s16': 0}, 's18': {'h19': 1, 's16': 2,
29                 's11': 3, 's12': 4, 's14': 5},
30                 'h19': {'s18': 0},
31 }
32
33 #Get connections from mininet
34 hostMappings = {
35     'h13': ('10.0.0.1', '00:00:00:00:00:01'),
36     'h15': ('10.0.0.2', '00:00:00:00:00:02'),
37     'h17': ('10.0.0.3', '00:00:00:00:00:03'),
38     'h19': ('10.0.0.4', '00:00:00:00:00:04'),
39 }
40
41 #Initiate dicts for storing values
42 delays = {}
43 switches = []
44 node_n = defaultdict(set)
45
46 #Open and store values from csv
47 with open(delayFile, 'r') as csvfile:
48     csvreader = csv.reader(csvfile, delimiter=',')
49     next(csvreader)
50     for link, delay in csvreader:
51         s1, s2 = link_to_node[link]
52
53         node_n[s1].add(s2)
54         node_n[s2].add(s1)
55
56         delays[(s1, s2)] = int(delay)
57         delays[(s2, s1)] = int(delay)
58
59         switches.append(s1)
60         switches.append(s2)
61
62 class Dijkstra(EventMixin):
63
64     def __init__(self):
65         self.listenTo(core.openflow)
66         log.debug("Enabling Dijkstra Module")
67
68     def _dijkstra(self, source):
69         to_check = switches.copy()
70         dist = defaultdict(lambda: float('inf'))
71         dist[source] = 0
72         prev = {}
73
74         while to_check != []:
75             minDist = float('inf')
76             close = min(to_check, key=lambda x: dist[x])
77             to_check.remove(close)
78
79             for node in node_n[close]:
80                 alt = dist[close] + delays[(close, node)]
81                 if alt <= dist[node]:
82                     dist[node] = alt
83                     prev[node] = close
84
85         return dist, prev

```

```

85
86 def _getPortMapping(self, source):
87     dist, prev = self._dijkstra(source)
88     ports = {}
89
90     for dest_H, dest_S in hosts.items():
91
92         if source == dest_S:
93             ports[dest_H] = switch_ports[source][dest_H]
94             continue
95
96         while source != prev[dest_S]:
97             dest_S = prev[dest_S]
98             ports[dest_H] = switch_ports[source][dest_S]
99
100     return ports
101
102 def _handle_ConnectionUp(self, event):
103     switch = 's' + str(event.dpid)
104     ports = self._getPortMapping(switch)
105
106     for host, (ip, mac) in hostMappings.iteritems():
107         port = ports[host]
108         msg_mac = of.ofp_flow_mod()
109         msg_mac.match.dl_dst = EthAddr(mac)
110         msg_mac.actions.append(of.ofp_action_output(port=port))
111         event.connection.send(msg_mac)
112         msg_ip = of.ofp_flow_mod()
113         msg_ip.match.nw_dst = IPAddr(ip)
114         msg_ip.match.dl_type = 2054
115         msg_ip.actions.append(of.ofp_action_output(port=port))
116         event.connection.send(msg_ip)
117
118     log.debug("Dijkstra installed on %s", dpidToStr(event.dpid))
119
120
121
122 def launch():
123     '''
124     Starting the Dijkstra module
125     '''
126     core.registerNew(Dijkstra)

```

Listing 4: Psuedo example

This code hardcodes the network topology of all the links and their neighboring hosts and switches. It then uses the delay.csv file to apply delays to each of those links and recursively runs destination/source nodes through Dijkstra's algorithm until it converges to the shortest path for each node.

9.4 Implement your code in dijkstra.py. Notice that the filename delay.csv is assigned to the variable delayFile. You may assume a static topology shape and hard-code link labels (e.g., g, h, i, etc.), but do not hard-code the delays: your implementation must be general enough to function properly even with different delay values.

****Done in python file****