

SDN Experiment 3

计算机75 姚杰 2174311698

Introduction

In this experiment we created two different methods of changing paths of packets delivery dynamically. In Part 1 we will introduce how to voluntarily change the path by time, while in part 2 we shall introduce how to change the path passively when the whole links disabled partially.

Environment

Operating System: Linux version 4.15.0-20-generic

RYU Controller: 4.30 version

Mininet: 2.3.0d4 version

Method

Part 1: Changing the path by time

Basic understanding

The flow of OpenFlow has two attributes, `idle_timeout` and `hard_timeout`. `Idle_timeout` means this flow will be destroyed after several seconds it does not match any packet. `Hard_timeout` means this flow will be destroyed after several seconds of installing. In the description of part 1, we are required to change the path in every 5 seconds. Therefore, we shall set `idle_timeout` to 0 and `hard_timeout` to 5.

Modify the shortest path function

Another question we should concern about is how to find the longest path. Since Dijkstra algorithm has been included in the original code, only a few changes shall be made to make it calculate the longest path. There are two ways:

1. Every time the algorithm chooses the shortest path, make it choose the longest.
2. Change the weight of every edge from 1 to -1.

In the code I chose the latter solution.

Add switch-mode condition in the packet in handler

To begin with, the switch will send trigger switch feature handler and then a basic flow whose match field is empty will be installed into the switch (This flow is significant in part 2). And then, when next new packet handler arrives, the switch will send a message to the controller and trigger packet in handler. That's when the get shortest path function and install path function will work.

Therefore, we can add a condition that the get shortest path function will be converted every **two** times the install function works. Two times is every important because the install function will install paths from destination to source and source to destination, which need to be the same.

The final result shall be as follows:

```
Terminal - test@sdnexp: ~/sdn/exp3
loading app ryu.controller.ofp_handler
creating context wsgi
instantiating app None of Network_Monitor
creating context Network_Monitor
instantiating app dynamic_rules.py of dynamic_rules
instantiating app ryu.topology.switches of Switches
instantiating app ryu.controller.ofp_handler of OFPHandler
(27110) wsgi starting up on http://0.0.0.0:8080
Pathmod: Shortest
path : [(1, 1), (1, 3), (4, 1), (4, 2), (5, 1), (5, 3)]
time_install: 2020-05-19 11:11:55.336480
Pathmod: Shortest
path : [(5, 3), (5, 1), (4, 2), (4, 1), (1, 3), (1, 1)]
time_install: 2020-05-19 11:11:56.344134
Pathmod: Longest
path : [(1, 1), (1, 2), (2, 1), (2, 2), (3, 1), (3, 2), (5, 2), (5, 3)]
time_install: 2020-05-19 11:12:00.441364
Pathmod: Longest
path : [(5, 3), (5, 2), (3, 2), (3, 1), (2, 2), (2, 1), (1, 2), (1, 1)]
time_install: 2020-05-19 11:12:02.491060
Pathmod: Shortest
path : [(1, 1), (1, 3), (4, 1), (4, 2), (5, 1), (5, 3)]
time_install: 2020-05-19 11:12:05.561115
```

```
Terminal - test@sdnexp: ~/sdn/exp3
path : [(1, 1), (1, 3), (4, 1), (4, 2), (5, 1), (5, 3)]
time_install: 2020-05-19 11:11:55.336480
Pathmod: Shortest
path : [(5, 3), (5, 1), (4, 2), (4, 1), (1, 3), (1, 1)]
time_install: 2020-05-19 11:11:56.344134
Pathmod: Longest
path : [(1, 1), (1, 2), (2, 1), (2, 2), (3, 1), (3, 2), (5, 2), (5, 3)]
time_install: 2020-05-19 11:12:00.441364
Pathmod: Longest
path : [(5, 3), (5, 2), (3, 2), (3, 1), (2, 2), (2, 1), (1, 2), (1, 1)]
time_install: 2020-05-19 11:12:02.491060
Pathmod: Shortest
path : [(1, 1), (1, 3), (4, 1), (4, 2), (5, 1), (5, 3)]
time_install: 2020-05-19 11:12:05.561115
Pathmod: Shortest
path : [(5, 3), (5, 1), (4, 2), (4, 1), (1, 3), (1, 1)]
time_install: 2020-05-19 11:12:07.614336
Pathmod: Longest
path : [(1, 1), (1, 2), (2, 1), (2, 2), (3, 1), (3, 2), (5, 2), (5, 3)]
time_install: 2020-05-19 11:12:10.680400
Pathmod: Longest
path : [(5, 3), (5, 2), (3, 2), (3, 1), (2, 2), (2, 1), (1, 2), (1, 1)]
time_install: 2020-05-19 11:12:12.728272
```

You can see that every 5 seconds a new path will be implemented and the path mode will be converted. The path is bidirectional so there will be two path changes at the same time. One of them is the path from source to destination, another is from destination to source. They have a slight delay, which is roughly 2 seconds.

Part 2: Disaster tolerance (Malfunction Recovery)

Basic understanding

A disaster tolerant controller shall switch to another feasible path with highest priority when the current path is malfunctioned due to some errors in the path. So every time changes happen in the topology, we will get its information and calculate another feasible path. After that, we will change the flows of the old path and install flows of the new path.

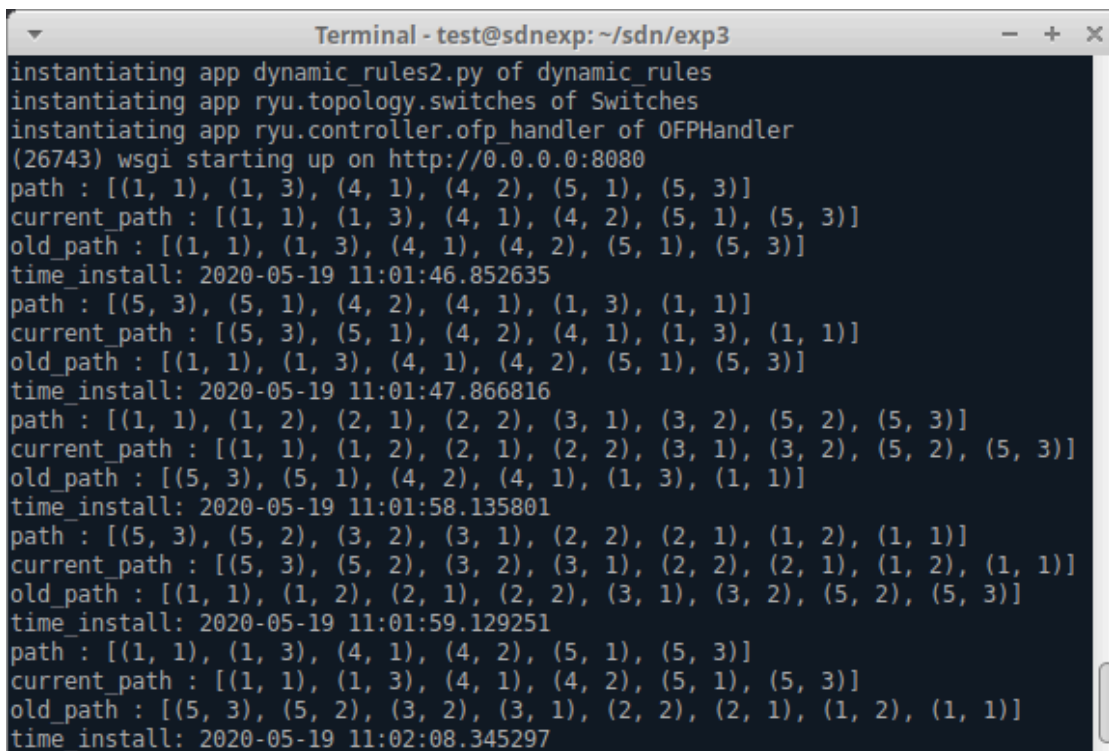
Flow changes of switches on the old path

The information of how to delete a flow is not enough in the internet. I have spent time in searching this yet the result is not satisfactory. Actually when we'd like to change the path, a certain flow is highly preferred to be deleted, yet the in port and out port are varied. It's hard to locate them. Therefore, I decided to delete all flows on the switches of the old path and right after that, install a basic flow whose match field is empty and will send a packet in message to the controller (the same to the flow the switch feature handler will install).

Getting the shortest path

Actually, when a port is aborted or a link is down, get topology handler will be triggered. Therefore, I did not use get port status message function. Instead, I add get shortest path function into the get topology function, so every time the topology change, it will calculate the newest shortest path right away.

The final result is as follows:

A terminal window titled "Terminal - test@sdnexp: ~/sdn/exp3" displays a series of log messages. The messages show the instantiation of various applications and the calculation of shortest paths at different times. Each calculation includes a 'path', 'current_path', 'old_path', and 'time_install' timestamp. The paths are represented as lists of coordinate pairs (x, y).

```
instantiating app dynamic_rules2.py of dynamic_rules
instantiating app ryu.topology.switches of Switches
instantiating app ryu.controller.ofp_handler of OFPHandler
(26743) wsgi starting up on http://0.0.0.0:8080
path : [(1, 1), (1, 3), (4, 1), (4, 2), (5, 1), (5, 3)]
current_path : [(1, 1), (1, 3), (4, 1), (4, 2), (5, 1), (5, 3)]
old_path : [(1, 1), (1, 3), (4, 1), (4, 2), (5, 1), (5, 3)]
time_install: 2020-05-19 11:01:46.852635
path : [(5, 3), (5, 1), (4, 2), (4, 1), (1, 3), (1, 1)]
current_path : [(5, 3), (5, 1), (4, 2), (4, 1), (1, 3), (1, 1)]
old_path : [(1, 1), (1, 3), (4, 1), (4, 2), (5, 1), (5, 3)]
time_install: 2020-05-19 11:01:47.866816
path : [(1, 1), (1, 2), (2, 1), (2, 2), (3, 1), (3, 2), (5, 2), (5, 3)]
current_path : [(1, 1), (1, 2), (2, 1), (2, 2), (3, 1), (3, 2), (5, 2), (5, 3)]
old_path : [(5, 3), (5, 1), (4, 2), (4, 1), (1, 3), (1, 1)]
time_install: 2020-05-19 11:01:58.135801
path : [(5, 3), (5, 2), (3, 2), (3, 1), (2, 2), (2, 1), (1, 2), (1, 1)]
current_path : [(5, 3), (5, 2), (3, 2), (3, 1), (2, 2), (2, 1), (1, 2), (1, 1)]
old_path : [(1, 1), (1, 2), (2, 1), (2, 2), (3, 1), (3, 2), (5, 2), (5, 3)]
time_install: 2020-05-19 11:01:59.129251
path : [(1, 1), (1, 3), (4, 1), (4, 2), (5, 1), (5, 3)]
current_path : [(1, 1), (1, 3), (4, 1), (4, 2), (5, 1), (5, 3)]
old_path : [(5, 3), (5, 2), (3, 2), (3, 1), (2, 2), (2, 1), (1, 2), (1, 1)]
time_install: 2020-05-19 11:02:08.345297
```

```
Terminal - test@sdnexp: ~/sdn/exp3
current_path : [(5, 3), (5, 1), (4, 2), (4, 1), (1, 3), (1, 1)]
old_path : [(1, 1), (1, 3), (4, 1), (4, 2), (5, 1), (5, 3)]
time_install: 2020-05-19 11:01:47.866816
path : [(1, 1), (1, 2), (2, 1), (2, 2), (3, 1), (3, 2), (5, 2), (5, 3)]
current_path : [(1, 1), (1, 2), (2, 1), (2, 2), (3, 1), (3, 2), (5, 2), (5, 3)]
old_path : [(5, 3), (5, 1), (4, 2), (4, 1), (1, 3), (1, 1)]
time_install: 2020-05-19 11:01:58.135801
path : [(5, 3), (5, 2), (3, 2), (3, 1), (2, 2), (2, 1), (1, 2), (1, 1)]
current_path : [(5, 3), (5, 2), (3, 2), (3, 1), (2, 2), (2, 1), (1, 2), (1, 1)]
old_path : [(1, 1), (1, 2), (2, 1), (2, 2), (3, 1), (3, 2), (5, 2), (5, 3)]
time_install: 2020-05-19 11:01:59.129251
path : [(1, 1), (1, 3), (4, 1), (4, 2), (5, 1), (5, 3)]
current_path : [(1, 1), (1, 3), (4, 1), (4, 2), (5, 1), (5, 3)]
old_path : [(5, 3), (5, 2), (3, 2), (3, 1), (2, 2), (2, 1), (1, 2), (1, 1)]
time_install: 2020-05-19 11:02:08.345297
path : [(1, 1), (1, 3), (4, 1), (4, 2), (5, 1), (5, 3)]
current_path : [(1, 1), (1, 3), (4, 1), (4, 2), (5, 1), (5, 3)]
old_path : [(1, 1), (1, 3), (4, 1), (4, 2), (5, 1), (5, 3)]
time_install: 2020-05-19 11:02:09.370632
path : [(5, 3), (5, 1), (4, 2), (4, 1), (1, 3), (1, 1)]
current_path : [(5, 3), (5, 1), (4, 2), (4, 1), (1, 3), (1, 1)]
old_path : [(1, 1), (1, 3), (4, 1), (4, 2), (5, 1), (5, 3)]
time_install: 2020-05-19 11:02:10.391672
```

You can see that at the beginning the path is "1-4-5", and then when link "s1-s4" is down, the path changes to "1-2-3-5". After the link is up online, the path changes to "1-4-5" again. In this process, we achieved the disaster tolerance.

Let's take a closer look.

```
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Unable to contact the remote controller at 127.0.0.1:6633
Setting remote controller to 127.0.0.1:6653
*** Adding hosts:
h1 h2
*** Adding switches:
s1 s2 s3 s4 s5
*** Adding links:
(h1, s1) (s1, s2) (s1, s4) (s2, s3) (s3, s5) (s4, s5) (s5, h2)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 5 switches
s1 s2 s3 s4 s5 ...
*** Starting CLI:
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.424 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.083 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.096 ms
^C
--- 10.0.0.2 ping statistics ---
5 packets transmitted, 3 received, 40% packet loss, time 4081ms
rtt min/avg/max/mdev = 0.083/0.201/0.424/0.157 ms
mininet> link s1 s4 down
mininet> dpctl dump-flows
*** s1 ***
cookie=0x0, duration=9.319s, table=0, n_packets=11, n_bytes=803, priority=0 actions=CONTROLLER:65535
*** s2 ***
cookie=0x0, duration=21.148s, table=0, n_packets=44, n_bytes=2640, priority=65535,dl_dst=01:80:c2:00:00:0e,dl_type=0x88cc actions=CONTROLLER:65535
cookie=0x0, duration=21.158s, table=0, n_packets=23, n_bytes=3478, priority=0 actions=CONTROLLER:65535
*** s3 ***
cookie=0x0, duration=21.164s, table=0, n_packets=48, n_bytes=2880, priority=65535,dl_dst=01:80:c2:00:00:0e,dl_type=0x88cc actions=CONTROLLER:65535
cookie=0x0, duration=21.195s, table=0, n_packets=21, n_bytes=3167, priority=0 actions=CONTROLLER:65535
*** s4 ***
cookie=0x0, duration=9.338s, table=0, n_packets=11, n_bytes=803, priority=0 actions=CONTROLLER:65535
*** s5 ***
cookie=0x0, duration=9.344s, table=0, n_packets=23, n_bytes=1676, priority=0 actions=CONTROLLER:65535
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.487 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.107 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.107 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.087 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.087 ms
```

To begin with, the ping process is successful. After we shut down the link from s1 to s4, the controller will delete specific flows on the old path, which is s1, s4 and s5. Next time when we restart to ping, the packet will trigger the packet in handler again, and thus the controller will start to calculate the newest and shortest path and install

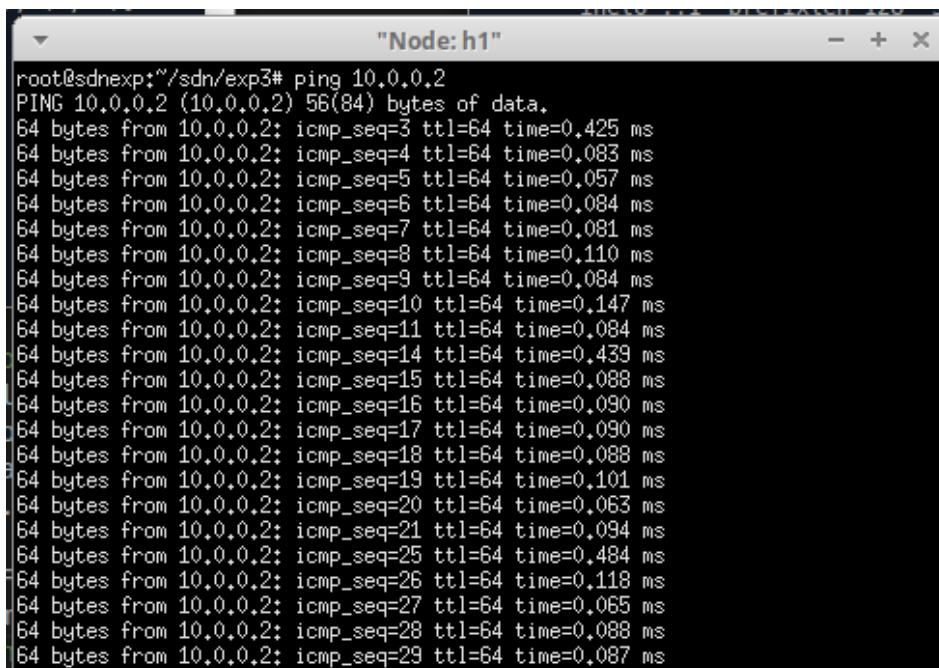
new flows.

See this:

```
8 packets transmitted, 6 received, 25% packet loss, time 7163ms
rtt min/avg/max/mdev = 0.087/0.163/0.487/0.145 ms
mininet> dpctl dump-flows
*** s1 ***
cookie=0x0, duration=56.095s, table=0, n_packets=7, n_bytes=686, priority=100,ip,in_port="s1-eth1",dl_src=fe:b0:d5:18:43:2c,dl_dst=0e:80:d5:88:c2:4e,n
h2"
cookie=0x0, duration=55.083s, table=0, n_packets=6, n_bytes=588, priority=100,ip,in_port="s1-eth2",dl_src=0e:80:d5:88:c2:4e,dl_dst=fe:b0:d5:18:43:2c,n
:b0:d5:18:43:2c,output:"s1-eth1"
cookie=0x0, duration=75.778s, table=0, n_packets=93, n_bytes=6069, priority=0 actions=CONTROLLER:65535
*** s2 ***
cookie=0x0, duration=87.611s, table=0, n_packets=192, n_bytes=11520, priority=65535,dl_dst=01:80:c2:00:00:0e,dl_type=0x88cc actions=CONTROLLER:65535
cookie=0x0, duration=56.105s, table=0, n_packets=7, n_bytes=686, priority=100,ip,in_port="s2-eth1",dl_src=fe:b0:d5:18:43:2c,dl_dst=0e:80:d5:88:c2:4e,n
h2"
cookie=0x0, duration=55.092s, table=0, n_packets=6, n_bytes=588, priority=100,ip,in_port="s2-eth2",dl_src=0e:80:d5:88:c2:4e,dl_dst=fe:b0:d5:18:43:2c,n
h1"
cookie=0x0, duration=87.621s, table=0, n_packets=31, n_bytes=4570, priority=0 actions=CONTROLLER:65535
*** s3 ***
cookie=0x0, duration=87.622s, table=0, n_packets=196, n_bytes=11760, priority=65535,dl_dst=01:80:c2:00:00:0e,dl_type=0x88cc actions=CONTROLLER:65535
cookie=0x0, duration=56.110s, table=0, n_packets=7, n_bytes=686, priority=100,ip,in_port="s3-eth1",dl_src=fe:b0:d5:18:43:2c,dl_dst=0e:80:d5:88:c2:4e,n
h2"
cookie=0x0, duration=55.096s, table=0, n_packets=6, n_bytes=588, priority=100,ip,in_port="s3-eth2",dl_src=0e:80:d5:88:c2:4e,dl_dst=fe:b0:d5:18:43:2c,n
h1"
cookie=0x0, duration=87.653s, table=0, n_packets=29, n_bytes=4259, priority=0 actions=CONTROLLER:65535
*** s4 ***
cookie=0x0, duration=75.798s, table=0, n_packets=89, n_bytes=5789, priority=0 actions=CONTROLLER:65535
*** s5 ***
cookie=0x0, duration=56.121s, table=0, n_packets=7, n_bytes=686, priority=100,ip,in_port="s5-eth2",dl_src=fe:b0:d5:18:43:2c,dl_dst=0e:80:d5:88:c2:4e,n
:80:d5:88:c2:4e,output:"s5-eth3"
cookie=0x0, duration=55.105s, table=0, n_packets=6, n_bytes=588, priority=100,ip,in_port="s5-eth3",dl_src=0e:80:d5:88:c2:4e,dl_dst=fe:b0:d5:18:43:2c,n
h2"
cookie=0x0, duration=75.802s, table=0, n_packets=182, n_bytes=11868, priority=0 actions=CONTROLLER:65535
mininet> link s1 s4 up
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.442 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.082 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.083 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.081 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.100 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.191 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.115 ms
^C
--- 10.0.0.2 ping statistics ---
9 packets transmitted, 7 received, 22% packet loss, time 8191ms
rtt min/avg/max/mdev = 0.081/0.156/0.442/0.122 ms
mininet> xterm h1
mininet> |
```

You can see that the flows have been installed on s1, s2, s3, s5.

Or you can ping h2 in the terminal of h1. Note that h1 does not know what is "h2". You should ping 10.0.0.2 instead of "h2".



```
root@sdnexp:~/sdn/exp3# ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.425 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.083 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.057 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.084 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.081 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.110 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.084 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=0.147 ms
64 bytes from 10.0.0.2: icmp_seq=11 ttl=64 time=0.084 ms
64 bytes from 10.0.0.2: icmp_seq=14 ttl=64 time=0.439 ms
64 bytes from 10.0.0.2: icmp_seq=15 ttl=64 time=0.088 ms
64 bytes from 10.0.0.2: icmp_seq=16 ttl=64 time=0.090 ms
64 bytes from 10.0.0.2: icmp_seq=17 ttl=64 time=0.090 ms
64 bytes from 10.0.0.2: icmp_seq=18 ttl=64 time=0.088 ms
64 bytes from 10.0.0.2: icmp_seq=19 ttl=64 time=0.101 ms
64 bytes from 10.0.0.2: icmp_seq=20 ttl=64 time=0.063 ms
64 bytes from 10.0.0.2: icmp_seq=21 ttl=64 time=0.094 ms
64 bytes from 10.0.0.2: icmp_seq=25 ttl=64 time=0.484 ms
64 bytes from 10.0.0.2: icmp_seq=26 ttl=64 time=0.118 ms
64 bytes from 10.0.0.2: icmp_seq=27 ttl=64 time=0.065 ms
64 bytes from 10.0.0.2: icmp_seq=28 ttl=64 time=0.088 ms
64 bytes from 10.0.0.2: icmp_seq=29 ttl=64 time=0.087 ms
```

Attention: Every time the path changes, the ping will be slow for a short while. That's because the controller needs time to get the new topology, recalculate the shortest path and install flows, during which some packets

might be lost.

Reference

[OpenFlow Switch Specification](#)

[How to delete flows](#)

Source Code

```
1 # the code of the topology
2
3 from mininet.topo import Topo
4
5 class Mytopo(Topo):
6     def __init__(self):
7         super(Mytopo, self).__init__()
8         h1 = self.addHost('h1')
9         h2 = self.addHost('h2')
10
11         s1 = self.addSwitch('s1')
12         s2 = self.addSwitch('s2')
13         s3 = self.addSwitch('s3')
14         s4 = self.addSwitch('s4')
15         s5 = self.addSwitch('s5')
16
17         self.addLink(h1, s1)
18         self.addLink(s1, s2)
19         self.addLink(s1, s4)
20         self.addLink(s2, s3)
21         self.addLink(s4, s5)
22         self.addLink(s3, s5)
23         self.addLink(s5, h2)
24
25 topos = { 'mytopo': (lambda : Mytopo()) }
```



```
1 # the code of part 1
2
3 from ryu.base import app_manager
4 from ryu.controller import ofp_event
5 from ryu.topology import event
6 from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER, DEAD_DISPATCHER,
    HANDSHAKE_DISPATCHER
7 from ryu.controller.handler import set_ev_cls
8 from ryu.ofproto import ofproto_v1_3
9 from ryu.lib.packet import packet
10 from ryu.lib.packet import ethernet
11 from ryu.lib.packet import arp
12 from ryu.lib.packet import ipv4
13 from ryu.lib.packet import tcp
14 from ryu.topology.api import get_link
```



```

15 from ryu.lib.packet import ether_types
16 from ryu.app.wsgi import WSGIApplication
17 from collections import defaultdict
18 import network_monitor
19 import datetime
20 class dynamic_rules(app_manager.RyuApp):
21     OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]
22     _CONTEXTS = {
23         "Network_Monitor": network_monitor.Network_Monitor,
24         "wsgi": WSGIApplication
25     }
26     def __init__(self, *args, **kwargs):
27         super(dynamic_rules, self).__init__(*args, **kwargs)
28         self.mac_to_port = {}
29         #mac_to_port maps [mac of src][dpid of switch] to port
30         self.ip_to_mac = {}
31         self.mac_to_dpid = {} # {mac:(dpid,port)}
32
33         self.datapaths = defaultdict(lambda: None)
34         self.topology_api_app = self
35         self.src_links = defaultdict(lambda: defaultdict(lambda: None))
36
37         self.check_ip_dpid = defaultdict(list)
38
39         self.qos_ip_bw_list = []
40
41         self.network_monitor = kwargs["Network_Monitor"]
42
43
44         self.ip_to_switch = {}
45         self.port_name_to_num = {}
46
47         self.ip_to_port = {} #{ip:(dpid,port)}
48         #promise me, use it well :)
49         # let 0 equals to shortest_path, 1 equals to longest_path
50         self.pathmod = 0
51         self.path = None
52         self.come_and_go = 0
53
54         @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
55         def switch_features_handler(self, ev):
56             datapath = ev.msg.datapath
57             ofproto = datapath.ofproto
58             parser = datapath.ofproto_parser
59             match = parser.OFPMatch()
60             actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
61                                             ofproto.OFPCML_NO_BUFFER)]
62             self.add_flow(datapath, 0, match, actions)
63
64         def add_flow(self, datapath, priority, match, actions, buffer_id=None, idle_timeout=0,
65                     hard_timeout=0):
66             ofproto = datapath.ofproto
67             parser = datapath.ofproto_parser
68
69             inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,

```

```

70         if buffer_id:
71             mod = parser.OFPFlowMod(datapath=datapath, buffer_id=buffer_id,
72                                     priority=priority, match=match,
73                                     idle_timeout=idle_timeout,
74                                     hard_timeout=hard_timeout,
75                                     instructions=inst)
76         else:
77             mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
78                                     idle_timeout=idle_timeout,
79                                     hard_timeout=hard_timeout,
80                                     match=match, instructions=inst)
81     datapath.send_msg(mod)
82
83 @set_ev_cls(ofp_event.EventOFPacketIn, MAIN_DISPATCHER)
84 def _packet_in_handler(self, ev):
85
86     if ev.msg.msg_len < ev.msg.total_len:
87         self.logger.debug("packet truncated: only %s of %s bytes",
88                             ev.msg.msg_len, ev.msg.total_len)
89     msg = ev.msg
90     datapath = msg.datapath
91     ofproto = datapath.ofproto
92     parser = datapath.ofproto_parser
93     in_port = msg.match['in_port']
94
95     pkt = packet.Packet(msg.data)
96     eth = pkt.get_protocols(ethernet.ethernet)[0]
97     pkt_arp = pkt.get_protocol(arp.arp)
98     pkt_ipv4 = pkt.get_protocol(ipv4.ipv4)
99     pkt_tcp = pkt.get_protocol(tcp.tcp)
100
101     if eth.ethertype == ether_types.ETH_TYPE_LLDP:
102         return
103     if eth.ethertype == ether_types.ETH_TYPE_IPV6:
104         return
105
106     dst = eth.dst
107     src = eth.src
108     dpid = datapath.id
109
110     # self.logger.info("packet in %s %s %s %s", dpid, src, dst, in_port)
111
112     # in rest_topology, self.mac_to_port is for the find for host
113     self.mac_to_port.setdefault(dpid, {})
114     self.mac_to_port[dpid][src] = in_port
115
116     # arp handle
117     if pkt_arp and pkt_arp.opcode == arp.ARP_REQUEST:
118         if pkt_arp.src_ip not in self.ip_to_mac:
119             self.ip_to_mac[pkt_arp.src_ip] = src
120             self.mac_to_dpid[src] = (dpid, in_port)
121             self.ip_to_port[pkt_arp.src_ip] = (dpid, in_port)
122
123         if pkt_arp.dst_ip in self.ip_to_mac:
124             #self.logger.info("[PACKET] ARP packet_in.")
125             self.handle_arpre(datapath=datapath, port=in_port,

```



```

126         src_mac=self.ip_to_mac[pkt_arp.dst_ip],
127         dst_mac=src, src_ip=pkt_arp.dst_ip, dst_ip=pkt_arp.src_ip)
128     else:
129         # to avoid flood when the dst ip not in the network
130         if datapath.id not in self.check_ip_dpid[pkt_arp.dst_ip]:
131             self.check_ip_dpid[pkt_arp.dst_ip].append(datapath.id)
132             out_port = ofproto.OFPP_FLOOD
133             actions = [parser.OFPAActionOutput(out_port)]
134             data = None
135             if msg.buffer_id == ofproto.OFP_NO_BUFFER:
136                 data = msg.data
137             out = parser.OFPPacketOut(datapath=datapath, buffer_id=msg.buffer_id,
138                                     in_port=in_port, actions=actions, data=data)
139             datapath.send_msg(out)
140         return
141
142     elif pkt_arp and pkt_arp.opcode == arp.ARP_REPLY:
143         if pkt_arp.src_ip not in self.ip_to_mac:
144             self.ip_to_mac[pkt_arp.src_ip] = src
145             self.mac_to_dpid[src] = (dpid, in_port)
146             self.ip_to_port[pkt_arp.src_ip] = (dpid, in_port)
147         dst_mac = self.ip_to_mac[pkt_arp.dst_ip]
148         (dst_dpid, dst_port) = self.mac_to_dpid[dst_mac]
149         self.handle_arpre(datapath=self.datapaths[dst_dpid], port=dst_port, src_mac=src,
dst_mac=dst_mac,
150                          src_ip=pkt_arp.src_ip, dst_ip=pkt_arp.dst_ip)
151         return
152
153         if pkt_ipv4 and (self.ip_to_port.get(pkt_ipv4.dst)) and
(self.ip_to_port.get(pkt_ipv4.src)):
154             (src_dpid, src_port) = self.ip_to_port[pkt_ipv4.src] # src dpid and port
155             (dst_dpid, dst_port) = self.ip_to_port[pkt_ipv4.dst] # dst dpid and port
156             self.install_path(src_dpid=src_dpid, dst_dpid=dst_dpid, src_port=src_port,
dst_port=dst_port,
157                             ev=ev, src=src, dst=dst, pkt_ipv4=pkt_ipv4, pkt_tcp=pkt_tcp)
158             self.come_and_go = self.come_and_go + 1
159             if self.come_and_go == 2:
160                 self.come_and_go = 0
161                 self.pathmod = not self.pathmod
162
163
164     def send_pkt(self, datapath, port, pkt):
165         ofproto = datapath.ofproto
166         parser = datapath.ofproto_parser
167         pkt.serialize()
168         data = pkt.data
169         actions = [parser.OFPAActionOutput(port=port)]
170         out = parser.OFPPacketOut(datapath=datapath, buffer_id=ofproto.OFP_NO_BUFFER,
in_port=ofproto.OFPP_CONTROLLER,
171                                  actions=actions, data=data)
172         datapath.send_msg(out)
173
174     def handle_arpre(self, datapath, port, src_mac, dst_mac, src_ip, dst_ip):
175         pkt = packet.Packet()
176         pkt.add_protocol(ethernet.ethernet(ethertype=0x0806, dst=dst_mac, src=src_mac))
177         pkt.add_protocol(arp.arp(opcode=arp.ARP_REPLY, src_mac=src_mac, src_ip=src_ip,

```

```

dst_mac=dst_mac, dst_ip=dst_ip))
178         self.send_pkt(datapath, port, pkt)
179
180     def install_path(self, src_dpid, dst_dpid, src_port, dst_port, ev, src, dst, pkt_ipv4,
pkt_tcp):
181         msg = ev.msg
182         datapath = msg.datapath
183         ofproto = datapath.ofproto
184         parser = datapath.ofproto_parser
185
186         mid_path = None
187         if self.pathmod == 0:
188             mid_path = self.short_path(src=src_dpid, dst=dst_dpid, bw=0)
189             self.logger.info('Pathmod: Shortest')
190         elif self.pathmod == 1:
191             mid_path = self.short_path(src=src_dpid, dst=dst_dpid, bw=1)
192             self.logger.info('Pathmod: Longest')
193
194         if mid_path is None:
195             return
196         self.path = None
197         self.path = [(src_dpid, src_port)] + mid_path + [(dst_dpid, dst_port)]
198
199         self.logger.info("path : %s", str(self.path))
200
201         for i in xrange(len(self.path) - 2, -1, -2):
202             datapath_path = self.datapaths[self.path[i][0]]
203             match = parser.OFPMatch(in_port=self.path[i][1], eth_src=src, eth_dst=dst,
eth_type=0x0800,
204                                     ipv4_src=pkt_ipv4.src, ipv4_dst=pkt_ipv4.dst)
205
206             if i < (len(self.path) - 2):
207                 actions = [parser.OFPACTIONOutput(self.path[i + 1][1])]
208             else:
209                 actions =
[parser.OFPACTIONSetField(eth_dst=self.ip_to_mac.get(pkt_ipv4.dst)),
210                             parser.OFPACTIONOutput(self.path[i + 1][1])]
211
212             self.add_flow(datapath_path, 100, match, actions, idle_timeout=0,
hard_timeout=5)
213             time_install = datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f')
214             self.logger.info("time_install: %s", time_install)
215
216
217     def short_path(self, src, dst, bw):
218         if src == dst:
219             return []
220         result = defaultdict(lambda: defaultdict(lambda: None))
221         distance = defaultdict(lambda: defaultdict(lambda: None))
222
223         # the node is checked
224         seen = [src]
225
226         # the distance to src
227         # w=1 means shortest path, w=-1 means longest path
228         distance[src] = 0

```

```

229         if bw==0:
230             w = 1 # weight
231         else:
232             w = -1
233
234         while len(seen) < len(self.src_links):
235             node = seen[-1]
236             if node == dst:
237                 break
238             for (temp_src, temp_dst) in self.src_links[node]:
239                 if temp_dst not in seen:
240                     temp_src_port = self.src_links[node][(temp_src, temp_dst)][0]
241                     temp_dst_port = self.src_links[node][(temp_src, temp_dst)][1]
242                     if (distance[temp_dst] is None) or (distance[temp_dst] >
distance[temp_src] + w):
243                         distance[temp_dst] = distance[temp_src] + w
244                         # result = {"dpid":(link_src, src_port, link_dst, dst_port)}
245                         result[temp_dst] = (temp_src, temp_src_port, temp_dst,
temp_dst_port)
246                     min_node = None
247                     min_path = 999
248                     # get the min_path node
249                     for temp_node in distance:
250                         if (temp_node not in seen) and (distance[temp_node] is not None):
251                             if distance[temp_node] < min_path:
252                                 min_node = temp_node
253                                 min_path = distance[temp_node]
254                     if min_node is None:
255                         break
256                     seen.append(min_node)
257
258             path = []
259
260             if dst not in result:
261                 return None
262
263             while (dst in result) and (result[dst] is not None):
264                 path = [result[dst][2:4]] + path
265                 path = [result[dst][0:2]] + path
266                 dst = result[dst][0]
267             #self.logger.info("path : %s", str(path))
268             return path
269
270 # this function might be useful, but who knows anyway
271 # def long_path(self, src, dst, bw=0):
272
273 @set_ev_cls(ofp_event.EventOFPPStateChange, [MAIN_DISPATCHER, DEAD_DISPATCHER])
274 def state_change_handler(self, ev):
275     datapath = ev.datapath
276     if ev.state == MAIN_DISPATCHER:
277         if datapath.id not in self.datapaths:
278             self.datapaths[datapath.id] = datapath
279     elif ev.state == DEAD_DISPATCHER:
280         if datapath.id in self.datapaths:
281             del self.datapaths[datapath.id]
282     #self.logger.info("datapaths : %s", self.datapaths)

```

```

283
284     @set_ev_cls([event.EventSwitchEnter, event.EventSwitchLeave, event.EventPortAdd,
event.EventPortDelete,
285                 event.EventPortModify, event.EventLinkAdd, event.EventLinkDelete])
286     def get_topology(self, ev):
287         links_list = get_link(self.topology_api_app, None)
288         self.src_links.clear()
289         for link in links_list:
290             sw_src = link.src.dpid
291             sw_dst = link.dst.dpid
292             src_port = link.src.port_no
293             dst_port = link.dst.port_no
294             src_port_name = link.src.name
295             dst_port_name = link.dst.name
296             self.port_name_to_num[src_port_name] = src_port
297             self.port_name_to_num[dst_port_name] = dst_port
298             self.src_links[sw_src][(sw_src, sw_dst)] = (src_port, dst_port)
299             self.src_links[sw_dst][(sw_dst, sw_src)] = (dst_port, src_port)
300             # self.logger.info("****src_port_name : %s", str(src_port_name))
301             # self.logger.info("src_links : %s", str(self.src_links))
302             # self.logger.info("port_name_to_num : %s", str(self.port_name_to_num))

1 # the code of part 2
2
3 from ryu.base import app_manager
4 from ryu.controller import ofp_event
5 from ryu.topology import event
6 from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER, DEAD_DISPATCHER,
HANDSHAKE_DISPATCHER
7 from ryu.controller.handler import set_ev_cls
8 from ryu.ofproto import ofproto_v1_3
9 from ryu.lib.packet import packet
10 from ryu.lib.packet import ethernet
11 from ryu.lib.packet import arp
12 from ryu.lib.packet import ipv4
13 from ryu.lib.packet import tcp
14 from ryu.topology.api import get_link
15 from ryu.lib.packet import ether_types
16 from ryu.app.wsgi import WSGIApplication
17 from collections import defaultdict
18 import network_monitor
19 import datetime
20 import time
21 class dynamic_rules(app_manager.RyuApp):
22     OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]
23     _CONTEXTS = {
24         "Network_Monitor": network_monitor.Network_Monitor,
25         "wsgi": WSGIApplication
26     }
27     def __init__(self, *args, **kwargs):
28         super(dynamic_rules, self).__init__(*args, **kwargs)
29         self.mac_to_port = {}
30         #mac_to_port maps [mac of src][dpid of switch] to port
31         self.ip_to_mac = {}
32         self.mac_to_dpid = {} # {mac:(dpid,port)}
33
34         self.datapaths = defaultdict(lambda: None)

```

```

35     self.topology_api_app = self
36     self.src_links = defaultdict(lambda: defaultdict(lambda: None))
37
38     self.check_ip_dpid = defaultdict(list)
39
40     self.qos_ip_bw_list = []
41
42     self.network_monitor = kwargs["Network_Monitor"]
43
44
45     self.ip_to_switch = {}
46     self.port_name_to_num = {}
47
48     self.ip_to_port = {}  #{ip:(dpid,port)}
49     #promise me, use it well :)
50     # let 0 equals to shortest_path, 1 equals to longest_path
51     self.pathmod = 0
52     self.path = None
53     self.old_path = None
54     self.current_path = None
55
56
57     @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
58     def switch_features_handler(self, ev):
59         datapath = ev.msg.datapath
60         ofproto = datapath.ofproto
61         parser = datapath.ofproto_parser
62         match = parser.OFPMatch()
63         actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
64                                           ofproto.OFPCML_NO_BUFFER)]
65         self.add_flow(datapath, 0, match, actions)
66
67     def add_flow(self, datapath, priority, match, actions, table_id=0, buffer_id=None,
68                 idle_timeout=0, hard_timeout=0):
69         ofproto = datapath.ofproto
70         parser = datapath.ofproto_parser
71
72         inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
73                                             actions)]
74         if buffer_id:
75             mod = parser.OFPFlowMod(datapath=datapath, table_id=table_id,
76                                     buffer_id=buffer_id,
77                                     priority=priority, match=match,
78                                     idle_timeout=idle_timeout,
79                                     hard_timeout=hard_timeout,
80                                     instructions=inst)
81         else:
82             mod = parser.OFPFlowMod(datapath=datapath, table_id=table_id, priority=priority,
83                                     idle_timeout=idle_timeout,
84                                     hard_timeout=hard_timeout,
85                                     match=match, instructions=inst)
86         datapath.send_msg(mod)
87
88     @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
89     def _packet_in_handler(self, ev):

```

```

89         if ev.msg.msg_len < ev.msg.total_len:
90             self.logger.debug("packet truncated: only %s of %s bytes",
91                               ev.msg.msg_len, ev.msg.total_len)
92         msg = ev.msg
93         datapath = msg.datapath
94         ofproto = datapath.ofproto
95         parser = datapath.ofproto_parser
96         in_port = msg.match['in_port']
97
98         pkt = packet.Packet(msg.data)
99         eth = pkt.get_protocols(ether_types.ethernet)[0]
100        pkt_arp = pkt.get_protocol(arp.arp)
101        pkt_ipv4 = pkt.get_protocol(ipv4.ipv4)
102        pkt_tcp = pkt.get_protocol(tcp.tcp)
103
104        if eth.ethertype == ether_types.ETH_TYPE_LLDP:
105            return
106        if eth.ethertype == ether_types.ETH_TYPE_IPV6:
107            return
108
109        dst = eth.dst
110        src = eth.src
111        dpid = datapath.id
112
113        # self.logger.info("packet in %s %s %s %s", dpid, src, dst, in_port)
114
115        # in rest_topology, self.mac_to_port is for the find for host
116        self.mac_to_port.setdefault(dpid, {})
117        self.mac_to_port[dpid][src] = in_port
118
119        # arp handle
120        if pkt_arp and pkt_arp.opcode == arp.ARP_REQUEST:
121            if pkt_arp.src_ip not in self.ip_to_mac:
122                self.ip_to_mac[pkt_arp.src_ip] = src
123                self.mac_to_dpid[src] = (dpid, in_port)
124                self.ip_to_port[pkt_arp.src_ip] = (dpid, in_port)
125
126            if pkt_arp.dst_ip in self.ip_to_mac:
127                #self.logger.info("[PACKET] ARP packet_in.")
128                self.handle_arpre(datapath=datapath, port=in_port,
129                                  src_mac=self.ip_to_mac[pkt_arp.dst_ip],
130                                  dst_mac=src, src_ip=pkt_arp.dst_ip, dst_ip=pkt_arp.src_ip)
131            else:
132                # to avoid flood when the dst ip not in the network
133                if datapath.id not in self.check_ip_dpid[pkt_arp.dst_ip]:
134                    self.check_ip_dpid[pkt_arp.dst_ip].append(datapath.id)
135                    out_port = ofproto.OFPP_FLOOD
136                    actions = [parser.OFPAActionOutput(out_port)]
137                    data = None
138                    if msg.buffer_id == ofproto.OFP_NO_BUFFER:
139                        data = msg.data
140                    out = parser.OFPPacketOut(datapath=datapath, buffer_id=msg.buffer_id,
141                                              in_port=in_port, actions=actions, data=data)
142                    datapath.send_msg(out)
143            return
144

```

```

145         elif pkt_arp and pkt_arp.opcode == arp.ARP_REPLY:
146             if pkt_arp.src_ip not in self.ip_to_mac:
147                 self.ip_to_mac[pkt_arp.src_ip] = src
148                 self.mac_to_dpid[src] = (dpid, in_port)
149                 self.ip_to_port[pkt_arp.src_ip] = (dpid, in_port)
150                 dst_mac = self.ip_to_mac[pkt_arp.dst_ip]
151                 (dst_dpid, dst_port) = self.mac_to_dpid[dst_mac]
152                 self.handle_arp(re=datapath=self.datapaths[dst_dpid], port=dst_port, src_mac=src,
dst_mac=dst_mac,
153                                     src_ip=pkt_arp.src_ip, dst_ip=pkt_arp.dst_ip)
154             return
155
156         if pkt_ipv4 and (self.ip_to_port.get(pkt_ipv4.dst)) and
(self.ip_to_port.get(pkt_ipv4.src)):
157             (src_dpid, src_port) = self.ip_to_port[pkt_ipv4.src] # src dpid and port
158             (dst_dpid, dst_port) = self.ip_to_port[pkt_ipv4.dst] # dst dpid and port
159             self.install_path(src_dpid=src_dpid, dst_dpid=dst_dpid, src_port=src_port,
dst_port=dst_port,
160                             ev=ev, src=src, dst=dst, pkt_ipv4=pkt_ipv4, pkt_tcp=pkt_tcp)
161             '''
162             time.sleep(5)
163             self.logger.info("Now Start Deleting...")
164             for i in xrange(len(self.path) - 2, -1, -2):
165                 datapath_path = self.datapaths[self.path[i][0]]
166                 self.delete_flow(datapath_path)
167             '''
168         def send_pkt(self, datapath, port, pkt):
169             ofproto = datapath.ofproto
170             parser = datapath.ofproto_parser
171             pkt.serialize()
172             data = pkt.data
173             actions = [parser.OFPAActionOutput(port=port)]
174             out = parser.OFPPacketOut(datapath=datapath, buffer_id=ofproto.OFP_NO_BUFFER,
in_port=ofproto.OFPP_CONTROLLER,
175                                     actions=actions, data=data)
176             datapath.send_msg(out)
177
178         def handle_arp(self, datapath, port, src_mac, dst_mac, src_ip, dst_ip):
179             pkt = packet.Packet()
180             pkt.add_protocol(ethernet.ethernet(ethertype=0x0806, dst=dst_mac, src=src_mac))
181             pkt.add_protocol(arp.arp(opcode=arp.ARP_REPLY, src_mac=src_mac, src_ip=src_ip,
dst_mac=dst_mac, dst_ip=dst_ip))
182             self.send_pkt(datapath, port, pkt)
183
184         def install_path(self, src_dpid, dst_dpid, src_port, dst_port, ev, src, dst, pkt_ipv4,
pkt_tcp):
185             msg = ev.msg
186             datapath = msg.datapath
187             ofproto = datapath.ofproto
188             parser = datapath.ofproto_parser
189
190             mid_path = None
191
192             mid_path = self.short_path(src=src_dpid, dst=dst_dpid, bw=0)
193
194             if mid_path is None:

```



```

195         return
196     self.path = None
197     self.path = [(src_dpид, src_port)] + mid_path + [(dst_dpид, dst_port)]
198     if self.old_path == None:
199         self.old_path = self.path
200         self.current_path = self.path
201     else:
202         self.old_path = self.current_path
203         self.current_path = self.path
204
205     self.logger.info("path : %s", str(self.path))
206     self.logger.info("current_path : %s", str(self.current_path))
207     self.logger.info("old_path : %s", str(self.old_path))
208
209     for i in xrange(len(self.path) - 2, -1, -2):
210         datapath_path = self.datapaths[self.path[i][0]]
211         match = parser.OFPMatch(in_port=self.path[i][1], eth_src=src, eth_dst=dst,
eth_type=0x0800,
212                                ipv4_src=pkt_ipv4.src, ipv4_dst=pkt_ipv4.dst)
213
214         if i < (len(self.path) - 2):
215             actions = [parser.OFPACTIONOutput(self.path[i + 1][1])]
216         else:
217             actions =
[parser.OFPACTIONSetField(eth_dst=self.ip_to_mac.get(pkt_ipv4.dst)),
218                          parser.OFPACTIONOutput(self.path[i + 1][1])]
219
220         self.add_flow(datapath_path, 100, match, actions, table_id=0, idle_timeout=0,
hard_timeout=0)
221         time_install = datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f')
222         self.logger.info("time_install: %s", time_install)
223
224
225     def short_path(self, src, dst, bw):
226         if src == dst:
227             return []
228         result = defaultdict(lambda: defaultdict(lambda: None))
229         distance = defaultdict(lambda: defaultdict(lambda: None))
230
231         # the node is checked
232         seen = [src]
233
234         # the distance to src
235         # w=1 means shortest path, w=-1 means longest path
236         distance[src] = 0
237         w = 1
238
239         while len(seen) < len(self.src_links):
240             node = seen[-1]
241             if node == dst:
242                 break
243             for (temp_src, temp_dst) in self.src_links[node]:
244                 if temp_dst not in seen:
245                     temp_src_port = self.src_links[node][(temp_src, temp_dst)][0]
246                     temp_dst_port = self.src_links[node][(temp_src, temp_dst)][1]
247                     if (distance[temp_dst] is None) or (distance[temp_dst] >

```

```

distance[temp_src] + w):
248         distance[temp_dst] = distance[temp_src] + w
249         # result = {"dpid":(link_src, src_port, link_dst, dst_port)}
250         result[temp_dst] = (temp_src, temp_src_port, temp_dst,
temp_dst_port)
251         min_node = None
252         min_path = 999
253         # get the min_path node
254         for temp_node in distance:
255             if (temp_node not in seen) and (distance[temp_node] is not None):
256                 if distance[temp_node] < min_path:
257                     min_node = temp_node
258                     min_path = distance[temp_node]
259             if min_node is None:
260                 break
261         seen.append(min_node)
262
263         path = []
264
265         if dst not in result:
266             return None
267
268         while (dst in result) and (result[dst] is not None):
269             path = [result[dst][2:4]] + path
270             path = [result[dst][0:2]] + path
271             dst = result[dst][0]
272         #self.logger.info("path : %s", str(path))
273         return path
274
275     # this function might be useful, but who knows anyway
276     # def long_path(self, src, dst, bw=0):
277
278     @set_ev_cls(ofp_event.EventOFPSwitchChange, [MAIN_DISPATCHER, DEAD_DISPATCHER])
279     def state_change_handler(self, ev):
280         datapath = ev.datapath
281         if ev.state == MAIN_DISPATCHER:
282             if datapath.id not in self.datapaths:
283                 self.datapaths[datapath.id] = datapath
284             elif ev.state == DEAD_DISPATCHER:
285                 if datapath.id in self.datapaths:
286                     del self.datapaths[datapath.id]
287             #self.logger.info("datapaths : %s", self.datapaths)
288
289     @set_ev_cls([event.EventSwitchEnter, event.EventSwitchLeave, event.EventPortAdd,
event.EventPortDelete,
290                 event.EventPortModify, event.EventLinkAdd, event.EventLinkDelete])
291     def get_topology(self, ev):
292         # self.logger.info("Get topo now...")
293         links_list = get_link(self.topology_api_app, None)
294         self.src_links.clear()
295         for link in links_list:
296             sw_src = link.src.dpid
297             sw_dst = link.dst.dpid
298             src_port = link.src.port_no
299             dst_port = link.dst.port_no
300             src_port_name = link.src.name

```

```

301     dst_port_name = link.dst.name
302     self.port_name_to_num[src_port_name] = src_port
303     self.port_name_to_num[dst_port_name] = dst_port
304     self.src_links[sw_src][(sw_src, sw_dst)] = (src_port, dst_port)
305     self.src_links[sw_dst][(sw_dst, sw_src)] = (dst_port, src_port)
306     # self.logger.info("****src_port_name : %s", str(src_port_name))
307     # self.logger.info("src_links : %s", str(self.src_links))
308     # self.logger.info("port_name_to_num : %s", str(self.port_name_to_num))
309     if self.old_path != None:
310         for i in xrange(len(self.current_path)-2, -1, -2):
311             datapath_path = self.datapaths[self.path[i][0]]
312             self.delete_flow(datapath=datapath_path)
313             ofproto = datapath_path.ofproto
314             parser = datapath_path.ofproto_parser
315             match = parser.OFPMatch()
316             actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
317                                             ofproto.OFPCML_NO_BUFFER)]
318             self.add_flow(datapath_path, 0, match, actions)
319
320
321     # these two functions need to be coded in your own way
322     def delete_flow(self, datapath, idle_timeout=10, hard_timeout=60):
323         ofproto = datapath.ofproto
324         parser = datapath.ofproto_parser
325         match = parser.OFPMatch()
326         instructions = []
327         mod = parser.OFPPFlowMod(datapath=datapath, command=ofproto.OFPFC_DELETE,
out_port=ofproto.OFPP_ANY, out_group=ofproto.OFPG_ANY)
328         datapath.send_msg(mod)
329         #self.logger.info("Deleted! "+ str(datapath.id))
330         '''
331         @set_ev_cls(ofp_event.EventOFPPortStatus, [CONFIG_DISPATCHER, MAIN_DISPATCHER,
DEAD_DISPATCHER, HANDSHAKE_DISPATCHER])
332         def get_OFPPortStatus_msg(self, ev):
333             '''

```