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 partically.

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 attributtes, idle_timeout and hard_time out. Idle_time out 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 descriptioin 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 questioin 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:

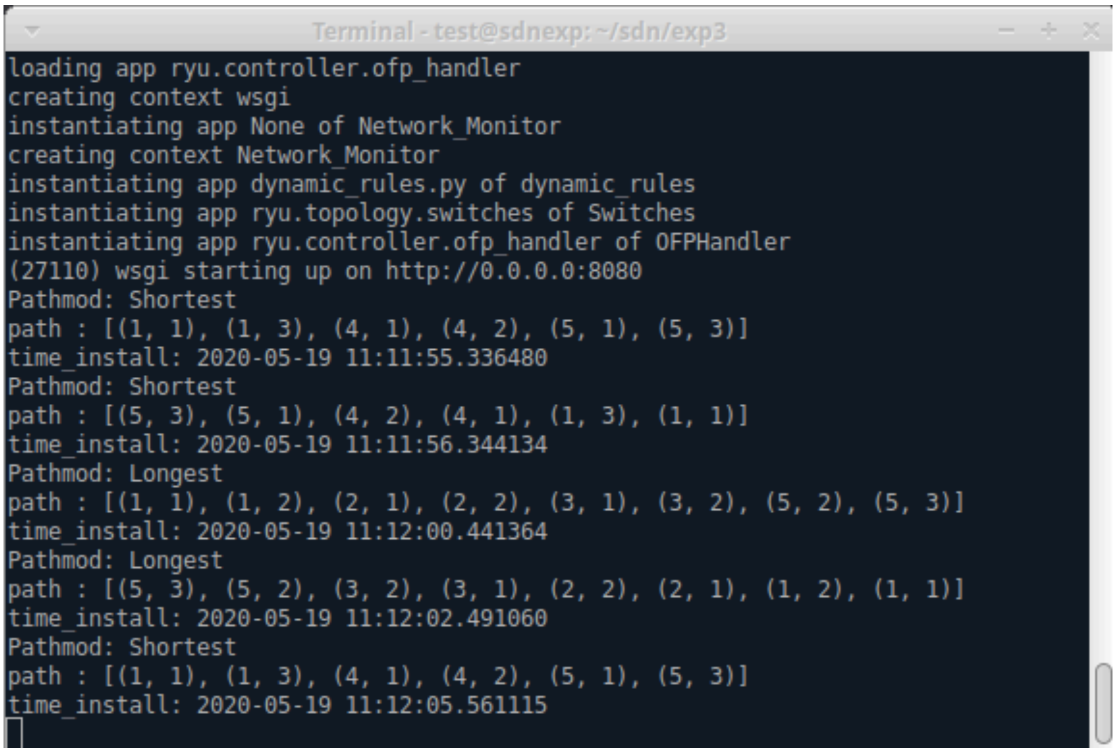
1. Every time the algorithm chooses the shortest path, make it choose the longest.
2. Change the weight of every edge from 1 to -1.
   In the code I chosed the latter solution.

Add switch-mode condition in the packet in handler

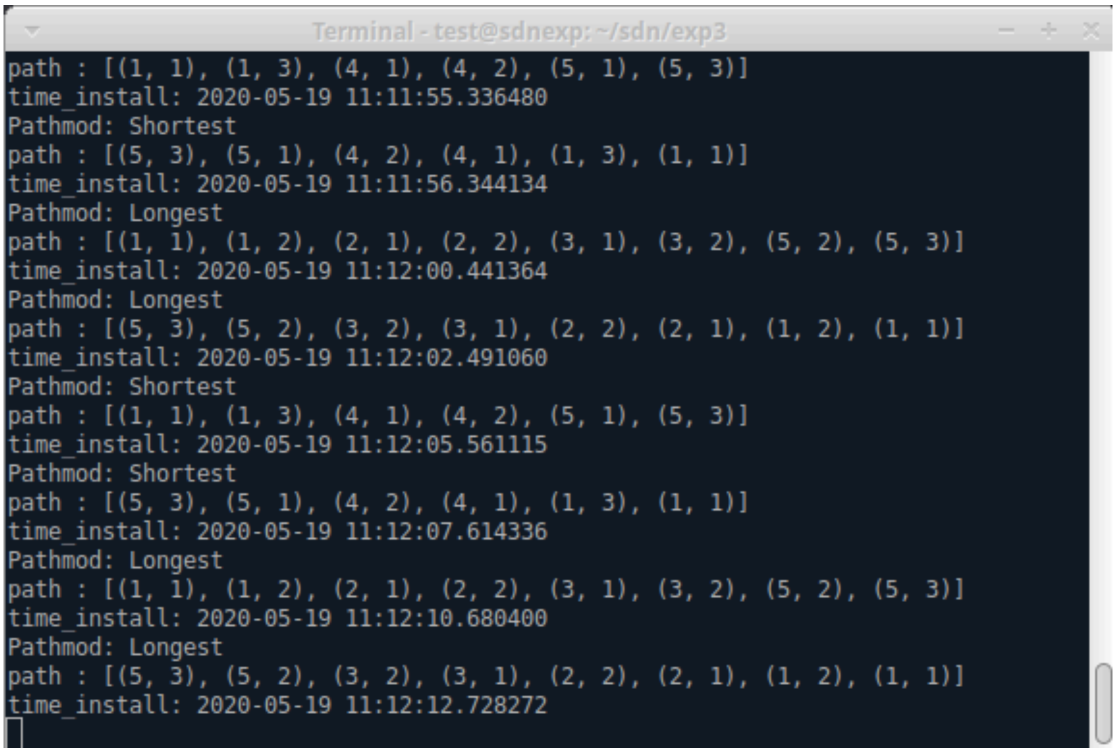
To begin with, the switch will send trigger swith feature hanlder and then a basic flow whose match field is empty will be installed into the switch (This flow is significant is part 2). And then, when next new packet hanler arrives, the switch will send a message to the controller and trigger packet 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:



```
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
```



```
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 destionation 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.

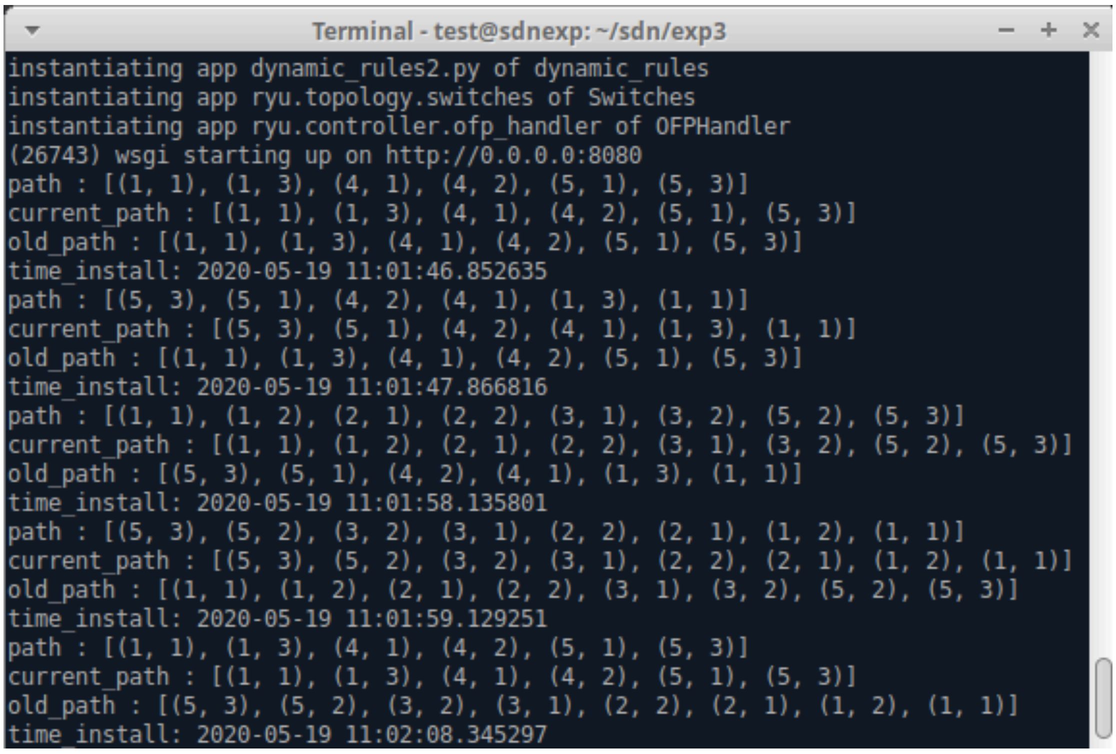
Flow changes of switches on the old path

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).

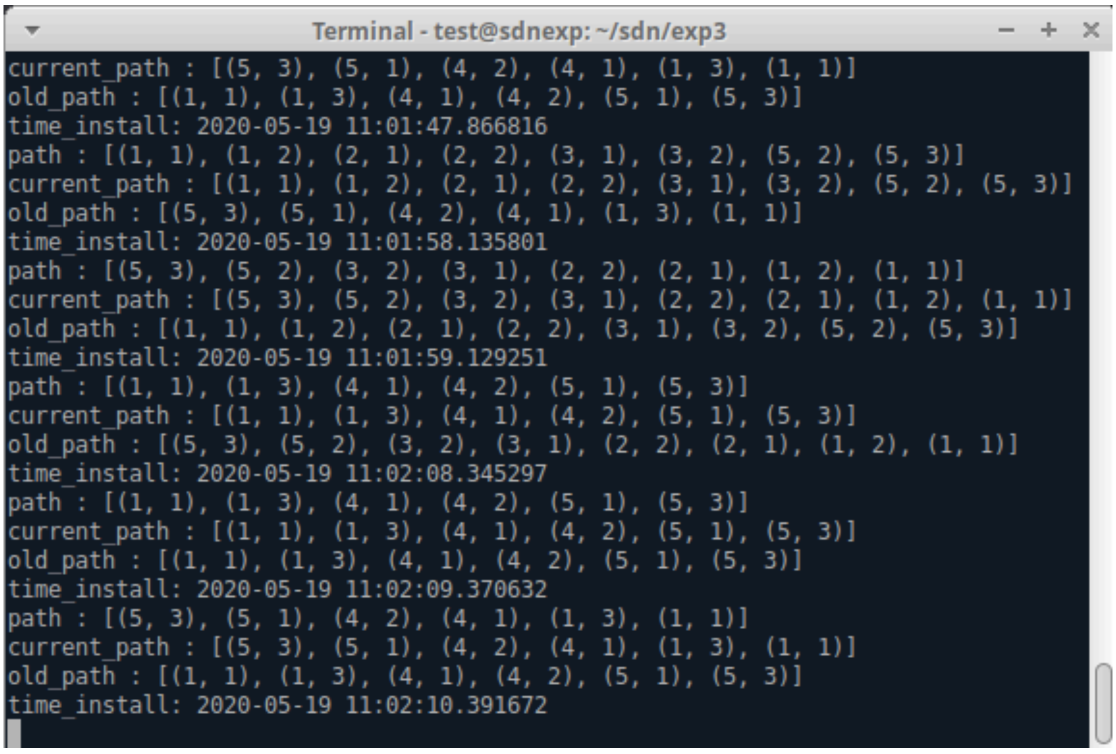
Getting the shortest path

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

The final result is as follows:



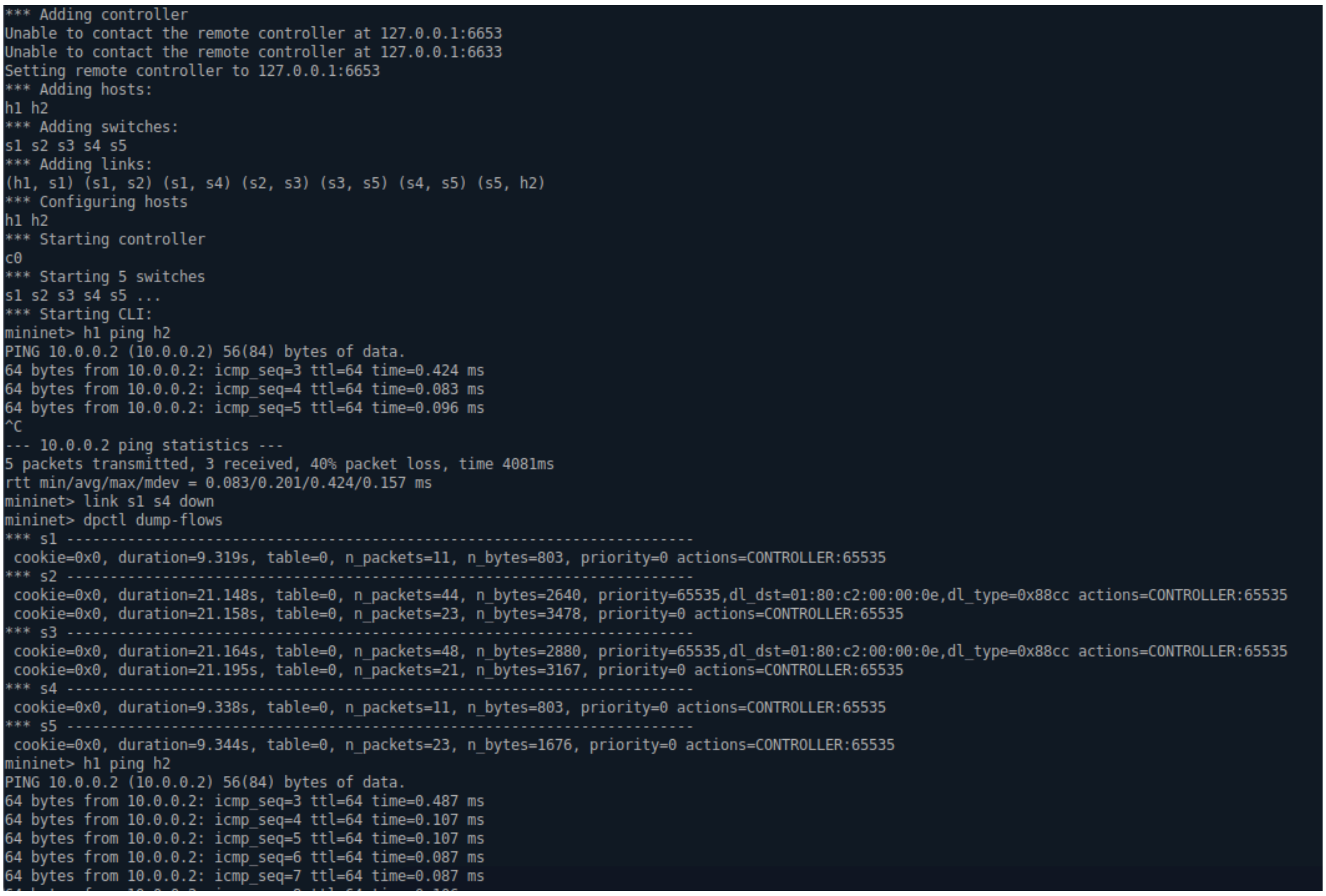
```
instantiating app dynamic_rules2.py of dynamic_rules
instantiating app ryu.topology.switches of Switches
instantiating app ryu.controller.ofp_handler of OFPHandler
(26743) wsgi starting up on http://0.0.0.0:8080
path : [(1, 1), (1, 3), (4, 1), (4, 2), (5, 1), (5, 3)]
current_path : [(1, 1), (1, 3), (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 disater tolerance.

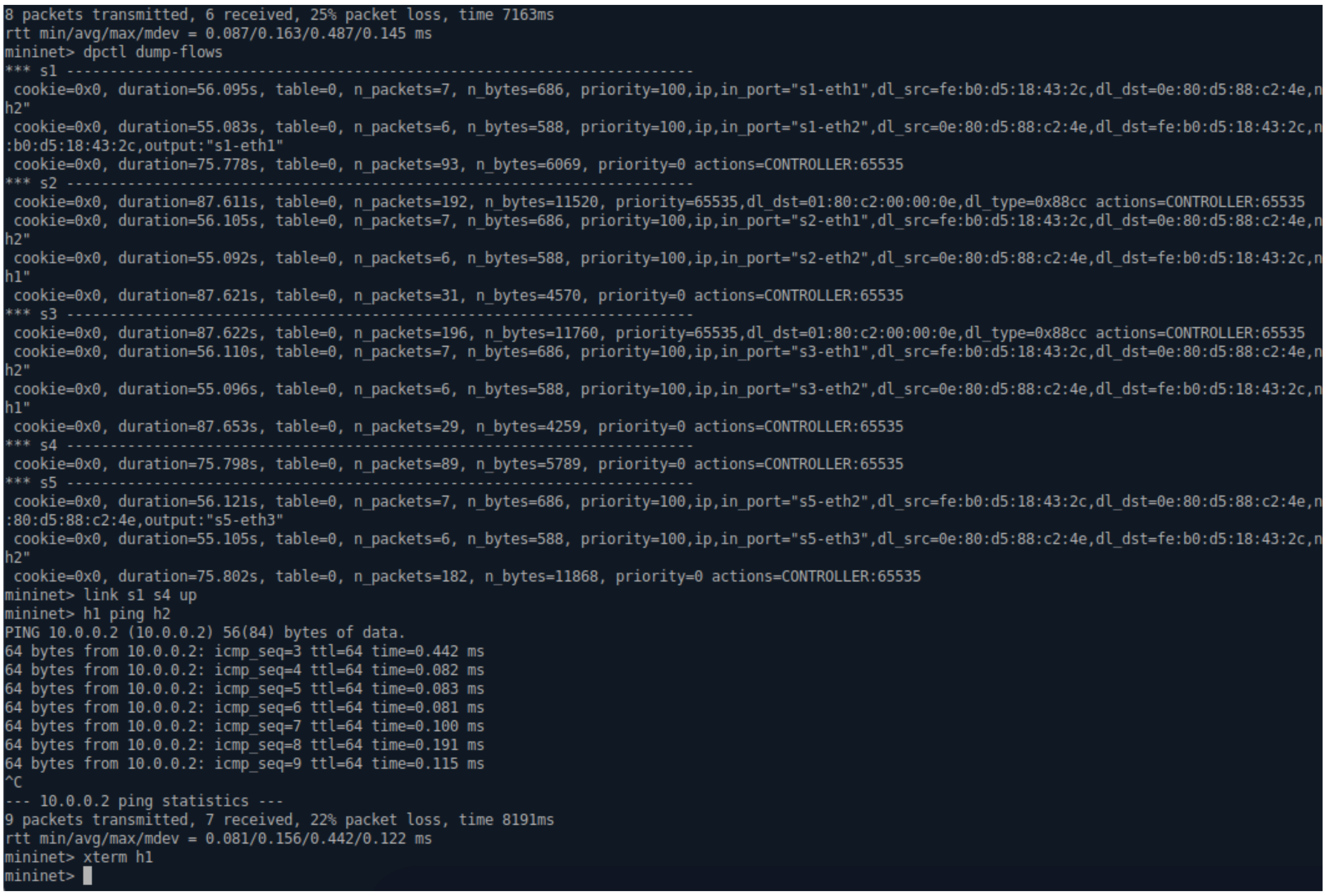
Let's take a closer look.



```
*** 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 form 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.

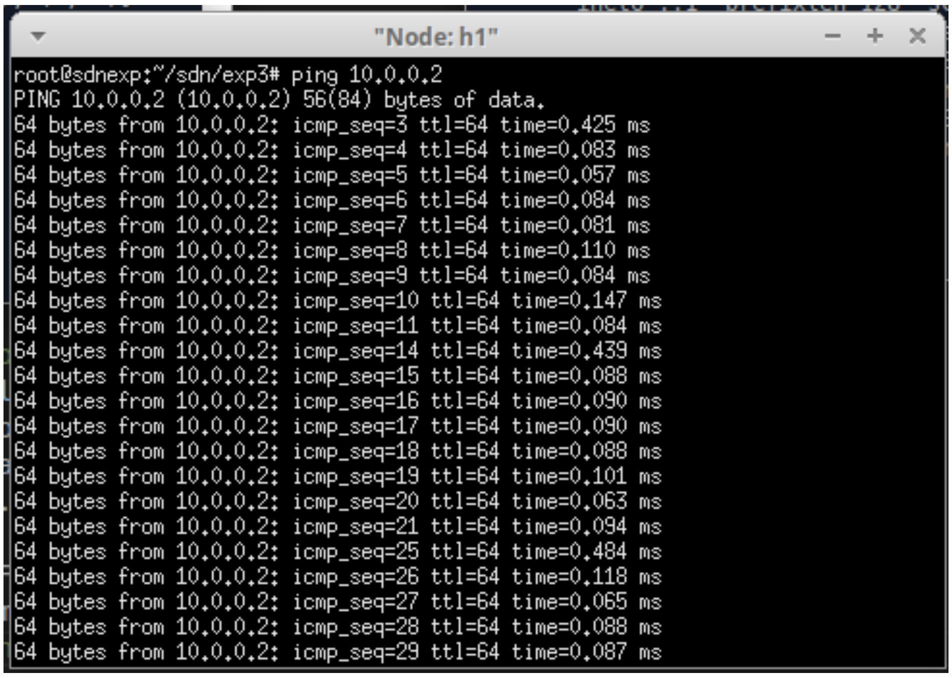
See this:



```
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, recalcucate the shortest path and install flows, during which some packets *might be lost.*

Reference

[OpenFlow Switch Specification]()
[How to delete flows]()

Source Code

```python
# the code of the topology

from mininet.topo import Topo

class Mytopo(Topo):
    def __init__(self):
        super(Mytopo, self).__init__()
        h1 = self.addHost('h1')
        h2 = self.addHost('h2')

        s1 = self.addSwitch('s1')
        s2 = self.addSwitch('s2')
        s3 = self.addSwitch('s3')
        s4 = self.addSwitch('s4')
        s5 = self.addSwitch('s5')

        self.addLink(h1, s1)
        self.addLink(s1, s2)
        self.addLink(s1, s4)
        self.addLink(s2, s3)
        self.addLink(s4, s5)
        self.addLink(s3, s5)
        self.addLink(s5, h2)

topos = { 'mytopo': (lambda : Mytopo())}
```

```python
# the code of part 1

from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.topology import event
from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER, DEAD_DISPATCHER, HANDSHAKE_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 ipv4
from ryu.lib.packet import tcp
from ryu.topology.api import get_link
from ryu.lib.packet import ether_types
from ryu.app.wsgi import  WSGIApplication
from collections import defaultdict
import network_monitor
import datetime
class dynamic_rules(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]
    _CONTEXTS = {
        "Network_Monitor": network_monitor.Network_Monitor,
        "wsgi": WSGIApplication
    }
    def __init__(self, *args, **kwargs):
        super(dynamic_rules, self).__init__(*args, **kwargs)
        self.mac_to_port = {}
        #mac_to_port maps [mac of src][dpid of switch] to port
        self.ip_to_mac = {}
        self.mac_to_dpid = {}  # {mac:(dpid,port)}

        self.datapaths = defaultdict(lambda: None)
        self.topology_api_app = self
        self.src_links = defaultdict(lambda: defaultdict(lambda: None))

        self.check_ip_dpid = defaultdict(list)

        self.qos_ip_bw_list = []

        self.network_monitor = kwargs["Network_Monitor"]


        self.ip_to_switch = {}
        self.port_name_to_num = {}

        self.ip_to_port = {}  #{ip:(dpid,port)}
        #promise me, use it well :)
        # let 0 equals to shortest_path, 1 equals to longest_path
        self.pathmod = 0
        self.path = None
        self.come_and_go = 0

    @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
    def switch_features_handler(self, ev):
        datapath = ev.msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
        match = parser.OFPMatch()
        actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
                                          ofproto.OFPCML_NO_BUFFER)]
        self.add_flow(datapath, 0, match, actions)

    def add_flow(self, datapath, priority, match, actions, buffer_id=None, idle_timeout=0,
hard_timeout=0):
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
                                             actions)]
        if buffer_id:
            mod = parser.OFPFlowMod(datapath=datapath, buffer_id=buffer_id,
                                    priority=priority, match=match,
                                    idle_timeout=idle_timeout,
                                    hard_timeout=hard_timeout,
                                    instructions=inst)
        else:
            mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
                                    idle_timeout=idle_timeout,
                                    hard_timeout=hard_timeout,
                                    match=match, instructions=inst)
        datapath.send_msg(mod)

    @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
    def _packet_in_handler(self, ev):

        if ev.msg.msg_len < ev.msg.total_len:
            self.logger.debug("packet truncated: only %s of %s bytes",
                              ev.msg.msg_len, ev.msg.total_len)
        msg = ev.msg
        datapath = msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
        in_port = msg.match['in_port']

        pkt = packet.Packet(msg.data)
        eth = pkt.get_protocols(ethernet.ethernet)[0]
        pkt_arp = pkt.get_protocol(arp.arp)
        pkt_ipv4 = pkt.get_protocol(ipv4.ipv4)
        pkt_tcp = pkt.get_protocol(tcp.tcp)

        if eth.ethertype == ether_types.ETH_TYPE_LLDP:
            return
        if eth.ethertype == ether_types.ETH_TYPE_IPV6:
            return

        dst = eth.dst
        src = eth.src
        dpid = datapath.id

        # self.logger.info("packet in %s %s %s %s", dpid, src, dst, in_port)

        # in rest_topology, self.mac_to_port is for the find for host
        self.mac_to_port.setdefault(dpid, {})
        self.mac_to_port[dpid][src] = in_port

        # arp handle
        if pkt_arp and pkt_arp.opcode == arp.ARP_REQUEST:
            if pkt_arp.src_ip not in self.ip_to_mac:
                self.ip_to_mac[pkt_arp.src_ip] = src
                self.mac_to_dpid[src] = (dpid, in_port)
                self.ip_to_port[pkt_arp.src_ip] = (dpid, in_port)

            if pkt_arp.dst_ip in self.ip_to_mac:
                #self.logger.info("[PACKET] ARP packet_in.")
                self.handle_arpre(datapath=datapath, port=in_port,
                                  src_mac=self.ip_to_mac[pkt_arp.dst_ip],
                                  dst_mac=src, src_ip=pkt_arp.dst_ip, dst_ip=pkt_arp.src_ip)
            else:
                # to avoid flood when the dst ip not in the network
                if datapath.id not in self.check_ip_dpid[pkt_arp.dst_ip]:
                    self.check_ip_dpid[pkt_arp.dst_ip].append(datapath.id)
                    out_port = ofproto.OFPP_FLOOD
                    actions = [parser.OFPActionOutput(out_port)]
                    data = None
                    if msg.buffer_id == ofproto.OFP_NO_BUFFER:
                        data = msg.data
                    out = parser.OFPPacketOut(datapath=datapath, buffer_id=msg.buffer_id,
                                              in_port=in_port, actions=actions, data=data)
                    datapath.send_msg(out)
            return

        elif pkt_arp and pkt_arp.opcode == arp.ARP_REPLY:
            if pkt_arp.src_ip not in self.ip_to_mac:
                self.ip_to_mac[pkt_arp.src_ip] = src
                self.mac_to_dpid[src] = (dpid, in_port)
                self.ip_to_port[pkt_arp.src_ip] = (dpid, in_port)
            dst_mac = self.ip_to_mac[pkt_arp.dst_ip]
            (dst_dpid, dst_port) = self.mac_to_dpid[dst_mac]
            self.handle_arpre(datapath=self.datapaths[dst_dpid], port=dst_port, src_mac=src, dst_mac=dst_mac,
                              src_ip=pkt_arp.src_ip, dst_ip=pkt_arp.dst_ip)
            return

        if pkt_ipv4 and (self.ip_to_port.get(pkt_ipv4.dst)) and (self.ip_to_port.get(pkt_ipv4.src)):
            (src_dpid, src_port) = self.ip_to_port[pkt_ipv4.src]  # src dpid and port
            (dst_dpid, dst_port) = self.ip_to_port[pkt_ipv4.dst]  # dst dpid and port
            self.install_path(src_dpid=src_dpid, dst_dpid=dst_dpid, src_port=src_port, dst_port=dst_port,
                              ev=ev, src=src, dst=dst, pkt_ipv4=pkt_ipv4, pkt_tcp=pkt_tcp)
            self.come_and_go = self.come_and_go + 1
            if self.come_and_go == 2:
                self.come_and_go = 0
                self.pathmod = not self.pathmod


    def send_pkt(self, datapath, port, pkt):
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
        pkt.serialize()
        data = pkt.data
        actions = [parser.OFPActionOutput(port=port)]
        out = parser.OFPPacketOut(datapath=datapath, buffer_id=ofproto.OFP_NO_BUFFER, in_port=ofproto.OFPP_CONTROLLER,
                                  actions=actions, data=data)
        datapath.send_msg(out)

    def handle_arpre(self, datapath, port, src_mac, dst_mac, src_ip, dst_ip):
        pkt = packet.Packet()
        pkt.add_protocol(ethernet.ethernet(ethertype=0x0806, dst=dst_mac, src=src_mac))
        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))
            self.send_pkt(datapath, port, pkt)

    def install_path(self, src_dpid, dst_dpid, src_port, dst_port, ev, src, dst, pkt_ipv4, pkt_tcp):
        msg = ev.msg
        datapath = msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        mid_path = None
        if self.pathmod == 0:
            mid_path = self.short_path(src=src_dpid, dst=dst_dpid, bw=0)
            self.logger.info('Pathmod: Shortest')
        elif self.pathmod == 1:
            mid_path = self.short_path(src=src_dpid, dst=dst_dpid, bw=1)
            self.logger.info('Pathmod: Longest')

        if mid_path is None:
            return
        self.path = None
        self.path = [(src_dpid, src_port)] + mid_path + [(dst_dpid, dst_port)]

        self.logger.info("path : %s", str(self.path))

        for i in xrange(len(self.path) - 2, -1, -2):
            datapath_path = self.datapaths[self.path[i][0]]
            match = parser.OFPMatch(in_port=self.path[i][1], eth_src=src, eth_dst=dst, eth_type=0x0800,
                                    ipv4_src=pkt_ipv4.src, ipv4_dst=pkt_ipv4.dst)

            if i < (len(self.path) - 2):
                actions = [parser.OFPActionOutput(self.path[i + 1][1])]
            else:
                actions = [parser.OFPActionSetField(eth_dst=self.ip_to_mac.get(pkt_ipv4.dst)),
                           parser.OFPActionOutput(self.path[i + 1][1])]

            self.add_flow(datapath_path, 100, match, actions, idle_timeout=0, hard_timeout=5)
        time_install = datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f')
        self.logger.info("time_install: %s", time_install)


    def short_path(self, src, dst, bw):
        if src == dst:
            return []
        result = defaultdict(lambda: defaultdict(lambda: None))
        distance = defaultdict(lambda: defaultdict(lambda: None))

        # the node is checked
        seen = [src]

        # the distance to src
        # w=1 means shortest path, w=-1 means longest path
        distance[src] = 0
```

```python
        if bw==0:
            w = 1  # weight
        else:
            w = -1

        while len(seen) < len(self.src_links):
            node = seen[-1]
            if node == dst:
                break
            for (temp_src, temp_dst) in self.src_links[node]:
                if temp_dst not in seen:
                    temp_src_port = self.src_links[node][(temp_src, temp_dst)][0]
                    temp_dst_port = self.src_links[node][(temp_src, temp_dst)][1]
                    if (distance[temp_dst] is None) or (distance[temp_dst] > distance[temp_src] + w):
                        distance[temp_dst] = distance[temp_src] + w
                        # result = {"dpid":(link_src, src_port, link_dst, dst_port)}
                        result[temp_dst] = (temp_src, temp_src_port, temp_dst, temp_dst_port)
            min_node = None
            min_path = 999
            # get the min_path node
            for temp_node in distance:
                if (temp_node not in seen) and (distance[temp_node] is not None):
                    if distance[temp_node] < min_path:
                        min_node = temp_node
                        min_path = distance[temp_node]
            if min_node is None:
                break
            seen.append(min_node)

        path = []

        if dst not in result:
            return None

        while (dst in result) and (result[dst] is not None):
            path = [result[dst][2:4]] + path
            path = [result[dst][0:2]] + path
            dst = result[dst][0]
        #self.logger.info("path : %s", str(path))
        return path

    # this function might be useful, but who knows anyway
    # def long_path(self, src, dst, bw=0):

    @set_ev_cls(ofp_event.EventOFPStateChange, [MAIN_DISPATCHER, DEAD_DISPATCHER])
    def state_change_handler(self, ev):
        datapath = ev.datapath
        if ev.state == MAIN_DISPATCHER:
            if datapath.id not in self.datapaths:
                self.datapaths[datapath.id] = datapath
        elif ev.state == DEAD_DISPATCHER:
            if datapath.id in self.datapaths:
                del self.datapaths[datapath.id]
        #self.logger.info("datapaths : %s", self.datapaths)

    @set_ev_cls([event.EventSwitchEnter, event.EventSwitchLeave, event.EventPortAdd, event.EventPortDelete,
        event.EventPortModify, event.EventLinkAdd, event.EventLinkDelete])
    def get_topology(self, ev):
        links_list = get_link(self.topology_api_app, None)
        self.src_links.clear()
        for link in links_list:
            sw_src = link.src.dpid
            sw_dst = link.dst.dpid
            src_port = link.src.port_no
            dst_port = link.dst.port_no
            src_port_name = link.src.name
            dst_port_name = link.dst.name
            self.port_name_to_num[src_port_name] = src_port
            self.port_name_to_num[dst_port_name] = dst_port
            self.src_links[sw_src][(sw_src, sw_dst)] = (src_port, dst_port)
            self.src_links[sw_dst][(sw_dst, sw_src)] = (dst_port, src_port)
            # self.logger.info("****src_port_name : %s", str(src_port_name))
            # self.logger.info("src_links : %s", str(self.src_links))
            # self.logger.info("port_name_to_num : %s", str(self.port_name_to_num))
```

```python
# the code of part 2

from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.topology import event
from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER, DEAD_DISPATCHER, HANDSHAKE_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 ipv4
from ryu.lib.packet import tcp
from ryu.topology.api import get_link
from ryu.lib.packet import ether_types
from ryu.app.wsgi import  WSGIApplication
from collections import defaultdict
import network_monitor
import datetime
import time
class dynamic_rules(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]
    _CONTEXTS = {
        "Network_Monitor": network_monitor.Network_Monitor,
        "wsgi": WSGIApplication
    }
    def __init__(self, *args, **kwargs):
        super(dynamic_rules, self).__init__(*args, **kwargs)
        self.mac_to_port = {}
        #mac_to_port maps [mac of src][dpid of switch] to port
        self.ip_to_mac = {}
        self.mac_to_dpid = {}  # {mac:(dpid,port)}

        self.datapaths = defaultdict(lambda: None)
        self.topology_api_app = self
        self.src_links = defaultdict(lambda: defaultdict(lambda: None))

        self.check_ip_dpid = defaultdict(list)

        self.qos_ip_bw_list = []

        self.network_monitor = kwargs["Network_Monitor"]


        self.ip_to_switch = {}
        self.port_name_to_num = {}

        self.ip_to_port = {}  #{ip:(dpid,port)}
        #promise me, use it well :)
        # let 0 equals to shortest_path, 1 equals to longest_path
        self.pathmod = 0
        self.path = None
        self.old_path = None
        self.current_path = None


    @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
    def switch_features_handler(self, ev):
        datapath = ev.msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
        match = parser.OFPMatch()
        actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
                                          ofproto.OFPCML_NO_BUFFER)]
        self.add_flow(datapath, 0, match, actions)

    def add_flow(self, datapath, priority, match, actions, table_id=0, buffer_id=None, idle_timeout=0, hard_timeout=0):
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
                                             actions)]
        if buffer_id:
            mod = parser.OFPFlowMod(datapath=datapath, table_id=table_id, buffer_id=buffer_id,
                                    priority=priority, match=match,
                                    idle_timeout=idle_timeout,
                                    hard_timeout=hard_timeout,
                                    instructions=inst)
        else:
            mod = parser.OFPFlowMod(datapath=datapath, table_id=table_id, priority=priority,
                                    idle_timeout=idle_timeout,
                                    hard_timeout=hard_timeout,
                                    match=match, instructions=inst)
        datapath.send_msg(mod)

    @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
    def _packet_in_handler(self, ev):

        if ev.msg.msg_len < ev.msg.total_len:
            self.logger.debug("packet truncated: only %s of %s bytes",
                              ev.msg.msg_len, ev.msg.total_len)
        msg = ev.msg
        datapath = msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
        in_port = msg.match['in_port']

        pkt = packet.Packet(msg.data)
        eth = pkt.get_protocols(ethernet.ethernet)[0]
        pkt_arp = pkt.get_protocol(arp.arp)
        pkt_ipv4 = pkt.get_protocol(ipv4.ipv4)
        pkt_tcp = pkt.get_protocol(tcp.tcp)

        if eth.ethertype == ether_types.ETH_TYPE_LLDP:
            return
        if eth.ethertype == ether_types.ETH_TYPE_IPV6:
            return

        dst = eth.dst
        src = eth.src
        dpid = datapath.id

        # self.logger.info("packet in %s %s %s %s", dpid, src, dst, in_port)

        # in rest_topology, self.mac_to_port is for the find for host
        self.mac_to_port.setdefault(dpid, {})
        self.mac_to_port[dpid][src] = in_port

        # arp handle
        if pkt_arp and pkt_arp.opcode == arp.ARP_REQUEST:
            if pkt_arp.src_ip not in self.ip_to_mac:
                self.ip_to_mac[pkt_arp.src_ip] = src
                self.mac_to_dpid[src] = (dpid, in_port)
                self.ip_to_port[pkt_arp.src_ip] = (dpid, in_port)

            if pkt_arp.dst_ip in self.ip_to_mac:
                #self.logger.info("[PACKET] ARP packet_in.")
                self.handle_arpre(datapath=datapath, port=in_port,
                                  src_mac=self.ip_to_mac[pkt_arp.dst_ip],
                                  dst_mac=src, src_ip=pkt_arp.dst_ip, dst_ip=pkt_arp.src_ip)
            else:
                # to avoid flood when the dst ip not in the network
                if datapath.id not in self.check_ip_dpid[pkt_arp.dst_ip]:
                    self.check_ip_dpid[pkt_arp.dst_ip].append(datapath.id)
                    out_port = ofproto.OFPP_FLOOD
                    actions = [parser.OFPActionOutput(out_port)]
                    data = None
                    if msg.buffer_id == ofproto.OFP_NO_BUFFER:
                        data = msg.data
                    out = parser.OFPPacketOut(datapath=datapath, buffer_id=msg.buffer_id,
                                              in_port=in_port, actions=actions, data=data)
                    datapath.send_msg(out)
            return

        elif pkt_arp and pkt_arp.opcode == arp.ARP_REPLY:
            if pkt_arp.src_ip not in self.ip_to_mac:
                self.ip_to_mac[pkt_arp.src_ip] = src
                self.mac_to_dpid[src] = (dpid, in_port)
                self.ip_to_port[pkt_arp.src_ip] = (dpid, in_port)
            dst_mac = self.ip_to_mac[pkt_arp.dst_ip]
            (dst_dpid, dst_port) = self.mac_to_dpid[dst_mac]
            self.handle_arpre(datapath=self.datapaths[dst_dpid], port=dst_port, src_mac=src, dst_mac=dst_mac,
                              src_ip=pkt_arp.src_ip, dst_ip=pkt_arp.dst_ip)
            return

        if pkt_ipv4 and (self.ip_to_port.get(pkt_ipv4.dst)) and (self.ip_to_port.get(pkt_ipv4.src)):
            (src_dpid, src_port) = self.ip_to_port[pkt_ipv4.src]  # src dpid and port
            (dst_dpid, dst_port) = self.ip_to_port[pkt_ipv4.dst]  # dst dpid and port
            self.install_path(src_dpid=src_dpid, dst_dpid=dst_dpid, src_port=src_port, dst_port=dst_port,
                              ev=ev, src=src, dst=dst, pkt_ipv4=pkt_ipv4, pkt_tcp=pkt_tcp)
        '''
        time.sleep(5)
        self.logger.info("Now Start Deleting...")
        for i in xrange(len(self.path) - 2, -1, -2):
            datapath_path = self.datapaths[self.path[i][0]]
            self.delete_flow(datapath_path)
        '''
    def send_pkt(self, datapath, port, pkt):
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
        pkt.serialize()
        data = pkt.data
        actions = [parser.OFPActionOutput(port=port)]
        out = parser.OFPPacketOut(datapath=datapath, buffer_id=ofproto.OFP_NO_BUFFER, in_port=ofproto.OFPP_CONTROLLER,
                                  actions=actions, data=data)
        datapath.send_msg(out)

    def handle_arpre(self, datapath, port, src_mac, dst_mac, src_ip, dst_ip):
        pkt = packet.Packet()
        pkt.add_protocol(ethernet.ethernet(ethertype=0x0806, dst=dst_mac, src=src_mac))
        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))
        self.send_pkt(datapath, port, pkt)

    def install_path(self, src_dpid, dst_dpid, src_port, dst_port, ev, src, dst, pkt_ipv4, pkt_tcp):
        msg = ev.msg
        datapath = msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        mid_path = None

        mid_path = self.short_path(src=src_dpid, dst=dst_dpid, bw=0)

        if mid_path is None:
            return
        self.path = None
        self.path = [(src_dpid, src_port)] + mid_path + [(dst_dpid, dst_port)]
        if self.old_path == None:
            self.old_path = self.path
            self.current_path = self.path
        else:
            self.old_path = self.current_path
            self.current_path = self.path

        self.logger.info("path : %s", str(self.path))
        self.logger.info("current_path : %s", str(self.current_path))
        self.logger.info("old_path : %s", str(self.old_path))

        for i in xrange(len(self.path) - 2, -1, -2):
            datapath_path = self.datapaths[self.path[i][0]]
            match = parser.OFPMatch(in_port=self.path[i][1], eth_src=src, eth_dst=dst,
eth_type=0x0800,
                                    ipv4_src=pkt_ipv4.src, ipv4_dst=pkt_ipv4.dst)

            if i < (len(self.path) - 2):
                actions = [parser.OFPActionOutput(self.path[i + 1][1])]
            else:
                actions =
[parser.OFPActionSetField(eth_dst=self.ip_to_mac.get(pkt_ipv4.dst)),
                            parser.OFPActionOutput(self.path[i + 1][1])]

            self.add_flow(datapath_path, 100, match, actions, table_id=0, idle_timeout=0,
hard_timeout=0)
        time_install = datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f')
        self.logger.info("time_install: %s", time_install)


    def short_path(self, src, dst, bw):
        if src == dst:
            return []
        result = defaultdict(lambda: defaultdict(lambda: None))
        distance = defaultdict(lambda: defaultdict(lambda: None))

        # the node is checked
        seen = [src]

        # the distance to src
        # w=1 means shortest path, w=-1 means longest path
        distance[src] = 0
        w = 1

        while len(seen) < len(self.src_links):
            node = seen[-1]
            if node == dst:
                break
            for (temp_src, temp_dst) in self.src_links[node]:
                if temp_dst not in seen:
                    temp_src_port = self.src_links[node][(temp_src, temp_dst)][0]
                    temp_dst_port = self.src_links[node][(temp_src, temp_dst)][1]
                    if (distance[temp_dst] is None) or (distance[temp_dst] >
distance[temp_src] + w):
                        distance[temp_dst] = distance[temp_src] + w
                        # result = {"dpid":(link_src, src_port, link_dst, dst_port)}
                        result[temp_dst] = (temp_src, temp_src_port, temp_dst, temp_dst_port)
            min_node = None
            min_path = 999
            # get the min_path node
            for temp_node in distance:
                if (temp_node not in seen) and (distance[temp_node] is not None):
                    if distance[temp_node] < min_path:
                        min_node = temp_node
                        min_path = distance[temp_node]
            if min_node is None:
                break
            seen.append(min_node)

        path = []

        if dst not in result:
            return None

        while (dst in result) and (result[dst] is not None):
            path = [result[dst][2:4]] + path
            path = [result[dst][0:2]] + path
            dst = result[dst][0]
        #self.logger.info("path : %s", str(path))
        return path

    # this function might be useful, but who knows anyway
    # def long_path(self, src, dst, bw=0):

    @set_ev_cls(ofp_event.EventOFPStateChange, [MAIN_DISPATCHER, DEAD_DISPATCHER])
    def state_change_handler(self, ev):
        datapath = ev.datapath
        if ev.state == MAIN_DISPATCHER:
            if datapath.id not in self.datapaths:
                self.datapaths[datapath.id] = datapath
        elif ev.state == DEAD_DISPATCHER:
            if datapath.id in self.datapaths:
                del self.datapaths[datapath.id]
        #self.logger.info("datapaths : %s", self.datapaths)

    @set_ev_cls([event.EventSwitchEnter, event.EventSwitchLeave, event.EventPortAdd, event.EventPortDelete,
        event.EventPortModify, event.EventLinkAdd, event.EventLinkDelete])
    def get_topology(self, ev):
        # self.logger.info("Get topo now...")
        links_list = get_link(self.topology_api_app, None)
        self.src_links.clear()
        for link in links_list:
            sw_src = link.src.dpid
            sw_dst = link.dst.dpid
            src_port = link.src.port_no
            dst_port = link.dst.port_no
            src_port_name = link.src.name
            dst_port_name = link.dst.name
            self.port_name_to_num[src_port_name] = src_port
            self.port_name_to_num[dst_port_name] = dst_port
            self.src_links[sw_src][(sw_src, sw_dst)] = (src_port, dst_port)
            self.src_links[sw_dst][(sw_dst, sw_src)] = (dst_port, src_port)
            # self.logger.info("****src_port_name : %s", str(src_port_name))
            # self.logger.info("src_links : %s", str(self.src_links))
            # self.logger.info("port_name_to_num : %s", str(self.port_name_to_num))
            if self.old_path != None:
                for i in xrange(len(self.current_path)-2, -1, -2):
                    datapath_path = self.datapaths[self.path[i][0]]
                    self.delete_flow(datapath=datapath_path)
                    ofproto = datapath_path.ofproto
                    parser = datapath_path.ofproto_parser
                    match = parser.OFPMatch()
                    actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
                                                    ofproto.OFPCML_NO_BUFFER)]
                    self.add_flow(datapath_path, 0, match, actions)


    # these two functions need to be coded in your own way
    def delete_flow(self, datapath, idle_timeout=10, hard_timeout=60):
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
        match = parser.OFPMatch()
        instructions = []
        mod = parser.OFPFlowMod(datapath=datapath, command=ofproto.OFPFC_DELETE, out_port=ofproto.OFPP_ANY, out_group=ofproto.OFPG_ANY)
        datapath.send_msg(mod)
        #self.logger.info("Deleted! "+ str(datapath.id))
    '''
    @set_ev_cls(ofp_event.EventOFPPortStatus, [CONFIG_DISPATCHER, MAIN_DISPATCHER, DEAD_DISPATCHER, HANDSHAKE_DISPATCHER])
    def get_OFPPortStatus_msg(self, ev):
    '''
```