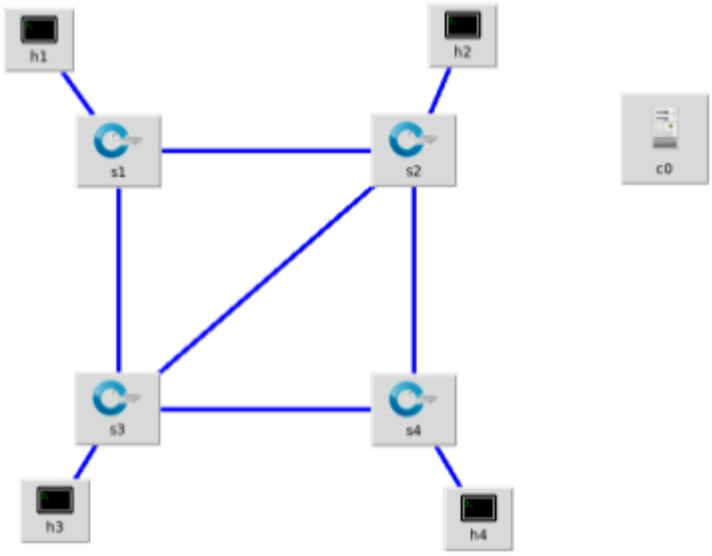
1-Introduction:

Software defined network a new approach that aims to solve problems related to flexibility and programmability. This would accelerate development, deployment and maintain process for networking infrastructires. In this work we will try to validate the key concept of SDN which is a centralized controller that calculate rules representing our network application than send these rules to simple forwarding elements. Our example consist of using Mininet an effective network simulator based on Linux network name-spaces, this simulator deploying simple forwarding devices that uses open-flow to interact with a controller in our case Pox controller.
The goal of this project is to implement a module to calculate routing tables, and each switch will receive his table. We used Dijkstra as our shortest path algorithm, and for tests will use a graphic module named Gephi to visualize our network topology changes in real time and also the recalculated routes due to these changes.

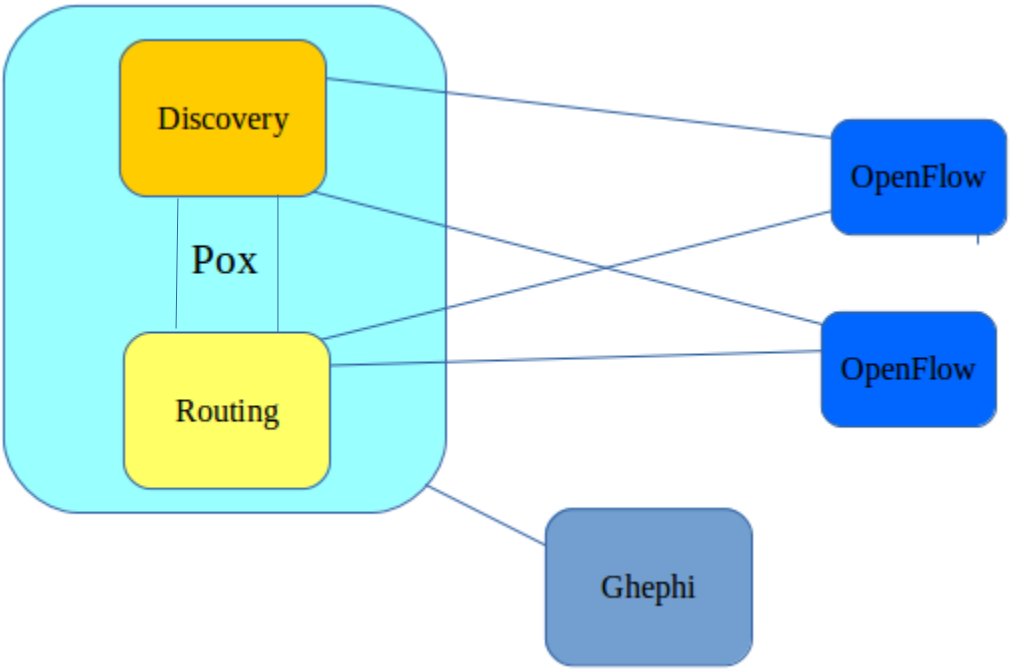
2-Architecture:



Our target Topology

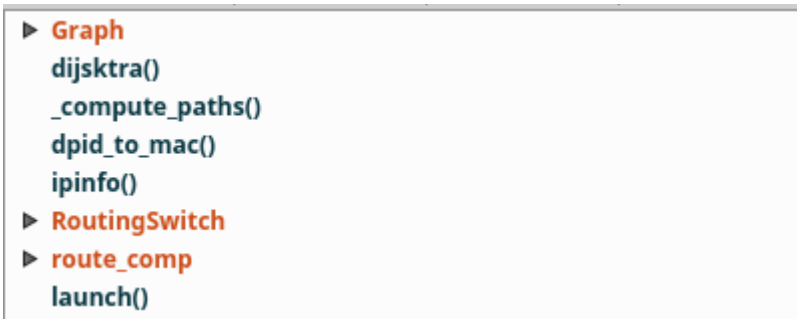
our project is consisting for three main comopenent:
-forwording.Routing :
This is our main programs whtich listen to events created by openflow.discovery, implements our routing algorithm and send rules to switches.
-openflow.discovery: This component sends specially-crafted LLDP messages out of OpenFlow switches so that it can discover the network topology. It raises events (which you can listen to) when links go up or down. More specifically, you can listen to `LinkEvent` events on `core.openflow_discovery`. When a link is detected, such an event is raised with the `.added` attribute set to True. When a link is detected as having been removed or failed, the `.removed` attribute is set to True. `LinkEvent` also has a .link attribute, which is a `Link` object, and a `port_for_dpid(<dpid>)` method (pass it the DPID of one end of the link and it will tell you the port used on that data-path)[Pox manual].

-Gephi: is a pretty awesome open-source, multi platform graph visualization/manipulation/analysis package. It has a plug-in for streaming graphs back and forth between it and something else over a network connection. The `gephi_topo` component uses this to provide visualization for switches, links, and (optionally) hosts detected by other POX components [pox manual].



Main components of our project

3- Code structure:



Functions and classes of our code

Graph: a class that represents our network topology
dijsktra(): fuction to calculate shortest path
_compute_paths(): uses the dijsktra
RoutingSwithc: a class represents our nodes objects
route_comp: a class where event are detected from discovery to create new nodes and links

	Time	Source	Destination	Protocol	Len	Info
643	19.565965994	00:00:00_00:00:02	Broadcast	OpenFlow	1…	Type: OFPT_PACKET_OUT
645	19.566280834	00:00:00_00:00:02		ARP	44	Who has 10.2.4.42? Tell 0.0.0.0
646	19.566297742	56:f7:54:27:6a:71		ARP	44	10.2.4.42 is at 56:f7:54:27:6a:71
647	19.566889924	56:f7:54:27:6a:71	00:00:00_00:00:02	OpenFlow	1…	Type: OFPT_PACKET_IN
674	20.600959205	00:00:00_00:00:01	Broadcast	OpenFlow	1…	Type: OFPT_PACKET_OUT
676	20.601225206	00:00:00_00:00:01		ARP	44	Who has 10.1.4.42? Tell 0.0.0.0
677	20.601243710	f2:8e:83:95:46:94		ARP	44	10.1.4.42 is at f2:8e:83:95:46:94
678	20.601731529	f2:8e:83:95:46:94	00:00:00_00:00:01	OpenFlow	1…	Type: OFPT_PACKET_IN

Messages when h1 ping h2 for the first time

```
mininet> dpctl dump-flows
*** s4 ------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
*** s1 ------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
*** s3 ------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
*** s2 ------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
mininet>
```

Table flow before running pox

```
*** s2 ------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=4.658s, table=0, n_packets=2, n_bytes=82, idle_age=1, prio
rity=65000,dl_dst=01:23:20:00:00:01,dl_type=0x88cc actions=CONTROLLER:65535
 cookie=0x0, duration=4.658s, table=0, n_packets=0, n_bytes=0, idle_age=4, ip,nw
_dst=10.1.0.0/16 actions=output:1
 cookie=0x0, duration=4.658s, table=0, n_packets=0, n_bytes=0, idle_age=4, ip,nw
_dst=10.3.0.0/16 actions=output:3
 cookie=0x0, duration=4.658s, table=0, n_packets=0, n_bytes=0, idle_age=4, ip,nw
_dst=10.4.0.0/16 actions=output:2
 cookie=0x0, duration=4.657s, table=0, n_packets=0, n_bytes=0, idle_age=4, prior
ity=32767,ip,nw_dst=10.2.0.0/16 actions=CONTROLLER:65535
 cookie=0x0, duration=4.657s, table=0, n_packets=0, n_bytes=0, idle_age=4, prior
ity=32767,ip,nw_dst=255.255.255.255 actions=output:4
```

s2 table flow after running Pox

These rules are sent by the controller at the first connection

```
*** s2 ------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=78.869s, table=0, n_packets=46, n_bytes=1886, idle_age=0,
priority=65000,dl_dst=01:23:20:00:00:01,dl_type=0x88cc actions=CONTROLLER:65535
 cookie=0x0, duration=78.869s, table=0, n_packets=2, n_bytes=196, idle_age=11, i
p,nw_dst=10.1.0.0/16 actions=output:1
 cookie=0x0, duration=78.869s, table=0, n_packets=0, n_bytes=0, idle_age=78, ip,
nw_dst=10.3.0.0/16 actions=output:3
 cookie=0x0, duration=78.869s, table=0, n_packets=0, n_bytes=0, idle_age=78, ip,
nw_dst=10.4.0.0/16 actions=output:2
 cookie=0x0, duration=78.868s, table=0, n_packets=1, n_bytes=98, idle_age=13, pr
iority=32767,ip,nw_dst=10.2.0.0/16 actions=CONTROLLER:65535
 cookie=0x0, duration=78.868s, table=0, n_packets=0, n_bytes=0, idle_age=78, pri
ority=32767,ip,nw_dst=255.255.255.255 actions=output:4
 cookie=0x0, duration=13.595s, table=0, n_packets=2, n_bytes=196, idle_age=11, i
p,nw_dst=10.2.4.42 actions=mod_dl_src:00:00:00:00:00:02,mod_dl_dst:ba:77:41:85:7
6:17,output:4
mininet>
```

s2 flow table after h1 pinging h2

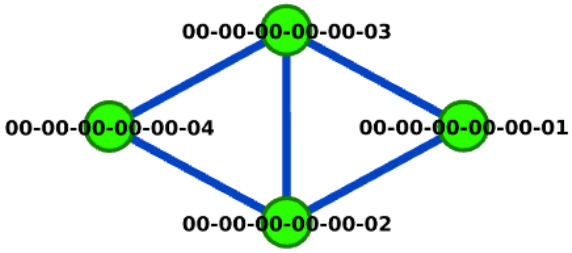
This rule is added by send_rewrite rule function(more details on code comments)

4-Tests:

I- Experience one:

here we made our tests without any changes to the graph

a- Graph:



Topology graph with macs of switches

b- Routing table:

```
[00-00-00-00-00-01/1]  is  defaultdict(<function <lambda> at 0x7f6d9a09ced8>, {[
00-00-00-00-00-03/3]: [00-00-00-00-00-01/1], [00-00-00-00-00-04/4]: [00-00-00-00
-00-02/2], [00-00-00-00-00-02/2]: [00-00-00-00-00-01/1]})
[00-00-00-00-00-02/2]  is  defaultdict(<function <lambda> at 0x7f6d9a09cf50>, {[
00-00-00-00-00-01/1]: [00-00-00-00-00-02/2], [00-00-00-00-00-04/4]: [00-00-00-00
-00-02/2], [00-00-00-00-00-03/3]: [00-00-00-00-00-02/2]})
[00-00-00-00-00-03/3]  is  defaultdict(<function <lambda> at 0x7f6d9a39a398>, {[
00-00-00-00-00-01/1]: [00-00-00-00-00-03/3], [00-00-00-00-00-04/4]: [00-00-00-00
-00-03/3], [00-00-00-00-00-02/2]: [00-00-00-00-00-03/3]})
[00-00-00-00-00-04/4]  is  defaultdict(<function <lambda> at 0x7f6d9a39a410>, {[
00-00-00-00-00-01/1]: [00-00-00-00-00-02/2], [00-00-00-00-00-02/2]: [00-00-00-00
-00-04/4], [00-00-00-00-00-03/3]: [00-00-00-00-00-04/4]})
```

Route table for each vertex in the network

for each node we have a dictionary where the key is the destination and the value is the previous node to this destination.

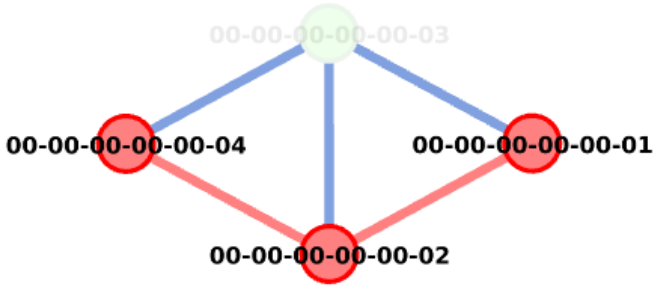
c- IP connectivity:

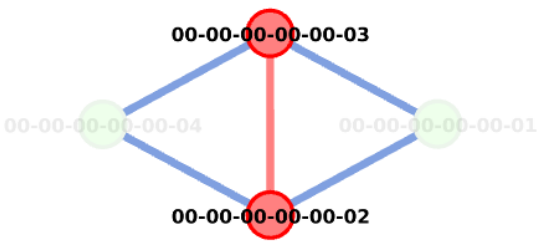
```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h3 h4 h2
h3 -> h1 h4 h2
h4 -> h1 h3 h2
h2 -> h1 h3 h4
*** Results: 0% dropped (12/12 received)
mininet>
```

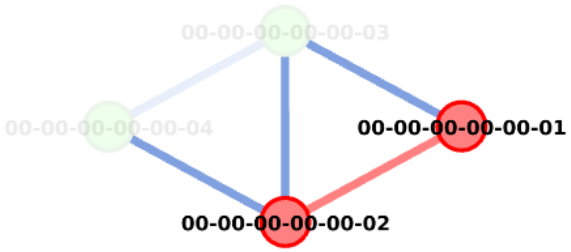
Ping test

we can see here that all the machine are reachable to each other

d- Paths:





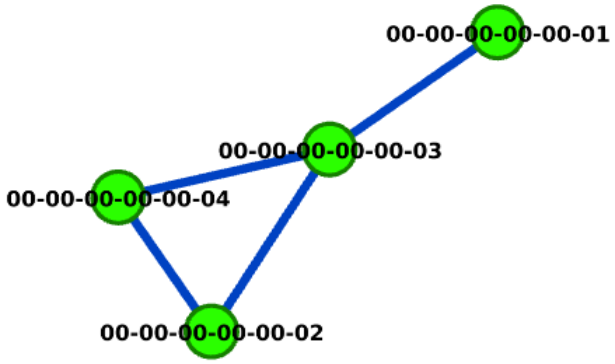


Some path shown with Gephi exactly the same as shown in route table above

II- Experience 2:

to the previous graph we run this command “link s2 s1 down” to make a rupture in the link between switch 1 and switch 2, and we observed how pox reacted to that change, we didn’t shutdown any thing everything was in real time.

a- Graph:



Our new graph with s1-- s2 link is down

b- Routing table:

```
[00-00-00-00-00-01/1]  is  defaultdict(<function <lambda> at 0x7f6d9a09ced8>, {
[00-00-00-00-00-02/2]: [00-00-00-00-00-03/3], [00-00-00-00-00-03/3]: [00-00-00-
00-00-01/1], [00-00-00-00-00-04/4]: [00-00-00-00-00-03/3]})
[00-00-00-00-00-02/2]  is  defaultdict(<function <lambda> at 0x7f6d9a09cf50>, {
[00-00-00-00-00-01/1]: [00-00-00-00-00-03/3], [00-00-00-00-00-04/4]: [00-00-00-
00-00-02/2], [00-00-00-00-00-03/3]: [00-00-00-00-00-02/2]})
[00-00-00-00-00-03/3]  is  defaultdict(<function <lambda> at 0x7f6d9a39a398>, {
[00-00-00-00-00-01/1]: [00-00-00-00-00-03/3], [00-00-00-00-00-04/4]: [00-00-00-
00-00-03/3], [00-00-00-00-00-02/2]: [00-00-00-00-00-03/3]})
[00-00-00-00-00-04/4]  is  defaultdict(<function <lambda> at 0x7f6d9a39a410>, {
[00-00-00-00-00-01/1]: [00-00-00-00-00-03/3], [00-00-00-00-00-02/2]: [00-00-00-
00-00-04/4], [00-00-00-00-00-03/3]: [00-00-00-00-00-04/4]})
```

New routing table in respond of new changes

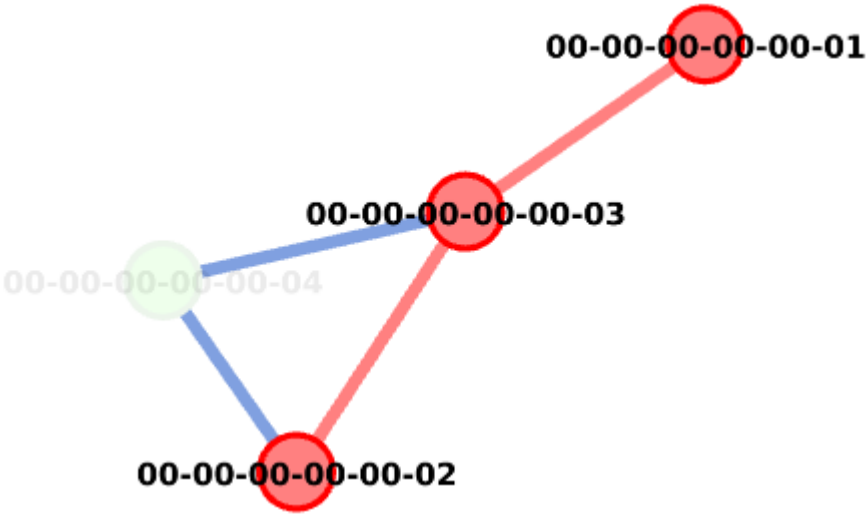
we notice here that previous node to node 2 is no longer node 1 but is node 3

c- IP connectivity:

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h3 h4 h2
h3 -> h1 h4 h2
h4 -> h1 h3 h2
h2 -> h1 h3 h4
*** Results: 0% dropped (12/12 received)
mininet>
```

Ping Test

d- Paths:

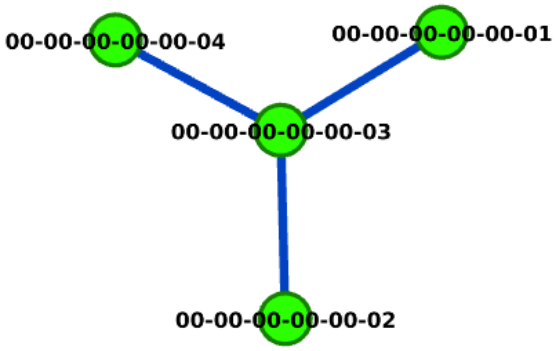


New path between vertex 2 and vertex 1

III- Experience 3:

Here in addition of link s1—s2 down we will shutdown also the link s2—s4

a- Graph:



New graph after link s2—s4 is down

-b Routing table:

```
[00-00-00-00-00-01/1]  is  defaultdict(<function <lambda> at 0x7f6d9a39a398>, {[
00-00-00-00-00-02/2]: [00-00-00-00-00-03/3], [00-00-00-00-00-03/3]: [00-00-00-00
-00-01/1], [00-00-00-00-00-04/4]: [00-00-00-00-00-03/3]})
[00-00-00-00-00-02/2]  is  defaultdict(<function <lambda> at 0x7f6d9a39a410>, {[
00-00-00-00-00-01/1]: [00-00-00-00-00-03/3], [00-00-00-00-00-03/3]: [00-00-00-00
-00-02/2], [00-00-00-00-00-04/4]: [00-00-00-00-00-03/3]})
[00-00-00-00-00-03/3]  is  defaultdict(<function <lambda> at 0x7f6d9a312a28>, {[
00-00-00-00-00-01/1]: [00-00-00-00-00-03/3], [00-00-00-00-00-04/4]: [00-00-00-00
-00-03/3], [00-00-00-00-00-02/2]: [00-00-00-00-00-03/3]})
[00-00-00-00-00-04/4]  is  defaultdict(<function <lambda> at 0x7f6d9a09cf50>, {[
00-00-00-00-00-01/1]: [00-00-00-00-00-03/3], [00-00-00-00-00-03/3]: [00-00-00-00
-00-04/4], [00-00-00-00-00-02/2]: [00-00-00-00-00-03/3]})
```

New routing table

We notice here that for node 4 previous node to node 2 is no longer node 2 but node 3.

c- IP connectivity:

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h3 h4 h2
h3 -> h1 h4 h2
h4 -> h1 h3 h2
h2 -> h1 h3 h4
*** Results: 0% dropped (12/12 received)
mininet>
```

d-Paths:

Some new paths in our topolgy

IV- Experience 4:

Here in addition of s2—s1 down, s2—s4 down, we will shutdown also link s4—s3, so we will get s4 out of the network

a- Graph:

New topology with s4 out of the network

b- Routing table:

```
[00-00-00-00-00-01/1]  is  defaultdict(<function <lambda> at 0x7f6d9a39a398>, {[
00-00-00-00-00-03/3]: [00-00-00-00-00-01/1], [00-00-00-00-00-02/2]: [00-00-00-00
-00-03/3]})
[00-00-00-00-00-02/2]  is  defaultdict(<function <lambda> at 0x7f6d9a39a410>, {[
00-00-00-00-00-03/3]: [00-00-00-00-00-02/2], [00-00-00-00-00-01/1]: [00-00-00-00
-00-03/3]})
[00-00-00-00-00-03/3]  is  defaultdict(<function <lambda> at 0x7f6d9a312a28>, {[
00-00-00-00-00-02/2]: [00-00-00-00-00-03/3], [00-00-00-00-00-01/1]: [00-00-00-00
-00-03/3]})
[00-00-00-00-00-04/4]  is  defaultdict(<function <lambda> at 0x7f6d9a09cf50>, {}
)
```

Routing table, no routes for s4

c- IP connectivity:

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h3 X h2
h3 -> h1 X h2
h4 -> X X X
h2 -> h1 h3 X
*** Results: 50% dropped (6/12 received)
mininet>
```

Ping test shows s4 not reachable

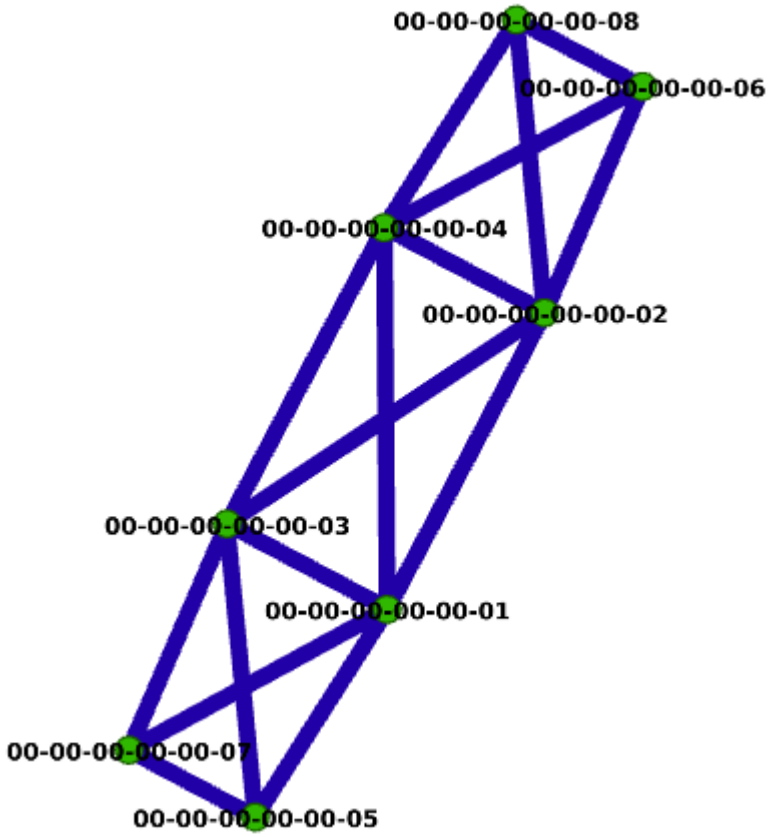
d- Paths:

The only path we got between vertex 1,2 and 3

V- Experience 5:

Here we will use a complex graph and we will see the performance giving by our routing module, we have created 8 vertex with many looped links. Test will be in route between s5 and s8.

a- Graph:



The new used topology graph

b- Routing table

```
[00-00-00-00-00-08/8]  is  defaultdict(<function <lambda> at 0x7f47e82a5938>, {[
00-00-00-00-00-02/2]: [00-00-00-00-00-08/8], [00-00-00-00-00-03/3]: [00-00-00-00
-00-02/2], [00-00-00-00-00-05/5]: [00-00-00-00-00-03/3], [00-00-00-00-00-06/6]:
[00-00-00-00-00-08/8], [00-00-00-00-00-01/1]: [00-00-00-00-00-02/2], [00-00-00-0
0-00-07/7]: [00-00-00-00-00-03/3], [00-00-00-00-00-04/4]: [00-00-00-00-00-08/8]}
)
```

s8 routing table

```
[00-00-00-00-00-05/5]  is  defaultdict(<function <lambda> at 0x7f47e82a5668>, {[
00-00-00-00-00-04/4]: [00-00-00-00-00-03/3], [00-00-00-00-00-01/1]: [00-00-00-00
-00-05/5], [00-00-00-00-00-06/6]: [00-00-00-00-00-02/2], [00-00-00-00-00-08/8]:
[00-00-00-00-00-02/2], [00-00-00-00-00-03/3]: [00-00-00-00-00-05/5], [00-00-00-0
0-00-07/7]: [00-00-00-00-00-05/5], [00-00-00-00-00-02/2]: [00-00-00-00-00-03/3]}
)
```

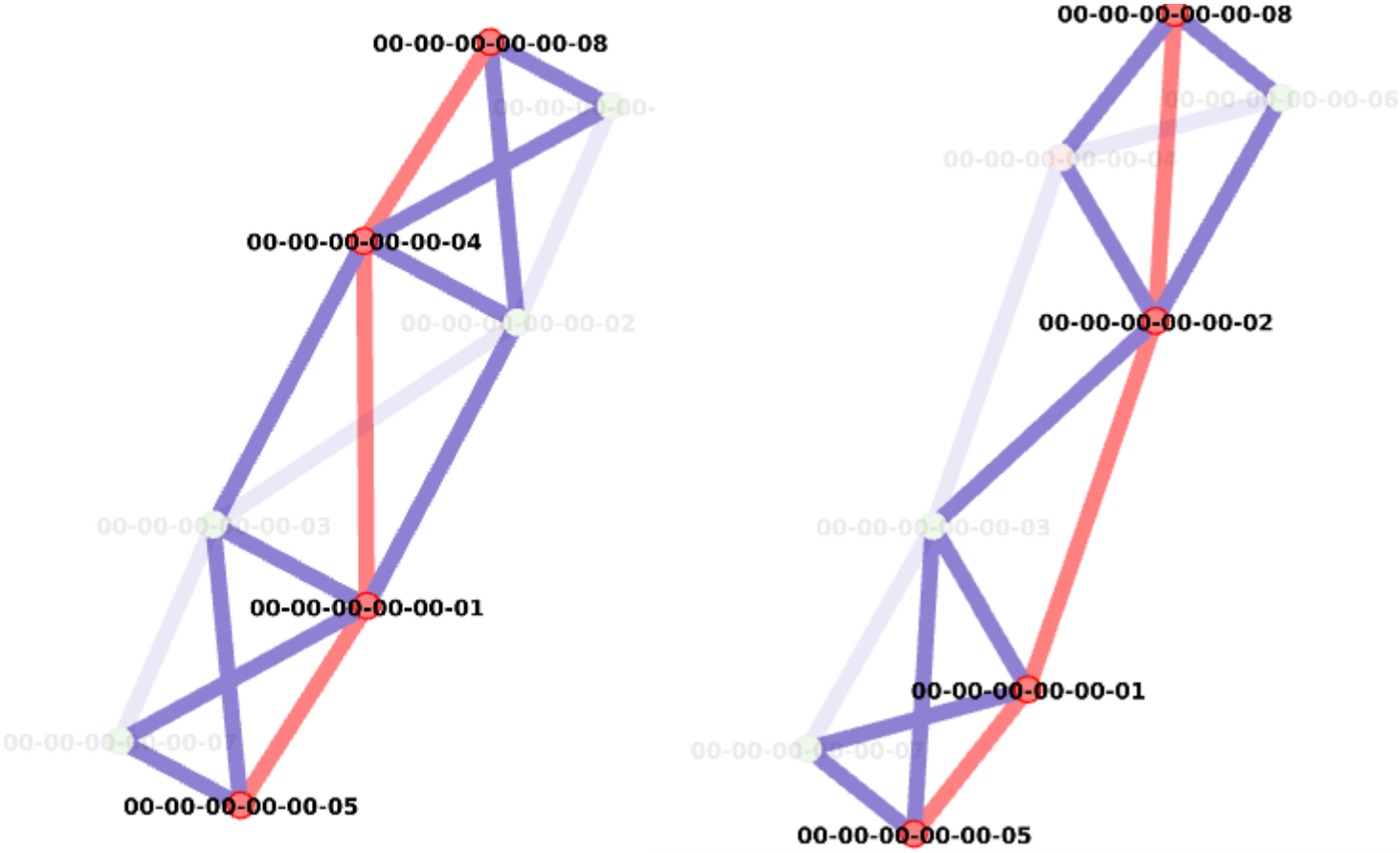
s5 routing table

c- IP connectivity:

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7 h8
h2 -> h1 h3 h4 h5 h6 h7 h8
h3 -> h1 h2 h4 h5 h6 h7 h8
h4 -> h1 h2 h3 h5 h6 h7 h8
h5 -> h1 h2 h3 h4 h6 h7 h8
h6 -> h1 h2 h3 h4 h5 h7 h8
h7 -> h1 h2 h3 h4 h5 h6 h8
h8 -> h1 h2 h3 h4 h5 h6 h7
*** Results: 0% dropped (56/56 received)
mininet>
```

Ping test succeeded

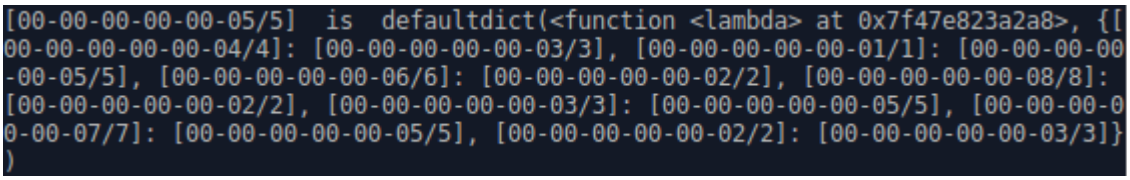
d-Paths:



On left path from s8 to s5 before make s1—s4 down, right new path s8 to s5 after change. Note here we may have different paths with the same cost, remember we have assumed that links have equal cost.

```
[00-00-00-00-00-08/8]  is  defaultdict(<function <lambda> at 0x7f47e823a230>, {[
00-00-00-00-00-02/2]: [00-00-00-00-00-08/8], [00-00-00-00-00-03/3]: [00-00-00-00
-00-02/2], [00-00-00-00-00-05/5]: [00-00-00-00-00-03/3], [00-00-00-00-00-06/6]:
[00-00-00-00-00-08/8], [00-00-00-00-00-01/1]: [00-00-00-00-00-02/2], [00-00-00-0
0-00-07/7]: [00-00-00-00-00-03/3], [00-00-00-00-00-04/4]: [00-00-00-00-00-08/8]}
)
```

S8 table after change



S5 table route after change

5-Conclusion:

We were able build and to test a modules that ensure connectivity between our network nodes, farther more it respond to real time rupture , and tries to find a new path. However there is more work to do, in Dijkstra algorithm link cost is an essential parameter, we assumed that links have equal cost which is not true in the real case, but we are able to do that using TCLink module of mininet. This is a future work.
Another idea to implement some rules that fit in the filtering context.
Another idea test mininet with real examples since it uses network name spaces we can build a real network, with many flexible options, test firewalls, applications behavior, customizing protocols headers, this will helps students to harden their skills with protocols in every TCP/IP layer
Another idea is to try mininet-wifi
the last thing is about the project: I think this project should be divided to many exercises, build a component from scratch is better and easier than reading and modifying someone's code.

6-Sources:

https://fr.wikipedia.org/wiki/Algorithme_de_Dijkstra
https://www.youtube.com/watch?v=pVfj6mxhdMw
http://csie.nqu.edu.tw/smallko/sdn/sdn.htm
https://noxrepo.github.io/pox-doc/html/
http://mininet.org/walkthrough/
https://www.youtube.com/watch?v=FyV4MoQ3T0I
https://www.youtube.com/watch?v=l25Ukkmk6Sk
https://gephi.org/
http://www.brianlinkletter.com/using-pox-components-to-create-a-software-defined-networking-application/
http://www.brianlinkletter.com/visualizing-software-defined-network-topologies-using-pox-and-gephi/