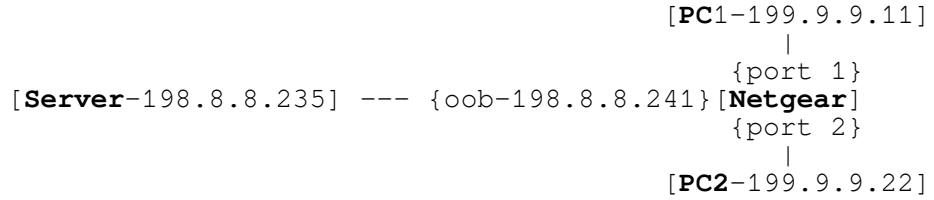
Dear Alessio, I meant to come back to you with a simple lab setup to demonstrate before now.



On an Ubuntu server (or Desktop) install the Ryu Controller.

```
~$ sudo apt-get install python3-ryu
```

Now setup OpenFlow on the switch and then disable it.

```
(OF-SW) #show openflow

Administrative Mode.......................... Enable
Operational Status........................... Enabled
Disable Reason............................... None
IP Address................................... 198.8.8.241
IP Mode...................................... ServicePort IP
Static IP Address............................ 0.0.0.0
OpenFlow Variant............................. OpenFlow 1.3
Passive Mode................................. Disable

(OF-SW) (Config)#no openflow enable
```

Run the Ryu controller with the REST API

```
~$ ryu-manager --app-lists ryu.app.simple_switch_13 ryu.app.ofctl_rest
```

The SDN Controller is now running with a RESTful interface on port 8080. Now enable OpenFlow on the Netgear switch (BTW My SDN Controller is on IP 198.8.8.235 and the serviceport on the Netgear is configured with IP 198.8.8.241. i.e. 198.8.8.0/24 is the management network and 199.9.9.0/24 and 2a99:9:9::/48 are the operational networks.

```
(OF-SW) (Config)#openflow enable
(OF-SW) (Config)#1970-01-01T01:30:53Z|00300|rconn|INFO|ofswitch<-
>tcp:198.8.8.235:6633: connecting...
1970-01-01T01:30:53Z|00301|rconn|INFO|ofswitch<->tcp:198.8.8.235:6633: connected
```

Get the switch Datapath ID (DPID) via the RESTful API. (Run this on any computer that is also on the 198.8.8.0/24 network. A second terminal on the same computer that is running the SDN controller is fine.

```
~$ curl -X GET http://198.8.8.235:8080/stats/switches; echo
[176199429686713]
```

Confirm it is the Netgear

```
~$ curl -X GET http://198.8.8.235:8080/stats/desc/176199429686713; echo
{"176199429686713": {"mfr_desc": "Netgear Inc.", "hw_desc": "Unknown",
"sw_desc": "12.0.7.10", "serial_num": "4G04745R000E6", "dp_desc": "M4300-28G
ProSAFE 24-port 1G and 2-port 10GBASE-T and 2-port 10G SFP+, 12.0.7.10,
B1.0.0.11"}}
```

Now extract the flows

```
~$ curl -X GET http://198.8.8.235:8080/stats/flow/176199429686713; echo
{"176199429686713": [{"priority": 0, "cookie": 0, "idle_timeout": 0,
"hard_timeout": 0, "byte_count": 34944, "duration_sec": 236, "duration_nsec":
737000000, "packet_count": 428, "length": 80, "flags": 0, "actions":
["OUTPUT:CONTROLLER"], "match": {}, "table_id": 0}, {"priority": 1, "cookie": 0,
"idle_timeout": 0, "hard_timeout": 0, "byte_count": 1408, "duration_sec": 232,
"duration_nsec": 49000000, "packet_count": 22, "length": 96, "flags": 0,
"actions": ["OUTPUT:2"], "match": {"in_port": 2, "dl_dst": "c4:71:fe:10:fe:00"},
"table_id": 0}, {"priority": 1, "cookie": 0, "idle_timeout": 0, "hard_timeout":
0, "byte_count": 1408, "duration_sec": 232, "duration_nsec": 49000000,
"packet_count": 22, "length": 96, "flags": 0, "actions": ["OUTPUT:1"], "match":
{"in_port": 1, "dl_dst": "00:1e:be:17:eb:9a"}, "table_id": 0}]}
```

Break these flows down.

Breaking this dictionary down

Flow 1: This flow OK, this is injected to the switch by the SDN controller to tell it where to send packets it doesn't know about.

```
{"priority": 0, "cookie": 0, "idle_timeout": 0, "hard_timeout": 0, "byte_count":
34944, "duration_sec": 236, "duration_nsec": 737000000, "packet_count": 428,
"length": 80, "flags": 0, "actions": ["OUTPUT:CONTROLLER"], "match": {},
"table_id": 0}
```

Flows 2: This flow is strange as the in_port in the match is is ("in_port": 2) and the action is ["OUTPUT:2"], i.e. the same port.

```
{"priority": 1, "cookie": 0, "idle_timeout": 0, "hard_timeout": 0, "byte_count":
1408, "duration_sec": 232, "duration_nsec": 49000000, "packet_count": 22,
"length": 96, "flags": 0, "actions": ["OUTPUT:2"], "match": {"in_port": 2,
"dl_dst": "c4:71:fe:10:fe:00"}, "table_id": 0},
```

This third flow also has match for port 1 and output for the same port 1.

```
{"priority": 1, "cookie": 0, "idle_timeout": 0, "hard_timeout": 0, "byte_count":
1408, "duration_sec": 232, "duration_nsec": 49000000, "packet_count": 22,
"length": 96, "flags": 0, "actions": ["OUTPUT:1"], "match": {"in_port": 1,
"dl_dst": "00:1e:be:17:eb:9a"}, "table_id": 0}]}
```

Why, well the following packets are sent from the Netgear switch to the SDN Controller.

```
packet in 176199429686713 00:1e:be:17:eb:9a 00:1e:be:17:eb:9a

packet in 176199429686713 c4:71:fe:10:fe:00 c4:71:fe:10:fe:00
```

To debug further edit the file: /usr/lib/python3/dist-packages/ryu/app/simple_switch_13.py by adding in the following lines to get more information:

At Line 87:

```
print(f"EVENT HANDLER: {datapath.id}, IN PORT: {in_port}, SRC_MAC: {eth.src},
DST_MAC: {eth.dst}")
```

Insert at line 60 (as a first line in the add_flow() method.

```
print(f"ADD FLOW: {datapath}, {priority}, {match}, {actions}, {buffer_id}")
```

Re-run the test and you will see the default flow injected as expected.

```
ADD FLOW: , 0, OFPMatch(oxm_fields={}),
[OFPActionOutput(len=16,max_len=65535,port=4294967293,type=0)], None
```

Now after this event (i.e. packet sent from the Netgear to the SDN Controller for a packet in on port 1 with a source and destination MAC the same, the SDN Controller responds with a corresponding flow. The same then happens for port 2. Therein lies the problem. Why is the netgear sending a packet received on a port with the same source and destination MAC addresses destined for output on the same port?

```
EVENT HANDLER: 176199429686713, IN PORT: 1, SRC_MAC: 00:1e:be:17:eb:9a, DST_MAC:
00:1e:be:17:eb:9a

ADD FLOW: , 1, OFPMatch(oxm_fields={'in_port': 1, 'eth_dst':
'00:1e:be:17:eb:9a'}), [OFPActionOutput(len=16,max_len=65509,port=1,type=0)],
267

EVENT HANDLER: 176199429686713, IN PORT: 2, SRC_MAC: c4:71:fe:10:fe:00, DST_MAC:
c4:71:fe:10:fe:00

ADD FLOW: , 1, OFPMatch(oxm_fields={'in_port': 2, 'eth_dst':
'c4:71:fe:10:fe:00'}), [OFPActionOutput(len=16,max_len=65509,port=2,type=0)],
274
```