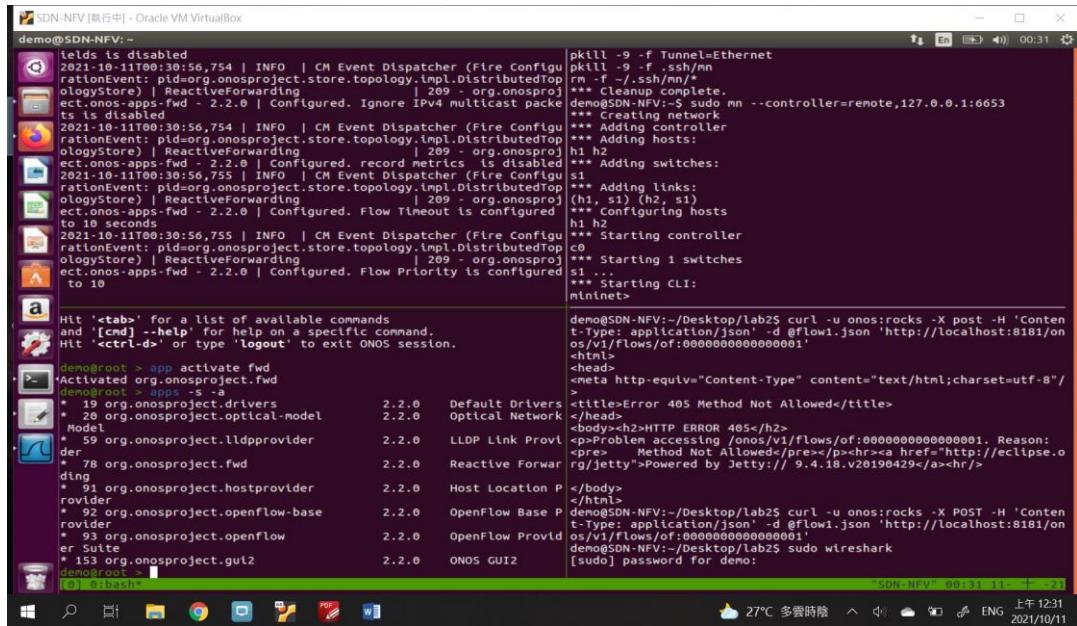


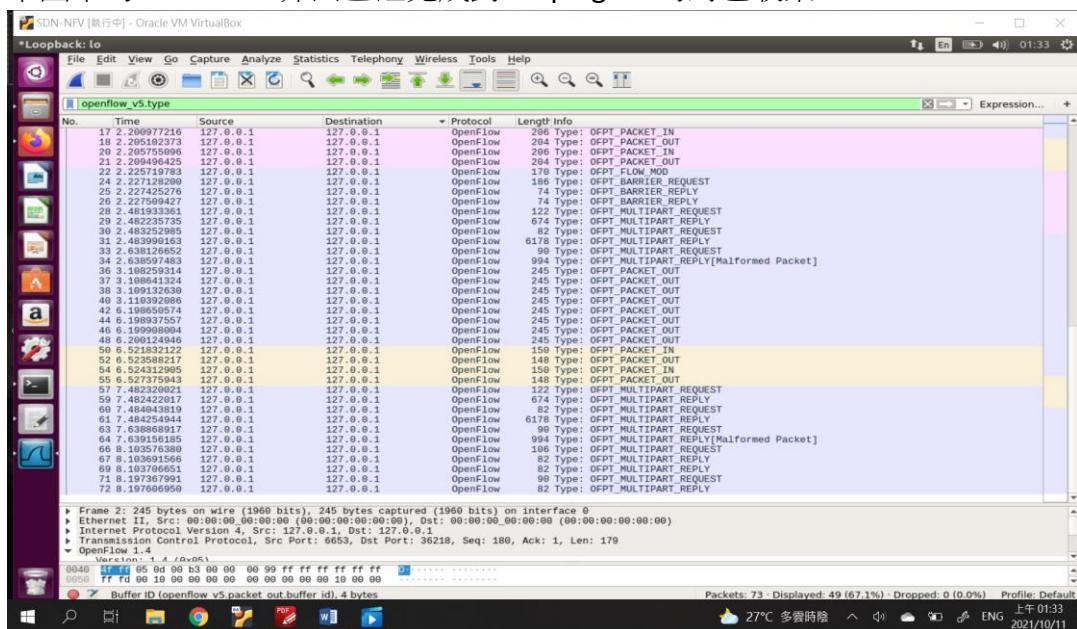
## SDN project2 report

### Part1

以下在完成 spec 中的 preparation 後的畫面(右上角為 mininet topology，左下角為 onos localhost)

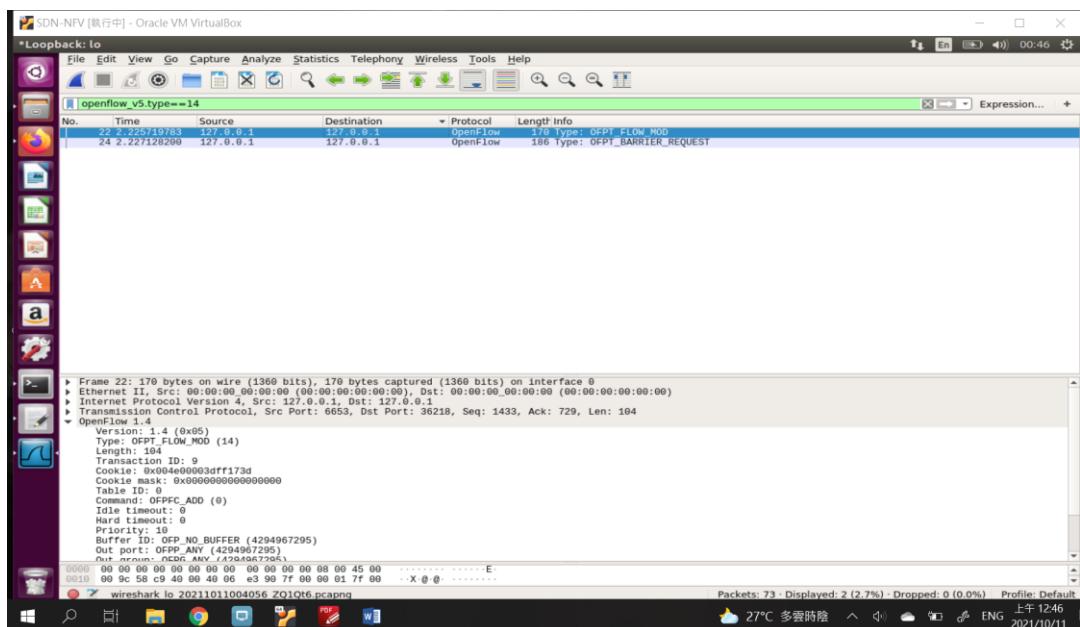
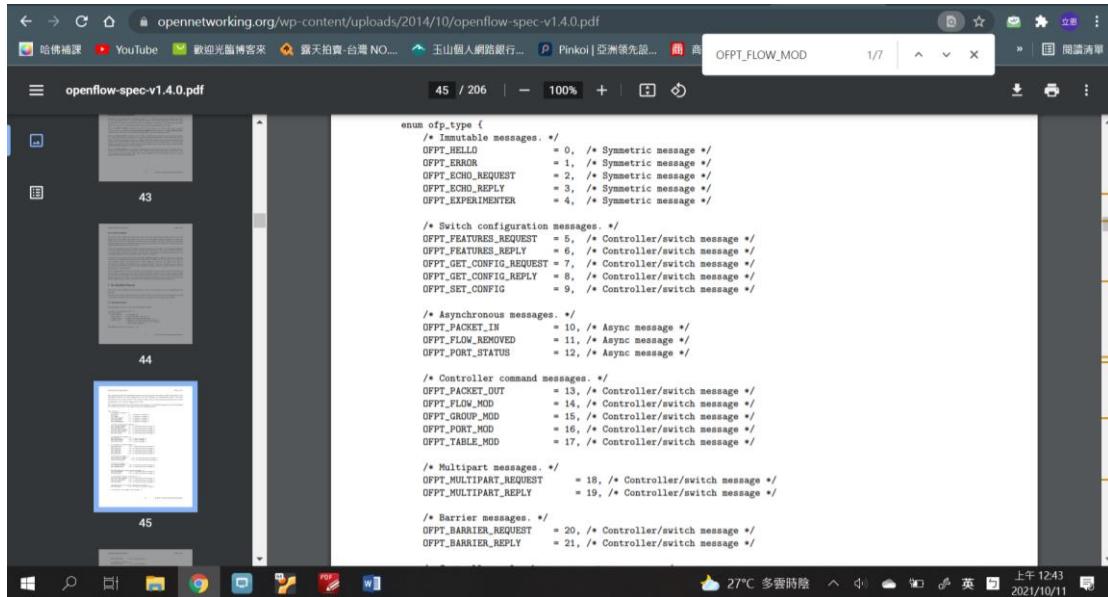


下圖中的 wireshark 介面已經完成對 h1 ping h2 的封包收集



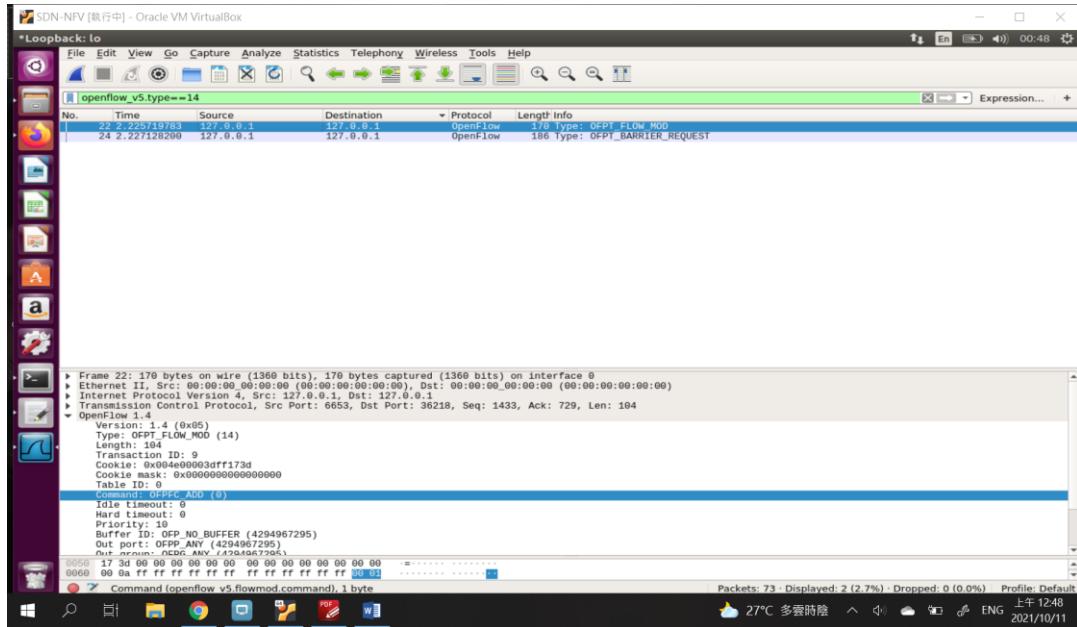
Q1 :

根據 Openflow1.4 版本的 spec，可知 OFPT\_FLOW\_MOD 的 type 對應到 14，因此用 wireshark 內建的 filter 搜尋 openflow\_v5.type==14 的方法尋找含有 OFPT\_FLOW\_MOD 的 packet

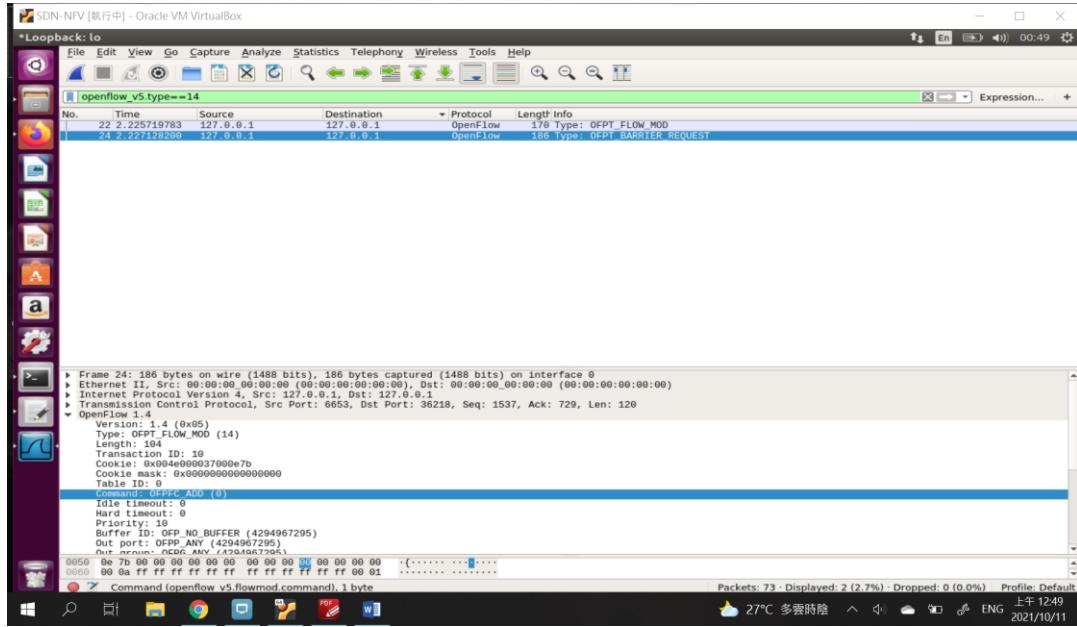


由上圖中可知，含有 OFPT\_FLOW\_MOD 的 packet 只有兩個，並且兩個 packet 的 command 皆為 OFPFC\_ADD，如下圖

第一個封包：



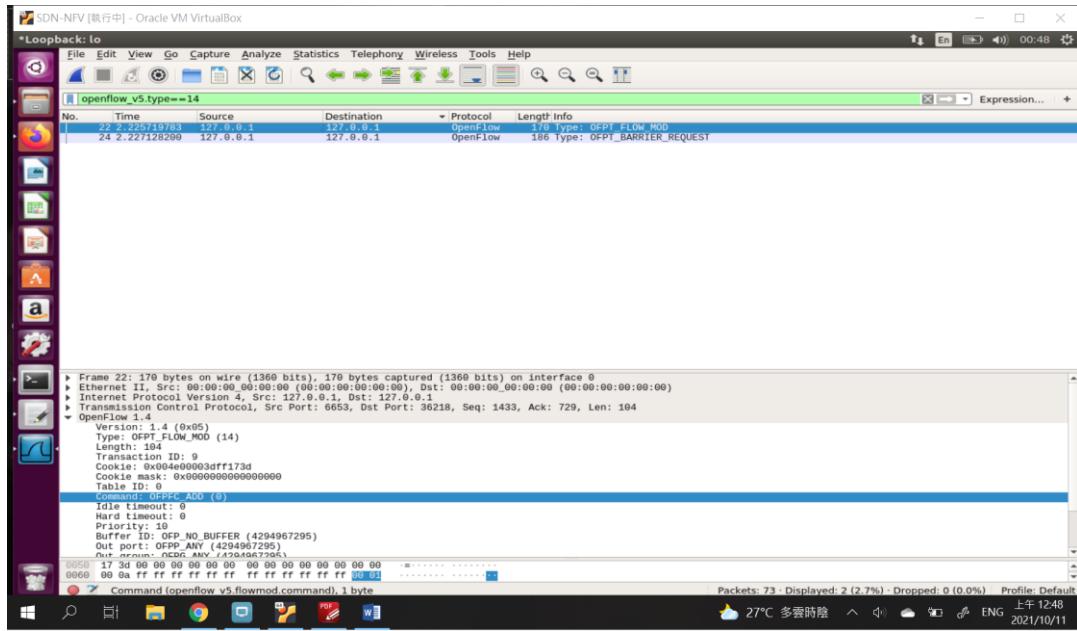
第二個封包



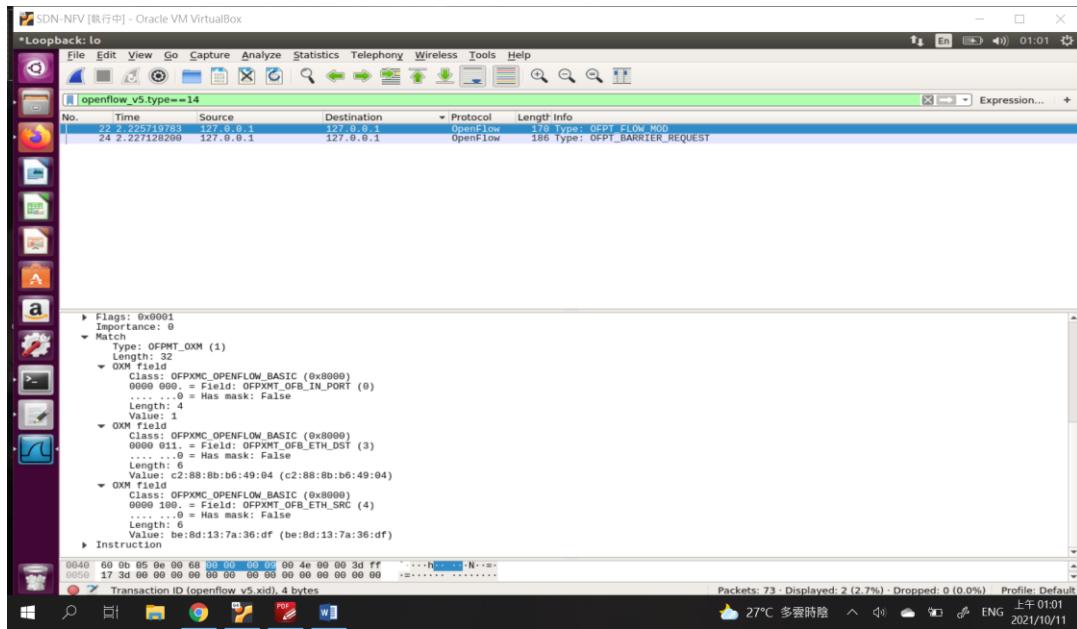
Q2:

根據 Q1，兩個封包的 match field 和 action 如下圖

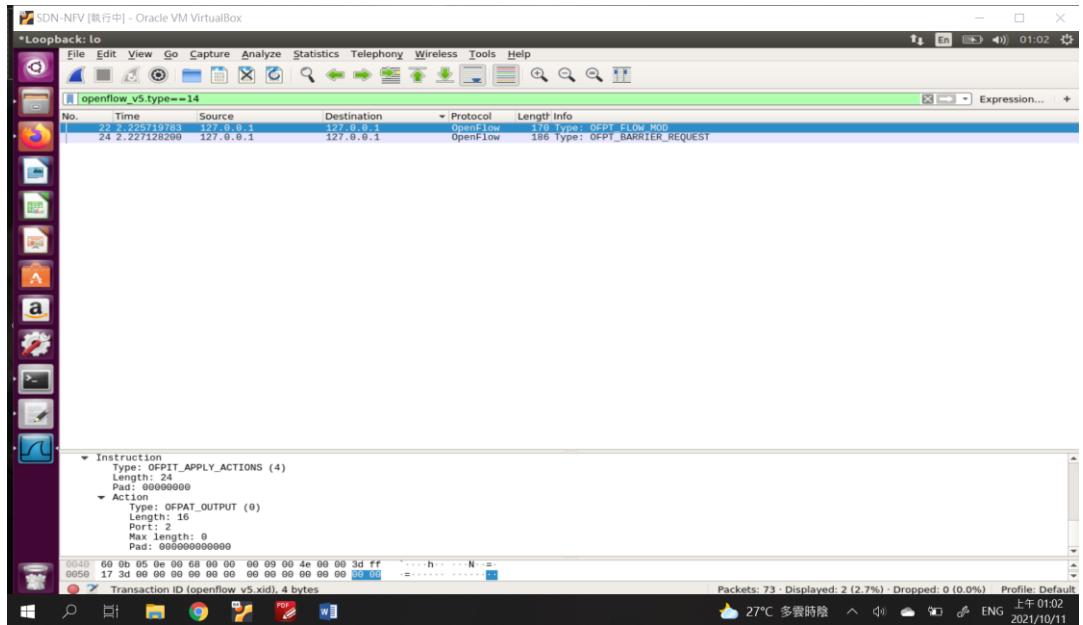
第一個封包：



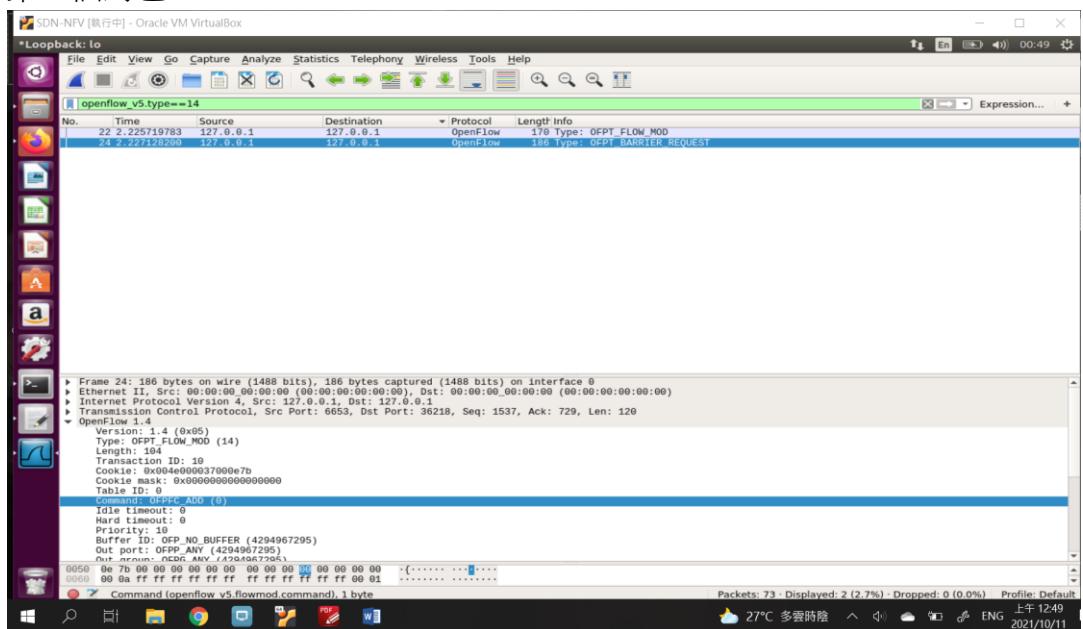
Match field :



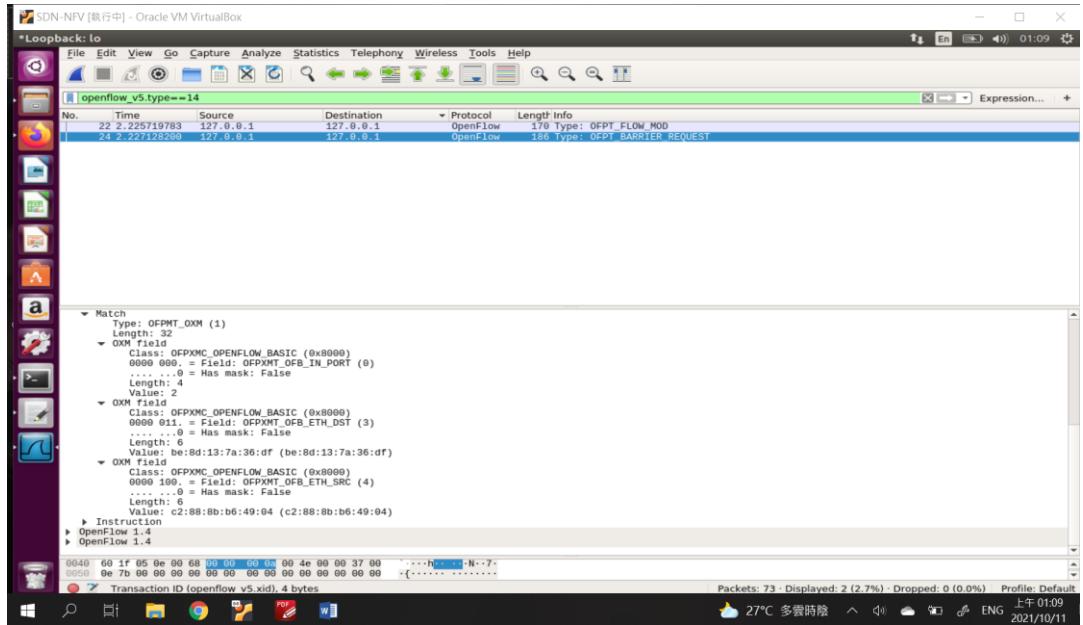
## Action :



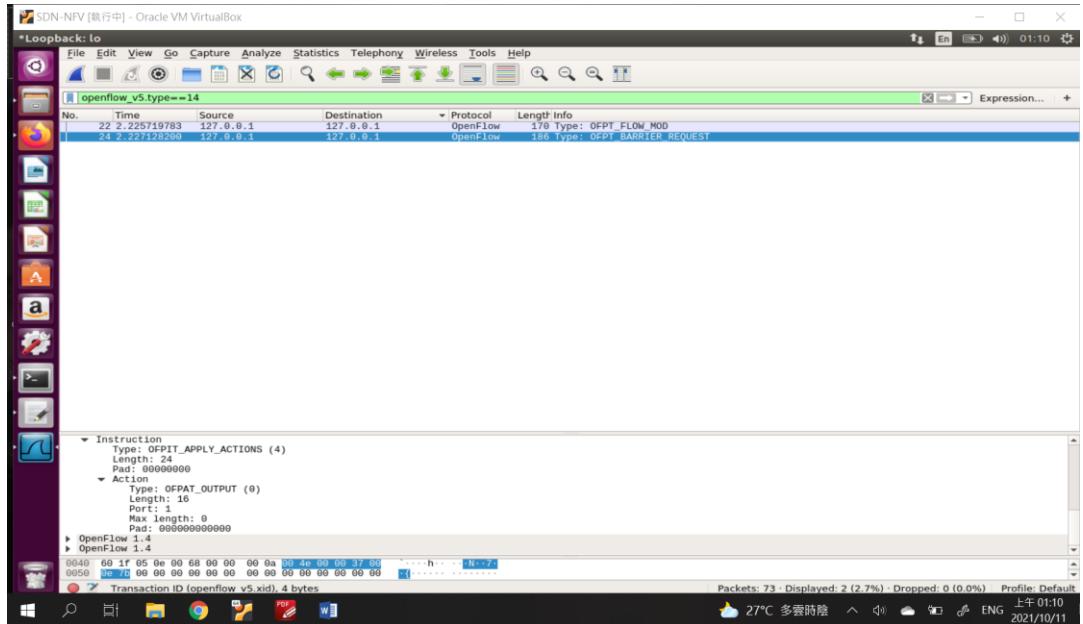
## 第二個封包 :



Match :



### Action :



Q3 :

根據下圖，s1 中共有四條 rule，各自的 flow rule timeout 如下，可知他們各自的 timeout(idle timeout 和 hard timeout)皆為 0

The screenshot shows the ONOS Flow Table for Device 0000000000000001. There are four flow entries listed:

STATE	PACKETS	DURATION	FLOW PRIORITY	TABLE NAME	SELECTOR	TREATMENT	APP NAME
Added	0	3,440	40000	0x100007a585b6f	Flow ID : 0x100007a585b6f State : Added Bytes : 0 Packets : 0 Duration : 3,430 Flow Priority : 40000 Table Name : 0 App Name : *core App ID : 1 Group ID : 0x0 Idle Timeout : 0 Hard Timeout : 0 Permanent : true	0x100007a585b6f	0x100007a585b6f
Added	4	3,440	40000	0x100007a585b6f	Flow ID : 0x100007a585b6f State : Added Bytes : 0 Packets : 0 Duration : 3,430 Flow Priority : 40000 Table Name : 0 App Name : *core App ID : 1 Group ID : 0x0 Idle Timeout : 0 Hard Timeout : 0 Permanent : true	0x100007a585b6f	0x100007a585b6f
Added	0	3,440	40000	0x100007a585b6f	Flow ID : 0x100007a585b6f State : Added Bytes : 0 Packets : 0 Duration : 3,430 Flow Priority : 40000 Table Name : 0 App Name : *core App ID : 1 Group ID : 0x0 Idle Timeout : 0 Hard Timeout : 0 Permanent : true	0x100007a585b6f	0x100007a585b6f
Added	4	3,428	5	0x100007a585b6f	Flow ID : 0x100007a585b6f State : Added Bytes : 0 Packets : 0 Duration : 3,430 Flow Priority : 40000 Table Name : 0 App Name : *core App ID : 1 Group ID : 0x0 Idle Timeout : 0 Hard Timeout : 0 Permanent : true	0x100007a585b6f	0x100007a585b6f

The screenshot shows the ONOS Flow Table for Device 0000000000000001. There are four flow entries listed:

STATE	PACKETS	DURATION	FLOW PRIORITY	TABLE NAME	SELECTOR	TREATMENT	APP NAME
Added	0	3,505	40000	0x10000ea6f4b8e	Flow ID : 0x10000ea6f4b8e State : Added Bytes : 168 Packets : 4 Duration : 3,500 Flow Priority : 40000 Table Name : 0 App Name : *core App ID : 1 Group ID : 0x0 Idle Timeout : 0 Hard Timeout : 0 Permanent : true	0x10000ea6f4b8e	0x10000ea6f4b8e
Added	4	3,505	40000	0x10000ea6f4b8e	Flow ID : 0x10000ea6f4b8e State : Added Bytes : 168 Packets : 4 Duration : 3,500 Flow Priority : 40000 Table Name : 0 App Name : *core App ID : 1 Group ID : 0x0 Idle Timeout : 0 Hard Timeout : 0 Permanent : true	0x10000ea6f4b8e	0x10000ea6f4b8e
Added	0	3,505	40000	0x10000ea6f4b8e	Flow ID : 0x10000ea6f4b8e State : Added Bytes : 168 Packets : 4 Duration : 3,500 Flow Priority : 40000 Table Name : 0 App Name : *core App ID : 1 Group ID : 0x0 Idle Timeout : 0 Hard Timeout : 0 Permanent : true	0x10000ea6f4b8e	0x10000ea6f4b8e
Added	4	3,493	5	0x10000ea6f4b8e	Flow ID : 0x10000ea6f4b8e State : Added Bytes : 168 Packets : 4 Duration : 3,500 Flow Priority : 40000 Table Name : 0 App Name : *core App ID : 1 Group ID : 0x0 Idle Timeout : 0 Hard Timeout : 0 Permanent : true	0x10000ea6f4b8e	0x10000ea6f4b8e

SDN-NFV [执行中] - Oracle VM VirtualBox

ONOS — Mozilla Firefox

ONOS

localhost:8181/onos/ui/#/flow?devid=oF:00000000000000000001

Flows for Device of:0000000000000001 (4 Total)

STATE	PACKETS	DURATION	FLOW PRIORITY	TABLE NAME	SELECTOR	TREATMENT	APP NAME
Added	0	3,535	40000				
Added	4	3,535	40000				
Added	0	3,535	40000				
Added	4	3,523	5				

**0x10000946555a**

Flow ID : 0x10000946555a  
State : Added  
Bytes : 0  
Packets : 0  
Duration : 3,525  
Flow Priority : 40000  
Table Name : 0  
App Name : \*core  
App ID : 1  
Group ID : 0x0  
Idle Timeout : 0  
Hard Timeout : 0  
Permanent : true

Selector

ETH\_TYPE : ETH\_TYPE:lldp

27°C 多雲時陰 上午 01:39 ENG 2021/10/11

SDN-NFV [执行中] - Oracle VM VirtualBox

ONOS — Mozilla Firefox

ONOS

localhost:8181/onos/ui/#/flow?devid=oF:0000000000000001

Flows for Device of:0000000000000001 (4 Total)

STATE	PACKETS	DURATION	FLOW PRIORITY	TABLE NAME	SELECTOR	TREATMENT	APP NAME
Added	0	3,560	40000				
Added	4	3,560	40000				
Added	0	3,560	40000				
Added	4	3,548	5				

**0x10000021b41dc**

Flow ID : 0x10000021b41dc  
State : Added  
Bytes : 392  
Packets : 4  
Duration : 3,548  
Flow Priority : 5  
Table Name : 0  
App Name : \*core  
App ID : 1  
Group ID : 0x0  
Idle Timeout : 0  
Hard Timeout : 0  
Permanent : true

Selector

ETH\_TYPE : ETH\_TYPE:ipv4

27°C 多雲時陰 上午 01:39 ENG 2021/10/11

## Part 2

在完成 part2 的準備工作後，測試 arping 和 ping 的結果，如下圖，可知目前 ARP packet 和 IP packet 皆無法在 h1 和 h2 之間流通

The screenshot shows a terminal window titled "SDN-NFV [執行中] - Oracle VM VirtualBox". The window displays the following log output:

```
2021-10-12T15:12:28,632 | INFO | onos-of-worker-1 | OFChannelHandler | Processing 0 pending port status messages for 00:00:01:00:00:00:00:01
2021-10-12T15:12:28,658 | INFO | onos-topo-build-1 | TopologyManager | 190 - org.onosproject.onos-core-net - 2.2.0 | Topology DefaultTopology@time=1082515389919, creationTime=1634022748574, computeCost=16893302, clusters=1, devices=1, links=0) changed
2021-10-12T15:12:28,714 | INFO | onos-of-worker-1 | OpenFlowControllerImpl$OpenFlowSwitchAgent | 203 - org.onosproject.onos-protocols-openflow-ctl | Transitioned switch 00:00:00:00:00:00:01 to MASTER
2021-10-12T15:12:28,729 | INFO | onos-topo-build-2 | TopologyManager | 190 - org.onosproject.onos-core-net - 2.2.0 | Topology DefaultTopology@time=1082516092979, creationTime=1634022748685, computeCost=84042, clusters=1, devices=1, links=0) changed
2021-10-12T15:12:29,521 | INFO | onos-of-event-stats-3 | DistributionGroupStore | 189 - org.onosproject.onos-core-dispatcher | Group AUDIT: Setting device of 00:00:00:00:00:00 initial AUDIT completed
*** Configuring hosts
*** Starting controller c0
*** Starting 1 switches
*** Starting CLI:
mininet> h1 arping h2
ARPING 10.0.0.2 from 10.0.0.1 h1-eth0
^CSent 6 probes (6 broadcast(s))
Received 0 response(s)
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable
From 10.0.0.1 icmp_seq=2 Destination Host Unreachable
From 10.0.0.1 icmp_seq=3 Destination Host Unreachable
^C
--- 10.0.0.2 ping statistics ---
5 packets transmitted, 0 received, +3 errors, 100% packet loss, time 4125ms
pipe 4
mininet>
```

Below the log, the terminal prompt is "demo@root: ~ /Desktop/lab2\$". The user then runs "apps -s -a" which lists various ONOS components. Finally, "demo@root: ~ /Desktop/lab2\$ sudo" is entered.

首先處理 ARP 的部分，將 flow1.json 改寫成下列兩個 json file(為容易在 onos 上辨別 rule，每個 rule 各自採用不同的 priority)，左方的 file 是讓 h2 到 h1 的 ARP packet 可以通行，而右方的 file 作用於相反的方向，讓 h1 的 ARP packet 可以到達 h2。因為 spec 指定 match field 需要有 ARP 的 ethernet type，因此新增了 ethernet type 為 ARP(0X0806)的 criteria

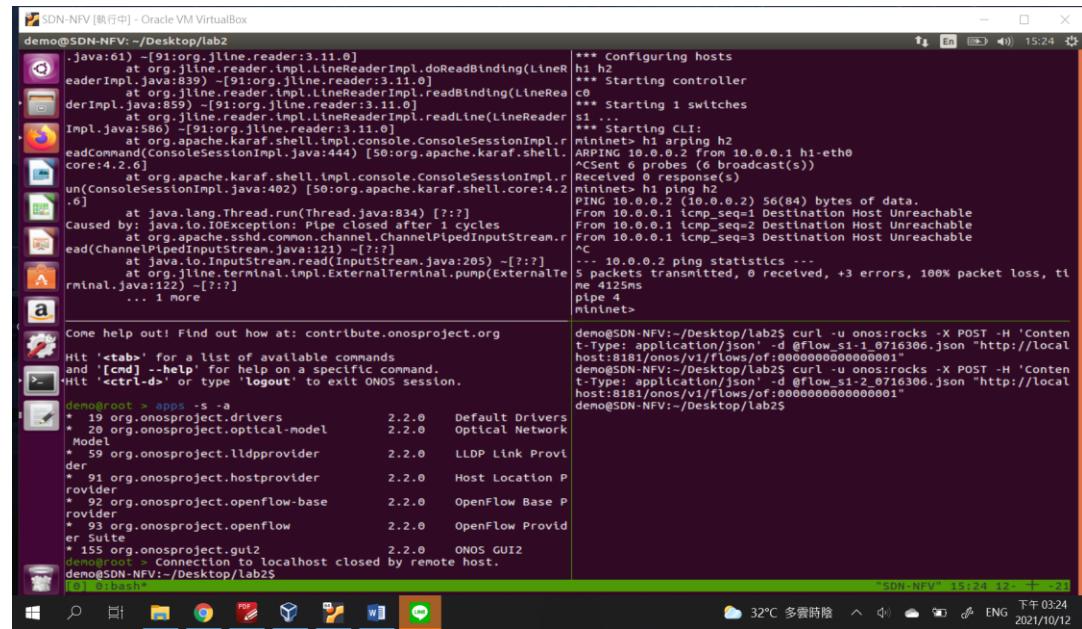
The screenshot shows two side-by-side text editors, both titled "Text Editor". The left editor contains the following JSON configuration for flow rule s1-1:

```
{
  "priority": 60001,
  "timeout": 0,
  "isPermanent": true,
  "selected": true,
  "criteria": [
    {
      "type": "ETH_TYPE",
      "ethType": "0x0806"
    },
    {
      "type": "IN_PORT",
      "port": "2"
    }
  ],
  "treatment": [
    {
      "instructions": [
        {
          "type": "OUTPUT",
          "port": "1"
        }
      ]
    }
  ]
}
```

The right editor contains the following JSON configuration for flow rule s1-2:

```
{
  "priority": 60002,
  "timeout": 0,
  "isPermanent": true,
  "selected": false,
  "criteria": [
    {
      "type": "ETH_TYPE",
      "ethType": "0x0806"
    },
    {
      "type": "IN_PORT",
      "port": "1"
    }
  ],
  "treatment": [
    {
      "instructions": [
        {
          "type": "OUTPUT",
          "port": "2"
        }
      ]
    }
  ]
}
```

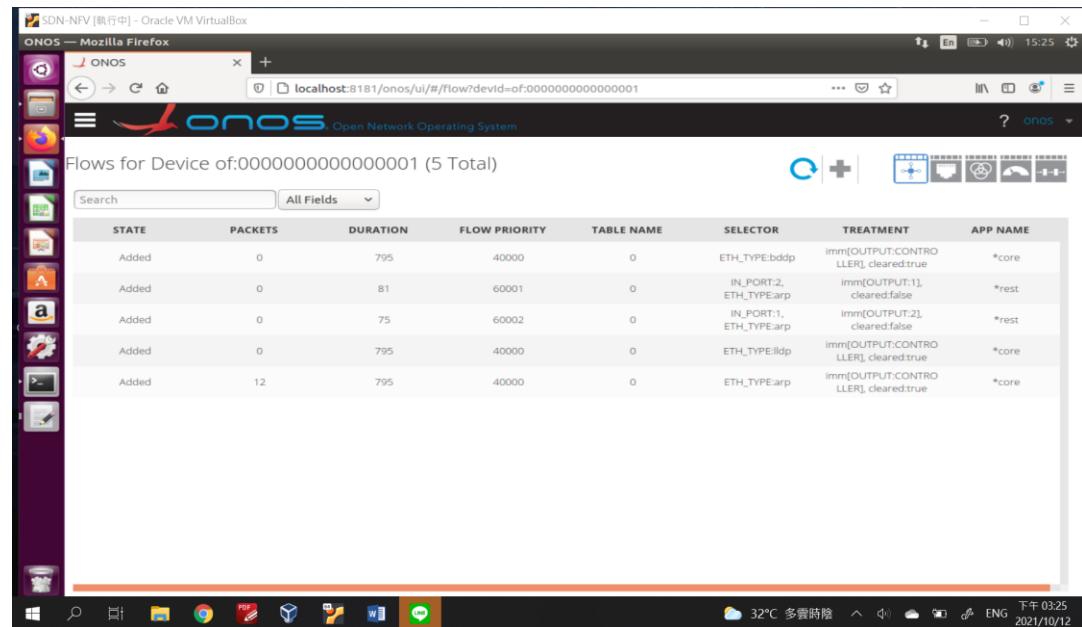
將兩個 json file 用 curl 傳輸到 ONOS，可看見在 ONOS 上已經有我們指定的 flow(priority 為 60001 和 60002)



```

demo@SDN-NFV:~/Desktop/lab2
[...]
*** Configuring hosts
h1 h2
*** Starting controller
controller
*** Starting 1 switches
...
*** Starting CLI:
mininet> h1 arping h2
ARPING 10.0.0.2 from 10.0.0.1 h1-eth0
^CSENT 6 probes (0 broadcast(s))
RECV 0 responses(s)
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable
From 10.0.0.1 icmp_seq=2 Destination Host Unreachable
From 10.0.0.1 icmp_seq=3 Destination Host Unreachable
^C
- 10.0.0.2 ping statistics --
5 packets transmitted, 0 received, +3 errors, 100% packet loss, time 4125ms
pipe 4
mininet>
[...]
Come help out! Find out how at: contribute.onosproject.org
Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.
Hit '<ctrl-d>' or type 'logout' to exit ONOS session.

demo@root > apps -s -a
* 19 org.onosproject.drivers          2.2.0  Default Drivers
* 49 org.onosproject.optical-model   2.2.0  Optical Network Model
* 59 org.onosproject.lldpprovider    2.2.0  LLDP Link Provider
* 91 org.onosproject.hostprovider    2.2.0  Host Location Provider
* 92 org.onosproject.openflow-base   2.2.0  OpenFlow Base Provider Suite
* 93 org.onosproject.openflow        2.2.0  OpenFlow Provider
* 155 org.onosproject.gui2           2.2.0  ONOS GUI2
demo@root > Connection to localhost closed by remote host.
demo@SDN-NFV:~/Desktop/lab2$
```



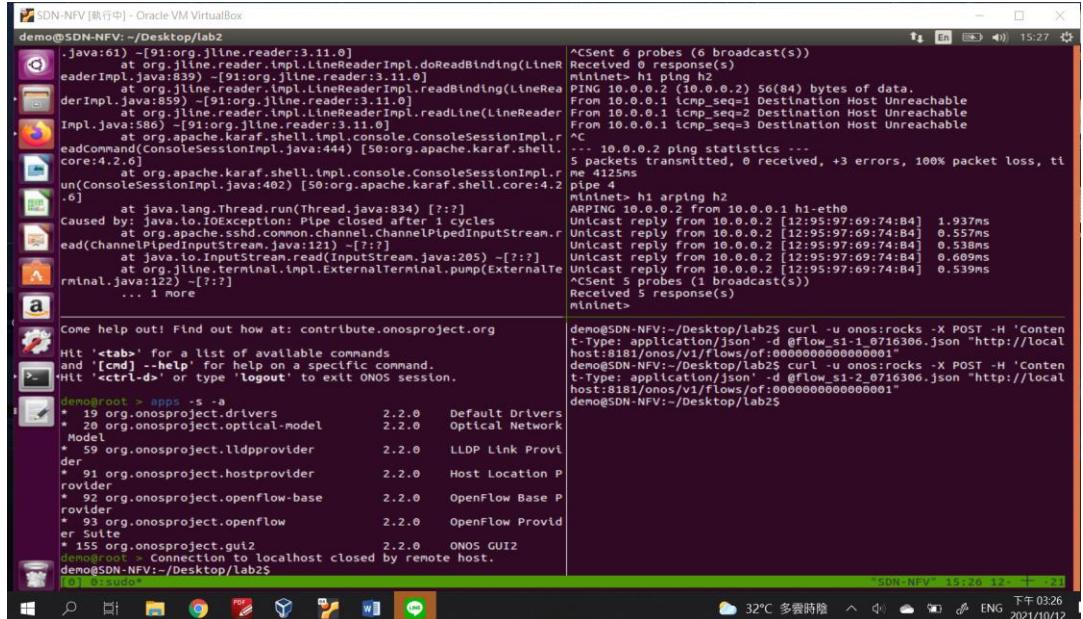
ONOS — Mozilla Firefox

Flows for Device of:0000000000000001 (5 Total)

STATE	PACKETS	DURATION	FLOW PRIORITY	TABLE NAME	SELECTOR	TREATMENT	APP NAME
Added	0	795	40000	0	ETH_TYPE:bddp	imm[OUTPUT:CONTROLLER], cleared:true	*core
Added	0	81	60001	0	IN_PORT:2, ETH_TYPE:arp	imm[OUTPUT:1], cleared:false	*rest
Added	0	75	60002	0	IN_PORT:1, ETH_TYPE:arp	imm[OUTPUT:2], cleared:false	*rest
Added	0	795	40000	0	ETH_TYPE:ldp	imm[OUTPUT:CONTROLLER], cleared:true	*core
Added	12	795	40000	0	ETH_TYPE:arp	imm[OUTPUT:CONTROLLER], cleared:true	*core

嘗試使用 h1 arping h2 和 h2 arping h1 得結果如下，可知 h1 和 h2 皆可成功使用 arping 找到對方

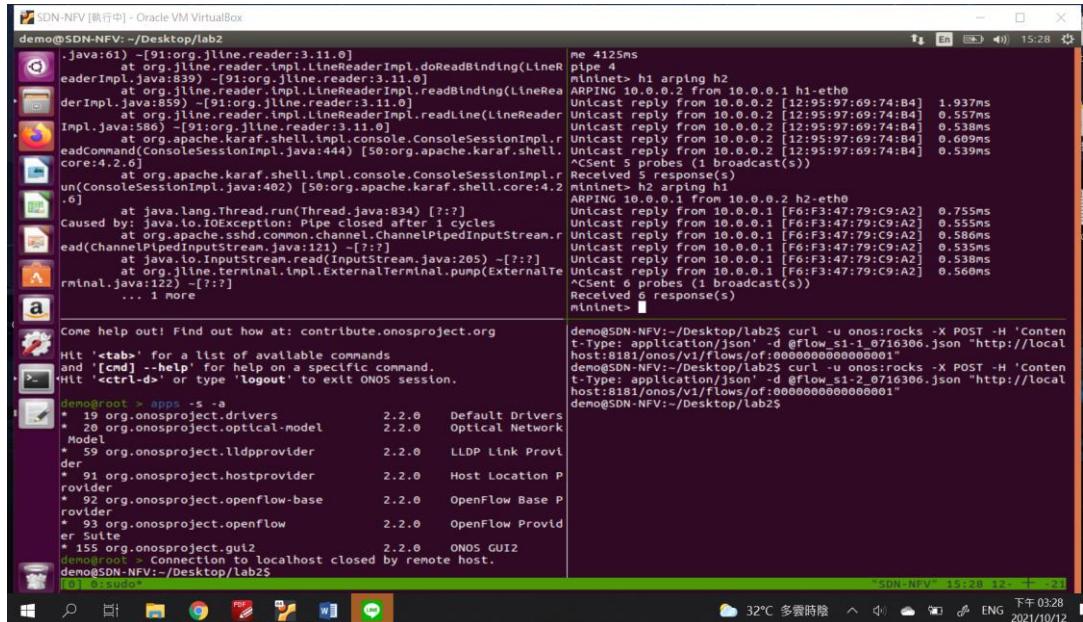
### h1 arping h2:



```
demo@SDN-NFV: ~/Desktop/lab2
[...]
^Csent 6 probes (6 broadcast(s))
Received 0 response(s)
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable
From 10.0.0.1 icmp_seq=2 Destination Host Unreachable
From 10.0.0.1 icmp_seq=3 Destination Host Unreachable
^C
sent 10.0.0.2 ping statistics ---
5 packets transmitted, 0 received, +3 errors, 100% packet loss, time
   4125ms
pipe 4
mininet> h1 arping h2
ARPING 10.0.0.1 from 10.0.0.1 h1-eth0
  Unicast reply from 10.0.0.2 [12:95:97:69:74:B4] 1.937ms
  Unicast reply from 10.0.0.2 [12:95:97:69:74:B4] 0.557ms
  Unicast reply from 10.0.0.2 [12:95:97:69:74:B4] 0.538ms
  Unicast reply from 10.0.0.2 [12:95:97:69:74:B4] 0.609ms
  Unicast reply from 10.0.0.2 [12:95:97:69:74:B4] 0.539ms
^Csent 5 probes (1 broadcast(s))
Received 5 response(s)
mininet>
```

demogSN-NFV:-/Desktop/lab2\$ curl -u onos:rocks -X POST -H 'Content-Type: application/json' -d @flow\_s1-1\_0716306.json "http://localhost:8181/onos/v1/flows/of:0000000000000001"
demogSN-NFV:-/Desktop/lab2\$ curl -u onos:rocks -X POST -H 'Content-Type: application/json' -d @flow\_s1-2\_0716306.json "http://localhost:8181/onos/v1/flows/of:0000000000000001"
demogSN-NFV:-/Desktop/lab2\$

### h2 arping h1:



```
demo@SDN-NFV: ~/Desktop/lab2
[...]
me 4125ms
mininet> h1 arping h2
ARPING 10.0.0.2 from 10.0.0.1 h1-eth0
  Unicast reply from 10.0.0.2 [12:95:97:69:74:B4] 1.937ms
  Unicast reply from 10.0.0.2 [12:95:97:69:74:B4] 0.557ms
  Unicast reply from 10.0.0.2 [12:95:97:69:74:B4] 0.538ms
  Unicast reply from 10.0.0.2 [12:95:97:69:74:B4] 0.609ms
  Unicast reply from 10.0.0.2 [12:95:97:69:74:B4] 0.539ms
^Csent 5 probes (1 broadcast(s))
Received 5 response(s)
mininet> h2 arping h1
ARPING 10.0.0.1 from 10.0.0.2 h2-eth0
  Unicast reply from 10.0.0.1 [F6:F3:47:79:C9:A2] 0.755ms
  Unicast reply from 10.0.0.1 [F6:F3:47:79:C9:A2] 0.555ms
  Unicast reply from 10.0.0.1 [F6:F3:47:79:C9:A2] 0.535ms
  Unicast reply from 10.0.0.1 [F6:F3:47:79:C9:A2] 0.535ms
  Unicast reply from 10.0.0.1 [F6:F3:47:79:C9:A2] 0.538ms
^Csent 6 probes (1 broadcast(s))
Received 6 response(s)
mininet>
```

demogSN-NFV:-/Desktop/lab2\$ curl -u onos:rocks -X POST -H 'Content-Type: application/json' -d @flow\_s1-1\_0716306.json "http://localhost:8181/onos/v1/flows/of:0000000000000001"
demogSN-NFV:-/Desktop/lab2\$ curl -u onos:rocks -X POST -H 'Content-Type: application/json' -d @flow\_s1-2\_0716306.json "http://localhost:8181/onos/v1/flows/of:0000000000000001"
demogSN-NFV:-/Desktop/lab2\$

接下來完成 IPv4 packet 的部分，下圖為兩個 json file  
 和前兩個負責 ARP packet 的 file 的不同處在於，flow 的篩選標準需要從 ethernet type 改成 IPv4 destination，因此 match field 的 ethType 需要改成 IPV4\_DST，並各自根據進入 switch 的 port 和 IPv4 destination 判斷 flow 的方向，並且使用 priority 為 60003 和 60004 以和前兩個 flow rule 區別

```

SDN-NFV (執行中) - Oracle VM VirtualBox
Text Editor
* flow_s1-3_0716306.json (-/Desktop/lab2) - gedit
Save
{
  "priority": 60003,
  "timeout": 0,
  "isPermanent": true,
  "selector": {
    "criteria": [
      {
        "type": "IPV4_DST",
        "ip": "10.0.0.2"
      },
      {
        "type": "IN_PORT",
        "port": "2"
      }
    ],
    "treatment": [
      {
        "instructions": [
          {
            "type": "OUTPUT",
            "port": "1"
          }
        ]
      }
    ]
  }
}

SDN-NFV (執行中) - Oracle VM VirtualBox
Text Editor
* flow_s1-4_0716306.json (-/Desktop/lab2) - gedit
Save
{
  "priority": 60004,
  "timeout": 0,
  "isPermanent": true,
  "selector": {
    "criteria": [
      {
        "type": "IPV4_DST",
        "ip": ""
      },
      {
        "type": "IN_PORT",
        "port": "1"
      }
    ],
    "treatment": [
      {
        "instructions": [
          {
            "type": "OUTPUT",
            "port": "2"
          }
        ]
      }
    ]
  }
}

```

由下圖中可知，h1 連接到 s1-eth1 且 h2 連接到 s1-eth2，而 s1-eth1 使用的 port 為 1、s1-eth2 使用的 port 為 2，因此上圖中左方的 file 是 h2 到 h1 的 flow、右方的 file 是 h1 到 h2 的 flow

```

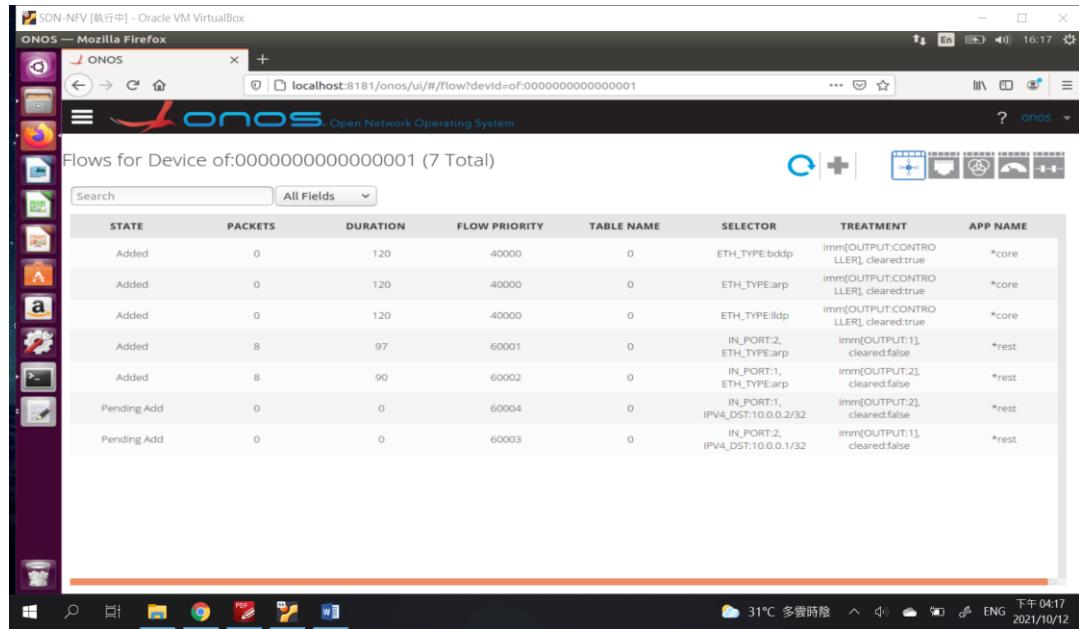
inet6 addr: fe80::1095:97ff:fe69:74b4/64 Scope:Link
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:1926 errors:0 dropped:1880 overruns:0 frame:0
TX packets:26 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:266806 (266.8 KB) TX bytes:1608 (1.6 KB)

lo      Link encap:Local Loopback
inet addr:127.0.0.1 Mask:255.0.0.0
inet6 addr: ::1/128 Scope:Host
UP LOOPBACK RUNNING MTU:65536 Metric:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

mininet> links
h1-eth0<->s1-eth1 (OK OK)
h2-eth0<->s1-eth2 (OK OK)
mininet> ports
s1 lo:0 s1-eth1:1 s1-eth2:2
mininet>

```

一樣用 curl 將兩個 file 傳送到 ONOS 後，可看見 ONOS 上已有 flow rule 存在



但之後無論是使用 h1 ping h2 或 h2 ping h1 都無法成功，根據 openflow spec 中的描述，原因是因為 IPV4\_DST 有 dependency，需要同時有 ETH\_TYPE="0x0800" 這個 criteria 存在

Table 11: Required match fields.

All match fields have different size, prerequisites and masking capability, as specified in Table 12. If not explicitly specified in the field description, each field type refer to the outermost occurrence of the field in the packet headers.

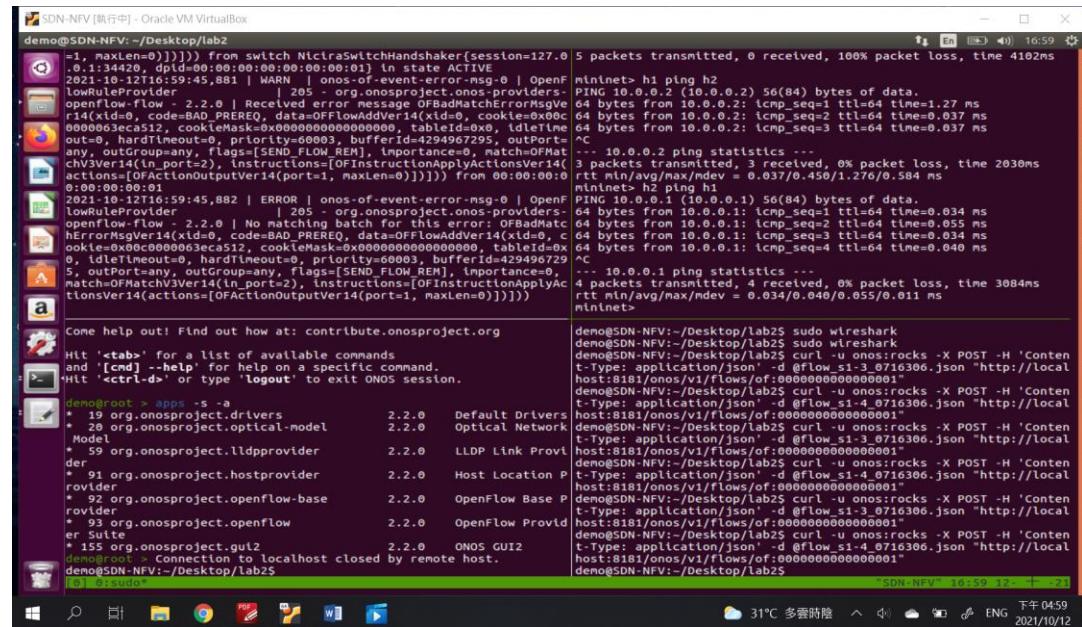
Field	Bits	Mask	Pre-requisite	Description
OXM_OF_IN_PORT	32	No	None	Ingress port. Numerical representation of incoming port, starting at 1. This may be a physical or switch-defined logical port.
OXM_OF_IN_PHY_PORT	32	No	IN_PORT present	Physical port. In <code>ofp_packet_in</code> messages, underlying physical port when packet received on a logical port.
OXM_OF_METADATA	64	Yes	None	Table metadata. Used to pass information between tables.
OXM_OF_ETH_DST	48	Yes	None	Ethernet destination MAC address.
OXM_OF_ETH_SRC	48	Yes	None	Ethernet source MAC address.
OXM_OF_ETH_TYPE	16	No	None	Ethernet type of the OpenFlow packet payload, after VLAN tags.
OXM_OF_VLAN_VID	12+1	Yes	None	VLAN-ID from 802.1Q header. The CFI bit indicates the presence of a valid VLAN-ID, see below.
OXM_OF_VLAN_PCP	3	No	VLAN.VID!=NONE	VLAN-PCP from 802.1Q header.
OXM_OF_IP_DSCP	6	No	ETH_TYPE=0x0800 or ETH_TYPE=0x86dd	Diff Serv Code Point (DSCP). Part of the IPv4 ToS field or the IPv6 Traffic Class field.
OXM_OF_IP_ECN	2	No	ETH_TYPE=0x0800 or ETH_TYPE=0x86dd	ECN bits of the IP header. Part of the IPv4 ToS field or the IPv6 Traffic Class field.
OXM_OF_IP_PROTO	8	No	ETH_TYPE=0x0800 or ETH_TYPE=0x86dd	IPv4 or IPv6 protocol number.
OXM_OF_IPV4_SRC	32	Yes	ETH_TYPE=0x0800	IPv4 source address. Can use subnet mask or arbitrary bitmask
OXM_OF_IPV4_DST	32	Yes	ETH_TYPE=0x0800	IPv4 destination address. Can use subnet mask or arbitrary bitmask

Table 12 – Continued on next page

因此在下圖中，可看見黑色外框中的 flow rule 因為沒有加上 ETH\_TYPE 作為 criteria，因此不會有 packet 使用(持續在 pending add 狀態)，而黃色外框中的 rule 因為有加上 ETH\_TYPE，因此會有 flow 經過(從 pending add 轉換為 added 的狀態)

STATE	PACKETS	DURATION	FLOW PRIORITY	TABLE NAME	SELECTOR	TREATMENT	APP NAME
Added	0	2,230	40000	0	ETH_TYPE:bdpp	imrn[OUTPUT:CONTROLLER], cleared:true	*core
Added	0	2,230	40000	0	ETH_TYPE:arp	imrn[OUTPUT:CONTROLLER], cleared:true	*core
Added	0	773	60004	0	ETH_TYPE:ip4, IPV4_DST:10.0.0.2/32	imrn[OUTPUT:2], cleared:false	*rest
Added	0	777	60003	0	ETH_TYPE:ip4, IPV4_DST:10.0.0.1/32	imrn[OUTPUT:1], cleared:false	*rest
Added	0	2,230	40000	0	ETH_TYPE:lldp	imrn[OUTPUT:CONTROLLER], cleared:true	*core
Added	27	2,207	60001	0	IN_PORT:2, ETH_TYPE:arp	imrn[OUTPUT:1], cleared:false	*rest
Added	27	2,200	60002	0	IN_PORT:1, ETH_TYPE:arp	imrn[OUTPUT:2], cleared:false	*rest
Pending Add	0	0	60004	0	IN_PORT:1, IPV4_DST:10.0.0.2/32	imrn[OUTPUT:2], cleared:false	*rest
Pending Add	0	0	60004	0	IPV4_DST:10.0.0.2/32	imrn[OUTPUT:2], cleared:false	*rest
Pending Add	0	0	60003	0	IPV4_DST:10.0.0.1/32	imrn[OUTPUT:1], cleared:false	*rest
Pending Add	0	0	60003	0	IN_PORT:2, IPV4_DST:10.0.0.1/32	imrn[OUTPUT:1], cleared:false	*rest

放入 ETH\_TYPE 作為 criteria 後，可看見 h1 ping h2 和 h2 ping h1 都成功



```
SDN-NFV [執行中] - Oracle VM VirtualBox
demo@SDN-NFV: ~/Desktop/lab2
[...]
=1, maxLen=0)]))) from switch NiciraSwitchHandshaker(session=127.0
2021-10-12T16:59:45,881 | WARN | onos-of-event-error-msg-0 | OpenFlow
lowRuleProvider      | 295 - org.onosproject.onos.providers-
r14(xid=0, code=BAD_PREREQ, data=OffFlowAddVer14(xid=0, cookie=0x0c
0000003eca512, cookieMask=0x0000000000000000, tableId=0x0, idleTime
out=0, hardTimeout=0, priority=60003, bufferId=4294967295, outPort
any, outgroup=any, flags=[SEND_FLOW_REM], importance=0, matchOfMatchV3Ver14(in_port=2), instructions=[OFInstructionApplyActionsVer14(actions=[OFActionOutputVer14(port=1, maxLen=0))]) from 00:00:00:0
0:00:00:00:00:00
2021-10-12T16:59:45,882 | ERROR | onos-of-event-error-msg-0 | OpenFlow
lowRuleProvider      | 295 - org.onosproject.onos.providers-
openflow-flow - 2.2.0 | No matching batch for this error: OFBadMatch
hErrorMsgVer14(xid=0, code=BAD_PREREQ, data=OffFlowAddVer14(xid=0, c
ookie=0x0c0000003eca512, cookieMask=0x0000000000000000, tableId=0x
0, idleTimeout=0, hardTimeout=0, priority=60003, bufferId=429496729
5, outPort=any, outgroup=any, flags=[SEND_FLOW_REM], importance=0,
matchOfMatchV3Ver14(in_port=2), instructions=[OFInstructionApplyActionsVer14(actions=[OFActionOutputVer14(port=1, maxLen=0))]) from 00:00:0
0:00:00:00:00:00
[...]
Come help out! Find out how at: contribute.onosproject.org
Hlt 'xtab' for a list of available commands
and 'fcmd --help' for help on a specific command.
Hlt '<ctrl-d>' or type 'logout' to exit ONOS session.

demo@root > apps -s -a
+ 19 org.onosproject.drivers          2.2.0  Default Drivers
+ 20 org.onosproject.optical-model    2.2.0  Optical Network
Mode
* 59 org.onosproject.lldpprovider     2.2.0  LLDP Link Provider
* 91 org.onosproject.hostprovider     2.2.0  Host Location Provider
* 92 org.onosproject.openflow-base    2.2.0  OpenFlow Base Provider
* 93 org.onosproject.openflow         2.2.0  OpenFlow Provider Suite
* 155 org.onosproject.gui2            2.2.0  ONOS GUI2
demo@root > Connection to localhost closed by remote host.
demo@SDN-NFV:~/Desktop/lab2$
```

5 packets transmitted, 0 received, 100% packet loss, time 4102ms  
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=1 ttl=64 time=1.27 ms  
64 bytes from 10.0.0.2: icmp\_seq=2 ttl=64 time=0.031 ms  
64 bytes from 10.0.0.2: icmp\_seq=3 ttl=64 time=0.037 ms  
--- 10.0.0.2 ping statistics ---  
3 packets transmitted, 3 received, 0% packet loss, time 2030ms  
rtt min/avg/max/mdev = 0.037/0.450/1.276/0.584 ms  
mininet>  
mininet> h2 ping h1  
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.  
64 bytes from 10.0.0.1: icmp\_seq=1 ttl=64 time=0.034 ms  
64 bytes from 10.0.0.1: icmp\_seq=2 ttl=64 time=0.055 ms  
64 bytes from 10.0.0.1: icmp\_seq=3 ttl=64 time=0.034 ms  
64 bytes from 10.0.0.1: icmp\_seq=4 ttl=64 time=0.040 ms  
--- 10.0.0.1 ping statistics ---  
4 packets transmitted, 4 received, 0% packet loss, time 3084ms  
rtt min/avg/max/mdev = 0.034/0.040/0.055/0.011 ms  
mininet>

demo@SDN-NFV:~/Desktop/lab2\$ sudo wireshark
demo@SDN-NFV:~/Desktop/lab2\$ sudo wireshark
demo@SDN-NFV:~/Desktop/lab2\$ curl -u onos:rocks -X POST -H 'Content-Type: application/json' -d @flow\_s1-3.0716306.json "http://localhost:8181/onos/v1/flows/of:0000000000000001"
demo@SDN-NFV:~/Desktop/lab2\$ curl -u onos:rocks -X POST -H 'Content-Type: application/json' -d @flow\_s1-4.0716306.json "http://localhost:8181/onos/v1/flows/of:0000000000000001"
demo@SDN-NFV:~/Desktop/lab2\$ curl -u onos:rocks -X POST -H 'Content-Type: application/json' -d @flow\_s1-5.0716306.json "http://localhost:8181/onos/v1/flows/of:0000000000000001"
demo@SDN-NFV:~/Desktop/lab2\$ curl -u onos:rocks -X POST -H 'Content-Type: application/json' -d @flow\_s1-6.0716306.json "http://localhost:8181/onos/v1/flows/of:0000000000000001"
demo@SDN-NFV:~/Desktop/lab2\$ curl -u onos:rocks -X POST -H 'Content-Type: application/json' -d @flow\_s1-7.0716306.json "http://localhost:8181/onos/v1/flows/of:0000000000000001"
demo@SDN-NFV:~/Desktop/lab2\$ curl -u onos:rocks -X POST -H 'Content-Type: application/json' -d @flow\_s1-8.0716306.json "http://localhost:8181/onos/v1/flows/of:0000000000000001"
demo@SDN-NFV:~/Desktop/lab2\$ curl -u onos:rocks -X POST -H 'Content-Type: application/json' -d @flow\_s1-9.0716306.json "http://localhost:8181/onos/v1/flows/of:0000000000000001"
demo@SDN-NFV:~/Desktop/lab2\$ curl -u onos:rocks -X POST -H 'Content-Type: application/json' -d @flow\_s1-10.0716306.json "http://localhost:8181/onos/v1/flows/of:0000000000000001"
demo@SDN-NFV:~/Desktop/lab2\$

SDN-NFV 16:59:12 + .23

31°C 多雲時陰 ^ ◄ ◂ ENG 下午 04:59 2021/10/12

## Part3

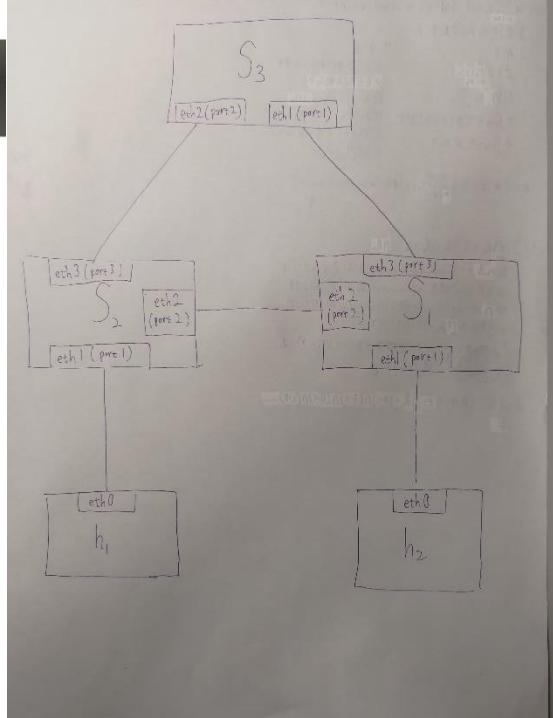
開啟 ONOS 後，確認無其他多餘的 app 被開啟

The screenshot shows two terminal windows side-by-side. The left window displays a Java stack trace from org.jline.reader, indicating multiple threads (h1, h2, s1, s2, s3) are reading from a pipeline. The right window shows the ONOS command-line interface (CLI) running on a host named 'demo'. The CLI output includes network creation commands like 'mn --custom=topo\_0710306.py --topo=mytopo --controller=remote,lp=127.0.0.1:6653' and configuration steps for hosts and switches. The bottom status bar shows system information including temperature (28°C), battery level (21%), and date/time (2021/10/12).

因為 broadcast storm 是起因於 ARP packet 的 broadcast 導致，若多個 switch 相連並且形成迴圈(loop)，則會發生 broadcast storm

以下是我設計的 topology，為方便設計，只使用 3 個 switch 形成一個 loop，code 及示意圖如下

The screenshot shows a dual-terminal window. The left terminal is a code editor displaying Python code for a Mininet topology named 'topo\_0716306.py'. The code defines a 'MyTopo' class that adds two hosts ('h1', 'h2') and three switches ('s1', 's2', 's3'). It connects each host to one switch and each switch to the others in a circular fashion. The right terminal shows the ONOS CLI running on a host named 'demo'. The CLI output shows the network configuration, including the addition of three switches and three hosts. The bottom status bar shows system information including temperature (28°C), battery level (21%), and date/time (2021/10/12).



```

*** Adding switches:
s1 s2 s3
*** Adding links:
(h1, s1) (h2, s2) (s1, s2) (s2, s3) (s3, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Starting CLI:
mininet> links
h1-eth0<->s1-eth1 (OK OK)
h2-eth0<->s2-eth1 (OK OK)
s1-eth2<->s2-eth2 (OK OK)
s2-eth3<->s3-eth1 (OK OK)
s3-eth2<->s1-eth3 (OK OK)
mininet> ports
s1 lo:0 s1-eth1:1 s1-eth2:2 s1-eth3:3
s2 lo:0 s2-eth1:1 s2-eth2:2 s2-eth3:3
s3 lo:0 s3-eth1:1 s3-eth2:2
mininet>

```

使用的 flow rule 如下，因為每個方向的 flow 都要一個 json file，故在 switch 中 s1 和 s2 因為額外連接了 h1 和 h2，各自有 6 個 rule，分別放在 flows\_s1-x\_0716306.json 和 flows\_s2-x\_0716306.json(其中 x 為 1~6)，而 s3 只連接 s1 和 s2，因此只需要 2 個 rule，分別為 flows\_s3-1\_0716306.json 和 flows\_s3-2\_0716306.json，使得一個 switch 若有 ARP packet 進入，能將這個 packet 送到另一個 switch，並且能讓 h1 arping h2 或 h2 arping h1 成功

### s1 的 rule

The screenshot shows two windows of a text editor, likely gedit, running on a Linux desktop. Both windows have tabs labeled 'flows\_s1-1\_0716306.json' and 'flows\_s1-2\_0716306.json'. The left window contains the configuration for switch s1, and the right window contains the configuration for switch s2. Both files are JSON objects defining flow rules. The rules involve matching specific Ethernet types (ARP) and ports (1 or 2) to output them to the other switch's port (2 or 1). The code is identical for both switches, differing only in the selector port number.

```

{
  "priority": 60011,
  "timeout": 0,
  "isPermanent": true,
  "selector": {
    "criteria": [
      {
        "type": "ETH_TYPE",
        "ethType": "0x0806"
      },
      {
        "type": "IN_PORT",
        "port": "1"
      }
    ],
    "treatment": {
      "instructions": [
        {
          "type": "OUTPUT",
          "port": "2"
        }
      ]
    }
  }
}

{
  "priority": 60012,
  "timeout": 0,
  "isPermanent": true,
  "selector": {
    "criteria": [
      {
        "type": "ETH_TYPE",
        "ethType": "0x0806"
      },
      {
        "type": "IN_PORT",
        "port": "2"
      }
    ],
    "treatment": {
      "instructions": [
        {
          "type": "OUTPUT",
          "port": "1"
        }
      ]
    }
  }
}

```

The screenshot displays three separate windows of the Oracle VM VirtualBox application, each showing a text editor with a JSON configuration file for SDN-NFV. The windows are arranged horizontally, showing different parts of the configuration.

**Top Left Window:** Shows the configuration for flow 0\_0716306.json. It includes a 'selector' section with an 'ethType' filter (0x0800) and an 'IN\_PORT' filter (port 1). The 'treatment' section contains two 'OUTPUT' instructions: one to port 3 and one to port 1.

```
{
  "priority": 60013,
  "timeout": 0,
  "isPermanent": true,
  "selector": {
    "criteria": [
      {
        "type": "ETH_TYPE",
        "ethType": "0x0800"
      },
      {
        "type": "IN_PORT",
        "port": "1"
      }
    ]
  },
  "treatment": [
    {
      "instructions": [
        {
          "type": "OUTPUT",
          "port": "3"
        }
      ]
    }
  ]
}
```

**Top Right Window:** Shows the configuration for flow 1\_0716306.json. It includes a 'selector' section with an 'ethType' filter (0x0800) and an 'IN\_PORT' filter (port 3). The 'treatment' section contains two 'OUTPUT' instructions: one to port 3 and one to port 1.

```
{
  "priority": 60014,
  "timeout": 0,
  "isPermanent": true,
  "selector": {
    "criteria": [
      {
        "type": "ETH_TYPE",
        "ethType": "0x0800"
      },
      {
        "type": "IN_PORT",
        "port": "3"
      }
    ]
  },
  "treatment": [
    {
      "instructions": [
        {
          "type": "OUTPUT",
          "port": "1"
        }
      ]
    }
  ]
}
```

**Bottom Left Window:** Shows the configuration for flow 5\_0716306.json. It includes a 'selector' section with an 'ethType' filter (0x0800) and an 'IN\_PORT' filter (port 2). The 'treatment' section contains two 'OUTPUT' instructions: one to port 3 and one to port 2.

```
{
  "priority": 60015,
  "timeout": 0,
  "isPermanent": true,
  "selector": {
    "criteria": [
      {
        "type": "ETH_TYPE",
        "ethType": "0x0800"
      },
      {
        "type": "IN_PORT",
        "port": "2"
      }
    ]
  },
  "treatment": [
    {
      "instructions": [
        {
          "type": "OUTPUT",
          "port": "3"
        }
      ]
    }
  ]
}
```

**Bottom Right Window:** Shows the configuration for flow 6\_0716306.json. It includes a 'selector' section with an 'ethType' filter (0x0800) and an 'IN\_PORT' filter (port 3). The 'treatment' section contains two 'OUTPUT' instructions: one to port 2 and one to port 1.

```
{
  "priority": 60016,
  "timeout": 0,
  "isPermanent": true,
  "selector": {
    "criteria": [
      {
        "type": "ETH_TYPE",
        "ethType": "0x0800"
      },
      {
        "type": "IN_PORT",
        "port": "3"
      }
    ]
  },
  "treatment": [
    {
      "instructions": [
        {
          "type": "OUTPUT",
          "port": "2"
        }
      ]
    }
  ]
}
```

## s2 的 rule

The screenshot displays four windows of the Oracle VM VirtualBox application, each showing a JSON configuration file for SDN-NFV. The windows are arranged in a 2x2 grid.

- Top Left Window:** Shows flows\_s2-1\_0716306.json. The configuration defines a selector with an 'ethType' criterion (0x8006) and an 'IN\_PORT' criterion (1). It has two treatments: one for port 2 (output) and one for port 1 (output).
- Top Right Window:** Shows flows\_s2-2\_0716306.json. This configuration is identical to the one in the top-left window, defining a selector with 'ethType' 0x8006 and 'IN\_PORT' 1, and two treatments for ports 2 and 1 respectively.
- Bottom Left Window:** Shows flows\_s2-3\_0716306.json. The configuration defines a selector with an 'ethType' criterion (0x8006) and an 'IN\_PORT' criterion (1). It has two treatments: one for port 3 (output) and one for port 1 (output).
- Bottom Right Window:** Shows flows\_s2-4\_0716306.json. This configuration is identical to the one in the bottom-left window, defining a selector with 'ethType' 0x8006 and 'IN\_PORT' 1, and two treatments for ports 3 and 1 respectively.

The status bar at the bottom of each window shows the date and time (e.g., 2021/10/13 16:23), system temperature (30°C), and language (ENG).

```
[{"priority": 60021, "timeout": 0, "isPermanent": true, "selector": { "criteria": [ { "type": "ETH_TYPE", "ethType": "0x8006" }, { "type": "IN_PORT", "port": "1" } ] }, "treatment": { "instructions": [ { "type": "OUTPUT", "port": "2" } ] }}, {"priority": 60022, "timeout": 0, "isPermanent": true, "selector": { "criteria": [ { "type": "ETH_TYPE", "ethType": "0x8006" }, { "type": "IN_PORT", "port": "2" } ] }, "treatment": { "instructions": [ { "type": "OUTPUT", "port": "1" } ] }}, {"priority": 60023, "timeout": 0, "isPermanent": true, "selector": { "criteria": [ { "type": "ETH_TYPE", "ethType": "0x8006" }, { "type": "IN_PORT", "port": "1" } ] }, "treatment": { "instructions": [ { "type": "OUTPUT", "port": "3" } ] }}, {"priority": 60024, "timeout": 0, "isPermanent": true, "selector": { "criteria": [ { "type": "ETH_TYPE", "ethType": "0x8006" }, { "type": "IN_PORT", "port": "3" } ] }, "treatment": { "instructions": [ { "type": "OUTPUT", "port": "1" } ] }}]
```

The screenshot shows two side-by-side gedit windows displaying JSON configuration files for network flows. Both windows have tabs for 'flows\_s2-5\_0716306.json' and 'flows\_s2-6\_0716306.json'. The left window's status bar shows 'Tab Width: 8' and 'Ln 2, Col 19'. The right window's status bar shows 'Tab Width: 8' and 'Ln 2, Col 19'. The desktop taskbar at the bottom includes icons for File Explorer, Task View, Edge, FileZilla, and others, along with system status like weather (30°C), battery, and date/time (2021/10/13).

```
[{"priority": 6002, "timeout": 0, "isPermanent": true, "selector": { "criteria": [ { "type": "ETH_TYPE", "ethType": "0x0800" }, { "type": "IN_PORT", "port": "2" } ] }, "treatment": { "instructions": [ { "type": "OUTPUT", "port": "3" } ] }}, {"priority": 60026, "timeout": 0, "isPermanent": true, "selector": { "criteria": [ { "type": "ETH_TYPE", "ethType": "0x0800" }, { "type": "IN_PORT", "port": "3" } ] }, "treatment": { "instructions": [ { "type": "OUTPUT", "port": "2" } ] }}
```

### s3 的 rule

The screenshot shows two side-by-side gedit windows displaying JSON configuration files for network flows. Both windows have tabs for 'flows\_s3-1\_0716306.json' and 'flows\_s3-2\_0716306.json'. The left window's status bar shows 'Tab Width: 8' and 'Ln 2, Col 19'. The right window's status bar shows 'Tab Width: 8' and 'Ln 2, Col 19'. The desktop taskbar at the bottom includes icons for File Explorer, Task View, Edge, FileZilla, and others, along with system status like weather (30°C), battery, and date/time (2021/10/13).

```
[{"priority": 60031, "timeout": 0, "isPermanent": true, "selector": { "criteria": [ { "type": "ETH_TYPE", "ethType": "0x0800" }, { "type": "IN_PORT", "port": "1" } ] }, "treatment": { "instructions": [ { "type": "OUTPUT", "port": "2" } ] }}, {"priority": 60032, "timeout": 0, "isPermanent": true, "selector": { "criteria": [ { "type": "ETH_TYPE", "ethType": "0x0800" }, { "type": "IN_PORT", "port": "2" } ] }, "treatment": { "instructions": [ { "type": "OUTPUT", "port": "1" } ] }}
```

使用 curl 將每個 switch 的 flow 傳送到 ONOS 上

s1:

```
[code .400, mesed Desktop/lab2/part3$ curl -u onos:rocks -X POST -H 'Content-Type: application/json' -d @flows_s1-1_0716306.json 'http://localhost:8181/onos/v1/flows/of:0000000000000001'demo@SDN-NFV:~/Desktop/lab2/part3$ curl -u onos:rocks -X POST -H 'Content-Type: application/json' -d @flows_s1-2_0716306.json 'http://localhost:8181/onos/v1/flows/of:0000000000000001'demo@SDN-NFV:~/Desktop/lab2/part3$ curl -u onos:rocks -X POST -H 'Content-Type: application/json' -d @flows_s1-3_0716306.json 'http://localhost:8181/onos/v1/flows/of:0000000000000001'demo@SDN-NFV:~/Desktop/lab2/part3$ curl -u onos:rocks -X POST -H 'Content-Type: application/json' -d @flows_s1-4_0716306.json 'http://localhost:8181/onos/v1/flows/of:0000000000000001'demo@SDN-NFV:~/Desktop/lab2/part3$ curl -u onos:rocks -X POST -H 'Content-Type: application/json' -d @flows_s1-5_0716306.json 'http://localhost:8181/onos/v1/flows/of:0000000000000001'demo@SDN-NFV:~/Desktop/lab2/part3$ curl -u onos:rocks -X POST -H 'Content-Type: application/json' -d @flows_s1-6_0716306.json 'http://localhost:8181/onos/v1/flows/of:0000000000000001'demo@SDN-NFV:~/Desktop/lab2/part3$
```

"SDN-NFV" 00:34 13- + -21

STATE	PACKETS	DURATION	FLOW PRIORITY	TABLE NAME	SELECTOR	TREATMENT	APP NAME
Added	0	53	60015	0	IN_PORT:2,ETH_TYPE:arp	immOUTPUT:3,cleared:false	*rest
Added	0	64	60012	0	IN_PORT:2,ETH_TYPE:arp	immOUTPUT:1,cleared:false	*rest
Added	0	60	60013	0	IN_PORT:1,ETH_TYPE:arp	immOUTPUT:3,cleared:false	*rest
Added	0	69	60011	0	IN_PORT:1,ETH_TYPE:arp	immOUTPUT:2,cleared:false	*rest
Added	0	105	40000	0	ETH_TYPE:arp	immOUTPUT:CONTROLLER,cleared:true	*core
Added	0	50	60016	0	IN_PORT:3,ETH_TYPE:arp	immOUTPUT:2,cleared:false	*rest
Added	0	56	60014	0	IN_PORT:3,ETH_TYPE:arp	immOUTPUT:1,cleared:false	*rest
Added	68	105	40000	0	ETH_TYPE:bddp	immOUTPUT:CONTROLLER,cleared:true	*core
Added	68	105	40000	0	ETH_TYPE:ildp	immOUTPUT:CONTROLLER,cleared:true	*core

s2

```
[code .400, mesed Desktop/lab2/part3$ curl -u onos:rocks -X POST -H 'Content-Type: application/json' -d @flows_s2-1_0716306.json 'http://localhost:8181/onos/v1/flows/of:0000000000000002'demo@SDN-NFV:~/Desktop/lab2/part3$ curl -u onos:rocks -X POST -H 'Content-Type: application/json' -d @flows_s2-2_0716306.json 'http://localhost:8181/onos/v1/flows/of:0000000000000002'demo@SDN-NFV:~/Desktop/lab2/part3$ curl -u onos:rocks -X POST -H 'Content-Type: application/json' -d @flows_s2-3_0716306.json 'http://localhost:8181/onos/v1/flows/of:0000000000000002'demo@SDN-NFV:~/Desktop/lab2/part3$ curl -u onos:rocks -X POST -H 'Content-Type: application/json' -d @flows_s2-4_0716306.json 'http://localhost:8181/onos/v1/flows/of:0000000000000002'demo@SDN-NFV:~/Desktop/lab2/part3$ curl -u onos:rocks -X POST -H 'Content-Type: application/json' -d @flows_s2-5_0716306.json 'http://localhost:8181/onos/v1/flows/of:0000000000000002'demo@SDN-NFV:~/Desktop/lab2/part3$ curl -u onos:rocks -X POST -H 'Content-Type: application/json' -d @flows_s2-6_0716306.json 'http://localhost:8181/onos/v1/flows/of:0000000000000002'demo@SDN-NFV:~/Desktop/lab2/part3$
```

"SDN-NFV" 00:36 13- + -21

STATE	PACKETS	DURATION	FLOW PRIORITY	TABLE NAME	SELECTOR	TREATMENT	APP NAME
Added	0	135	40000	0	ETH_TYPE:arp	imm[OUTPUT:CONTROLLER], cleared:true	*core
Added	0	66	60024	0	IN_PORT:3, ETH_TYPE:arp	imm[OUTPUT:1], cleared:false	*rest
Added	0	58	60026	0	IN_PORT:3, ETH_TYPE:arp	imm[OUTPUT:2], cleared:false	*rest
Added	0	73	60022	0	IN_PORT:2, ETH_TYPE:arp	imm[OUTPUT:1], cleared:false	*rest
Added	0	62	60025	0	IN_PORT:2, ETH_TYPE:arp	imm[OUTPUT:3], cleared:false	*rest
Added	0	76	60021	0	IN_PORT:1, ETH_TYPE:arp	imm[OUTPUT:2], cleared:false	*rest
Added	0	70	60023	0	IN_PORT:1, ETH_TYPE:arp	imm[OUTPUT:3], cleared:false	*rest
Added	86	134	40000	0	ETH_TYPE:lldp	imm[OUTPUT:CONTROLLER], cleared:true	*core
Added	86	134	40000	0	ETH_TYPE:bddp	imm[OUTPUT:CONTROLLER], cleared:true	*core

s3

```
demo@SDN-NFV:~/Desktop/lab2/part3$ curl -u onos:rocks -X POST -H 'Content-Type: application/json' -d @flows_s3-1_0716306.json 'http://localhost:8181/onos/v1/flows/of:0000000000000003'
demo@SDN-NFV:~/Desktop/lab2/part3$ curl -u onos:rocks -X POST -H 'Content-Type: application/json' -d @flows_s3-2_0716306.json 'http://localhost:8181/onos/v1/flows/of:0000000000000003'
demo@SDN-NFV:~/Desktop/lab2/part3$ █
```

STATE	PACKETS	DURATION	FLOW PRIORITY	TABLE NAME	SELECTOR	TREATMENT	APP NAME
Added	0	199	40000	0	ETH_TYPE:arp	imm[OUTPUT:CONTROLLER], cleared:true	*core
Added	0	115	60032	0	IN_PORT:2, ETH_TYPE:arp	imm[OUTPUT:1], cleared:false	*rest
Added	0	119	60031	0	IN_PORT:1, ETH_TYPE:arp	imm[OUTPUT:2], cleared:false	*rest
Added	128	199	40000	0	ETH_TYPE:lldp	imm[OUTPUT:CONTROLLER], cleared:true	*core
Added	128	199	40000	0	ETH_TYPE:bddp	imm[OUTPUT:CONTROLLER], cleared:true	*core

開始使用 h1 arping h2 及 h2 arping h1 後，得到以下結果(我在執行 arping 時先執行 h1 arping h2，因此下列每個 switch 的圖示中，第一張圖都只會有一個方向的 flow 有封包在傳輸，直到第二個封包執行後，相反方向的 flow 才有封包)

s1

h1 arping h2

STATE	PACKETS	DURATION	FLOW PRIORITY	TABLE NAME	SELECTOR	TREATMENT	APP NAME
Added	0	255	60012	0	IN_PORT:2, ETH_TYPE:arp	imm[OUTPUT:1], cleared:false	*rest
Added	0	260	60011	0	IN_PORT:1, ETH_TYPE:arp	imm[OUTPUT:2], cleared:false	*rest
Added	0	296	40000	0	ETH_TYPE:arp	imm[OUTPUT:CONTROLLER], cleared:true	*core
Added	0	241	60016	0	IN_PORT:3, ETH_TYPE:arp	imm[OUTPUT:2], cleared:false	*rest
Added	0	248	60014	0	IN_PORT:3, ETH_TYPE:arp	imm[OUTPUT:1], cleared:false	*rest
Added	5	251	60013	0	IN_PORT:1, ETH_TYPE:arp	imm[OUTPUT:3], cleared:false	*rest
Added	172	296	40000	0	ETH_TYPE:bddp	imm[OUTPUT:CONTROLLER], cleared:true	*core
Added	172	296	40000	0	ETH_TYPE:lldp	imm[OUTPUT:CONTROLLER], cleared:true	*core
Added	111,591,539	244	60015	0	IN_PORT:2, ETH_TYPE:arp	imm[OUTPUT:3], cleared:false	*rest

h2 arping h1

STATE	PACKETS	DURATION	FLOW PRIORITY	TABLE NAME	SELECTOR	TREATMENT	APP NAME
Added	0	294	60012	0	IN_PORT:2, ETH_TYPE:arp	imm[OUTPUT:1], cleared:false	*rest
Added	0	299	60011	0	IN_PORT:1, ETH_TYPE:arp	imm[OUTPUT:2], cleared:false	*rest
Added	0	335	40000	0	ETH_TYPE:arp	imm[OUTPUT:CONTROLLER], cleared:true	*core
Added	0	286	60014	0	IN_PORT:3, ETH_TYPE:arp	imm[OUTPUT:1], cleared:false	*rest
Added	5	290	60013	0	IN_PORT:1, ETH_TYPE:arp	imm[OUTPUT:3], cleared:false	*rest
Added	194	335	40000	0	ETH_TYPE:bddp	imm[OUTPUT:CONTROLLER], cleared:true	*core
Added	194	335	40000	0	ETH_TYPE:lldp	imm[OUTPUT:CONTROLLER], cleared:true	*core
Added	2,255,434	280	60016	0	IN_PORT:3, ETH_TYPE:arp	imm[OUTPUT:2], cleared:false	*rest
Added	177,208,894	283	60015	0	IN_PORT:2, ETH_TYPE:arp	imm[OUTPUT:3], cleared:false	*rest

s2

h1 arping h2

STATE	PACKETS	DURATION	FLOW PRIORITY	TABLE NAME	SELECTOR	TREATMENT	APP NAME
Added	0	270	40000	0	ETH_TYPE:arp	imrn[OUTPUT:CONTROLLER], cleared:true	*core
Added	0	201	60024	0	IN_PORT:3, ETH_TYPE:arp	imrn[OUTPUT:1], cleared:false	*rest
Added	0	208	60022	0	IN_PORT:2, ETH_TYPE:arp	imrn[OUTPUT:1], cleared:false	*rest
Added	0	197	60025	0	IN_PORT:2, ETH_TYPE:arp	imrn[OUTPUT:3], cleared:false	*rest
Added	0	212	60021	0	IN_PORT:1, ETH_TYPE:arp	imrn[OUTPUT:2], cleared:false	*rest
Added	0	205	60023	0	IN_PORT:1, ETH_TYPE:arp	imrn[OUTPUT:3], cleared:false	*rest
Added	162	269	40000	0	ETH_TYPE:lldp	imrn[OUTPUT:CONTROLLER], cleared:true	*core
Added	162	269	40000	0	ETH_TYPE:bBGP	imrn[OUTPUT:CONTROLLER], cleared:true	*core
Added	74,162,982	193	60026	0	IN_PORT:3, ETH_TYPE:arp	imrn[OUTPUT:2], cleared:false	*rest

h2 arping h1

STATE	PACKETS	DURATION	FLOW PRIORITY	TABLE NAME	SELECTOR	TREATMENT	APP NAME
Added	0	350	40000	0	ETH_TYPE:arp	imrn[OUTPUT:CONTROLLER], cleared:true	*core
Added	0	281	60024	0	IN_PORT:3, ETH_TYPE:arp	imrn[OUTPUT:1], cleared:false	*rest
Added	0	288	60022	0	IN_PORT:2, ETH_TYPE:arp	imrn[OUTPUT:1], cleared:false	*rest
Added	0	292	60021	0	IN_PORT:1, ETH_TYPE:arp	imrn[OUTPUT:2], cleared:false	*rest
Added	6	285	60023	0	IN_PORT:1, ETH_TYPE:arp	imrn[OUTPUT:3], cleared:false	*rest
Added	202	349	40000	0	ETH_TYPE:lldp	imrn[OUTPUT:CONTROLLER], cleared:true	*core
Added	202	349	40000	0	ETH_TYPE:bBGP	imrn[OUTPUT:CONTROLLER], cleared:true	*core
Added	9,685,270	277	60025	0	IN_PORT:2, ETH_TYPE:arp	imrn[OUTPUT:3], cleared:false	*rest
Added	194,230,842	273	60026	0	IN_PORT:3, ETH_TYPE:arp	imrn[OUTPUT:2], cleared:false	*rest

s3

h1 arping h2

This screenshot shows the ONOS flow table for device 0 (0000000000000003). The table displays five flows, all in the 'Added' state. The flows are categorized by their selector and treatment:

STATE	PACKETS	DURATION	FLOW PRIORITY	TABLE NAME	SELECTOR	TREATMENT	APP NAME
Added	0	241	40000	0	ETH_TYPE:arp	lmm[OUTPUT:CONTROLLER], cleared:true	*core
Added	0	161	60031	0	IN_PORT:1, ETH_TYPE:arp	lmm[OUTPUT:2], cleared:false	*rest
Added	152	241	40000	0	ETH_TYPE:lldp	lmm[OUTPUT:CONTROLLER], cleared:true	*core
Added	152	241	40000	0	ETH_TYPE:bddp	lmm[OUTPUT:CONTROLLER], cleared:true	*core
Added	28,640,517	157	60032	0	IN_PORT:2, ETH_TYPE:arp	lmm[OUTPUT:1], cleared:false	*rest

h2 arping h1

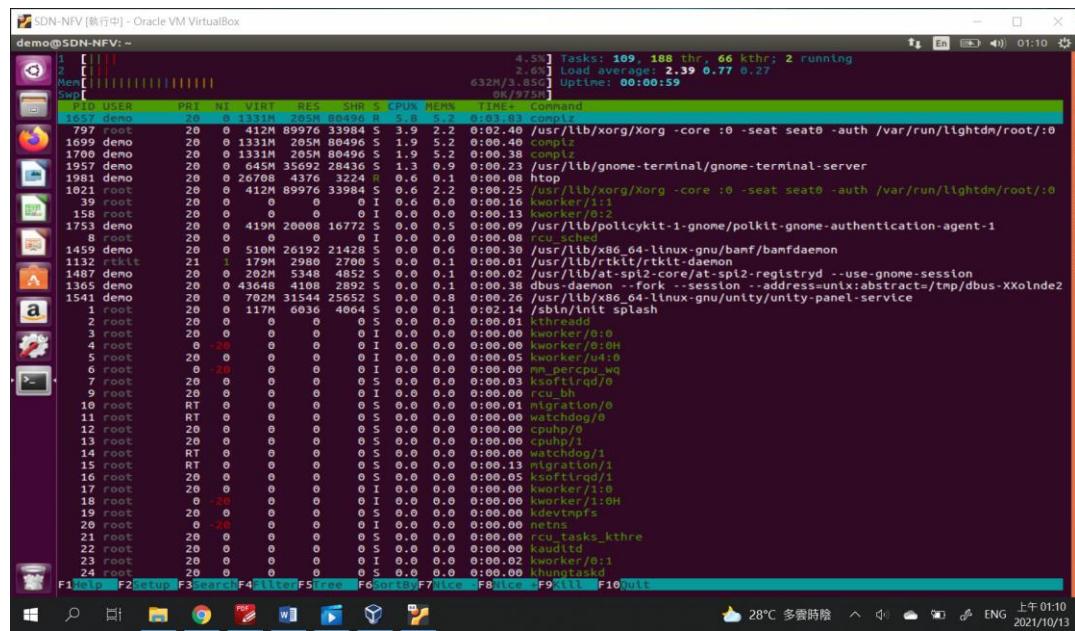
This screenshot shows the ONOS flow table for device 0 (0000000000000003). The table displays five flows, all in the 'Added' state. The flows are categorized by their selector and treatment:

STATE	PACKETS	DURATION	FLOW PRIORITY	TABLE NAME	SELECTOR	TREATMENT	APP NAME
Added	0	370	40000	0	ETH_TYPE:arp	lmm[OUTPUT:CONTROLLER], cleared:true	*core
Added	210	370	40000	0	ETH_TYPE:lldp	lmm[OUTPUT:CONTROLLER], cleared:true	*core
Added	210	370	40000	0	ETH_TYPE:bddp	lmm[OUTPUT:CONTROLLER], cleared:true	*core
Added	20,630,934	290	60031	0	IN_PORT:1, ETH_TYPE:arp	lmm[OUTPUT:2], cleared:false	*rest
Added	220,292,849	285	60032	0	IN_PORT:2, ETH_TYPE:arp	lmm[OUTPUT:1], cleared:false	*rest

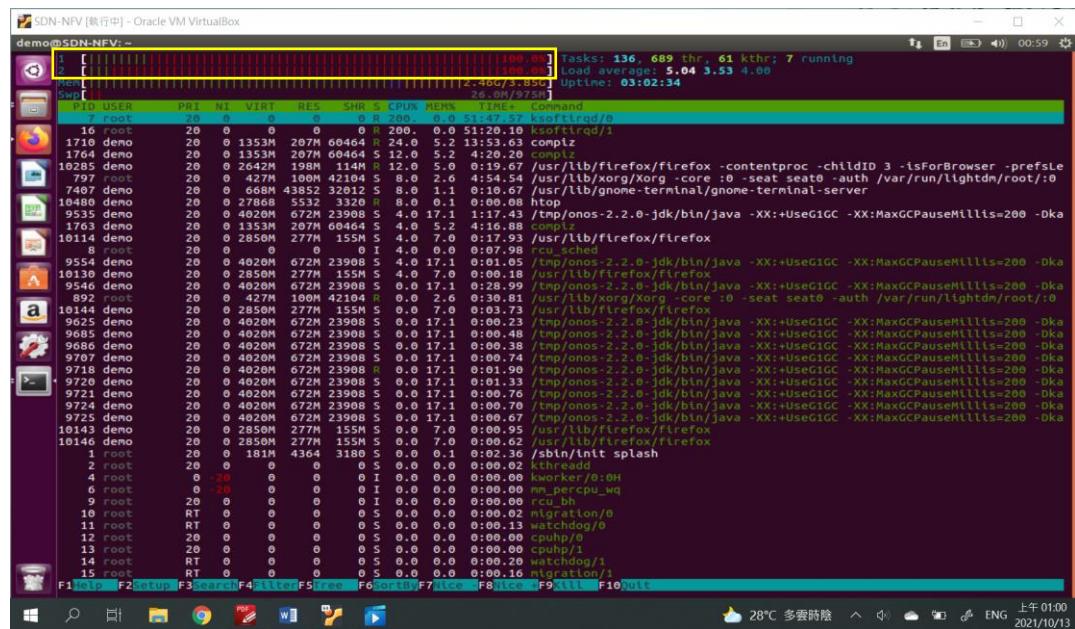
## CPU 使用率

在偵測 CPU 使用率的工具方面，我選擇使用 `htop`，平常狀態下的 CPU 和處於 broadcast storm 下的 CPU 對比如下

一般狀態



## Broadcast storm



由上圖黃框處可知 2 個 CPU 都處於滿負載的情況

## Part4

完成準備工作，如下圖

The screenshot shows a terminal window titled "SDN-NFV [執行中] - Oracle VM VirtualBox". The terminal displays several log entries from the ONOS project, including configuration of event dispatchers, reactive forwarding, and flow rules. It also shows a user navigating through the ONOS command-line interface (CLI) to activate various apps like fwd, reactive, and hostprovider.

```
demo@SDN-NFV:~  
2.0 | Configured. Matching TCP/UDP Fields is disabled | demo@SDN-NFV:~$ sudo mn --controller=remote,127.0.0.1:6653  
2021-10-13T14:29:33,117 | INFO | CM Event Dispatcher (Fire ConfigurationEvent: pid=org.onosproject.fwd.ReactiveForwarding) | Reactive Forwarding | 209 - org.onosproject.onos-apps-fwd - 2.0 | Configured. Matching ICMP (v4 and v6) fields is disabled | demo@SDN-NFV:~$  
2021-10-13T14:29:33,118 | INFO | CM Event Dispatcher (Fire ConfigurationEvent: pid=org.onosproject.fwd.ReactiveForwarding) | Reactive Forwarding | 209 - org.onosproject.onos-apps-fwd - 2.0 | Configured. Ignore IPv4 multicasts packets is disabled | demo@SDN-NFV:~$  
2021-10-13T14:29:33,118 | INFO | CM Event Dispatcher (Fire ConfigurationEvent: pid=org.onosproject.fwd.ReactiveForwarding) | Reactive Forwarding | 209 - org.onosproject.onos-apps-fwd - 2.0 | Configured. Flow Timeout is configured to 10 seconds | demo@SDN-NFV:~$  
2021-10-13T14:29:33,119 | INFO | CM Event Dispatcher (Fire ConfigurationEvent: pid=org.onosproject.fwd.ReactiveForwarding) | Reactive Forwarding | 209 - org.onosproject.onos-apps-fwd - 2.0 | Configured. Flow Priority is configured to 10 | demo@SDN-NFV:~$  
and '[cmd] --help' for help on a specific command.  
Hit <ctrl-d> or type 'logout' to exit ONOS session.  
demo@root > app activate fwd  
'Activated org.onosproject.fwd'  
* 19 org.onosproject.drivers 2.2.0 Default Drivers  
* 20 org.onosproject.optical-model 2.2.0 Optical Network Model  
* 59 org.onosproject.lldpprovider 2.2.0 LLDP Link Provider  
* 78 org.onosproject.fwd 2.2.0 Reactive Forwarding  
* 91 org.onosproject.hostprovider 2.2.0 Host Location Provider  
* 92 org.onosproject.openflow-base 2.2.0 OpenFlow Base Provider  
* 93 org.onosproject.openflow 2.2.0 OpenFlow Provider  
* 153 org.onosproject.gui2 2.2.0 ONOS GUI2  
demo@root >  
demo@root ~
```

首先 s1 一開始不會有能讓 h1 ping h2 的 flow rule

The screenshot shows the ONOS web interface in a Mozilla Firefox browser. The title bar says "ONOS — Mozilla Firefox". The main content area displays a table titled "Flows for Device of:00000000000000000001 (4 Total)". The table lists four flow entries:

STATE	PACKETS	DURATION	FLOW PRIORITY	TABLE NAME	SELECTOR	TREATMENT	APP NAME
Added	0	165	40000	0	ETH_TYPE:bddp	imrn[OUTPUT:CONTROLLER], cleared:true	*core
Added	0	165	40000	0	ETH_TYPE:arp	imrn[OUTPUT:CONTROLLER], cleared:true	*core
Added	0	165	40000	0	ETH_TYPE:lldp	imrn[OUTPUT:CONTROLLER], cleared:true	*core
Added	0	106	5	0	ETH_TYPE:ipv4	imrn[OUTPUT:CONTROLLER], cleared:true	*core

當執行 h1 ping h2 後，封包由 h1 傳送到 s1，s1 這時候的 flow table 沒有任何的 rule 存在，因此第一個進入到 switch 的 packet 會被包裝成 openflow packet in message 並送到 ONOS，ONOS 再將訊息送到 reactive forwarding app，並使得這個 app 可以發出 flowmod 或 packetout 的訊息

Reactive forwarding app 對封包的處理如下(參考 reactive forwarding 在 github 上的 code)

Reactive forwarding 主要會執行 Reactive processor 這個 function 以完成封包的轉送和 flow rule 的設定

Reactive processor 執行步驟

1. 從 switch 抓封包上來，並將封包 parse 好

```
InboundPacket pkt = context.inPacket();
Ethernet ethPkt = pkt.parsed();
```

2. Processor 會根據自己的 function 判斷 packet 的性質(control packet、IPV6 multicast packet...)

```
MacAddress macAddress = ethPkt.getSourceMAC();
ReactiveForwardMetrics macMetrics = null;
macMetrics = createCounter(macAddress);
inPacket(macMetrics);

// Bail if this is deemed to be a control packet.
if (isControlPacket(ethPkt)) {
    droppedPacket(macMetrics);
    return;
}

// Skip IPv6 multicast packet when IPv6 forward is disabled.
if (!ipv6Forwarding && isIpv6Multicast(ethPkt)) {
    droppedPacket(macMetrics);
    return;
}

HostId id = HostId.hostId(ethPkt.getDestinationMAC());

// Do not process LLDP MAC address in any way.
if (id.mac().isLldp()) {
    droppedPacket(macMetrics);
    return;
}
```

3. 從封包中取出 source 和 destination mac address，並確認這個 switch 是否知道 packet 的 destination 在哪，如果不知道就用 flood 的方式尋找

```
HostId id = HostId.hostId(ethPkt.getDestinationMAC());

// Do we know who this is for? If not, flood and bail.
Host dst = hostService.getHost(id);
if (dst == null) {
    flood(context, macMetrics);
    return;
}
```

4. 因為我們使用的是 default topology，h2 會直接連到現在這個 switch，此時就不會使用 flood，而是經過 deviceid 和 port 的比對後，呼叫 installRule 這個 function

```
// Do we know who this is for? If not, flood and bail.  
Host dst = hostService.getHost(id);  
if (dst == null) {  
    flood(context, macMetrics);  
    return;  
}  
  
// Are we on an edge switch that our destination is on? If so,  
// simply forward out to the destination and bail.  
if (pkt.receivedFrom().deviceId().equals(dst.location().deviceId())) {  
    if (!context.inPacket().receivedFrom().port().equals(dst.location().port())) {  
        installRule(context, dst.location().port(), macMetrics);  
    }  
    return;  
}
```

5. InstallRule 會將 packet 的 source、destination MAC address 及 port 作為篩選標準，且因為是 IPV4 message，篩選標準還會再加上 ethernet type 和 protocol

```
    selectorBuilder.matchInPort(context.inPacket().receivedFrom().port())
        .matchEthSrc(inPkt.getSourceMAC())
        .matchEthDst(inPkt.getDestinationMAC());

    // If configured Match Vlan ID
    if (matchVlanId && inPkt.getVlanID() != Ethernet.VLAN_UNTAGGED) {
        selectorBuilder.matchVlanId(vlanId.vlanId(inPkt.getVlanID()));
    }

    //
    // If configured and EtherType is IPv4 - Match IPv4 and
    // TCP/UDP/ICMP fields
    //
    if (matchIpv4Address && inPkt.getEtherType() == Ethernet.TYPE_IPV4) {
        IPv4 ipv4Packet = (IPv4) inPkt.getPayload();
        byte ipv4Protocol = ipv4Packet.getProtocol();
        Ip4Prefix matchIp4SrcPrefix =
            Ip4Prefix.valueOf(ipv4Packet.getSourceAddress(),
                Ip4Prefix.MAX_MASK_LENGTH);
        Ip4Prefix matchIp4DstPrefix =
            Ip4Prefix.valueOf(ipv4Packet.getDestinationAddress(),
                Ip4Prefix.MAX_MASK_LENGTH);
        selectorBuilder.matchEthType(Ethernet.TYPE_IPV4)
            .matchIPSrc(matchIp4SrcPrefix)
            .matchIPDst(matchIp4DstPrefix);

        if (matchIpv4Dscp) {
            byte dscp = ipv4Packet.getDscp();
            byte ecn = ipv4Packet.getEcn();
        }
    }
}
```

```

        byte ecn = ipv4Packet.getEcn();
        selectorBuilder.matchIPDscp(dscp).matchIPEcn(ecn);
    }

    if (matchTcpUdpPorts && ipv4Protocol == IPv4.PROTOCOL_TCP) {
        TCP tcpPacket = (TCP) ipv4Packet.getPayload();
        selectorBuilder.matchIPProtocol(ipv4Protocol)
            .matchTcpSrc(TpPort.tpPort(tcpPacket.getSourcePort()))
            .matchTcpDst(TpPort.tpPort(tcpPacket.getDestinationPort()));
    }
    if (matchTcpUdpPorts && ipv4Protocol == IPv4.PROTOCOL_UDP) {
        UDP udpPacket = (UDP) ipv4Packet.getPayload();
        selectorBuilder.matchIPProtocol(ipv4Protocol)
            .matchUdpSrc(TpPort.tpPort(udpPacket.getSourcePort()))
            .matchUdpDst(TpPort.tpPort(udpPacket.getDestinationPort()));
    }
    if (matchIcmpFields && ipv4Protocol == IPv4.PROTOCOL_ICMP) {
        ICMP icmpPacket = (ICMP) ipv4Packet.getPayload();
        selectorBuilder.matchIPProtocol(ipv4Protocol)
            .matchIcmpType(icmpPacket.getIcmpType())
            .matchIcmpCode(icmpPacket.getIcmpCode());
    }
}

```

## 6. 將 packet 送到指定的 port 並在 switch 上放置 rule

```

        .matchIcmpv6Type(icmp6Packet.getIcmpType())
        .matchIcmpv6Code(icmp6Packet.getIcmpCode());
    }
}
TrafficTreatment treatment = DefaultTrafficTreatment.builder()
    .setOutput(portNumber)
    .build();

ForwardingObjective forwardingObjective = DefaultForwardingObjective.builder()
    .withSelector(selectorBuilder.build())
    .withTreatment(treatment)
    .withPriority(flowPriority)
    .withFlag(ForwardingObjective.Flag.VERSATILE)
    .fromApp(appId)
    .makeTemporary(flowTimeout)
    .add();

flowObjectiveService.forward(context.inPacket().receivedFrom().deviceId(),
    forwardingObjective);
forwardPacket(macMetrics);

```

之後 packet 就進入 data plane 並送到 h2，然後 h2 會回傳 h1 一個 ICMP reply，因為 h2 到 h1 的 flow 並沒有在 flow table 中，上面的 control plane 流程會再走一次然後這個封包才會回傳到 h1

如此可以看到 s1 的 flow table 中有 h1 到 h2 方向及 h2 到 h1 方向的 flow rule

STATE	PACKETS	DURATION	FLOW PRIORITY	TABLE NAME	SELECTOR	TREATMENT	APP NAME
Added	0	6,155	40000	0	ETH_TYPE:bddp	imm[OUTPUT:CONTROLLER], cleared:true	*core
Added	0	6,155	40000	0	ETH_TYPE:lldp	imm[OUTPUT:CONTROLLER], cleared:true	*core
Added	5	5	10	0	IN_PORT:1, ETH_DST:02:18:54:E5:AE :40, ETH_SRC:C2:A5:CD:4C:C 0:9D	imm[OUTPUT:2], cleared:false	*fwd
Added	5	5	10	0	IN_PORT:2, ETH_DST:C2:A5:CD:4C:C 0:9D, ETH_SRC:02:18:54:E5:AE :40	imm[OUTPUT:1], cleared:false	*fwd
Added	6	6,096	5	0	ETH_TYPE:ip4	imm[OUTPUT:CONTROLLER], cleared:true	*core
Added	8	6,155	40000	0	ETH_TYPE:arp	imm[OUTPUT:CONTROLLER], cleared:true	*core