

# 教育部補助「5G 行動寬頻跨校教學聯盟計畫」

## 下世代 Network Slicing 模組設計

### 實驗單元：以 Mininet 搭配 Controller 使用 OpenFlow 模擬 SDN 網路環境

授課教師：李宗南

教材編撰：曾國維

## 一、課程單元目標

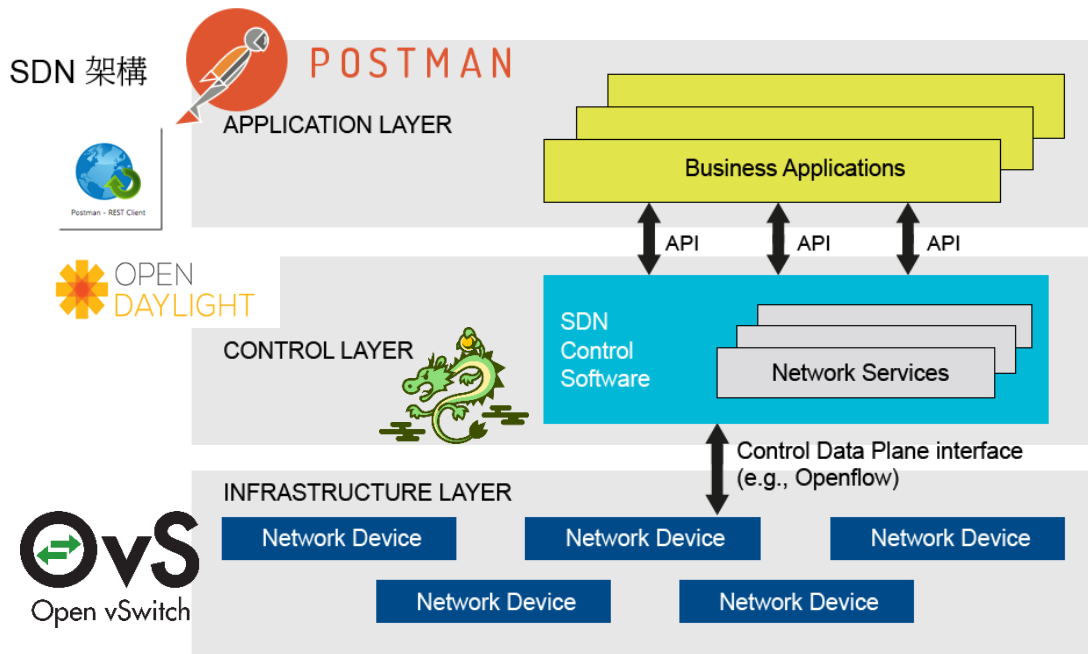
1. 修課學生得以了解 SDN 的基本觀念及架構
2. 修課學生得以理解 Mininet 的網路環境模擬以及 SDN 控制器(Ryu、OpenDaylight)的使用以及 SDN 應用程式 Postman 的使用。
3. 修課學生得以完成軟體定義網路環境進行實驗及，驗證 OpenFlow 執行

## 二、SDN 簡介與基本架構

### (一) SDN 簡介

SDN(Soft Defined Network)是一種新的網路架構。利用 OpenFlow 協定，把路由器的控制平面（control plane）從資料平面（data plane）中分離出來，以軟體方式實作。這個架構可以讓網路管理員，在不更動硬體裝置的前提下，以中央控制方式，用程式重新規劃網路，為控制網路流量提供了新的方法，也提供了核心網路及應用創新的良好平台。

### (二) SDN 基本架構



圖一、SDN 的基本架構

在圖一中可以看到，SDN 分為 3 個層面，分別為 Application Layer、Control Layer 及 Infrastructure Layer。

- Mininet

Mininet 是一個可以透過一些虛擬終端機、路由器、交換器等連接創建虛擬網路拓撲的平台，使用者可以輕易的在自己的個人電腦中創作支援 SDN 的區域網路，以驗證實驗方法，除此之外也可以造出的虛擬的 host 並以真實電腦般發送封包。

- Ryu

Ryu 是來自於日本 NTT 所開發以及設計，針對 SDN 的控制器開發框架(Framework)，開發時有明確的定義：Ryu is a component-based software defined networking framework.

Ryu 包含了 OpenFlow(以及其他部分協定) Controller 的功能，並且使用 Python 進行開發 Controller。

- OpenDaylight

OpenDaylight 是 Linux 的基金會負責管理的開源項目，提供一套基於 SDN 開發的模塊化，可擴展，可升級，支持多協議的控制器框架，其項目的設計目標是降低網路運營的複雜度，擴展現有網路架構中硬體的生命期，並且使用 Java 進行開發 controller。

- OpenvSwitch

OpenvSwitch 是 Open Networking Foundation 的一個開源計畫，顧名思義是一個 virtual switch，它的目的是讓大規模網路透過可編程或來進行擴展，可用於切割網域，QoS 或是流量監控，同時支持標準的管理接口服務和各項協議(sFlow, NetFlow, OpenFlow 等)。

本次實驗將在 Infrastructure Layer 中將使用 Mininet 進行實驗，Mininet 提供了 OpenvSwitch 元件，用以支援和 Control Layer 溝通所使用的 OpenFlow。在 Control Layer 分別將使用 Ryu 及 OpenDaylight 進行實驗 Task1~Task3 及 Task4。在 Application Layer 中將使用 Postman REST Client 來和 Controller 進行 REST API 的溝通，包含取得目前 Switch 及下達 OpenFlow 等。

### 三、SDN 實驗設備與版本

- 硬體:

- 電腦：Ubuntu 作業系統

- 軟體:

- Mininet：2.3.0d4

- Ryu：ryu-manager 4.30

- OpenDaylight: Lithium-SR1

- Postman REST Client

- gcc 編譯器

- vi/vim 文字編輯器

#### 四、安裝及執行

##### [1]Mininet+Ryu

[http://140.117.164.22/data/SDN\\_NFV\\_class/SDN\\_Lab4.pdf](http://140.117.164.22/data/SDN_NFV_class/SDN_Lab4.pdf)

##### [2]Mininet+OpenDaylight

[http://140.117.164.22/data/SDN\\_NFV\\_class/SDN\\_Lab1.pdf](http://140.117.164.22/data/SDN_NFV_class/SDN_Lab1.pdf)

請同學配合參考以上文件和以下步驟，並進行安裝實驗環境及完成 Task1~Task4

#### Mininet 安裝

```
git clone git://github.com/mininet/mininet
```

```
cd mininet
```

```
util/install.sh -a
```

完成上述指令後，輸入：

```
Sudo mn
```

```
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> _
```

出現上述結果代表 Mininet 安裝成功，預設的 Mininet 拓樸是一台 Switch 連接預設的 Controller 及 2 台 Host。

如果 Switch 為 OpenvSwitch，則可輸入以下指令直接查看 Switch 中的 OpenFlow 內容：

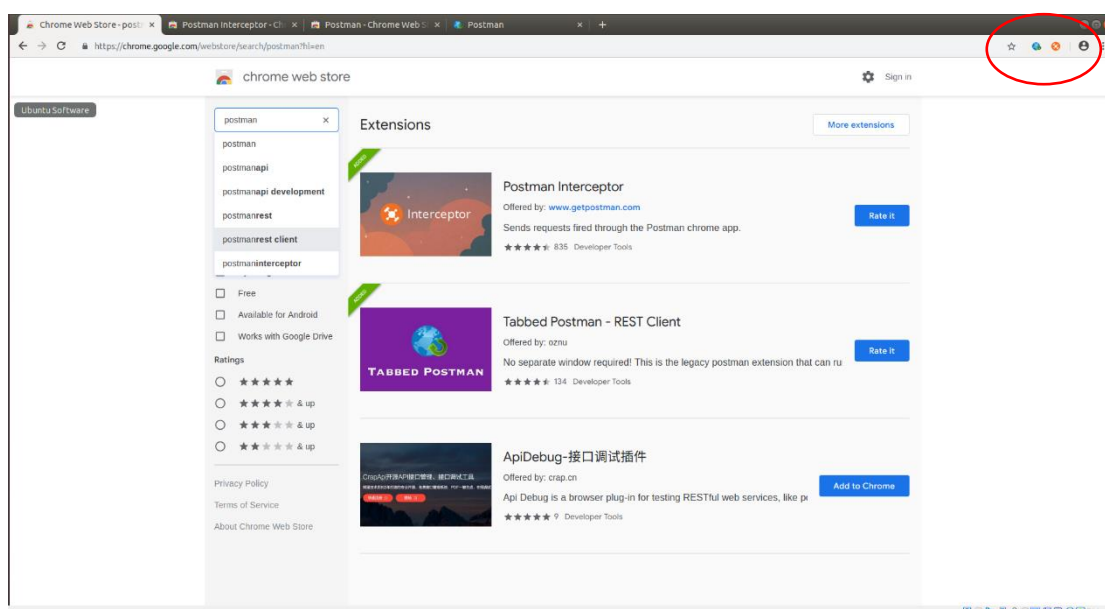
```
sh ovs-ofctl -O openflow13 dump-flows $SwitchName
```

另外 Switch 和 Controller 建立的 Hidden Flow 也可由以下查看：

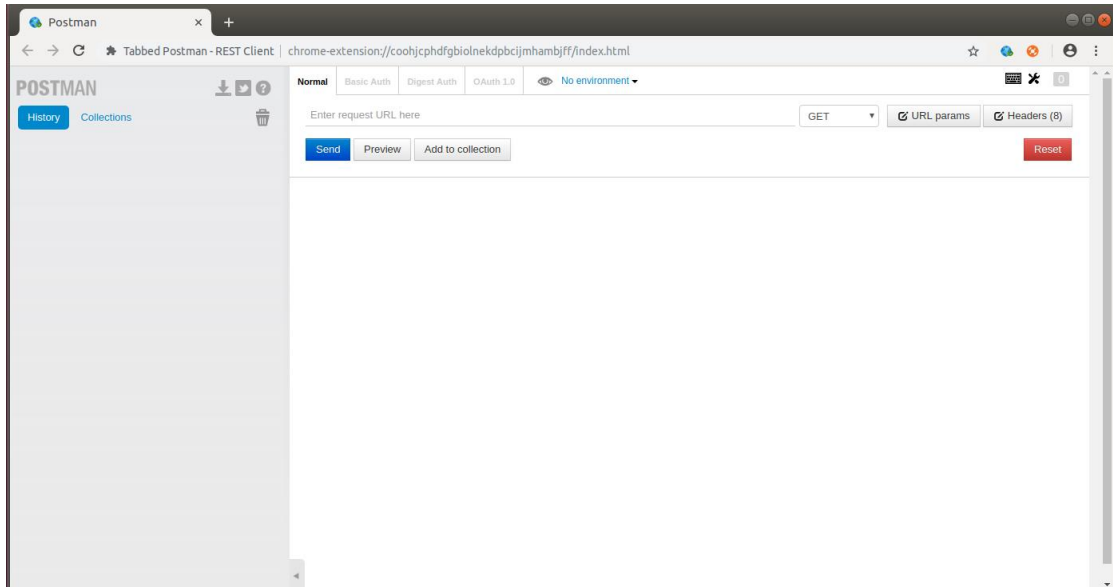
```
sh ovs-appctl bridge/dump-flows $SwitchName
```

## Postman REST 安裝及 Get Collections

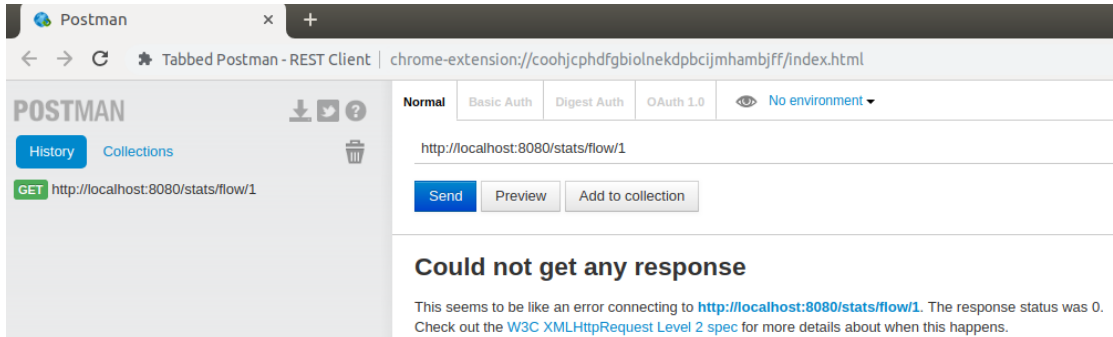
在 Chrome web store 中，輸入 postman 找到 Postman REST APP



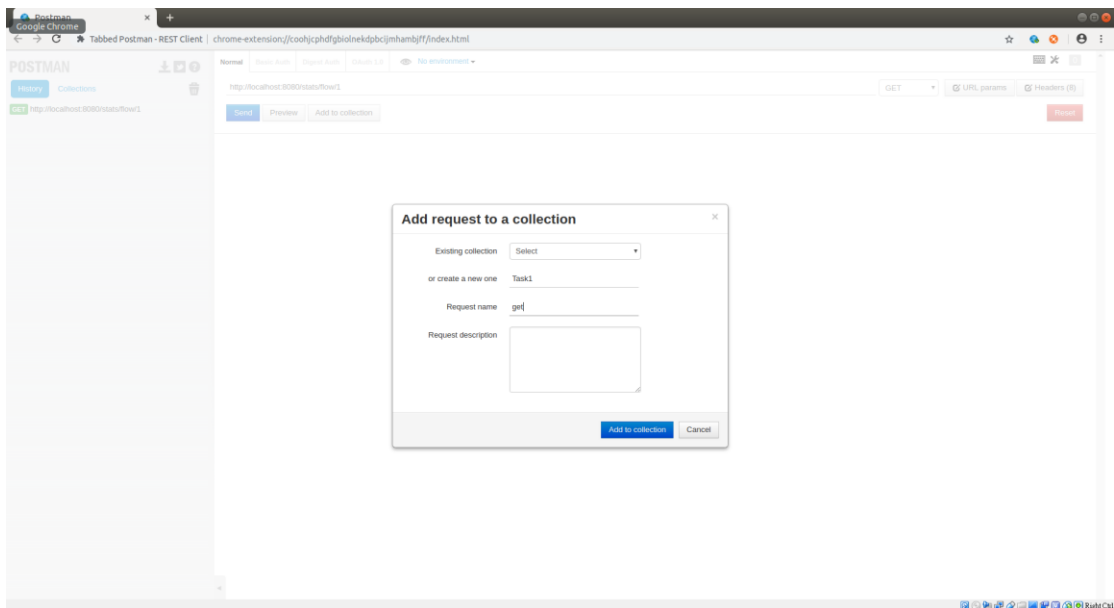
安裝完成後，在標籤列按下圖示即可進入 Postman 應用程式介面



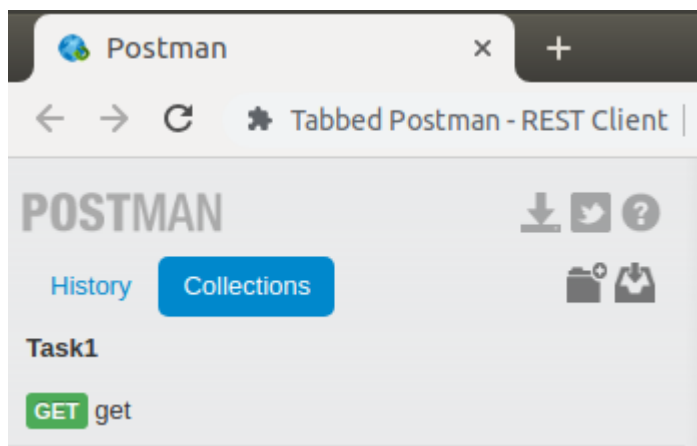
左邊會顯示 Request 的歷史紀錄，右邊部分上面為 Request REST 的 URL，右邊為選擇的 REQUEST 型態，通常使用 GET 向 Controller 取得 Switch 資訊，使用 PUT 或 POST 向 controller 通知對 Switch 下達 OpenFlow，下方為 Controller 的 RESPONSE。



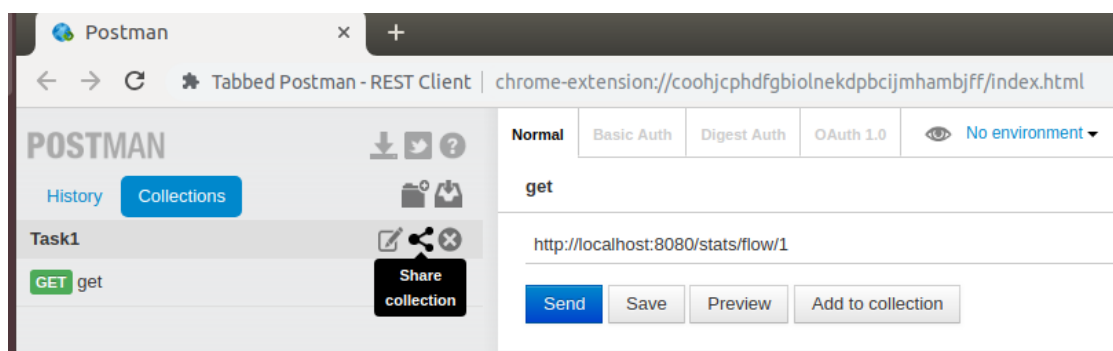
可以看到 REQUEST 一次後出在歷史紀錄中，按下 Send 按鈕旁的 Add to collection:

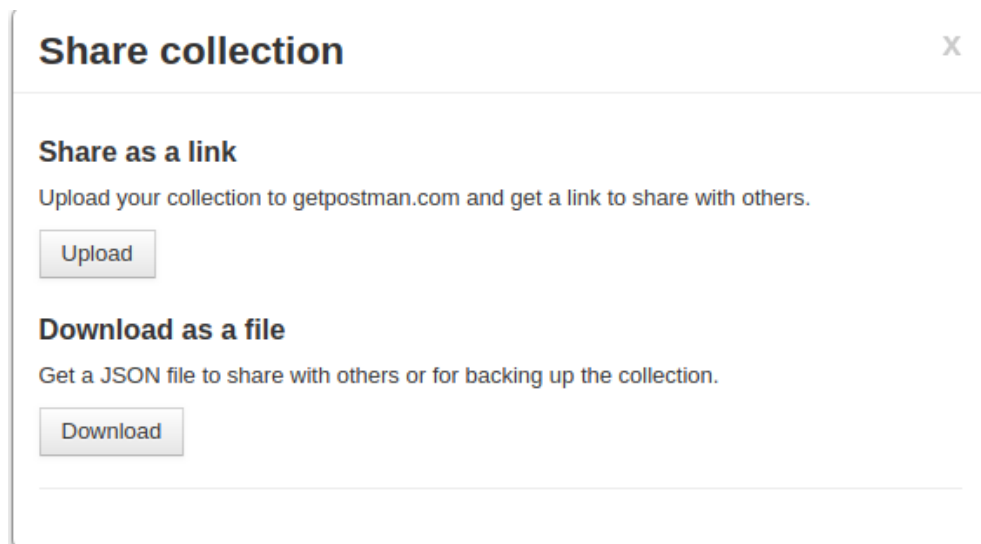


按下 Add to collection 後可以看到在 Collections 中出現剛剛設定的 collection 名稱及 REST request 的名稱。

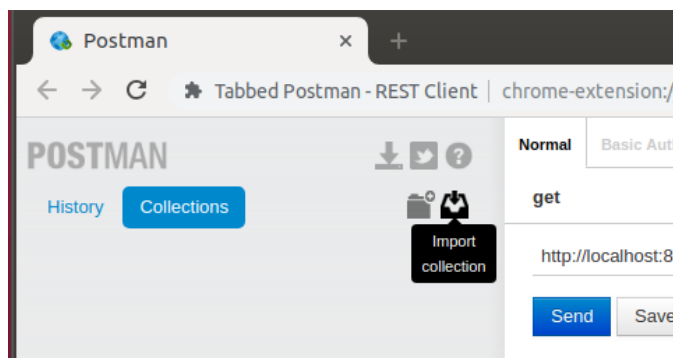


滑鼠移至 collection 名稱旁按下 share collection

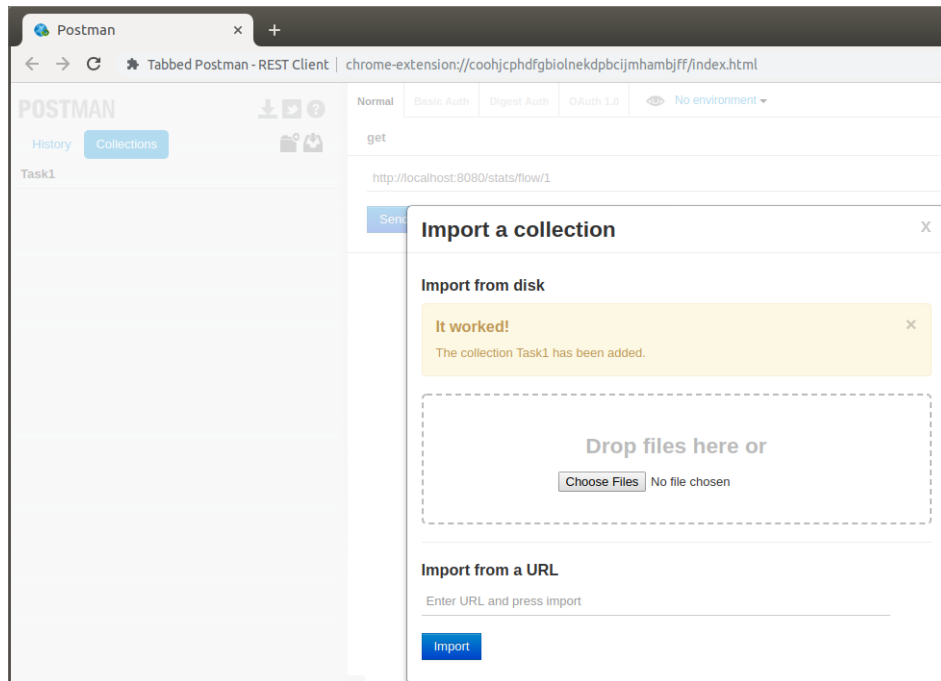




按下 Download 後會出現新的分頁，內容為 json 格式，就可將內文複製存成 json 檔並在下次進行 import







## Ryu Controller 安裝

```
sudo apt-get update
```

```
sudo apt-get install libxml2-dev libxslt1-dev python-pip python-eventlet python-routes python-webob python-paramiko
```

```
sudo pip install msgpack-python eventlet==0.15.2
```

```
sudo pip install six --upgrade
```

```
sudo pip install oslo.config q --upgrade
```

```
sudo pip install ryu
```

```
ryu-manager --verbose ryu.app.ofctl_rest
```

安裝完成後，啟動 Ryu controller:

```
ryu-manager --verbose ryu.app.ofctl_rest
```

```
cliis@cliis-VirtualBox: ~  
File Edit View Search Terminal Help  
CONSUMES EventOFPEchoReply  
CONSUMES EventOFPPortDescStatsReply  
BRICK RestStatsApi  
CONSUMES EventOFPMeterConfigStatsReply  
CONSUMES EventOFPPGroupDescStatsReply  
CONSUMES EventOFPPGroupStatsReply  
CONSUMES EventOFPPQueueGetConfigReply  
CONSUMES EventOFPPFlowStatsReply  
CONSUMES EventOFPPTableStatsReply  
CONSUMES EventOFPPQueueDescStatsReply  
CONSUMES EventOFPMeterStatsReply  
CONSUMES EventOFPSwitchFeatures  
CONSUMES EventOFPPStatsReply  
CONSUMES EventOFPPTableFeaturesStatsReply  
CONSUMES EventOFPPPortStatsReply  
CONSUMES EventOFPMeterFeaturesStatsReply  
CONSUMES EventOFPPAggregateStatsReply  
CONSUMES EventOFPPDescStatsReply  
CONSUMES EventOFPPGroupFeaturesStatsReply  
CONSUMES EventOFPPPortDescStatsReply  
CONSUMES EventOFPPQueueStatsReply  
CONSUMES EventOFPPRoleReply  
(11699) wsgi starting up on http://0.0.0.0:8080
```

得到以上結果代表 Ryu controller 啟動成功，並載入 ofctl\_rest 模組在 8080 port 上進行 REST API 的監聽。

## OpenDaylight Controller 安裝

```
sudo apt-get install openjdk-8-jdk openjdk-8-jre
```

```
wget
```

<https://nexus.opendaylight.org/content/repositories/public/org/opendaylight/integration/distribution-karaf/0.3.1-Lithium-SR1/distribution-karaf-0.3.1-Lithium-SR1.tar.gz>

```
tar -xzf distribution-karaf-0.3.1-Lithium-SR1.tar.gz
```

```
mv distribution-karaf-0.3.1-Lithium-SR1 karaf-0.3.1
```

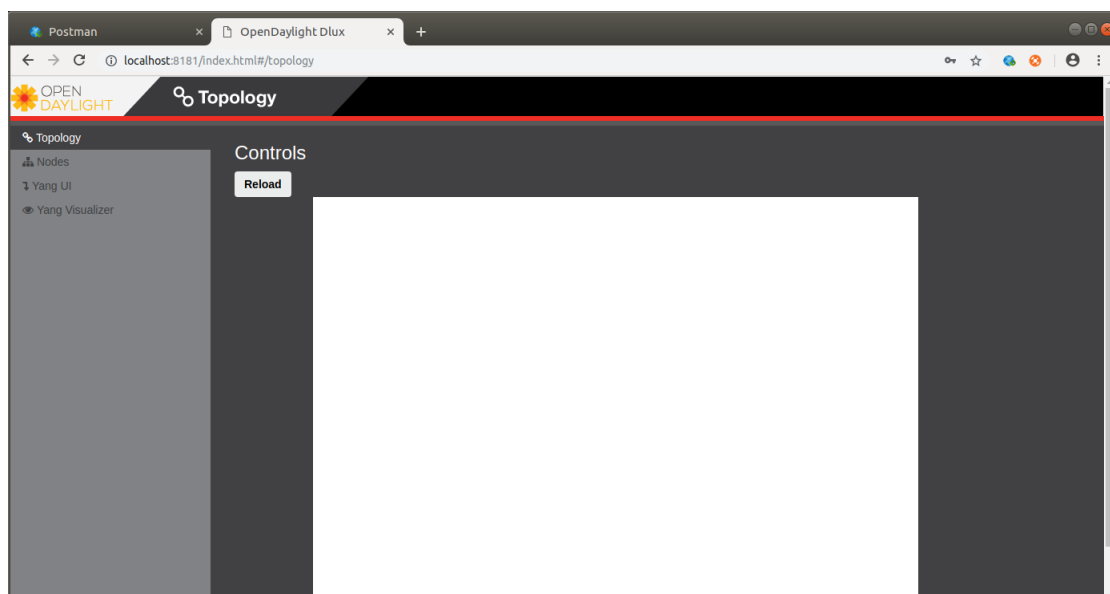
安裝完成後，啟動 OpenDaylight controller:

```
./karaf-0.3.1/bin/karaf -of13
```





登入帳號密碼皆為 admin，登入後即可看到目前接上 controller 的 switch 及 host 的拓模，及各 Switch 內容，以及 OpenDaylight Yang UI 的 controller 使用套件。



- **Task 1: Let h1 and h2 be able to access each other. (pingall)**

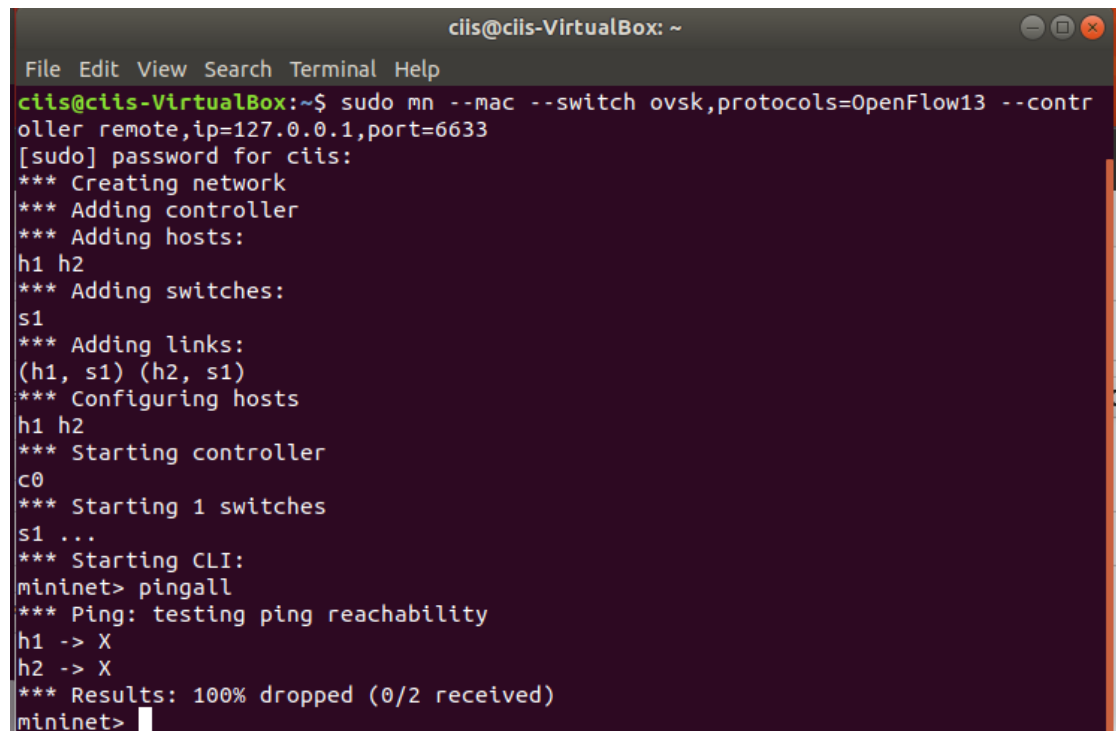
Task1 目標為對 Ryu 下達 REST API，Controller 對 Switch 下達 OpenFlow 使 Host1 與 Host2 互相 ping 成功。

Ryu

```
$ ryu-manager --verbose ryu.app.ofctl_rest
```

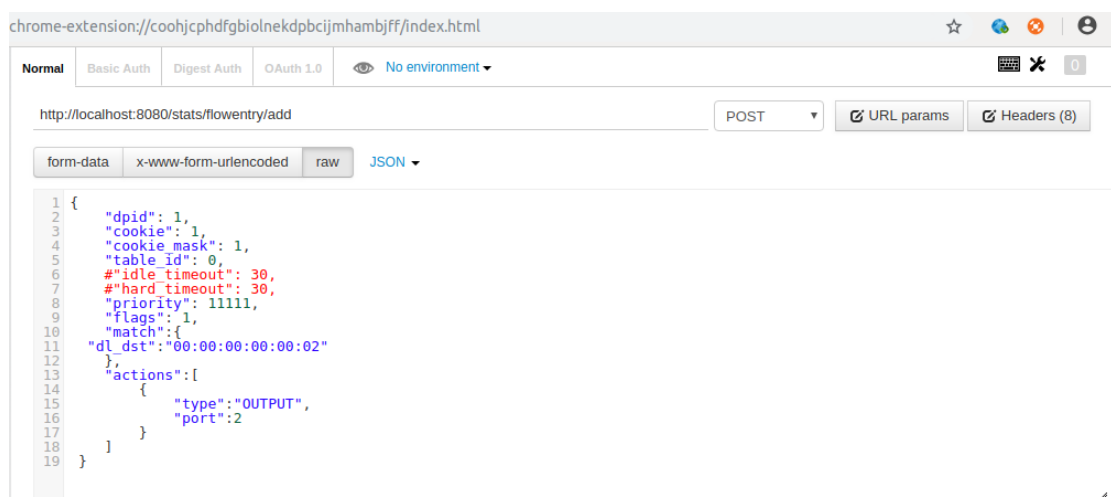
Mininet

```
$ sudo mn --mac --switch ovsk,protocols=OpenFlow13 --controller remote,ip=127.0.0.1,port=6633
```



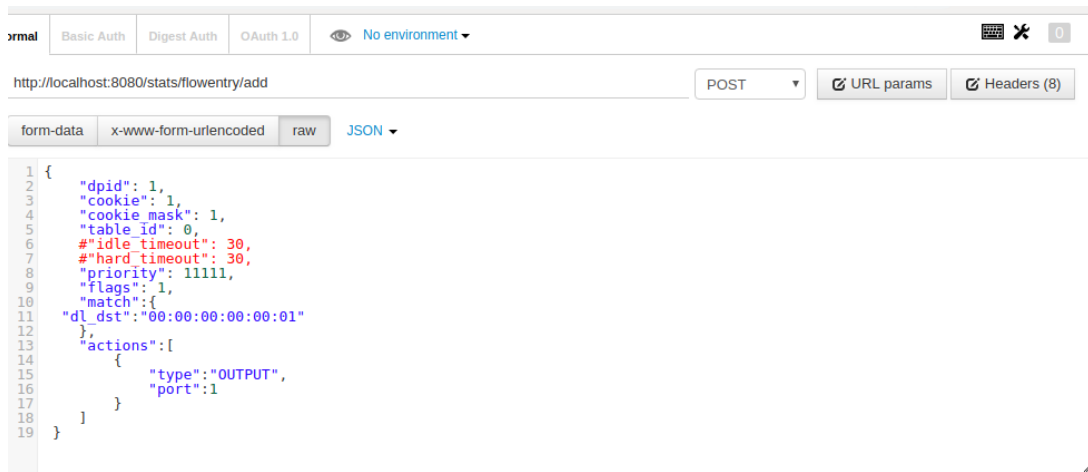
```
ciis@ciis-VirtualBox: ~  
File Edit View Search Terminal Help  
ciis@ciis-VirtualBox:~$ sudo mn --mac --switch ovsk,protocols=OpenFlow13 --controller remote,ip=127.0.0.1,port=6633  
[sudo] password for ciis:  
*** Creating network  
*** Adding controller  
*** Adding hosts:  
h1 h2  
*** Adding switches:  
s1  
*** Adding links:  
(h1, s1) (h2, s1)  
*** Configuring hosts  
h1 h2  
*** Starting controller  
c0  
*** Starting 1 switches  
s1 ...  
*** Starting CLI:  
mininet> pingall  
*** Ping: testing ping reachability  
h1 -> X  
h2 -> X  
*** Results: 100% dropped (0/2 received)  
mininet>
```

執行後，會發現 h1 與 h2 無法 ping 通，因此必須下達 3 條 OpenFlow 讓 Switch 得以轉傳 h1 與 h2 間的封包，第一條 Flow 為目標為 h2 的 MAC address 則送往 port 2(s1-eth2)



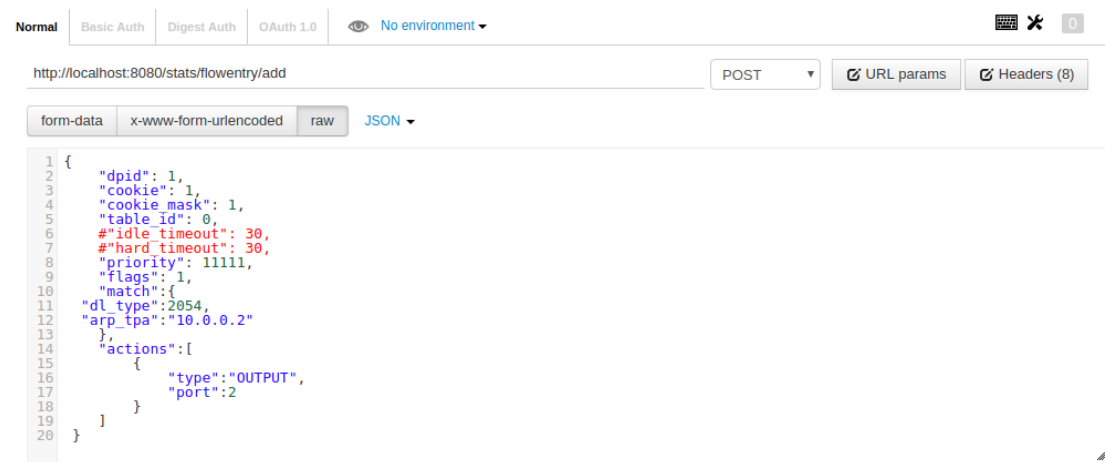
```
chrome-extension://coohjcpdhdfgbiolnekdpcijmhambjff/index.html  
Normal Basic Auth Digest Auth OAuth 1.0 No environment  
http://localhost:8080/stats/flowentry/add POST URL params Headers (8)  
form-data x-www-form-urlencoded raw JSON  
1 {  
2   "dpid": 1,  
3   "cookie": 1,  
4   "cookie_mask": 1,  
5   "table_id": 0,  
6   "#idle_timeout": 30,  
7   "#hard_timeout": 30,  
8   "priority": 11111,  
9   "flags": 1,  
10  "match": {  
11    "dl_dst": "00:00:00:00:00:02"  
12  },  
13  "actions": [  
14    {  
15      "type": "OUTPUT",  
16      "port": 2  
17    }  
18  ]  
19 }
```

第二條 Flow 為目標為 h1 的 MAC address 則送往 port 1(s1-eth1)



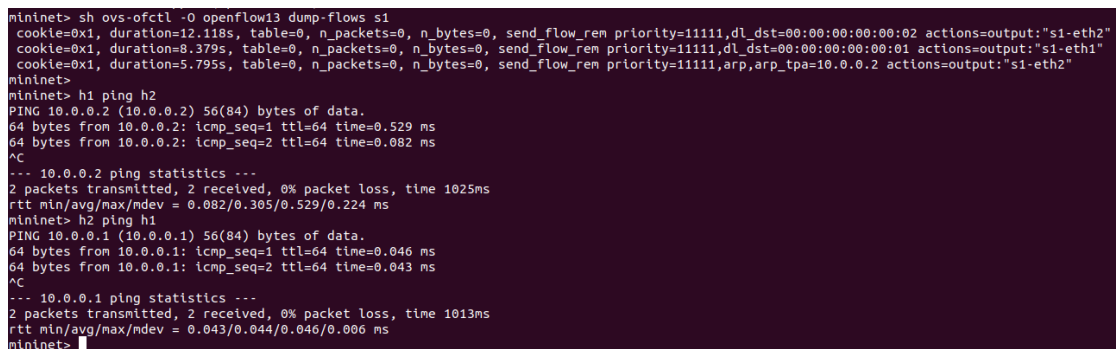
```
1 {
2   "dpid": 1,
3   "cookie": 1,
4   "cookie_mask": 1,
5   "table_id": 0,
6   "#idle_timeout": 30,
7   "#hard_timeout": 30,
8   "priority": 11111,
9   "flags": 1,
10  "match": {
11    "dl_dst": "00:00:00:00:00:01"
12  },
13  "actions": [
14    {
15      "type": "OUTPUT",
16      "port": 1
17    }
18  ]
19 }
```

下達這兩條 Flow 後，由於 h1 的 ARP Table 中並無 h2 的 MAC address，所以會進行 ARP broadcast，但 Switch 收到後沒有 Match 上述二條 Flow 會直接丟掉，因此須再下達第三條為 ARP protocol 2054 (hex 0806)目標為 h2 的 IP address 則送往 port 2(s1-eth2)，以更新 h1 的 ARP table



```
1 {
2   "dpid": 1,
3   "cookie": 1,
4   "cookie_mask": 1,
5   "table_id": 0,
6   "#idle_timeout": 30,
7   "#hard_timeout": 30,
8   "priority": 11111,
9   "flags": 1,
10  "match": {
11    "dl_type": 2054,
12    "arp_tpa": "10.0.0.2"
13  },
14  "actions": [
15    {
16      "type": "OUTPUT",
17      "port": 2
18    }
19  ]
20 }
```

查看 Switch 中 OpenFlow 的內容後，確認 h1 與 h2 可互相 ping 通



```
mininet> sh ovs-ofctl -O openflow13 dump-flows s1
cookie=0x1, duration=12.118s, table=0, n_packets=0, n_bytes=0, send_flow_rem priority=11111,dl_dst=00:00:00:00:00:02 actions=output:"s1-eth2"
cookie=0x1, duration=8.379s, table=0, n_packets=0, n_bytes=0, send_flow_rem priority=11111,dl_dst=00:00:00:00:00:01 actions=output:"s1-eth1"
cookie=0x1, duration=5.795s, table=0, n_packets=0, n_bytes=0, send_flow_rem priority=11111,arp,arp_tpa=10.0.0.2 actions=output:"s1-eth2"
mininet>
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=0.529 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.082 ms
^C
--- 10.0.0.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1025ms
rtt min/avg/max/mdev = 0.082/0.305/0.529/0.224 ms
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=0.046 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.043 ms
^C
--- 10.0.0.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1013ms
rtt min/avg/max/mdev = 0.043/0.044/0.046/0.006 ms
mininet>
```

- **Task 2: Develop an app which allows controller to flood every packet that comes to it.**

Task2 目標為在 Ryu 中開發一個模組，讓 Switch 收到封包後直接轉傳給 Controller，Controller 再處理 Switch 收到的封包，並以 FLOOD 型態轉發到 Switch 的 port 上。在 OpenFlow 中 Switch 的 output port 有以下保留名稱：

保留埠種類	保留埠說明
ALL	這種模式的保留埠只能當作Output Port，用於轉發特定的網路封包。轉發時，除了原本的Ingress Port以及所有被標示為OFPPC_NO_FWD的埠外，其他的埠都會收到轉發的網路封包
CONTROLLER	這種模式的保留埠代表Controller的Control Channel，可以當作Ingress Port或是Output Port
TABLE	這個保留埠代表OpenFlow Pipeline的起點，只有在Output動作指令中才有作用
IN_PORT	代表Ingress Port
ANY	當一個埠沒有被指定是哪一種保留埠的時候，就會標示成ANY，也因此，這個時候的保留埠無法當成Ingress Port，也不能當作Output Port
LOCAL	這個可以被當成是Ingress Port或是Output Port，可以允許遠端設備透過OpenFlow網路與這台交換機互動
NORMAL	代表要使用一般傳統非OpenFlow協定的Pipeline處理方式，如果交換機無法將網路封包從OpenFlow Pipeline傳到一般的Pipeline，則必須指名並不支援這種保留埠
FLOOD	代表用來做Flooding動作的埠，可以只Flood到Output Port。一般來說，這種埠會送往所有其他的標識埠，而不會傳送到Ingress Port，也不會傳送到標示為OFPPS_BLOCKED狀態的埠。當然交換機也可以指定要往VLAN中進行這種廣播的動作。之前提到過，一個VLAN可以看成同一個廣播網域

開發的模組程式碼路徑須放置在 python 中 ryu app 下：

```
/usr/local/lib/python2.7/dist-packages/ryu/app/
```

```
vim /usr/local/lib/python2.7/dist-packages/ryu/app/ mysw_basic.py
```

```

from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3
class L2Switch(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]
    def __init__(self, *args, **kwargs):
        super(L2Switch, self).__init__(*args, **kwargs)
    @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
    def switch_features_handler(self, ev):
        self.logger.info("***** Add Default Flow *****")
        dp = ev.msg.datapath
        ofp = dp.ofproto
        parser = dp.ofproto_parser
        match = parser.OFPMatch()
        actions = [parser.OFPActionOutput(ofp.OFPP_CONTROLLER, ofp.OFPCML_NO_BUFFER)]
        inst = [parser.OFPInstructionActions(ofp.OFPIT_APPLY_ACTIONS, actions)]
        mod = parser.OFPFlowMod(datapath=dp, priority=0, match=match, instructions=inst)
        dp.send_msg(mod)
    @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
    def packet_in_handler(self, ev):
        msg = ev.msg
        dp = msg.datapath
        ofp = dp.ofproto
        parser = dp.ofproto_parser
        self.logger.debug("***** Debug *****")
        # output port
        data = None
        if msg.buffer_id == ofp.OFP_NO_BUFFER:
            data = msg.data
        actions = [parser.OFPActionOutput(ofp.OFPP_FLOOD)]
        out = parser.OFPacketOut(datapath=dp, buffer_id=msg.buffer_id, in_port=msg.match['in_port'], actions=actions, data=data)
        dp.send_msg(out)

```

Ryu

\$ ryu-manager --verbose ryu.app.ofctl\_rest ryu.app.mysw\_basic

Mininet

\$ sudo mn --mac --switch ovsk,protocols=OpenFlow13 --controller  
remote,ip=127.0.0.1,port=6633

The screenshot shows two terminal windows. The left window displays a series of log messages from the L2Switch, including '\*\*\*\*\* Debug \*\*\*\*\*' and 'EVENT ofp\_event->L2Switch EventOFPPacketIn'. The right window shows the output of the 'mn' command, including '\*\*\* Adding switches:', '\*\*\* Adding links:', '\*\*\* Configuring hosts', '\*\*\* Starting controller', and '\*\*\* Starting 1 switches'. Below these, it shows the output of 'mininet> h1 ping h2', which displays a series of ping results for 10.0.0.2, showing times ranging from 4.43 ms to 5.38 ms.

可以看到 Switch 在收到封包後會先傳給 Controller，Controller 再處理所有封包，包含 ARP 與 ICMP，並且使用 FLOOD 的方式轉傳 Switch 上的 port，因此在 ping 的 response time 是明顯比 Task1 及 Task3 長的。



- **Task 3: Develop an app which allows controller to add a flow.**

Task3 目標為在 Ryu 中開發一個模組，讓 Switch 收到封包後轉傳給 Controller，Controller 處理 Switch 收到的封包後，下達 output port 型態為 FLOOD 的 OpenFlow 至 Switch。Switch 再依照流表的內容轉傳封包。

```
vim /usr/local/lib/python2.7/dist-packages/ryu/app/ mysw_flow.py
```

```
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3
class L2Switch(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]
    def __init__(self, *args, **kwargs):
        super(L2Switch, self).__init__(*args, **kwargs)
    @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
    def switch_features_handler(self, ev):
        self.logger.info("***** Add Default Flow *****")
        dp = ev.msg.datapath
        ofp = dp.ofproto
        parser = dp.ofproto_parser
        match = parser.OFPMatch()
        actions = [parser.OFPActionOutput(ofp.OFPP_CONTROLLER, ofp.OFPCML_NO_BUFFER)]
        inst = [parser.OFPInstructionActions(ofp.OFPIT_APPLY_ACTIONS, actions)]
        mod = parser.OFPFlowMod(datapath=dp, priority=0, match=match, instructions=inst)
        dp.send_msg(mod)
    @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
    def packet_in_handler(self, ev):
        msg = ev.msg
        dp = msg.datapath
        ofp = dp.ofproto
        parser = dp.ofproto_parser

        self.logger.debug("***** Add FLOOD Flow *****")
        match = parser.OFPMatch()
        actions = [parser.OFPActionOutput(ofp.OFPP_FLOOD)]
        inst = [parser.OFPInstructionActions(ofp.OFPIT_APPLY_ACTIONS, actions)]
        mod = parser.OFPFlowMod(datapath=dp, priority=2048, match=match, instructions=inst)
        dp.send_msg(mod)
        ## output port
        #data = None
        #if msg.buffer_id == ofp.OFP_NO_BUFFER:
        #    data = msg.data
        #actions = [parser.OFPActionOutput(ofp.OFPP_FLOOD)]
        #out = parser.OFPPacketOut(datapath=dp, buffer_id=msg.buffer_id, in_port=msg.match['in_port'], actions=actions, data=data)
        #dp.send_msg(out)
```

Ryu

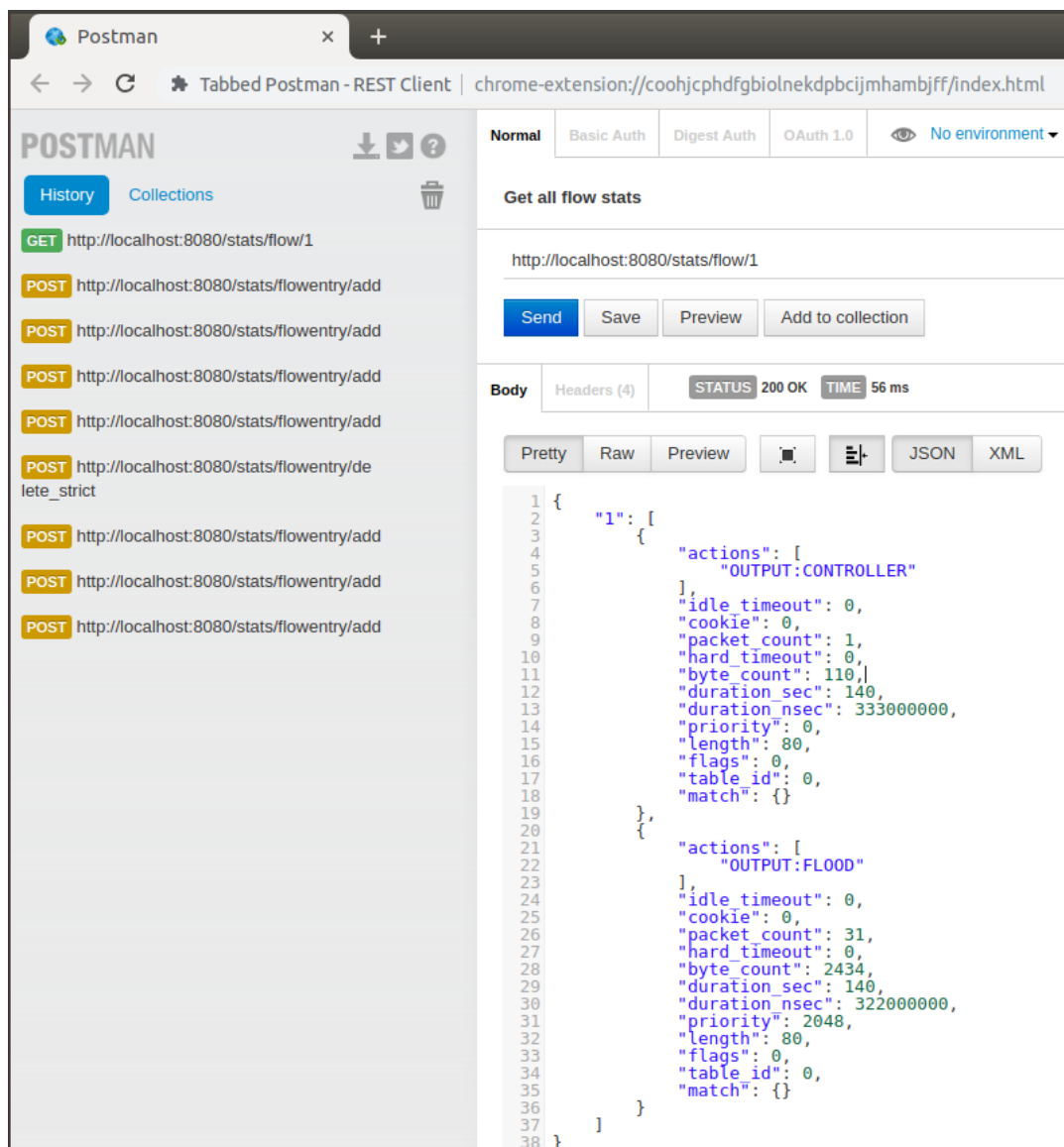
```
$ ryu-manager --verbose ryu.app.ofctl_rest ryu.app.mysw_flow
```

Mininet

```
$ sudo mn --mac --switch ovsk,protocols=OpenFlow13 --controller
remote,ip=127.0.0.1,port=6633
```

```
File Edit View Search Terminal Help
(13659) wsgi starting up on http://0.0.0.0:8080
connected socket: <eventlet.greenio.base.GreenSocket object at 0x7f8a156c2a90> address: ('127.0.0.1', 51990)
EVENT ofp_event->dpset EventOFPPStateChange
connected socket: <eventlet.greenio.base.GreenSocket object at 0x7f8a156c2d10> address: ('127.0.0.1', 51992)
hello ev <ryu.controller.ofp_event.EventOFPHello object at 0x7f8a156db210>
move onto config mode
EVENT ofp_event->dpset EventOFPPSwitchFeatures
EVENT ofp_event->L2Switch EventOFPPSwitchFeatures
switch features ev version=0x4,msg_type=0x6,msg_len=0x20,xid=0x4125a892,OFPPSwitchFeatures(auxiliary_id=0,capabilities=79,datapath_id=1,n_buffers=0,n_tables=254)
***** Add Default Flow *****
move onto main mode
EVENT ofp_event->dpset EventOFPPStateChange
DPSET: register datapath <ryu.controller.controller.Datapath object at 0x7f8a156c2b90>
EVENT ofp_event->L2Switch EventOFPPPacketIn
***** Add FLOOD Flow *****
EVENT ofp_event->dpset EventOFPPPortStatus
DPSET: A port was modified.(datapath id = 0000000000000001, port number = 1)
EVENT ofp_event->dpset EventOFPPPortStatus
DPSET: A port was modified.(datapath id = 0000000000000001, port number = 2)

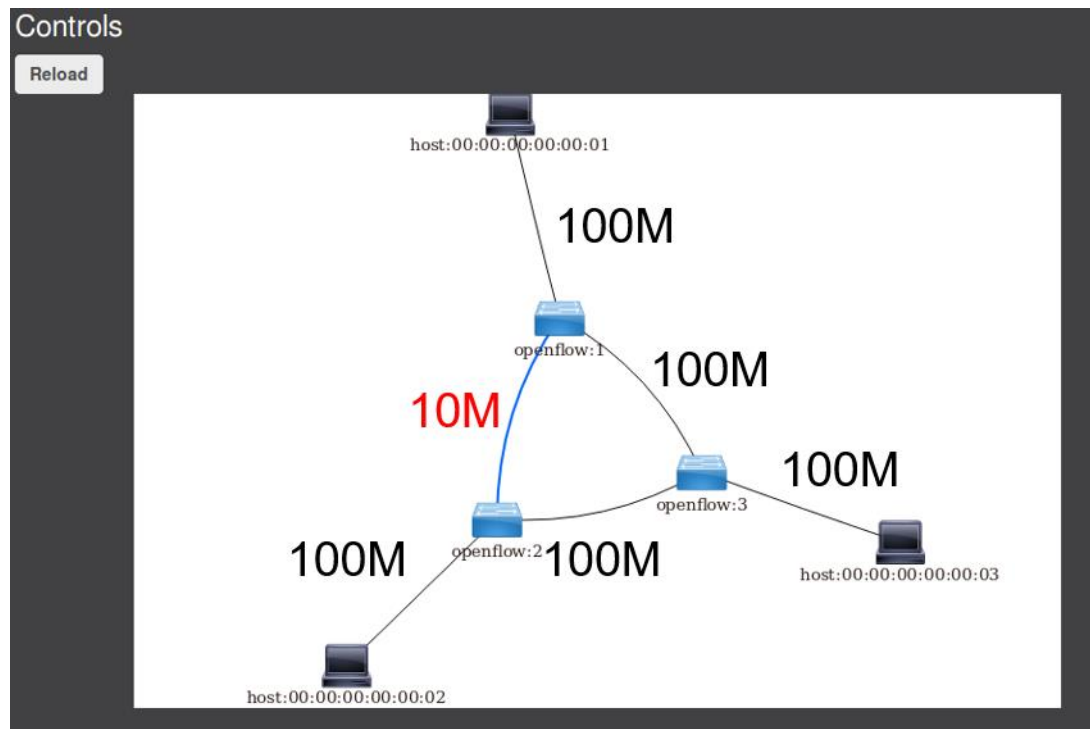
File Edit View Search Terminal Tabs Help
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data:
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.734 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.060 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.053 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.050 ms
^C
--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3054ms
rtt min/avg/max/mdev = 0.050/0.224/0.734/0.294 ms
mininet>
```



可以看到 Controller 和 Switch 連接後，對 Switch 下達 OpenFlow，一條為傳給 Controller，另一條為 output port 為 FLOOD。

## ● Task 4

Task4 目標為在 Mininet 中自訂拓樸方式，撰寫迴路拓樸，並將 openflow:1 與 openflow:2 間的 link bandwidth 設為 10M，其餘的 link 為 100M，接上 OpenDaylight 後，在 Topology 中如下所示(不會顯示 bandwidth)



(Hint: self.addLink( A, B cls=TCLink, bw=10))

Mininet

```
$ sudo mn --custom mininet/custom/YOURCUSTOM.py --topo mytopo --
controller remote --switch ovsk,protocols=OpenFlow13 --mac
```

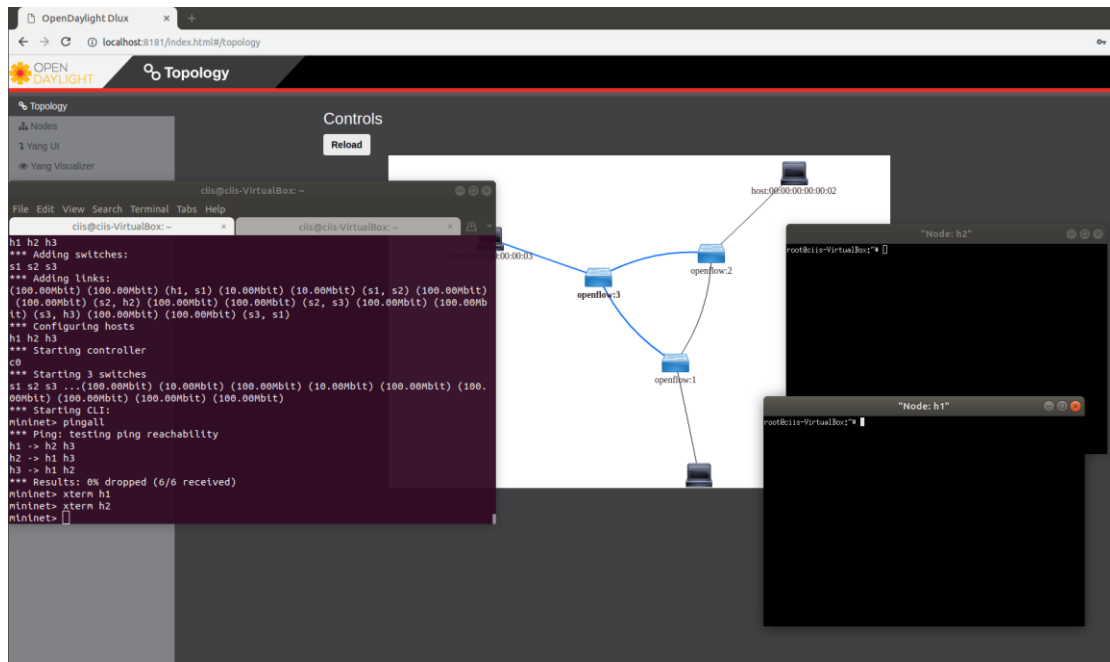
Open h1 and h2

```
$ xterm h1
```

```
$ xterm h2
```

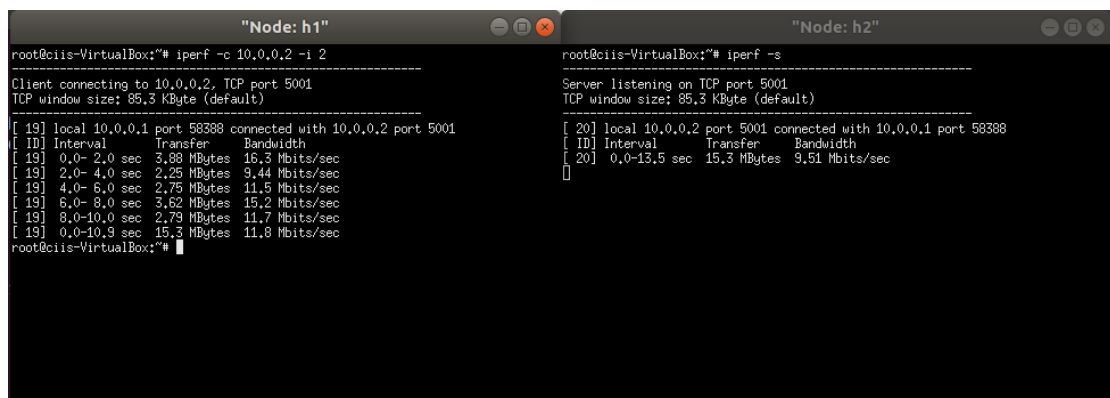
OpenDaylight

```
$ ./karaf-0.3.1/bin/karaf -of13
```



h1: iperf -c 10.0.0.2 -i 2

h2: iperf -s



在 OpenDaylight 控制下，會偵測迴路，因此 h1 與 h2 間的路徑不會經過 s3，故 bandwidth 約為 10Mbit/sec，此次實驗須下達 OpenFlow 後讓 h1 與 h2 間的 bandwidth 提升至 100Mbit/sec。

Hint:6 flows

Switch1:

dst=h1,send to host1(host1 mac addr)

dst=h2,send to switch3(openflow:)

Switch2:

dst=h2,send to host2 (host2 mac addr)

dst=h1,send to switch3(openflow:3)

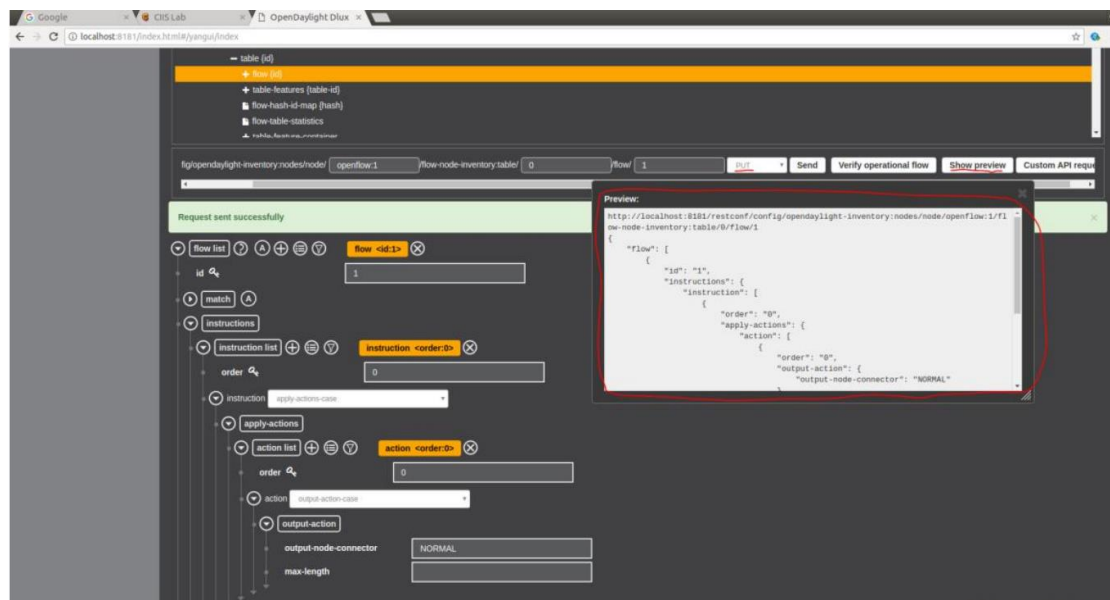
Switch3:

dst=h1,send to switch1(openflow:1)

dst=h2,send to switch2(openflow:2)

請參考[2]P.32~，經由 Yang UI 讓 Controller 對 Switch 下達 OpenFlow，並將由 Yang UI 的 API request 以 json 格式儲存。

Yang UI Json 格式取得方式:



## 五、實驗要求

Task1:請將 Postman 下達的 flow 集成一 Collection 並儲存，並也將實驗結果一併截圖繳交

Task2 及 Task3 :請在程式碼中加入學號，並將實驗結果截圖繳交

Task4:請將 REST 以 json 格式儲存並也將實驗結果一併截圖繳交

## 六、參考資料

### 1. 軟體定義網路 (SDN)

<https://www.xinguard.com/content.aspx?id=34>

### 2. OpenFlow 通訊協定

[https://osrg.github.io/ryu-book/zh\\_tw/html/openflow\\_protocol.html](https://osrg.github.io/ryu-book/zh_tw/html/openflow_protocol.html)