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：在 Minenet 模擬環境下實現速率控制

實驗單元：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：陳　陞

內容

一、 課程單元目標

實驗目標 1：修課學生得以了解 SDN 的基本觀念及架構。

實驗目標 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)的使用以及如何透過 Mininet 實現速率控制的概念。

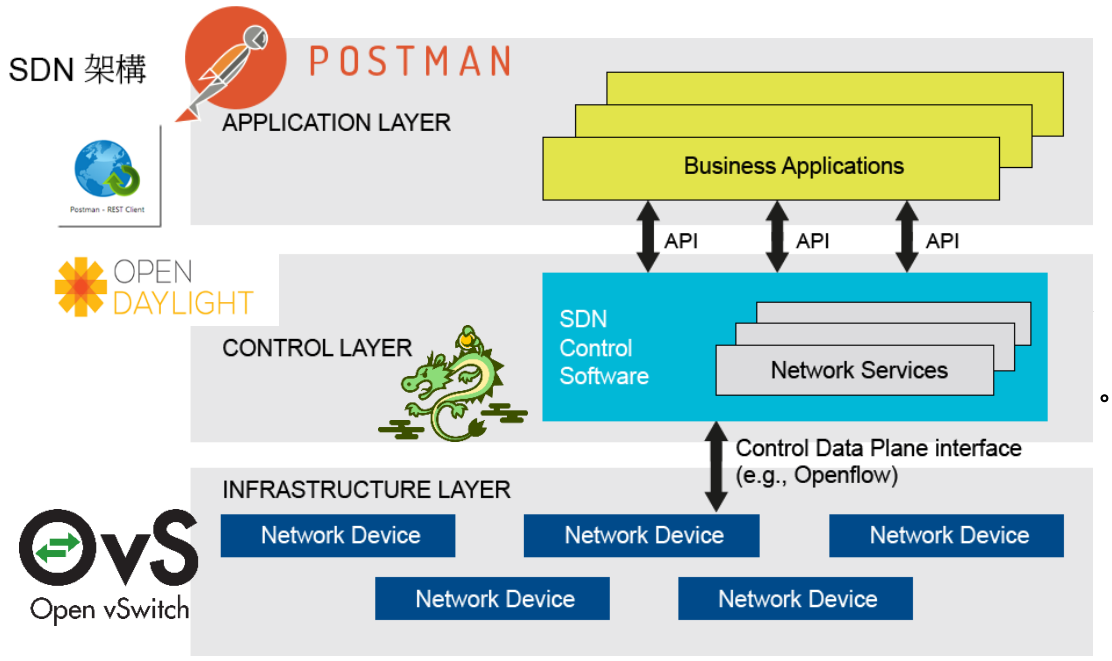
實驗目標 3：修課學生得以完成軟體定義網路(SDN)環境進行實驗及驗證 OpenFlow 執行。

二、 SDN 簡介與基本架構

2.1、 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。

2.2、 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樸的平台，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
  - OpenDaylight: Lithium-SR1
  - gcc 編譯器
  - Python 2.072
  - vi/vim 文字編輯器

四、 軟體介紹

4.1、 Mininet

Mininet 是一個網路仿真平台，通過這個平台，我們可以很方便的模擬真實環境中的網路操作與架構。

Mininet 自帶 switch、host、controller，同時，在 mininet 上可以安裝 OpenvSwitch 以及多種控制器（NOX\POX\RYU\Floodlight\OpenDaylight 等），同時，Mininet 可以運行在多種操作系統上（windows\linux\Mac OS），具有很強的系統兼容性。

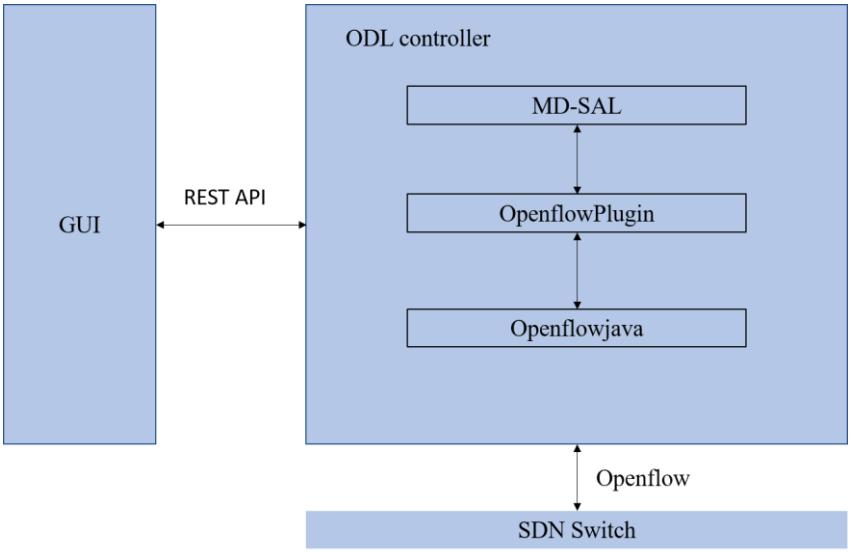
建立拓譜主要分成 2 種方式：

1. 利用終端機指令下參數，建立拓譜。

2. 撰寫 python 檔直接利用.py 檔撰寫程式指令以及建立拓譜。

4.2、 OpenDaylight

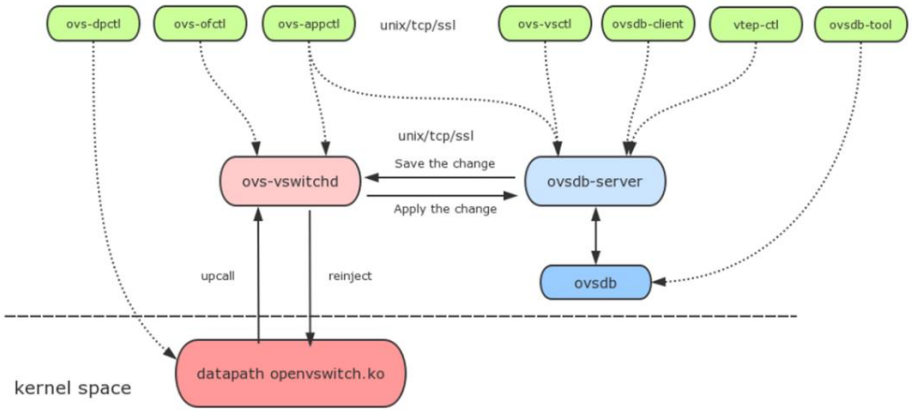
在架構中，Openflowjava 主要負責下方完成 Openflow 序列化、反序列化、端口監聽以及消息分布，OpenflowPlugin 則主要負責完成 Openflow 協議中的狀態管理、向 SAL 層提供服務，此外，OpenflowPlugin 也提供了 REST API 接口，提供使用者撰寫程式透過它完成對 Controller 的溝通。



4.3、 OpenvSwitch

一種虛擬交換器，可用來作為 L2switch，切割網域，QoS 或是流量監控，同時支持 OpenFlow 協定，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)。



1. ovs-dpctl ： 管理 ovs datapath，大部分資訊都是透過 netlink 反應。

2. ovs-vsctl : 對 ovsdb 操作，bridge,interface 新增，刪除，查詢，操作指令。

3. ovs-ofctl : openflow switch 管理工具，可以操作與 openflow 相關的設定。

4. ovs-appctl : ovs-vswithd 的管理工具，可以跟 ovs-vswitchd 程序溝通，ex:ofproto/trace 可用來追蹤封包 flow。

5. ovs-vswitchd：主要模組，實現內核 datapath upcall 處理以及 ofproto 查表，同時是 dpdk datapath 處理程序。

五、 安裝以及執行步驟教學

以下為實驗教學步驟，針對此次要做的實驗部分，做出詳細說明，請參考下面步驟完成實驗。

5.1、 安裝 Mininet 步驟教學

Step 1: 更新系統以及安裝 git 套件

```
$ sudo apt-get    update

$ sudo apt-get    upgrade

$ sudo apt-get install -y git
```

小提醒

update:更新我們的套件清單 /etc/apt/sources.list，這樣在我們更新時才能比對最新的套件清單，決定是否更新。

upgrade:相較於 update，它判斷是否更新套件標準是依據是否有相依性問題。

dist-upgrade:相較於 upgrade，它遇到相依性問題時，會試著透過安裝以及移除，將相依性解決並更新。(相對不安全的更新)

Step 2: 安裝 Mininet

利用 cd 指令到你想安裝 Mininet 的位置

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

$ sudo cd mininet/util

$ sudo ./install.sh –a

小提醒

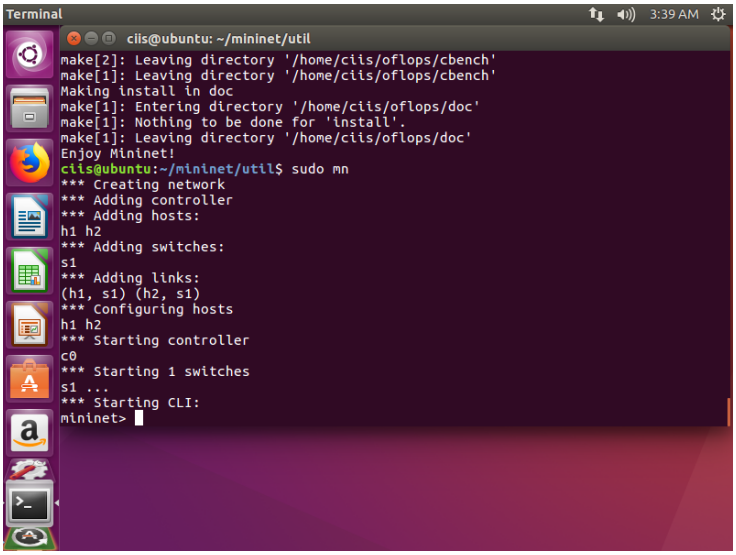
install.sh -a：安裝所有 mininet 的套件。

install.sh -s ：指定目錄 -a，在指定目錄下安裝所有 mininet 的套件 。

install.sh -nfv ：安裝 mininet + 自訂 switch + Open vSwitch。

Step 3: 測試安裝完成 Mininet 是否可用

$ sudo mn：自動建立簡單拓譜。



實驗 1：

任務 1：顯示 h1 的 IP address 以及 h2 的 MAC address，實驗結果以截圖方式呈現。

任務 2：顯示彼此節點透過 pingall 所得到的結果，實驗結果以截圖方式呈現。

5.2、 安裝及執行-OpenDaylight

Step 1:安裝 java jdk

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

Step 2:設置 java 環境變數，可以透過 vim 指令修改

$ sudo vim /etc/profile

將“JAVA_HOME=java 安裝路徑“ 加入此檔案最後一行，接著儲存之後離開。

$ sudo source /etc/profile　//引入環境變數

主要下載方式有 2 種：

1.官網中直接透過瀏覽器下載(tar / zip 格式均可)

2.透過 wget 指令下載

Step 3:透過以上 2 種方式擇一下載檔案並進行解壓縮

$ sudo wget　“[OpenDayLight 的網站位址]”

$ sudo tar -zxvf [檔案名稱].tar.gz　//解壓縮

Step 4:安裝 ODL 以及所需 feature

$ sudo cd [資料夾名稱]

$ sudo cd bin

$ sudo ./karaf

$ feature:install odl-mdsal-clustering odl-restconf odl-l2switch-switch-ui odl-dlux-all

提醒：安裝 feature 相關指令

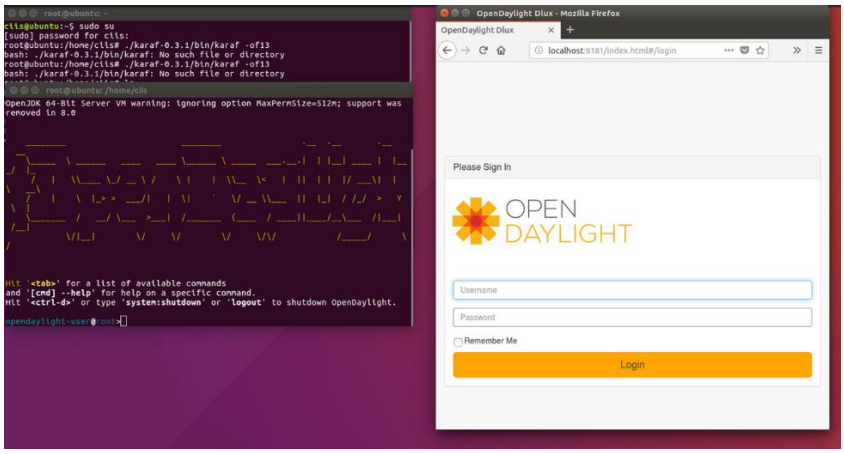
feature:list　　//查看所有 features

feature:list -i　　// 查看已安裝的 features

feature:install [feature 名稱]　　//安裝 features

Step 5:開啟 dlux 介面

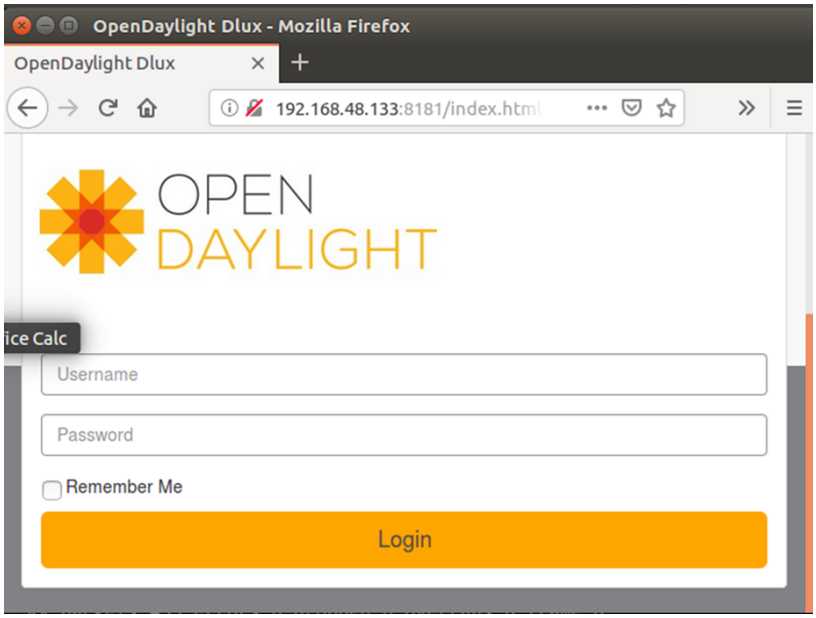
開啟 browser 輸入{ip}:8181/index.html，如果安裝完成就能看到 dlux 介面，帳密預設都是 admin。



5.3、 安裝及執行-OpenDaylight 下發 flow 教學

在這個階段，開始介紹及教學有關如何透過 OpenDaylight dlux 介面下發 Openflow。

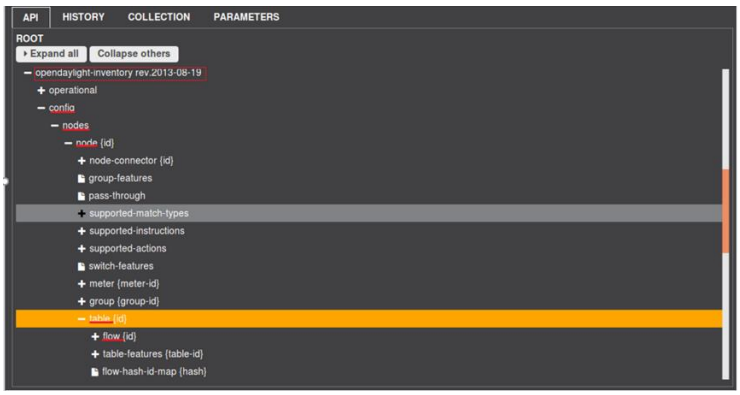
Step 6：開啟 dlux 介面，(http://<自己的 ip>:8181/index.html)，帳密皆為 admin



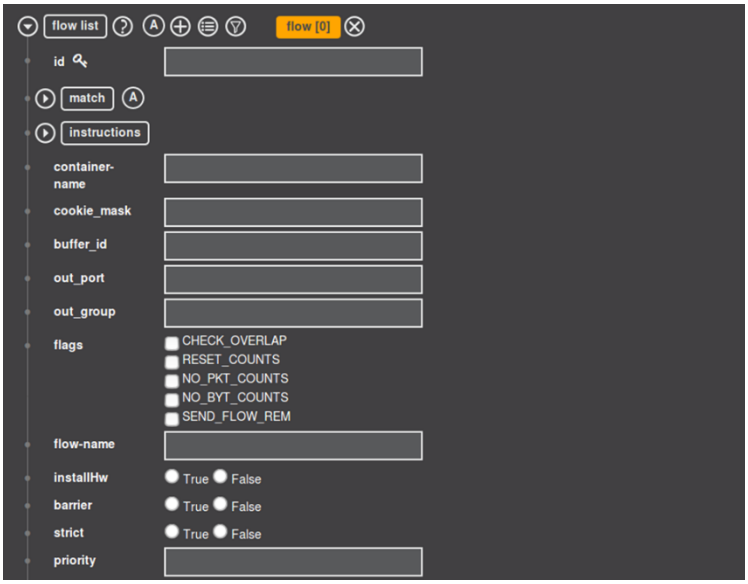
Step 7：根據你的使用目的，你可以從下面表格中找到你要的 direction 以及 table。

propose	action	direction
Add Flow	PUT	config/opendaylight-inventory:nodes/node/openflow:1/table/0/flow/1
Delete Flow	DELETE	config/opendaylight-inventory:nodes/node/openflow:1/table/0/flow/1
Get Flow Config	GET	config/opendaylight-inventory:nodes/node/openflow:1/table/0
Get Flow Operational	GET	operational/opendaylight-inventory:nodes/node/openflow:1/table/0
GET Inventory	GET	operational/opendaylight-inventory:nodes/
GET Topology	GET	operational/network-topology:network-topology/

Step 8：透過下圖找到 direction 並且開始管理 flows，以 Add flow 做為範例。



Step 9：透過 dlux 介面，可以透過選單引導，一步步填寫完成增加新的 flow。

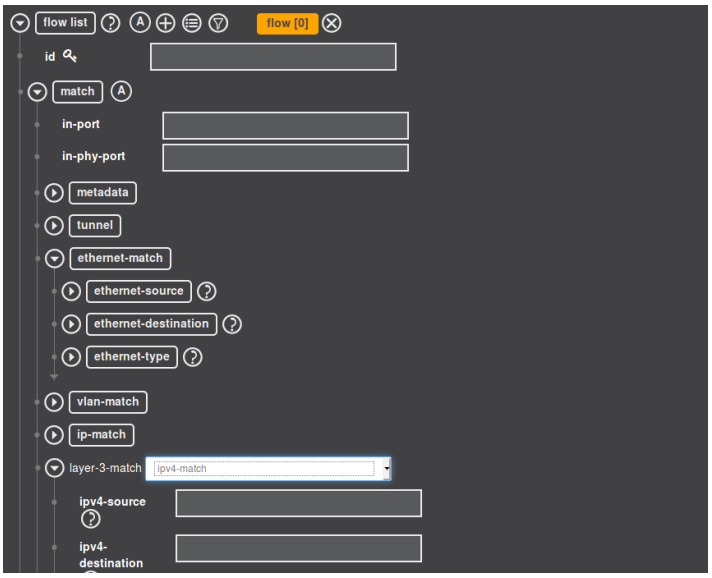


5.4、 OpenDaylight 下發 flow 教學

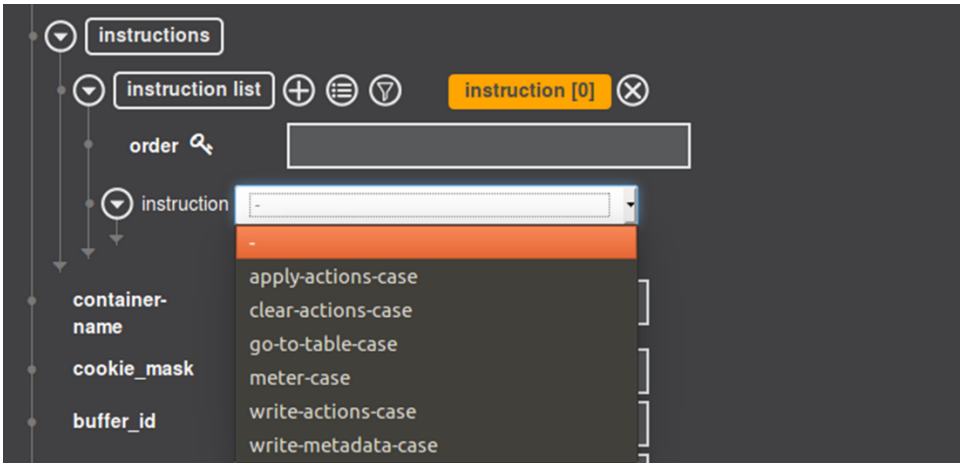
Step 1：透過”+” 來 add flow。

Step 2：擴展 match，透過展開按鈕。

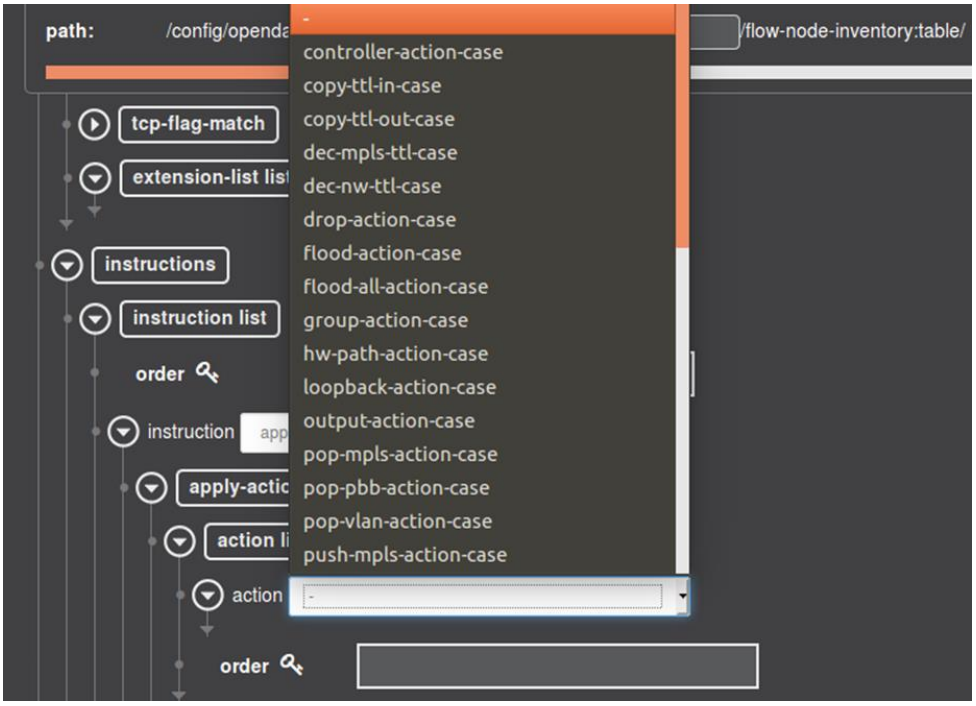
Step 3：打開”ethernet-match”，設定”layer-3-match”。



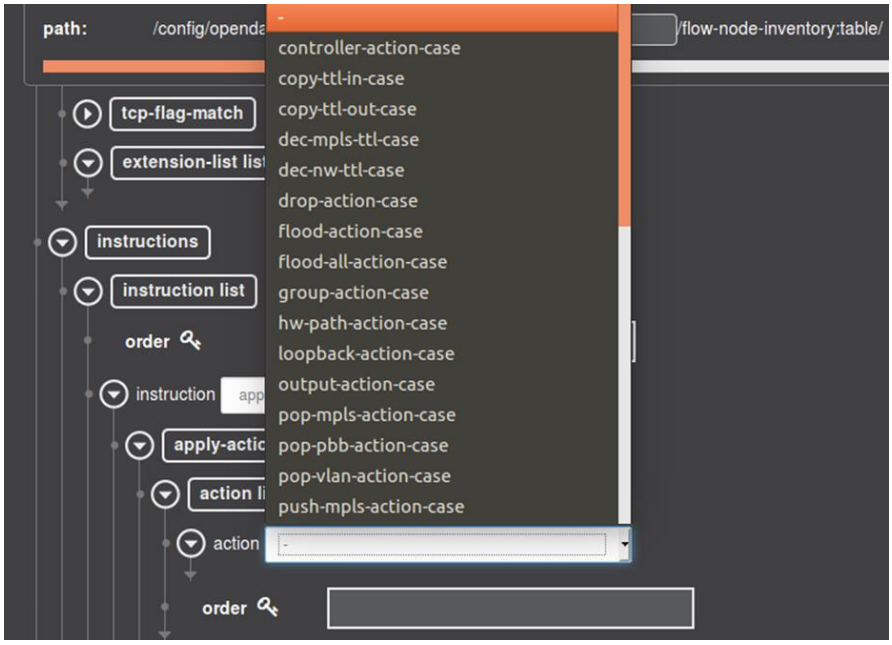
Step 4：在”instruction set”，可以挑選你想要的操作指令。



Step 5：選擇”apply-action-case”並且加入”action list”，選擇你要的行為。



Step 6：選擇”apply-action-case”並且加入”action list”，選擇你要的行為。



實驗 2：

任務 1：透過指令操作，將 mininet 建立的拓譜上的 controller 設定成 OpenDaylight controller，並且透過 dlux 顯示 mininet 拓譜情形，結果以截圖呈現。

任務 2：創建一個拓譜有 2 個 switch(s1、s2)，分別 s1 有 1 個 host(h1)，s2 有 3 個 host(h2、h3、h4)，並且透過 dlux 顯示 mininet 拓譜情形，結果以截圖呈現。

任務 3：利用 mininet 建立一個有三個 host 的拓譜，並且利用 OpenDaylight 下發 openflow 使得有一個 host 無法與其他 host 溝通，而另外 2 個 host 仍能保持聯繫，結果以截圖呈現。

5.5、 利用 OpenDaylight 下發 openflow meter

Meter Table 是由多個 Meter Enties 構成，每個 Meter Entry 定義每個 Flow 的 meters。基於此結構，OpenFlow Switch 可以實現各種簡單的 QoS 功能，例如：速率限制。

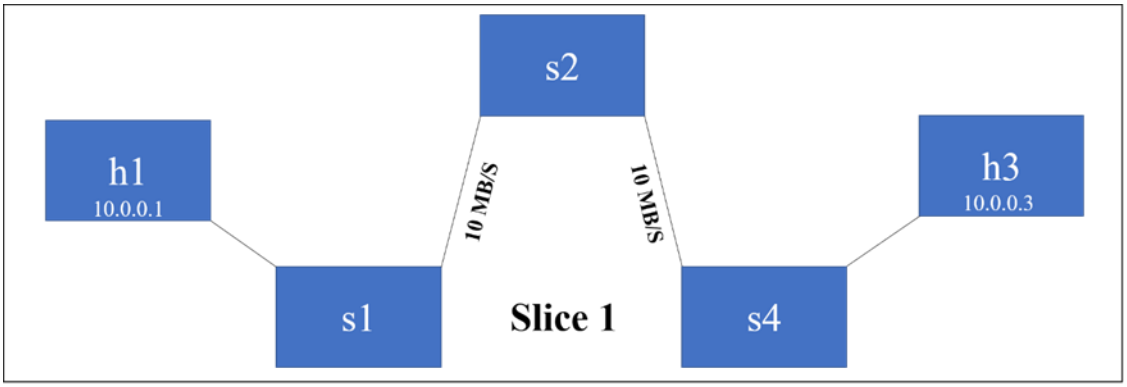
下圖為 meter 封包架構：

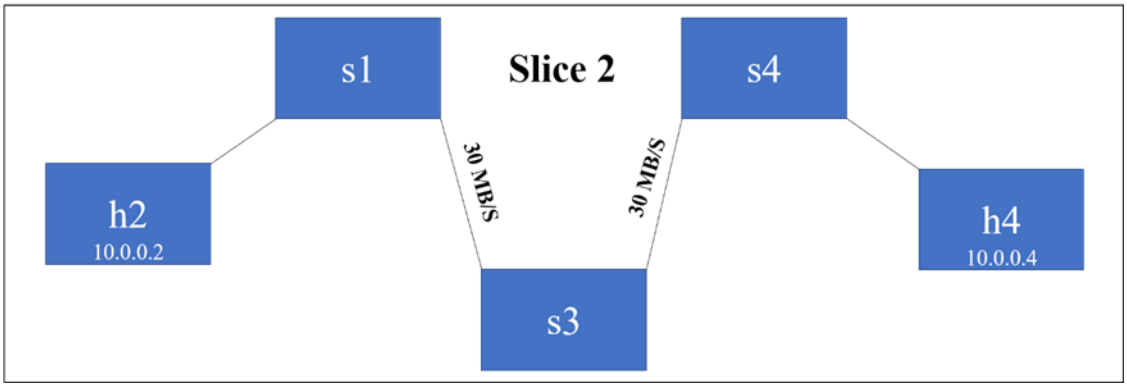
Meter Identifier	Meter Bands	Counters

一個 meter 可以衡量與它關聯的數據包的速率，並進而可以控制其聚合速率。任何一個 Flow Entry 都可以在其 Instructions Set 裡指定某一個 Meter，從而控制與該 Flow Entry 能夠成功匹配的數據包的聚合速率。每個 Meter Entry 都是由其 Meter Identifier 來唯一定位，詳情如下：

(1) Meter Identifier：一個 32 符號整数，作為一個 Meter Entry 的唯一標誌。
(2) Counters：被該 Meter Entry 處理過的數據包的統計量。
(3) Meter Bands：一個無序的 Meter Band 集合，每個 Meter Band 指明了頻寬速率以及處理數據包的行為。

接下來請利用前面學到的方法，嘗試利用.py 檔撰寫以下拓譜：





小提醒：如果無法完成參考下方拓譜 code 進行修改

buntu: /home/cils

```python
#!/usr/bin/python
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.node import CPULimitedHost
from mininet.link import TCLink
from mininet.util import dumpNodeConnections
from mininet.log import setLogLevel

class SingleSwitchTopo(Topo):

    def __init__(self, n=2, **opts):

        Topo.__init__(self, **opts)
        switch = self.addSwitch('s1')

        for h in range(n):
            host = self.addHost('h%s' % (h + 1), cpu=.5/n)
            self.addLink(host, switch, bw=10, delay='5ms', loss=0, max_queue_size=1000, use_htb=True)
def perfTest():
    topo = SingleSwitchTopo(n=4)
    net = Mininet(topo=topo,host=CPULimitedHost, link=TCLink)
    net.start()
    print "Dumping host connections"
    dumpNodeConnections(net.hosts)
    print "Testing network connectivity"
    net.pingAll()
    print "Testing bandwidth between h1 and h3"
    h1, h3 = net.get('h1', 'h3')
    net.iperf((h1, h3))
    net.stop()
if __name__=='__main__':

    setLogLevel('info')

    perfTest()
```

1. 可以參考前面 Add flow 範例，透過 PUT 指令對 Switch 添加一個 Meter，可以參考下圖 URI 完成細節步驟。

```
<meter xmlns="urn:opendaylight:flow:inventory">
    <meter-id>5</meter-id>
    <flags>meter-kbps meter-burst</flags>
    <container-name>abcd</container-name>
    <meter-band-headers>
        <meter-band-header>
            <band-id>0</band-id>
            <meter-band-types>
                <flags>ofpmbt-drop</flags>
            </meter-band-types>
            <drop-rate>8000</drop-rate>
            <drop-burst-size>100</drop-burst-size>
            </meter-band-header>
    </meter-band-headers>
    <meter-name>Foo</meter-name>
</meter>
```

2. 可以透過前面 Add flow 範例，透過 PUT 指令對 Switch 添加一個帶有 Meter 的 flow entry，從來源 IP 到目的 IP 可以參考下圖 URI 完成細節步驟。

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
    <priority>1</priority>
    <flow-name>Foo</flow-name>
    <match>
        <ethernet-match>
            <ethernet-type>
                <type>2048</type>
            </ethernet-type>
        </ethernet-match>
        <in-port>48</in-port>
    </match>
    <id>2</id>
    <table_id>0</table_id>
    <instructions>
        <instruction>
            <order>0</order>
            <apply-actions>
                <action>
                    <order>0</order>
                    <output-action>
                <output-node-connector>68</output-node-connector>
                    </output-action>
                </action>
            </apply-actions>
        </instruction>
        <instruction>
            <order>1</order>
            <meter>
                <meter-id>5</meter-id>
            </meter>
        </instruction>
    </instructions>
</flow>
```

3. 可以透過前面 Add flow 範例，透過 PUT 指令對 Switch 添加一個 flow entry，從目的 IP 到來源 IP 可以參考下圖 URI 完成細節步驟。

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
    <priority>1</priority>
    <flow-name>Foo</flow-name>
    <match>
        <ethernet-match>
            <ethernet-type>
                <type>2048</type>
            </ethernet-type>
        </ethernet-match>
        <in-port>68</in-port>
    </match>
    <id>2</id>
    <table_id>0</table_id>
    <instructions>
        <instruction>
            <order>0</order>
            <apply-actions>
                <action>
                    <order>0</order>
                    <output-action>
                        <output-node-connector>48</output-node-connector>
                    </output-action>
                </action>
            </apply-actions>
        </instruction>
    </instructions>
</flow>
```

可以透過觀看 switch 的流表得知添加的 Openflow，並且透過 Iperf 驗證實驗結果。

提醒 1：必須先下 Meter 才能指定 Flow entry

提醒 2：burst size 如果沒有設置，可能會導致實驗失敗。

實驗 3：

透過上面參考實驗教學以及.py 撰寫實驗拓譜，並且透過 OpenDaylight 下發 Openflow，實現速率限制，並且透過測速軟體驗證結果。

Iperf 指令：

Iperf －s Server 端

iperf –c 192.168.3.58 –w 100M –t 120 –i 10

-c 192.168.3.58 :Server 端的 IP

-w 100M :測試的檔案大小

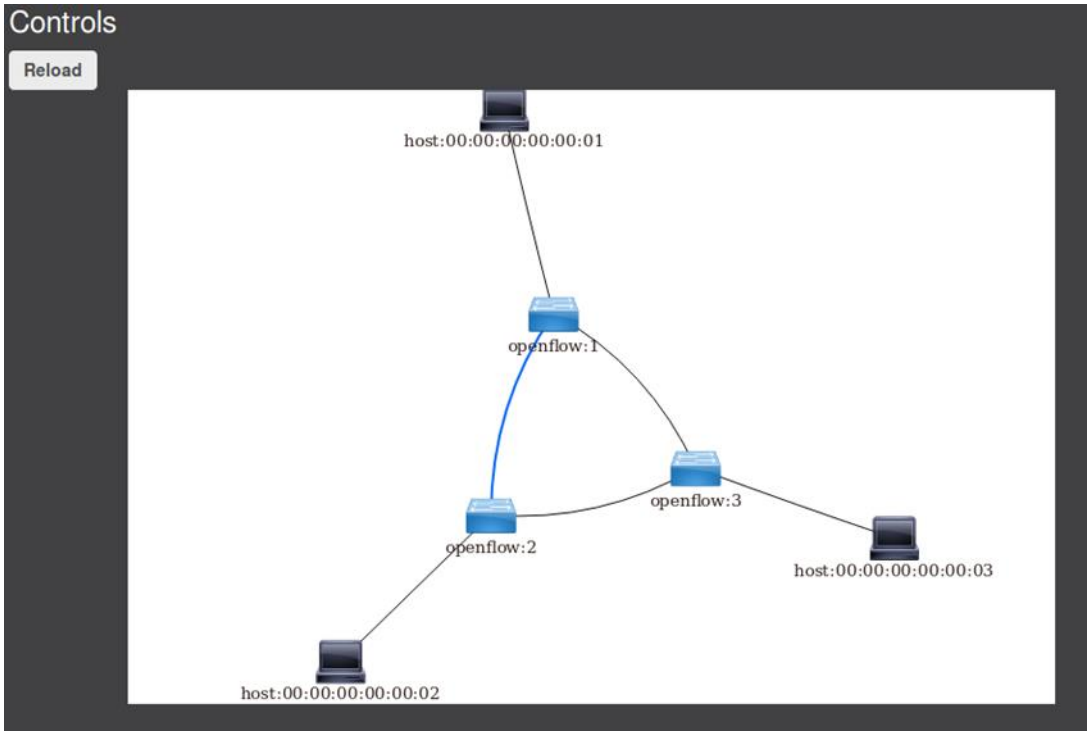
-t 120 :監視測量數據時間為 120 秒

-i 10 :每隔 10 秒將數據顯示出來

實驗 4：

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

self.addLink(Host1, Switch1, cls=TCLink, bw=10)



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)

啟動 OpenDaylight

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

透過 Mininet 執行拓譜以及實驗

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

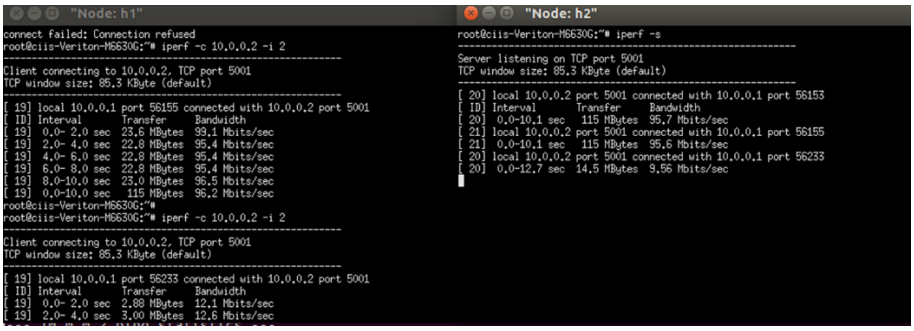
Iperf 指令：

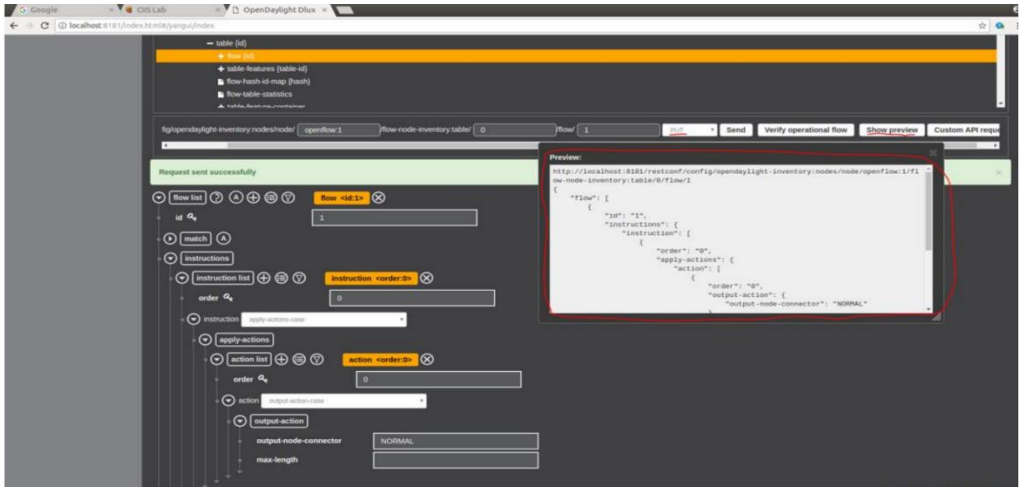
H2:server

H1:client

$ xterm h1; iperf -c h2 IPADDR –i 2

$ xterm h2; iperf -s





六、 實驗要求

實驗 1：需繳交實驗結果圖以及拓譜圖。

實驗 2：需繳交從 OpenDaylight 觀察 mininet 拓譜情形，並且說明如何連結完成以及下發 openflow(必須有步驟以及流程)。

實驗 3：需繳交拓譜.py 檔、每一個 switch 的流表截圖以及下發 Openflow 的 URI，最後說明步驟以及流程並且實際驗證結果。

實驗 4 ：請將 REST 以 Json 格式儲存並也將實驗結果一併截圖繳交。

七、 參考資料

[1] https://www.opendaylight.org/

[2] https://www.openvswitch.org/

[3] https://github.com/OSE-Lab/Learning-SDN/tree/master/Switch/OpenvSwitch/Walkthrough

[4] https://www.sdnlab.com/19448.html