

下世代Network Slicing模組設計

實驗單元：在Minenet模擬環 境下實現速率控制

國立中山大學 資訊工程系
授課教師：李宗南教授
教材編撰：陳 陞

目錄

CONTENTS

- 01 實驗目標
 - 02 實驗設備
 - 03 軟體介紹
 - 04 安裝以及執行步驟教學
 - 05 實驗要求
- 

01 課程目標



課程目標

- 課程目標1：修課學生得以理解 Mininet的網路環境模擬以及SDN控制器 OpenDaylight的使用以及OpenDaylight提供應用程式工具DLUX的使用。
- 課程目標2：修課學生得以完成軟體定義網路(SDN)環境下如何透過 Mininet與OpenFlow 實現速率控制的概念。

02 實驗設備



- 硬體:
 - 電腦：Ubuntu作業系統16.04
- 軟體
 - Mininet: 2.3.0d4
 - OpenDaylight: Lithium-SR1
 - gcc編譯器
 - Python 2.072
 - vi/vim文字編輯器

03 軟體介紹



軟體介紹-OpenDaylight(1)

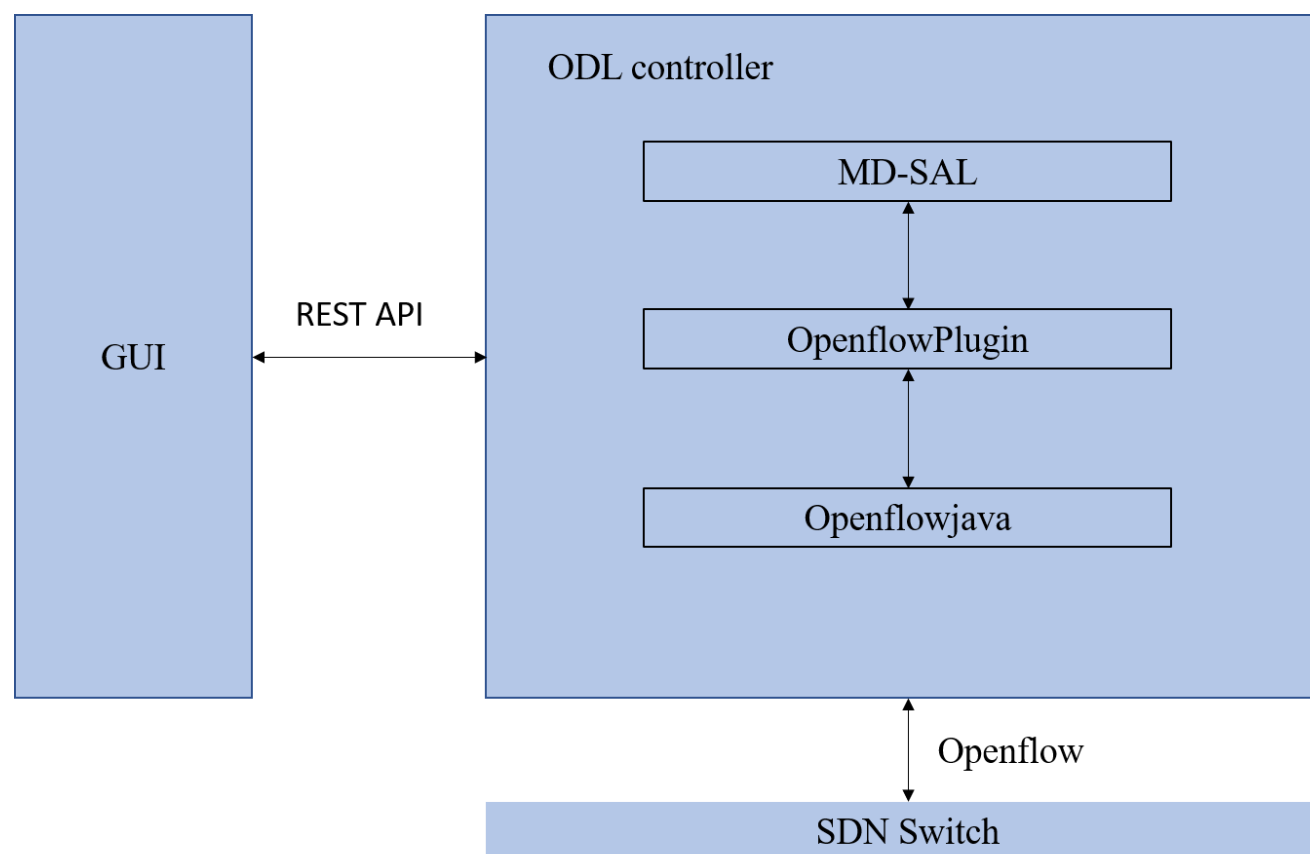
1.Linux的基金會負責管理的開源項目，提供一套基於SDN開發的模塊化，可擴展，可升級，支持多協議的控制器框架。

2.OpenDaylight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。



軟體介紹-OpenDaylight(2)

- 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。



04 安裝以及執行步驟教學



安裝及執行-Mininet(1)

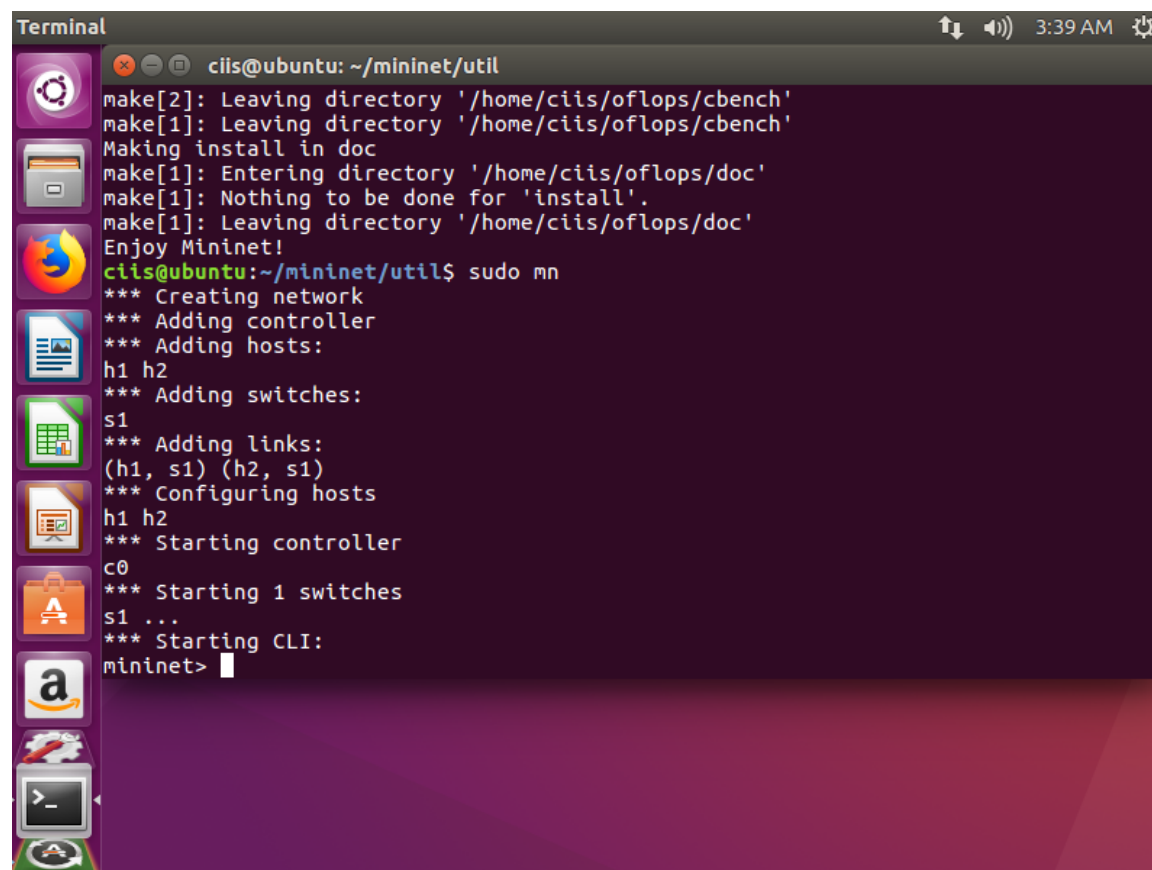
- Step I: 更新系統以及安裝git套件
 - \$ sudo apt-get update
 - \$ sudo apt-get upgrade
 - \$ sudo apt-get install -y git
- 小提醒
 - update:更新我們的套件清單 /etc/apt/sources.list, 這樣在我們更新時才能比對最新的套件清單, 決定是否更新。
 - upgrade:相較於update, 它判斷是否更新套件標準是依據是否有相依性問題。
 - dist-upgrade:相較於upgrade, 它遇到相依性問題時, 會試著透過安裝以及移除, 將相依性解決並更新。(相對不安全的更新)

安裝及執行-Mininet(2)

- Step II: 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

安裝及執行-Mininet(3)

- Step III: 測試安裝完成Mininet是否可用
 - \$ sudo mn
 - 自動建立簡單拓譜



A terminal window titled "Terminal" showing the execution of Mininet commands. The user is logged in as "ciis" on an "ubuntu" machine, with the current directory being "~/mininet/util". The terminal output shows the completion of previous steps, followed by the execution of "sudo mn". This command triggers a series of actions: creating a network, adding a controller, adding hosts (h1, h2), adding switches (s1), adding links ((h1, s1), (h2, s1)), configuring hosts, starting the controller (c0), starting one switch (s1), and finally starting the CLI. The prompt changes to "mininet>" after the last step.

```
Terminal
ciis@ubuntu: ~/mininet/util
make[2]: Leaving directory '/home/ciis/oflops/cbench'
make[1]: Leaving directory '/home/ciis/oflops/cbench'
Making install in doc
make[1]: Entering directory '/home/ciis/oflops/doc'
make[1]: Nothing to be done for 'install'.
make[1]: Leaving directory '/home/ciis/oflops/doc'
Enjoy Mininet!
ciis@ubuntu:~/mininet/util$ 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>
```

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。

安裝及執行-OpenDaylight(1)

- 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

安裝及執行-OpenDaylight(2)

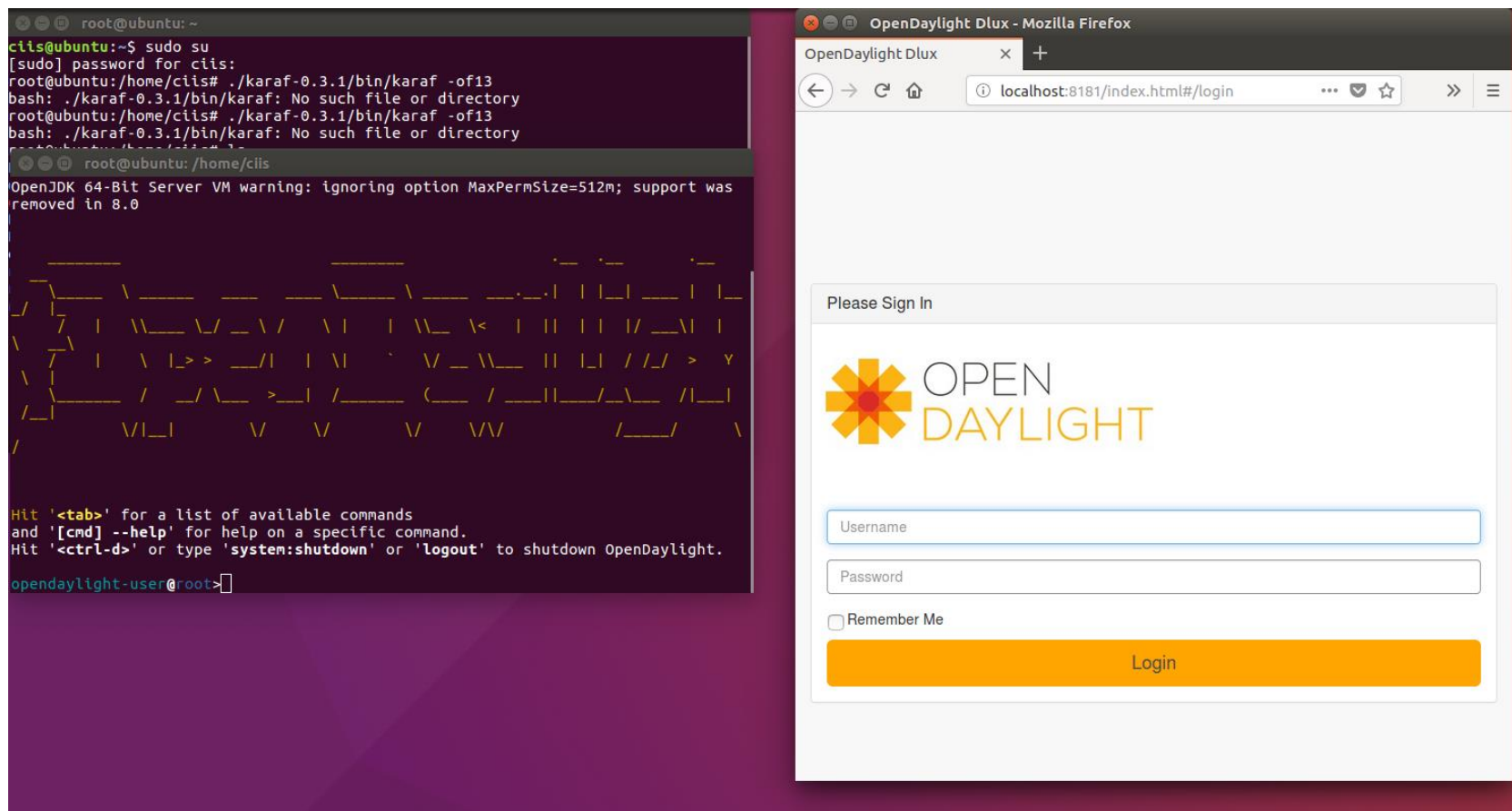
- 主要下載方式有2種：
 - 1.官網中直接透過瀏覽器下載(tar / zip 格式均可)
 - 2.透過wget 指令下載
- Step 3:透過以上2種方式擇一下載檔案並進行解壓縮
 - `$ sudo wget "[OpenDayLight的網部位址]"`
 - `$ sudo tar -zxvf [檔案名稱].tar.gz //解壓縮`

安裝及執行-OpenDaylight(3)

- 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

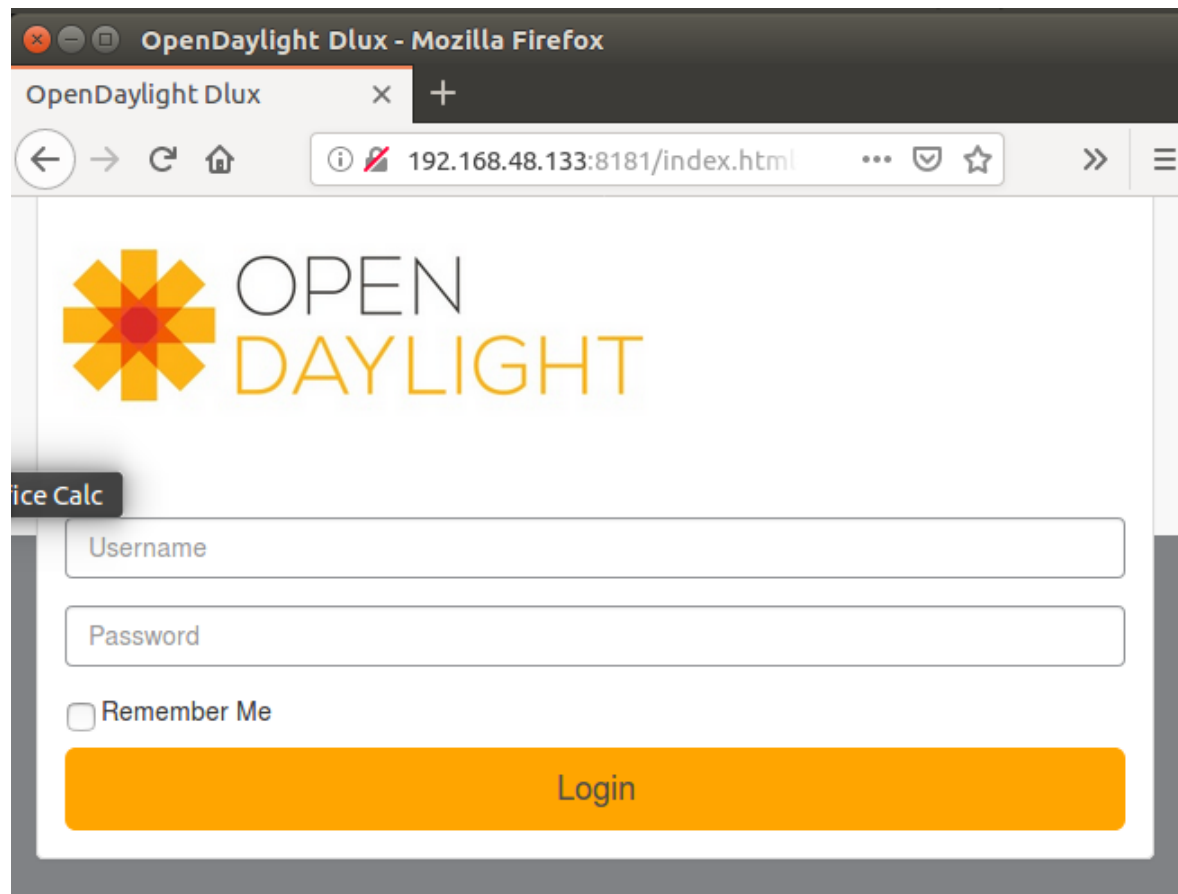
安裝及執行-OpenDaylight(4)

- Step 5: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。



安裝及執行-OpenDaylight下發flow教學(1)

- 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。



安裝及執行-OpenDaylight下發flow教學(2)

- 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/

安裝及執行-OpenDaylight下發flow教學(3)

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



安裝及執行-OpenDaylight下發flow教學(4)

- 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。

The screenshot shows the OpenDaylight dlux interface for adding a new flow. The interface is dark-themed and features a sidebar on the left with a vertical list of configuration steps: 'id', 'match', 'instructions', 'container-name', 'cookie_mask', 'buffer_id', 'out_port', 'out_group', 'flags', 'flow-name', 'installHw', 'barrier', 'strict', and 'priority'. The 'id' step is currently selected, and its configuration area is visible on the right. At the top of the interface, there is a 'flow list' dropdown, a help icon, and a 'flow [0]' button. The 'id' configuration area includes a text input field for the flow ID. Below the 'id' step, the 'match' and 'instructions' steps are visible, each with a play button icon. The 'container-name' step has a text input field. The 'cookie_mask' step has a text input field. The 'buffer_id' step has a text input field. The 'out_port' step has a text input field. The 'out_group' step has a text input field. The 'flags' step has a list of checkboxes: 'CHECK_OVERLAP', 'RESET_COUNTS', 'NO_PKT_COUNTS', 'NO_BYT_COUNTS', and 'SEND_FLOW_REM'. The 'flow-name' step has a text input field. The 'installHw' step has radio buttons for 'True' and 'False'. The 'barrier' step has radio buttons for 'True' and 'False'. The 'strict' step has radio buttons for 'True' and 'False'. The 'priority' step has a text input field.

安裝及執行-OpenDaylight下發flow教學(5)

- Step1: 透過“ + ” 來add flow。

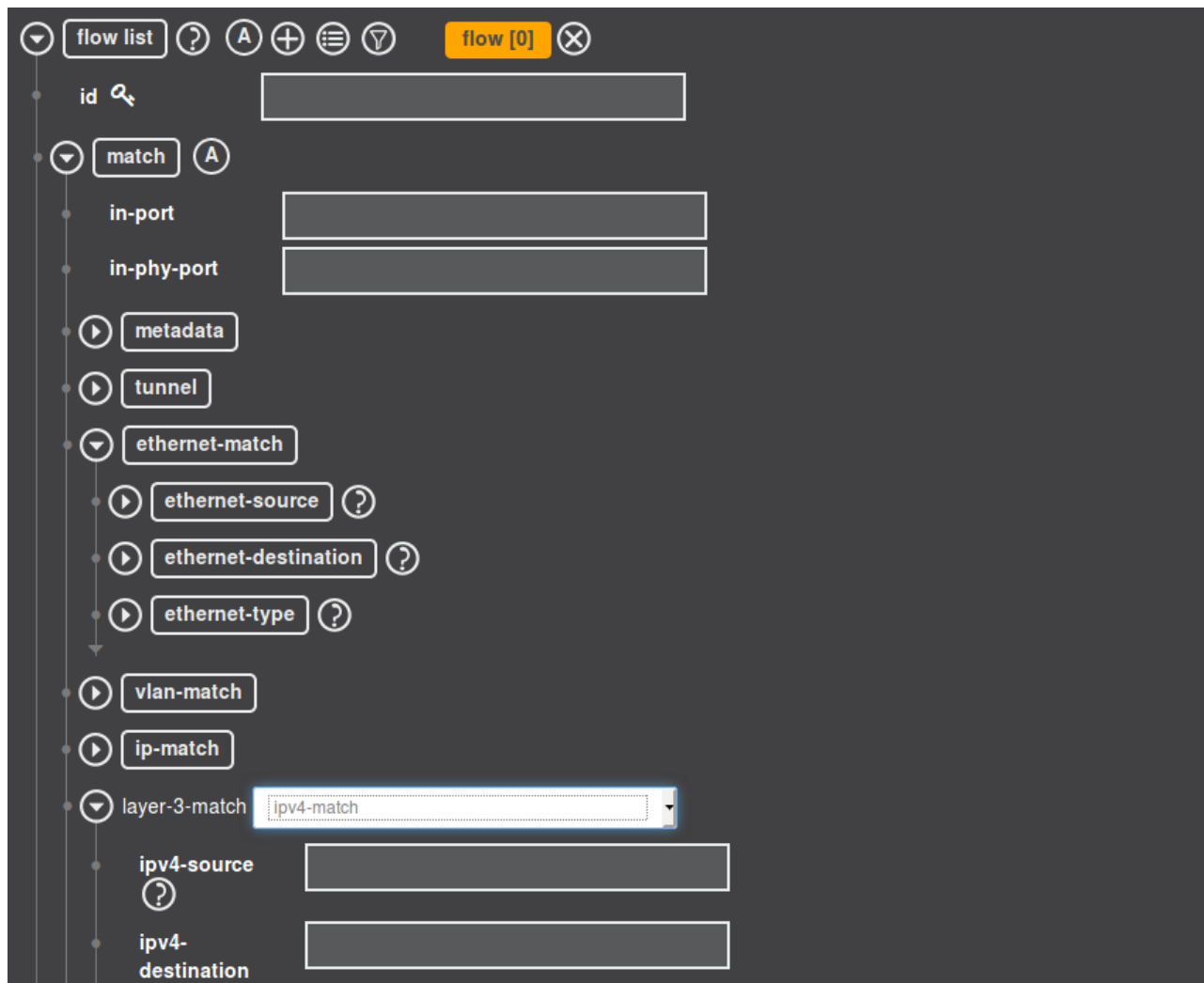


- Step2: 擴展match，透過展開按鈕。



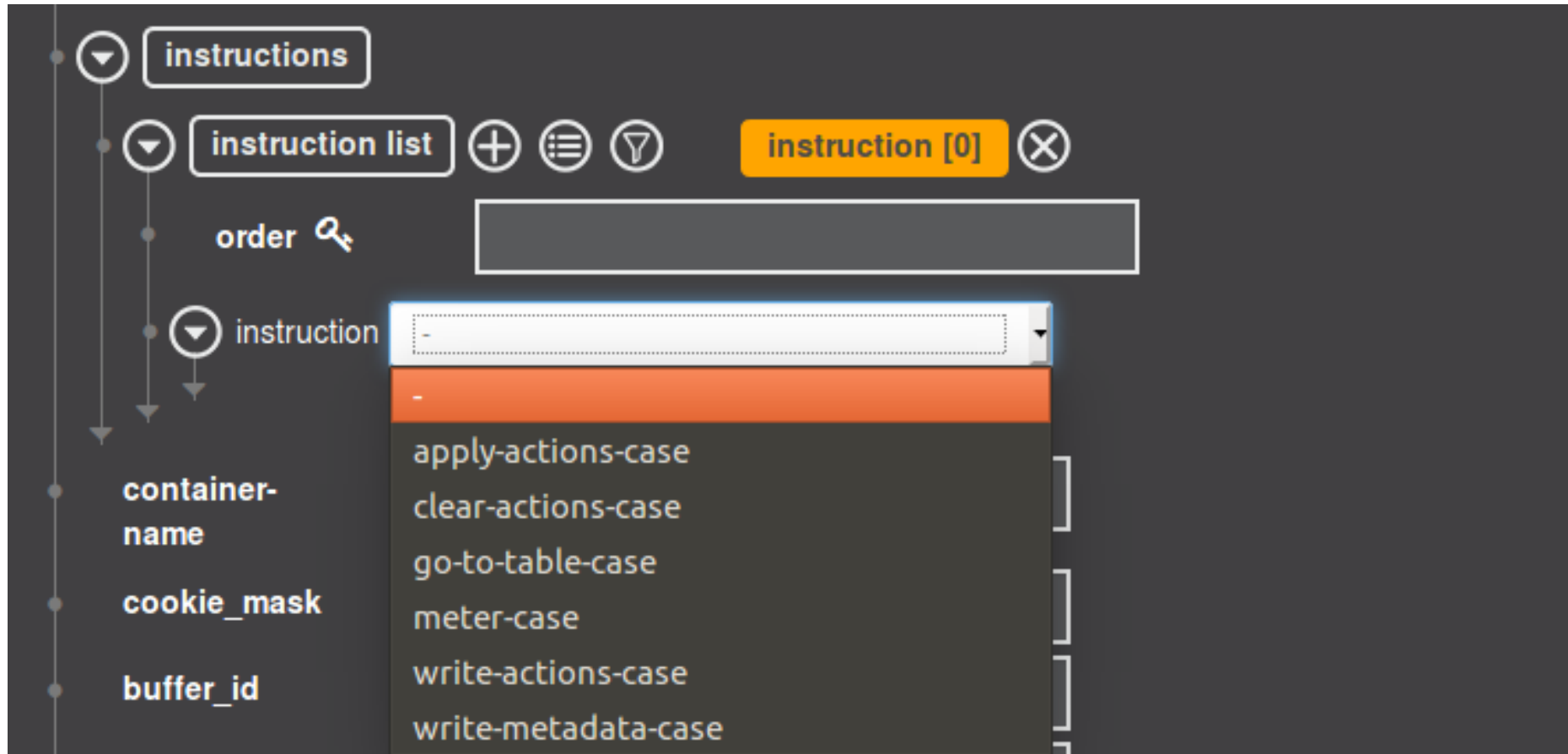
安裝及執行-OpenDaylight下發flow教學(6)

- Step3: 打開“ ethernet-match ”，設定“ layer-3-match ”。



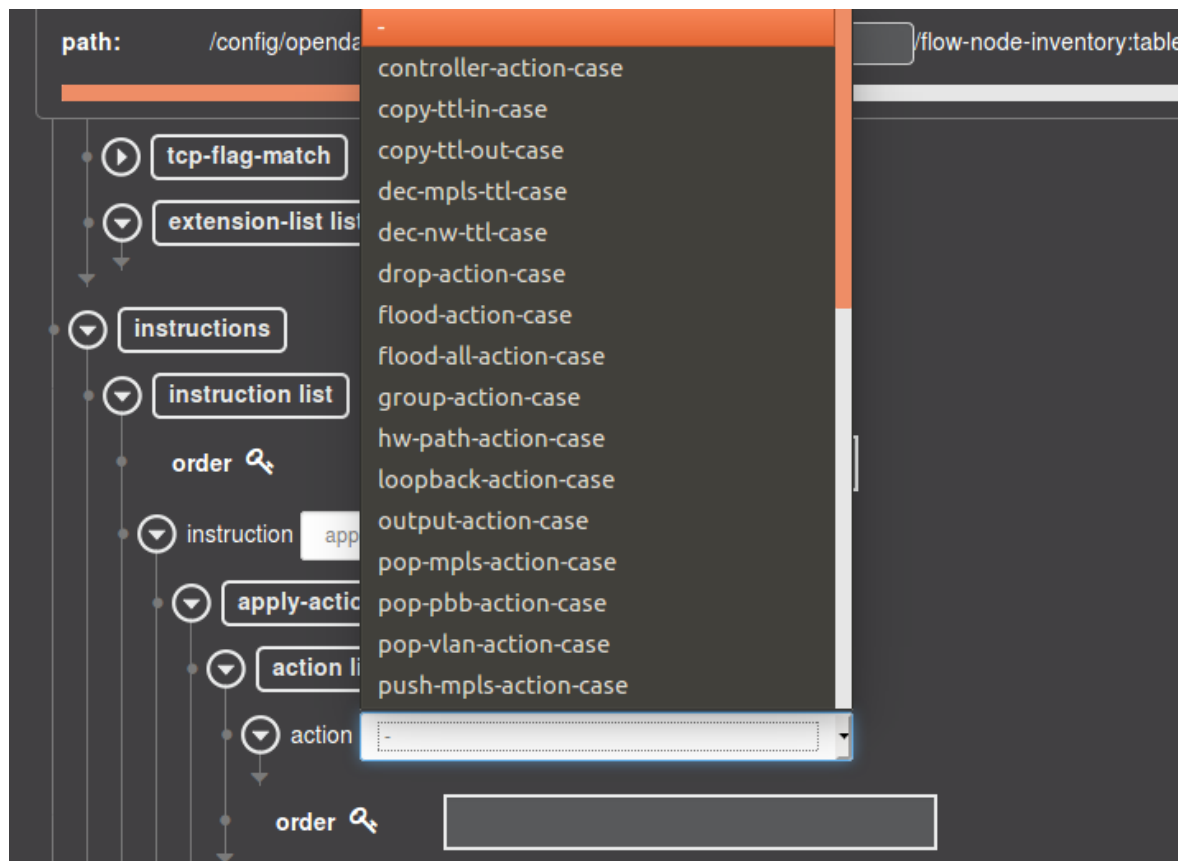
安裝及執行-OpenDaylight下發flow教學(7)

- Step4: 在“ instruction set” , 可以挑選你想要的操作指令。



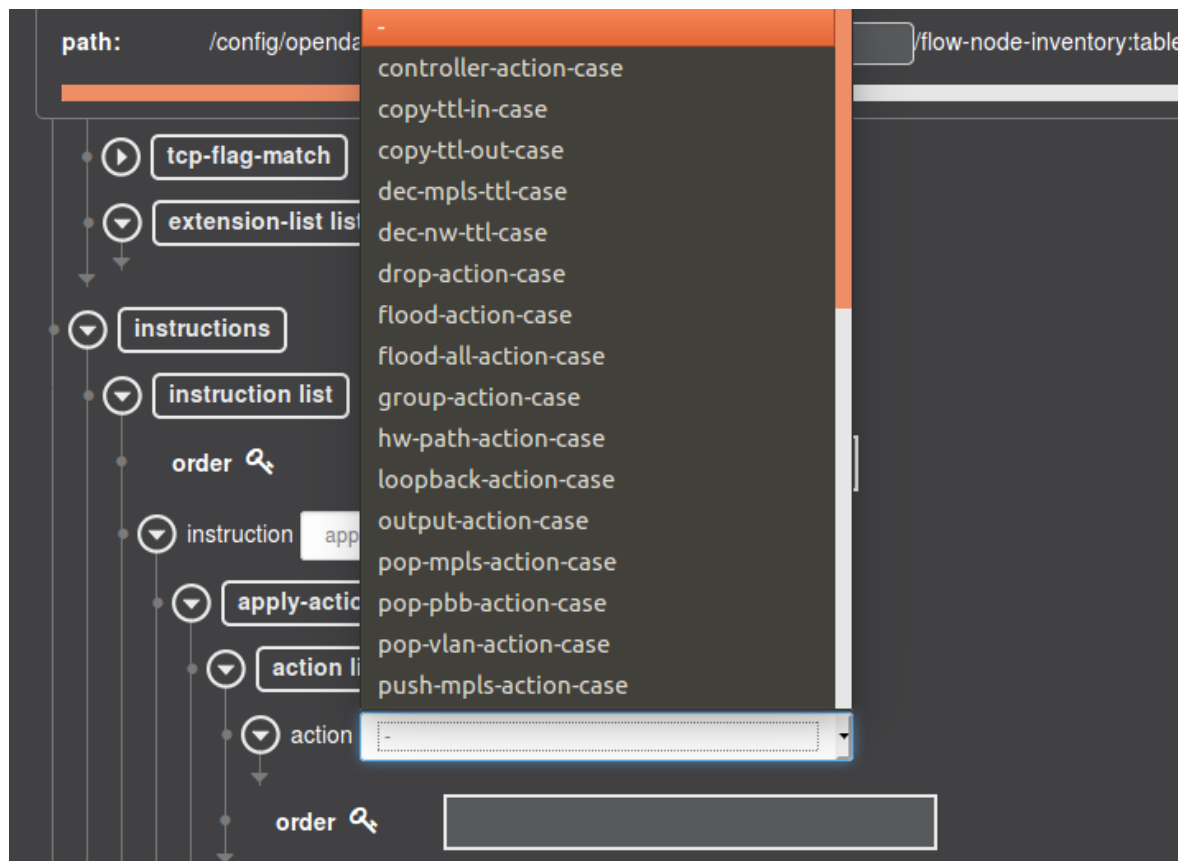
安裝及執行-OpenDaylight下發flow教學(8)

- Step5: 選擇“ apply-action-case” 並且加入“ action list” , 選擇你要的行為。



安裝及執行-OpenDaylight下發flow教學(8)

- 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。

Openflow Meter(1)

- Meter Table是由多個Meter Entries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：

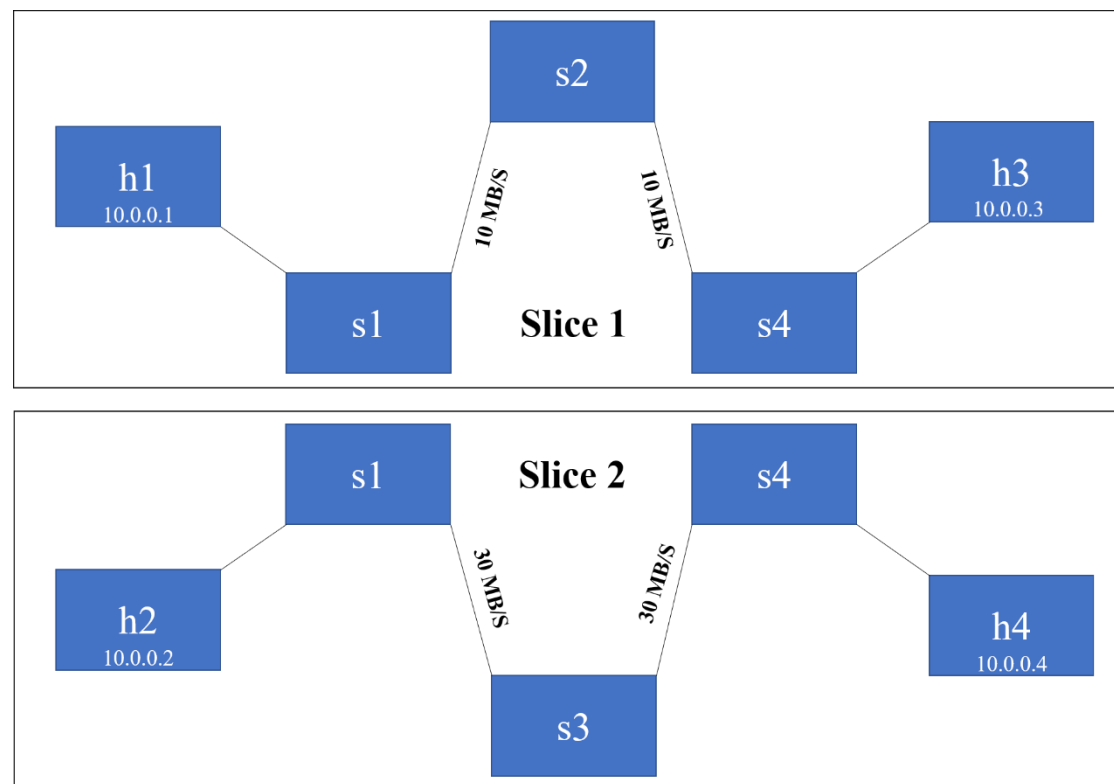
Meter Identifier	Meter Bands	Counters
-------------------------	--------------------	-----------------

Openflow Meter(2)

- 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符號整數，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。

實驗拓譜(1)

- 透過.py檔建立以下拓譜，作為這次實驗拓譜。



實驗拓譜(2)

- 可以參考下圖範例完成實驗拓譜。

```
buntu: /home/ciis
#!/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()
```


實驗OpenDaylight操作(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>
```

實驗OpenDaylight操作(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>
```

實驗OpenDaylight操作(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 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>
```

實驗提醒

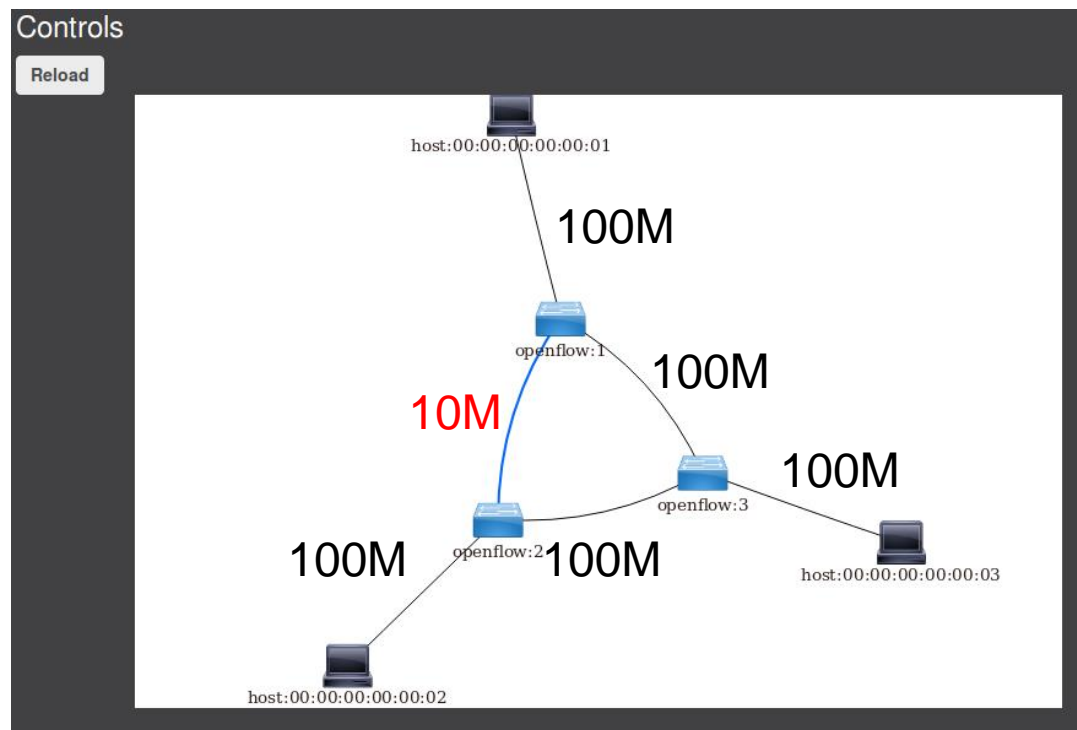
- 可以透過觀看switch的流表得知添加的Openflow，並且透過Iperf驗證實驗結果。
- 提醒1：必須先下Meter才能指定Flow entry
- 提醒2：burst size如果沒有設置，可能會導致實驗失敗。

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)`



實驗(4)

- 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)

實驗(4)

- OpenDaylight
- `$./karaf-0.3.1/bin/karaf -of13`
- Mininet
- `$ sudo mn --custom mininet/custom/yourtopo.py --topo yourtopo -
-controller remote --switch ovsk,protocols=OpenFlow13 --mac`

實驗(4)

- Iperf
 - H2:server
 - H1:client
- \$ xterm h1; iperf -c h2 IPADDR -i 2
- \$ xterm h2; iperf -s

```
"Node: h1"
connect failed: Connection refused
root@ciis-Veriton-M6630G:~# iperf -c 10.0.0.2 -i 2
-----
Client connecting to 10.0.0.2, TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 19] local 10.0.0.1 port 56155 connected with 10.0.0.2 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 19] 0.0- 2.0 sec   23.6 MBytes 99.1 Mbits/sec
[ 19] 2.0- 4.0 sec   22.8 MBytes 95.4 Mbits/sec
[ 19] 4.0- 6.0 sec   22.8 MBytes 95.4 Mbits/sec
[ 19] 6.0- 8.0 sec   22.8 MBytes 95.4 Mbits/sec
[ 19] 8.0-10.0 sec   23.0 MBytes 96.5 Mbits/sec
[ 19] 0.0-10.0 sec   115 MBytes 96.2 Mbits/sec
root@ciis-Veriton-M6630G:~#
root@ciis-Veriton-M6630G:~# iperf -c 10.0.0.2 -i 2
-----
Client connecting to 10.0.0.2, TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 19] local 10.0.0.1 port 56233 connected with 10.0.0.2 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 19] 0.0- 2.0 sec   2.88 MBytes 12.1 Mbits/sec
[ 19] 2.0- 4.0 sec   3.00 MBytes 12.6 Mbits/sec
-----
[ 19] 0.0-4.0 sec 6.88 MBytes 12.3 Mbits/sec

"Node: h2"
root@ciis-Veriton-M6630G:~# iperf -s
-----
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 20] local 10.0.0.2 port 5001 connected with 10.0.0.1 port 56153
[ ID] Interval      Transfer    Bandwidth
[ 20] 0.0-10.1 sec   115 MBytes 95.7 Mbits/sec
[ 21] local 10.0.0.2 port 5001 connected with 10.0.0.1 port 56155
[ 21] 0.0-10.1 sec   115 MBytes 95.6 Mbits/sec
[ 20] local 10.0.0.2 port 5001 connected with 10.0.0.1 port 56233
[ 20] 0.0-12.7 sec   14.5 MBytes 9.56 Mbits/sec
```

實驗(4)

The screenshot displays the OpenDaylight DLUX web interface for configuring a RESTCONF endpoint. The browser address bar shows `localhost:8181/index.html#/yangui/index`.

Tree View (Left):

- table {id}
- + flow {id}
- + table-features {table-id}
- flow-hash-id-map {hash}
- flow-table-statistics
- + table-features-mappings

Configuration Form (Center):

URL: `fig/opendaylight-inventory:nodes/node/` `openflow:1` `/flow-node-inventory:table/` `0` `/flow/` `1` `PUT` `Send` `Verify operational flow` `Show preview` `Custom API request`

Status: Request sent successfully

Flow Configuration:

- flow list: `flow <id:1>`
- id: `1`
- match: `match`
- instructions: `instructions`
- instruction list: `instruction <order:0>`
- order: `0`
- instruction: `apply-actions-case`
- apply-actions: `apply-actions`
- action list: `action <order:0>`
- order: `0`
- action: `output-action-case`
- output-action: `output-action`
- output-node-connector: `NORMAL`
- max-length:

Preview Window (Right):

Preview:

```
http://localhost:8181/restconf/config/opendaylight-inventory:nodes/node/openflow:1/flow-node-inventory:table/0/flow/1
{
  "flow": [
    {
      "id": "1",
      "instructions": {
        "instruction": [
          {
            "order": "0",
            "apply-actions": {
              "action": [
                {
                  "order": "0",
                  "output-action": {
                    "output-node-connector": "NORMAL"
                  }
                }
              ]
            }
          }
        ]
      }
    }
  ]
}
```

05 實驗要求



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

- [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>