

下世代Network Slicing模組設計

實驗單元： 開源軟體透過 Queue
設計實現保障優先度及流量管控

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 技術介紹
- 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：讓學生了解如何修改switch內部指令。
- 課程目標2：讓學生了解Queue的原理以及如何設計不同Queue的權重值以及優先度。
- 課程目標3：透過Queue來實現網路切片的流量保障以及優先度。

02 實驗設備



實驗設備

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

03 技術介紹



Switch架構介紹

- 當Data frame進入到Switch Interface會先進行classification以及making, 也可同時先做policing, 但對於router來說, 通常判斷packet的重要性, 是透過利用header的IPP或者DSCP來判斷packet的重要性, 但在switch的部份Data frame header中多一個選項, 就是參考Cos(class of service)值, 它跟IPP有點相似, 是一個3bit長度的值, 由0-7來判斷Data frame的重要性。

- Class of Service(CoS), 值是0-7, 透過設定Cos能過更精確的設定重要的 data frame。

用戶優先順序	流量類型
0	盡力服務 (Best Effort)
1	背景流量 (Background)
2	備用 (Spare)
3	很好服務 (Excellent Effort)
4	控制流量 (Controlled Load)
5	視頻, 小於 100ms的時延和抖動 ("Video," < 100 ms latency and jitter)
6	語音, 小於 10ms的時延和抖動 ("Voice," < 10 ms latency and jitter)
7	網路控制 (Network Control)

Queue 方法介紹

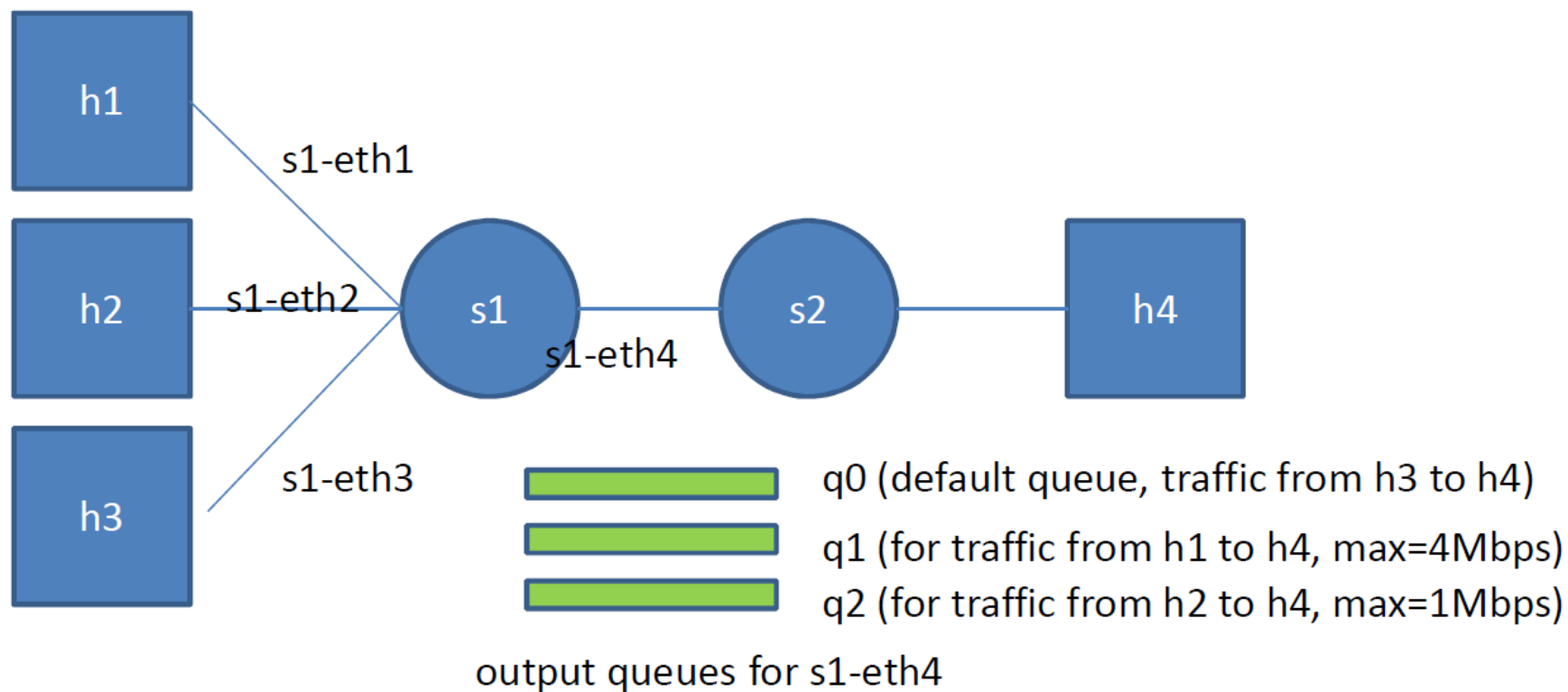
- Strictly Priority Queuing (SPQ): 這個方法很簡單，就是Q7送完換Q6送依序下去如果Q7一直沒送完則之後的Queue都不能送了
- Weighted Fair Queuing (WFQ): 每個Queue都可以設定權重，Switch會依據這些權重來分配最基本的保證頻寬，所以每個Queue都會有機會送出去資料。
- Weighted Round Robin Scheduling (WRR): Round Robin Scheduling的作法是指像排隊一樣，輪到了某一個Queue他就擁有全部的頻寬送完換下一個，自己就排到最後WRR也是用相同的作法，但是會根據priority跟weight去決定傳送的頻寬。

04 實驗以及執行步驟教學



實驗步驟(1)

- 首先安裝Mininet，在”在Mininet模擬環境下實現速率控制”這個實驗已經撰寫過如何安裝，這邊跳過。
- Step 1: 參考”在Mininet模擬環境下實現速率控制”撰寫出下圖拓譜的.py檔



實驗步驟(2)

- 可以參考下圖布建拓譜。

```
Terminal File Edit View Search Terminal Help
from mininet.topo import Topo

class MyTopo( Topo ):
    "Simple topology example."

    def __init__( self ):
        "Create custom topo."

        # Initialize topology
        Topo.__init__( self )

        # Add hosts and switches
        h1 = self.addHost( 'h1' )
        h2 = self.addHost( 'h2' )
        h3 = self.addHost( 'h3' )
        h4 = self.addHost( 'h4' )

        s1 = self.addSwitch( 's1' )
        s2 = self.addSwitch( 's2' )

        # Add links
        self.addLink( h1, s1 )
        self.addLink( h2, s1 )
        self.addLink( h3, s1 )
        self.addLink( s1, s2 )
        self.addLink( s2, h4 )

topos = { 'mytopo': ( lambda: MyTopo() ) }
```

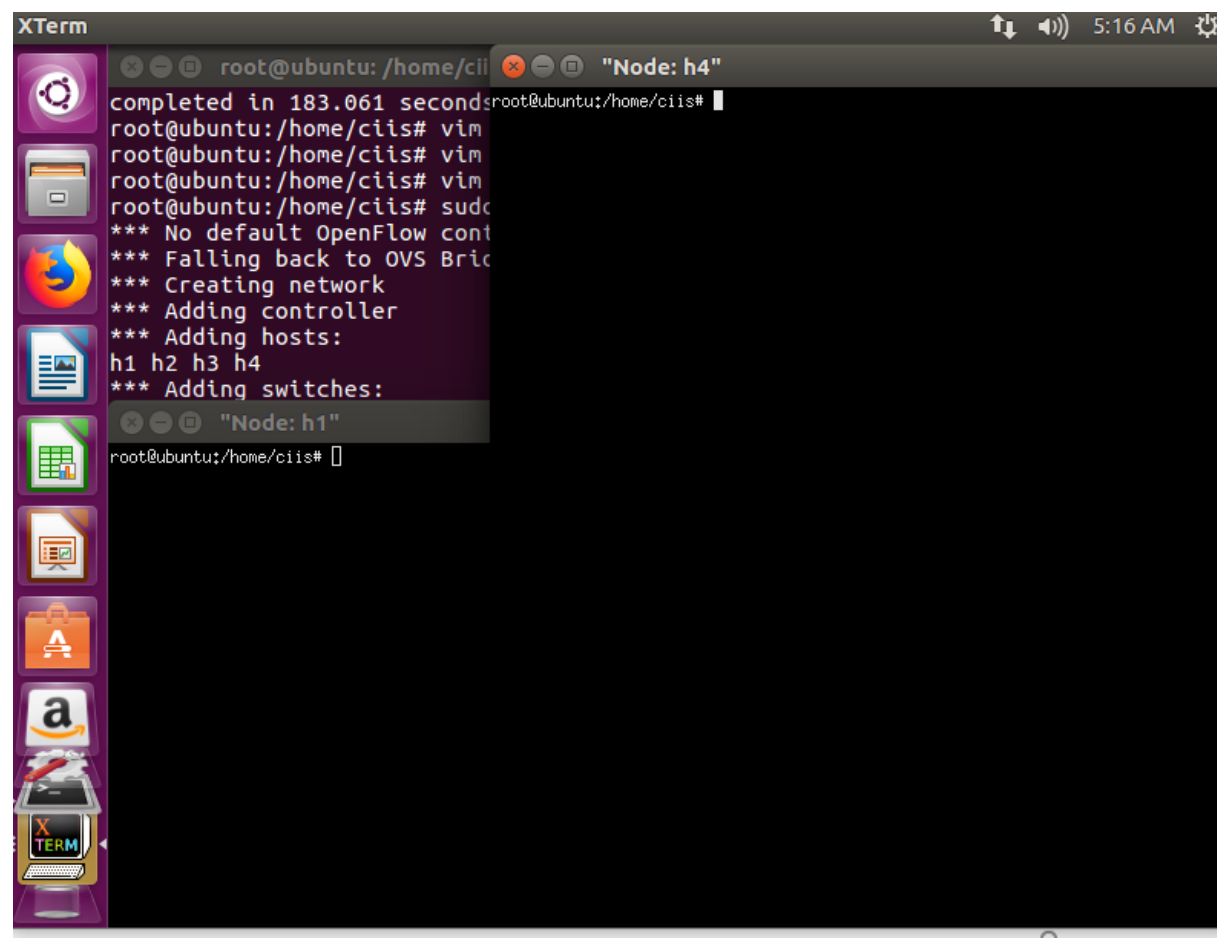
實驗步驟(3)

- 撰寫一個controller.py用來調整switch queue設定，並且監聽剛剛用Mininet建立的拓譜，並透過Mininet模擬一般使用情況製造流量，以驗證建立的queue是否能夠跟預期一樣效果。

```
root@ubuntu: /home/ciis
from pox.core import core
import pox.openflow.libopenflow_01 as of
from pox.lib.util import dpidToStr
log = core.getLogger()
s1_dpid=0
s2_dpid=0
def _handle_ConnectionUp (event):
    global s1_dpid, s2_dpid
    print "ConnectionUp: ",
    dpidToStr(event.connection.dpid)
    #remember the connection dpid for switch
    for m in event.connection.features.ports:
        if m.name == "s1-eth1":
            s1_dpid = event.connection.dpid
            print "s1_dpid=", s1_dpid
        elif m.name == "s2-eth1":
            s2_dpid = event.connection.dpid
            print "s2_dpid=", s2_dpid
    ~
    ~
    ~
    ~
    ~
"controller.py" 18L, 509C
```

實驗步驟(4)

- 透過Mininet xterm功能針對各個節點，並且分別透過iperf指令從h1、h2、h3到h4做測試一開始的速率。如下圖：



實驗步驟(5)

- 透過iperf分別進行測試，將h4當作server監聽port4000，可以發現再分開測試中三個host到達h4的上傳以及下載速度是相距不多的。

```
h3
"Node: h2"
UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
RX packets:67 errors:0 dropped:0 overruns:0 frame:0
TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:7212 (7.2 KB)  TX bytes:656 (656.0 B)

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)

root@ubuntu:/home/ciis# iperf -c 10.0.0.4 -p 4000
-----
Client connecting to 10.0.0.4, TCP port 4000
TCP window size: 85.3 KByte (default)
-----
[ 17] local 10.0.0.2 port 50184 connected with 10.0.0.4 port 4000
[ ID] Interval      Transfer    Bandwidth
[ 17] 0.0-10.0 sec  43.1 GBytes  37.0 Gbits/sec
root@ubuntu:/home/ciis#

"Node: h4"
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)

root@ubuntu:/home/ciis# iperf -s -p 4000 & [1] 12811
[1] 48255

Server listening on TCP port 4000
TCP window size: 85.3 KByte (default)
-----
[1]: command not found
root@ubuntu:/home/ciis# iperf -s -p 4000
bind failed: Address already in use
root@ubuntu:/home/ciis# [ 18] local 10.0.0.4 port 4000 connected with 10.0.0.2 port 49828
[ ID] Interval      Transfer    Bandwidth
[ 18] 0.0-10.0 sec  58.0 GBytes  49.8 Gbits/sec
[ 19] local 10.0.0.4 port 4000 connected with 10.0.0.3 port 47064
[ ID] Interval      Transfer    Bandwidth
[ 19] 0.0-10.0 sec  46.4 GBytes  39.9 Gbits/sec
[ 18] local 10.0.0.4 port 4000 connected with 10.0.0.2 port 50184
[ ID] Interval      Transfer    Bandwidth
[ 18] 0.0-10.0 sec  43.1 GBytes  37.0 Gbits/sec
[ 19] local 10.0.0.4 port 4000 connected with 10.0.0.1 port 49834
[ ID] Interval      Transfer    Bandwidth
[ 19] 0.0-10.0 sec  45.2 GBytes  38.8 Gbits/sec
root@ubuntu:/home/ciis#

"Node: h1"
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)

root@ubuntu:/home/ciis# iperf -c 10.0.0.4 -p 4000
connect failed: Connection refused
root@ubuntu:/home/ciis# iperf -c 10.0.0.4 -p 4000
-----
Client connecting to 10.0.0.4, TCP port 4000
TCP window size: 85.3 KByte (default)
-----
[ 17] local 10.0.0.1 port 49828 connected with 10.0.0.4 port 4000
[ ID] Interval      Transfer    Bandwidth
[ 17] 0.0-10.0 sec  58.0 GBytes  49.8 Gbits/sec
root@ubuntu:/home/ciis# iperf -c 10.0.0.4 -p 4000
-----
Client connecting to 10.0.0.4, TCP port 4000
TCP window size: 1.42 MByte (default)
-----
[ 17] local 10.0.0.1 port 49834 connected with 10.0.0.4 port 4000
[ ID] Interval      Transfer    Bandwidth
[ 17] 0.0-10.0 sec  45.2 GBytes  38.8 Gbits/sec
root@ubuntu:/home/ciis#
```


實驗步驟(6)

- 透過iperf分別進行測試，將h4當作server監聽port4000，可以發現再同時測試中三個host到達h4的上傳下載速度有什麼不同，並將原因解釋在你的回答中。

```
"Node: h2"
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)

root@ubuntu:/home/ciis# iperf -c 10.0.0.4 -p 4000
Client connecting to 10.0.0.4, TCP port 4000
TCP window size: 85.3 KByte (default)
[ 17] local 10.0.0.2 port 50184 connected with 10.0.0.4 port 4000
[ ID] Interval      Transfer    Bandwidth
[ 17] 0.0-10.0 sec  43.1 GBytes 37.0 Gbits/sec
root@ubuntu:/home/ciis# iperf -c 10.0.0.4 -p 4000
Client connecting to 10.0.0.4, TCP port 4000
TCP window size: 85.3 KByte (default)
[ 17] local 10.0.0.2 port 50192 connected with 10.0.0.4 port 4000
[ ID] Interval      Transfer    Bandwidth
[ 17] 0.0-10.0 sec  18.7 GBytes 16.1 Gbits/sec
root@ubuntu:/home/ciis#

"Node: h3"
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)

root@ubuntu:/home/ciis# iperf -c 10.0.0.4 -p 4000
Client connecting to 10.0.0.4, TCP port 4000
TCP window size: 85.3 KByte (default)
[ 17] local 10.0.0.3 port 47064 connected with 10.0.0.4 port 4000
[ ID] Interval      Transfer    Bandwidth
[ 17] 0.0-10.0 sec  46.4 GBytes 39.9 Gbits/sec
root@ubuntu:/home/ciis# iperf -c 10.0.0.4 -p 4000
Client connecting to 10.0.0.4, TCP port 4000
TCP window size: 85.3 KByte (default)
[ 17] local 10.0.0.3 port 47070 connected with 10.0.0.4 port 4000
[ ID] Interval      Transfer    Bandwidth
[ 17] 0.0-10.0 sec  19.1 GBytes 16.4 Gbits/sec
root@ubuntu:/home/ciis#

"Node: h4"
[1]: command not found
[2]+ Done iperf -s -p 4000
root@ubuntu:/home/ciis# iperf -s -p 5000 & [1] 12816
[2] 48358

Server listening on TCP port 5000
TCP window size: 85.3 KByte (default)

[1]: command not found
root@ubuntu:/home/ciis# iperf -s -p 6000 & [1] 12819
[3] 48363

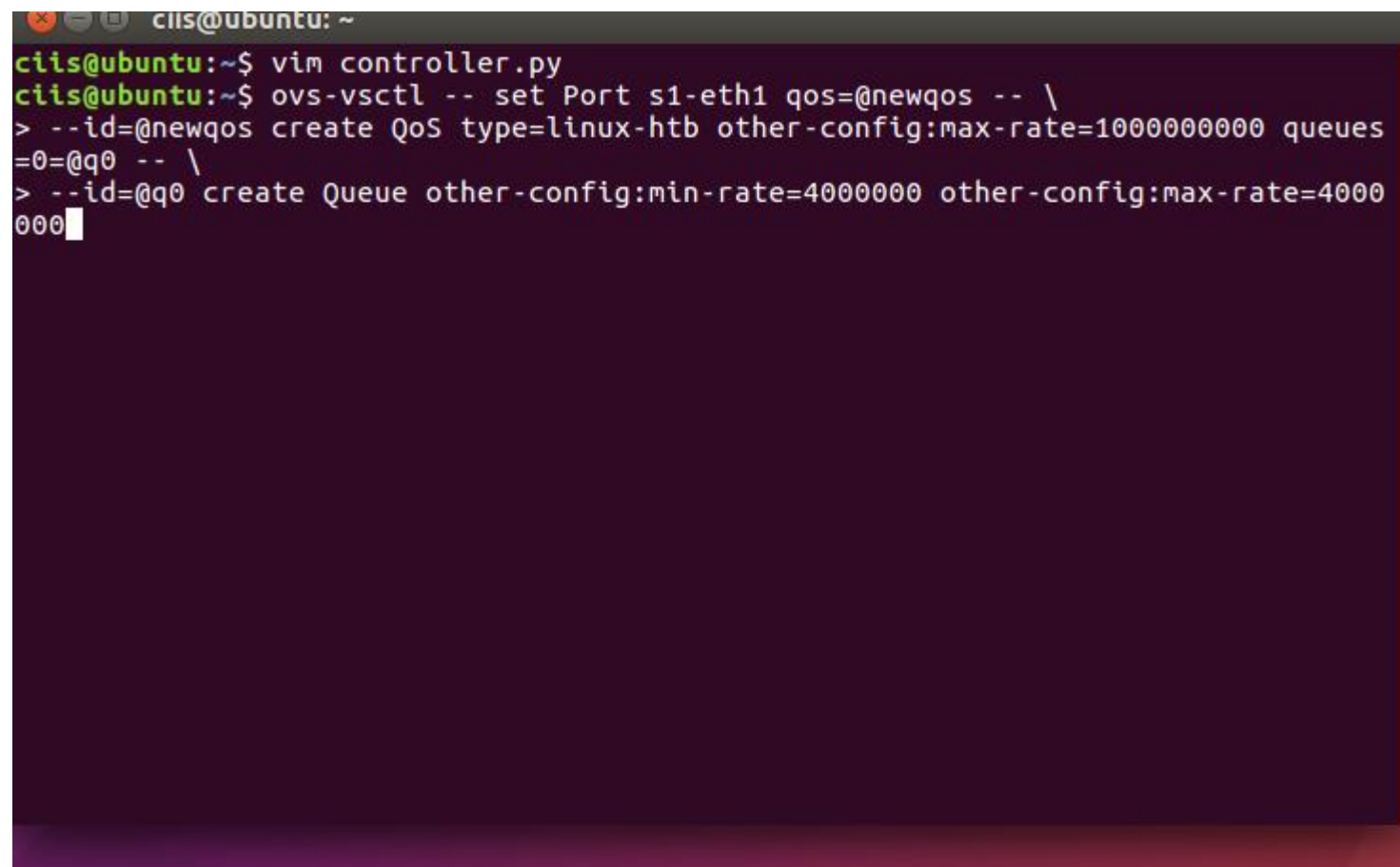
Server listening on TCP port 6000
TCP window size: 85.3 KByte (default)

[1]: command not found
root@ubuntu:/home/ciis# [ 18] local 10.0.0.4 port 4000 connected with 10.0.0.1 port 49838
[ 19] local 10.0.0.4 port 4000 connected with 10.0.0.1 port 49838
[ 18] 0.0-10.0 sec  19.1 GBytes 16.4 Gbits/sec
[ 19] 0.0-10.0 sec  16.8 GBytes 14.5 Gbits/sec
[ 20] 0.0-10.0 sec  18.7 GBytes 16.1 Gbits/sec
root@ubuntu:/home/ciis#

"Node: h1"
Client connecting to 10.0.0.4, TCP port 4000
TCP window size: 85.3 KByte (default)
[ 17] local 10.0.0.1 port 49828 connected with 10.0.0.4 port 4000
[ ID] Interval      Transfer    Bandwidth
[ 17] 0.0-10.0 sec  58.0 GBytes 49.8 Gbits/sec
root@ubuntu:/home/ciis# iperf -c 10.0.0.4 -p 4000
Client connecting to 10.0.0.4, TCP port 4000
TCP window size: 1.42 MByte (default)
[ 17] local 10.0.0.1 port 49834 connected with 10.0.0.4 port 4000
[ ID] Interval      Transfer    Bandwidth
[ 17] 0.0-10.0 sec  45.2 GBytes 38.8 Gbits/sec
root@ubuntu:/home/ciis# iperf -c 10.0.0.4 -p 4000
Client connecting to 10.0.0.4, TCP port 4000
TCP window size: 85.3 KByte (default)
[ 17] local 10.0.0.1 port 49838 connected with 10.0.0.4 port 4000
[ ID] Interval      Transfer    Bandwidth
[ 17] 0.0-10.0 sec  16.8 GBytes 14.5 Gbits/sec
root@ubuntu:/home/ciis#
```

實驗步驟(7)

- 接下來加入controller.py來調整queue的優先度，然後再次進行測試，g. 首先建立三個queue針對s1-eth4，接下來調整queue設定將h1、h2、h3的流量分別導向queue0、 queue1、 queue2，如下圖分別建立三條queue。



```
ciis@ubuntu: ~  
ciis@ubuntu:~$ vim controller.py  
ciis@ubuntu:~$ ovs-vsctl -- set Port s1-eth1 qos=@newqos -- \  
> --id=@newqos create QoS type=linux-htb other-config:max-rate=1000000000 queues  
=0=@q0 -- \  
> --id=@q0 create Queue other-config:min-rate=4000000 other-config:max-rate=4000  
000
```

實驗步驟(8)

- 接下來開啟另一個終端機開啟剛剛的Mininet模擬拓譜，另一個開啟controller.py，再進行一次剛剛的實驗，並且將結果截圖以及試著分析解析為什麼會有不同原因為何，controller.py如下圖。

```
cliis@ubuntu: ~  
from pox.core import core  
import pox.openflow.libopenflow_01 as of  
from pox.lib.util import dpidToStr  
log = core.getLogger()  
s1_dpid=0  
s2_dpid=0  
def _handle_ConnectionUp (event):  
    global s1_dpid, s2_dpid  
    print "ConnectionUp: ",  
        dpidToStr(event.connection.dpid)  
    #remember the connection dpid for switch  
    for m in event.connection.features.ports:  
        if m.name == "s1-eth1":  
            s1_dpid = event.connection.dpid  
            print "s1_dpid=", s1_dpid  
        elif m.name == "s2-eth1":  
            s2_dpid = event.connection.dpid  
            print "s2_dpid=", s2_dpid
```

05 實驗要求



實驗要求

- 任務1：參考實驗步驟拓譜圖，撰寫拓譜.py將檔案內容以截圖方式呈現。
- 任務2：參考實驗步驟，透過Mininet xterm功能，分別對h1、h2、h3到h4同時做測試以及分開做測試，並且將實驗過程以及結果截圖方式呈現，並且在檔案中需附上對於這2個不同結果的解釋。
- 任務3：參考實驗步驟，撰寫controller.py，並且透過指令添加3個queue，分別對h1、h2、h3到h4做測試，並且將實驗過程以及結果截圖方式呈現，並且在檔案中需附上對於結果的解釋。

- [1]<https://guiderworld.blogspot.com/2009/01/queue-method.html>
- [2]<https://www.jannet.hk/zh-Hant/post/quality-of-service-qos-switch/>
- [3]https://www.southampton.ac.uk/~drn1e09/ofertie/openflow_qos_mininet.pdf