313551137 官劉翔

**#Pingall結果**

一張含有 文字, 字型, 螢幕擷取畫面 的圖片

自動產生的描述

A、B、C可以自由連接

因為ping是使用ICMP，而作業要求Node D has access to A and B by tcp port 22 跟80，所以使用ping， D無法連接到任何其他host

C、D不可互聯

**實作細節:**

**實作可以分成這三個步驟**

1. **環境確認** 
   1. **確認Ryu SDN控制器已正確安裝且可運行**
   2. **確認Mininet已正確安裝且可運行**
2. **建立網路拓樸：** 
   1. **需要建立4個節點(A、B、C、D)**
   2. **需要建立3個交換機(S1、S2、S3)**
   3. **需要將這些節點與交換機按照題目圖示連接**
3. **實現網路策略與修改ryu：** 
   1. **策略1：節點A、B、C之間可以自由通訊**
   2. **策略2：節點D只能訪問A和B的22端口(SSH)和80端口(HTTP)**
   3. **策略3：節點D和節點C之間禁止所有通訊**
4. **環境確認**

先按照助教提供的方法安裝ryu與mininet

#啟動ryu by ryu.app.simple\_switch\_13 (之後會換成lab1\_Controller.py)

ryu-manager ryu.app.simple\_switch\_13

# 啟動 mininet

sudo mn --custom=lab1\_topo.py --topo=lab1\_topo --controller=remote,ip=127.0.0.1,port=6653 --switch=ovsk,protocols=OpenFlow13

1. **建立網路拓樸**

**lab1\_topology.py**

一張含有 文字, 螢幕擷取畫面, 陳列, 軟體 的圖片

自動產生的描述

我使用 Python 建立了一個包含 4 個主機（A、B、C、D）和 3 個交換機（S1、S2、S3）的網路拓撲。

*# 添加主機*

h1 = self.addHost('A', ip='10.0.0.1/24', mac='00:00:00:00:00:01')

h2 = self.addHost('B', ip='10.0.0.2/24', mac='00:00:00:00:00:02')

h3 = self.addHost('C', ip='10.0.0.3/24', mac='00:00:00:00:00:03')

h4 = self.addHost('D', ip='10.0.0.4/24', mac='00:00:00:00:00:04')

* 使用 addHost() 方法新增主機
* 為每個主機指定固定的 IP 和 MAC 地址
* 主機命名改為有意義的 A、B、C、D，而不是默認的 h1、h2、h3、h4

*# 添加交換機*

s1 = self.addSwitch('s1')

*# 添加link*

self.addLink(h1, s3, port1=1, port2=1) *# A 連接到 S3*

* 使用 addSwitch() 方法新增三個OpenFlow交換機
* 使用 addLink() 方法建立網路連接
* port1 和 port2 參數指定連接的端口號
* 連接配置形成了一個三角形的交換機拓撲，可能存在cycle問題

1. **實現網路策略與修改ryu**

**為什麼lab1需要 STP？**

在這個拓撲中，交換機 S1、S2、S3 形成了一個三角形link：

* S1 ←→ S2
* S2 ←→ S3
* S1 ←→ S3

當我們發送flood訊息時，會產生**廣播風暴**

因此需要 STP（生成樹協議）來：

1. 阻塞冗餘鏈路
2. 防止網絡環路
3. 在主要鏈路故障時提供備份路徑

**lab1\_Controller.py 程式碼解釋**

**Import:**

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

我上網查每個模組：

* app\_manager：這是 Ryu 應用程序的基礎，我們的控制器要繼承它
* ofp\_event 和 set\_ev\_cls：用來處理 OpenFlow 的事件，比如當封包進入交換機時
* ofproto\_v1\_3：我們使用 OpenFlow 1.3 版本的協議
* stplib：因為我們的拓撲有環路，需要用 STP 來處理
* packet 相關模組：用來解析不同類型的網路封包

**Init:**

一張含有 文字, 螢幕擷取畫面, 軟體, 陳列 的圖片

自動產生的描述

這裡很重要的是：

1. 我們的控制器繼承了 RyuApp
2. OFP\_VERSIONS 指定使用 OpenFlow 1.3
3. \_CONTEXTS 啟用了 STP 功能
4. mac\_to\_port 字典用來記住每個交換機的 MAC 地址表

下一段設定了三個交換機的 STP 優先級。當時想了很久不懂為什麼要設這個，後來明白了：

* 優先級決定哪個交換機會成為 root bridge
* 數字越小優先級越高
* S1 的優先級最高（0x8000），所以它會成為 root bridge
* 這樣可以控制 STP 如何阻斷環路

實際測試時，我發現 STP 確實會阻斷一條鏈路，防止cycle發生。但剛開始時我很困惑為什麼網路不通，後來發現是因為 STP 在收斂，需要等待一下下。

**Flow rule:**

首先，我們需要一個方法來添加flow rule。這個方法是所有策略實現的基礎：

一張含有 文字, 螢幕擷取畫面, 字型, 陳列 的圖片

自動產生的描述

一開始我對這個函數很困惑，後來理解了：

1. datapath 就是交換機
2. priority 決定規則的優先順序
3. match 定義了什麼樣的封包會匹配這條規則
4. actions 定義匹配後要做什麼

重要的一點是 priority（優先級）：較高的數字代表較高的優先級。這對實現我們的策略非常重要，因為我們需要確保某些規則能覆蓋其他rule。

接下來在switch\_features\_handler底下加入新函數，來符合lab1的要求

設置這些規則時，優先級的設置很關鍵：

* 300：D和C的阻止規則（最高優先級）
* 200：允許訪問特定端口的規則
* 100：默認規則
* 1：一般的MAC學習規則
* 0：table-miss 規則

一張含有 文字, 螢幕擷取畫面, 軟體 的圖片

自動產生的描述

策略一:

A、B、C 可以自由通信

* 這個最簡單，因為默認的學習交換機行為就能實現
* 不需要特別的flow rule

#策略二:

只要C、D 兩個host使用tcp port 22 與80就可以正常通過，並設置優先級200

另外設一個100優先級的封鎖條件，就是當D不遵守限制，action就為空

策略三:

這樣的優先級設置確保：

1. D和C的通信一定會被阻止
2. 允許的端口訪問不會被阻止規則影響
3. 其他正常的通信可以進行

**封包處理器（Packet-In Handler）**

當交換機收到一個不知道如何處理的封包時，就會送到控制器，這時就會觸發這個處理器：

**一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述**

因為我們使用了 STP，所以要改用 stplib.EventPacketIn。這個差別很重要，因為：

1. ofp\_event.EventOFPPacketIn 是普通的封包進入事件
2. stplib.EventPacketIn 是經過 STP 處理後的封包事件

一張含有 文字, 螢幕擷取畫面, 軟體, 作業系統 的圖片

自動產生的描述

第一部分:

* 取得封包的源和目的 MAC 地址
* 創建交換機的 MAC 地址表
* 記錄源 MAC 地址來自哪個端口

第二部分:

* 查找目的 MAC 地址對應的端口
* 如果找到，從該端口轉發
* 如果找不到，泛洪到所有端口
* 最後安裝流表規則避免下次重新詢問控制器

一張含有 文字, 螢幕擷取畫面, 軟體, 字型 的圖片

自動產生的描述

**安裝流表項以避免下次 packet\_in**

**STP 處理:**

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

從log中我觀察到端口狀態的變化過程：

1. DISABLE → LISTEN：端口開始工作
2. LISTEN → LEARN：開始學習MAC地址
3. LEARN → FORWARD：可以轉發數據
4. 某些端口會變成 BLOCK 狀態，阻止環路

當Topology改變時（比如鏈路斷開）：

1. 刪除所有流表
2. 清空MAC地址表
3. 重新添加flow rule