國立暨南國際大學資訊工程學系

碩士論文

# 基於 OpenFlow 之網頁認證網路存取控制機制

# An OpenFlow-based Captive Portal Access Control Mechanism

指導教授：吳坤熹博士

研 究 生：楊國呈

中華民國 106 年 1 月

國立暨南國際大學碩士論文考試審定書

# 誌謝

不免俗的要感謝指導老師 吳坤熹 老師，他讓我知道實踐的重要，至今仍持續的在運動與 coding。老師所教導的道理，大多都在網路文章與書中可以找到接近的觀念。雖然這些道理與知識我都經常在閱讀，也是我在研究之餘的消遣，卻無法頻繁地去實現它們。不論是專業技術或是興趣，總是三分鐘熱度，對於事物的理解也經常流於表面。在與人對談時，常常無法深入淺出的展現出自己對於事物的看法。老師的演示總是精采與深入並進，由此可知道對事物理解是不夠的，還必須要融會貫通。對事物理解也許只需要一小段時間，但是要融會貫通卻需要更勝於十倍的時間。於是我在老師身上學習到持續與堅持，持續的學習、堅持做正確的事。再來要謝謝所有實驗室的夥伴，其中特別想提到詠瑜。看到大學生對於實作基本知識的熱情與扎實的學習，實在是不能讓我繼續含混下去。

論文名稱:基於 OpenFlow 之網頁認證網路存取控制機制

校院系:國立暨南國際大學科技學院資訊工程學系　　頁數:76

畢業時間：中華民國 106 年 1 月　　學位別:碩士

研究生:楊國呈　　指導教授: 吳坤熹博士

# 摘要

傳統的網頁認證(captive portal)機制，常常會因為存取控制清單(access control list，ACL)設定的位置接近網路骨幹的關係，在網路中未能有效阻斷連線。而設定存取控制清單的位置接近末端網路設備時，又因為要允許受限制的主機在各個網段間移動，必須設定在各式不同廠牌不同介面的設備上，過程十分繁複。

本文使用軟體定義網路(software-defined networking，SDN)的架構，將存取控制清單的設定由末端支援 OpenFlow 的網路設備執行以減少瓶頸形成，並統一由 Ryu 控制器設定於各個末端支援 OpenFlow 的網路設備以減少設定次數，以協助網路管理者透過應用程式設定存取控制清單，並協助使用者透過網頁認證能順利使用網路。

關鍵字：Captive portal、DNS 重導流、OpenFlow、軟體定義網路、存取控制清單

Title of thesis: An OpenFlow-based Captive Portal Access Control Mechanism

Name of Institute: Department of Computer Science and Information Engineering, College of Science and Technology, National Chi Nan University

Pages:76

Graduation Time: 01/2017 Degree: Master

Student Name: Kuo-Cheng Yang Advisor Name: Dr. Quincy Wu

# Abstract

The traditional captive portal mechanism can't block connections efficiently because ACL (access control list) is applied to core switches. On the contrary, if the restriction is applied on access switches, it has to be configured many times.

In this thesis, we propose an SDN (software-defined networking) architecture to block unauthorized network traffic and reduce the times of configuration. A manager can configure ACL on OpenFlow devices through a Ryu application; users can obtain networking services through a captive portal authentication mechanism.

Key words: ACL, captive portal, DNS redirect, OpenFlow, SDN

# 目次

# 表目次

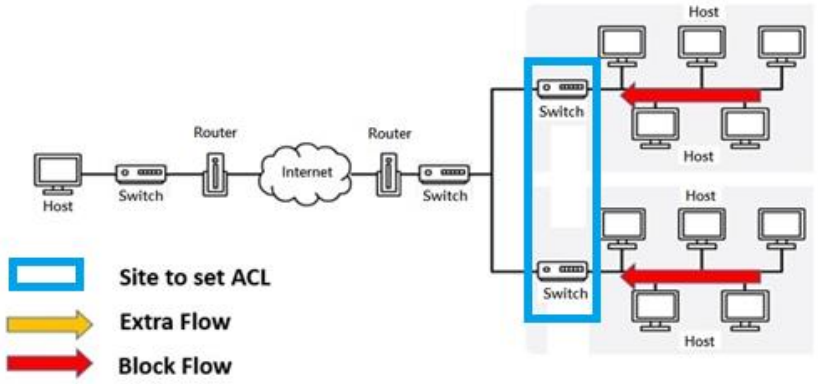
# 圖目次

# 第一章　緒論

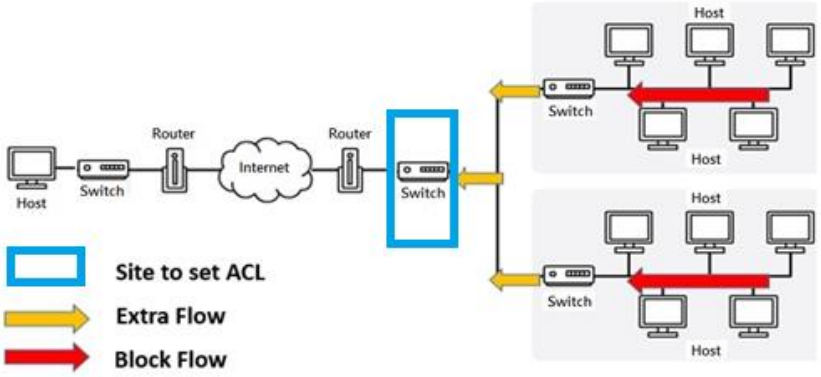
## 第一節 研究動機

網際網路（Internet）自 1980 發展至今已數十餘載，資訊產業一直像是依循摩爾定律（Moore's Law[1]）成長。個人電腦逐漸普及，從一個家庭共用一台，轉變到人手一台，至今甚至一個人擁有數個裝置：在家用桌上型電腦，上班帶筆記型電腦，隨身攜帶智慧型手機，消遣時使用平板，運動時配戴智慧手環。所有的設備無一不需要網路。在網路迅速的普及化下，需要使用網路來完成的事情越來越普遍，讓使用者存取網路的環境也不受限在固定的場所。因此就算不是網路服務提供商（Internet Service Provider，ISP）的產業，也面臨必須提供網路服務的情況：像是學生要去咖啡廳可能會優先選擇有提供網路的咖啡廳，或是商務人士出差時選擇提供上網服務的飯店。對於網路使用者流動性較高的區域，例如校園、商店、飯店、公司等，往往需要對不同類型的使用者提供上網服務。固定的網路使用者像是企業內部的工作人員，經常在該區域中使用；也有一些流動性高的使用者如旅客或顧客，僅會短暫停留及使用網路。網路管理員需要一套存取控制（Access Control[2]）機制作為網路安全之第一道防線，如果在使用者連接上網路後就立即管制，流量經過第一線的網路設備就可以過濾，就可以減少流量匯集處網路設備的負擔了。傳統的存取控制清單（Access Control List，ACL）難以有效管理流動的使用者，只能針對通訊協定來管理。此外，存取控制清單的數量也會影響到設備運作的效能。在博科（Brocade）的網路系統故障排除指南[3]中就指出，當存取控制清單到達上限而沒有即時的調整，可能會影響設備處理存取控制清單的效率或產生故障，所以存取控制清單的維護就很重要。然而存取控制清單設定的位置也相當的關鍵。通常要設定的對象都是針對末端的設備，第一個反應是會設定在

最靠近末端的網路設備如圖一,但是如果使用者離開了運行設定的網路設備底下，移動到另一個網路設備底下，理論上應該可以繼續使用原網路服務。而為了達到這個目的，管理者就必須在另一個網路設備上進行對應的設定。當末端網路設備數量為 n 時，管理者就必須要設定 n 台設備以免有漏網之魚。如果存取控制清單所管理的設備很多，像是流動的人所使用的設備，在人離開後對應的存取控制清單就必須移除，所以存取控制清單會常常更新，就必須在各個設備之間來回新增與刪除。如果管理者在靠近網路骨幹位於藍色箭頭處的網路設備進行設定如圖二，這樣不僅可以減少設定的次數，還可以阻擋來自任何一個末端網路設備的流量，但是這樣就發現該設備在流量到達骨幹被阻斷之前，可以在網路間肆意妄為，如圖二中黃色部分的流量。




圖一　ACL applied on access switches




圖二　ACL applied on core switches

所以本研究提出透過集中控制設定存取控制清單,讓管理者只需要設定一次，

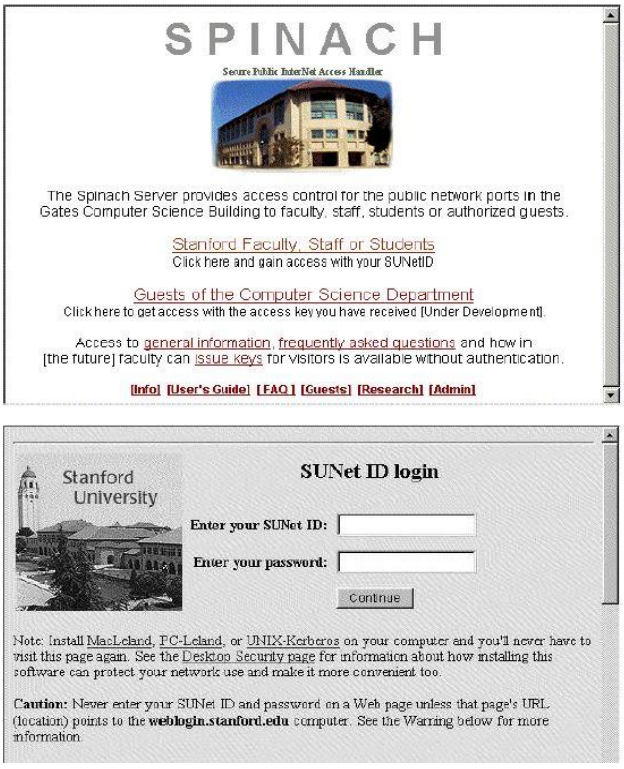
即可以在所有的末端網路設備上生效。存取控制清單在運作一段時間後，將會自動消失，不需要由管理者逐一登入設備以刪除。由於設定登錄存取控制清單的規則可以自動化，管理者將可節省更多的時間。目前公眾網路大多運用網頁認證（Captive Portal）的方式來進行認證（如圖三為星巴克的網頁認證介面），使用者只需要透過網頁瀏覽器登錄網站，接著輸入帳號密碼即可取得網路的服務。


圖三　Starbucks captive portal user interface[4]

透過網頁認證來管理存取控制清單是一個相當方便的方法，網頁認證的雛形始於 1999 年史丹佛大學提出的 SPINACH[5]( Secure Public Internet Access Handler）系統，用來管理資訊科學系系館公共空間的網路存取，包括休息室、會議室與大廳。在[5]中已提出認證、授權和計量( Authentication, Authorization, and Accounting，AAA）的概念且透過網頁介面供使用者進行認證（如圖四所示）。至今已有許多新設備與架構推陳出新因應各式不同的網路需求，像是[6]中的架構就是為配合網際網路協定第六版（Internet Protocol version 6，IPv6）來完成對漫遊使用者的存取控

制。在網路中要達到網頁認證，除了網路設備本身限制用戶上網的功能，還需要有用戶認證的機制來配合。

總結來說一個好的網路存取控制管理系統必須預先規劃好使用者能使用網路的權限，像是賓客登入學校的網路後，未認證前仍可以在學校網頁查找學校地圖，認證後則可經由學校提供的公共網路位址去拜訪校外網站。所以要提供什麼服務是一個必須優先考慮到的問題。將使用者分成長期有網路需求的使用者與短期需求的使用者。長期需求使用者的設定會在系統中存在較長的時間，不需要經常的進行更新或刪除，而短期需求使用者因為流動性高的緣故，設定會常常更新或刪除。所以在系統啟動時，給予長期需求網路使用者登錄網路的權限，接著再透過網頁認證帳號與密碼來管理短期需求的網路使用者,以達到網路管理之安全需求。而在尋找適當工具來處理這個問題的時候，發現軟體定義網路（software-defined networking，SDN[7]）的概念相當適合處理這種需求。所以本研究將會透過軟體定義網路結合網頁認證技術，達成網路管理存取控制之目的。




圖四　User's view of SPINACH[5]

## 第二節 研究目的

本研究目的為提出一個管理機制適用於網路使用者可能會在各個不同提供網路服務的設備之間移動的情境，透過集中控制網路設備結合網頁認證技術，達成網路管理存取控制之目的。本系統預計能達到之功能如下：

1. 統一設定登錄存取控制清單於所有末端網路設備。
2. 自動設定新認證用戶於存取控制清單。
3. 已註冊之用戶，連接上網路即可存取 Internet 網路資源。
4. 未註冊之用戶需經過網頁認證方可使用網路。

希望透過此系統所提供的功能以減少網路管理員重複於設備間反覆設定的負擔，得以將時間分配於其他工作，像是資安問題產生時能加速釐清問題並解決提升整體網路連線品質。由於 SDN 可程式化的特性，在系統提供服務時，管理者可以持續開發新的服務如負載平衡或災難復原，得以迅速的適應網路中發生的各種情境。以本系統的架構為例，原設計僅支援 IPv4，後來為了同時支援 IPv4/IPv6，只花了一星期，修改 90 行的程式碼即可順利完成。SDN 的彈性及威力由此可見。

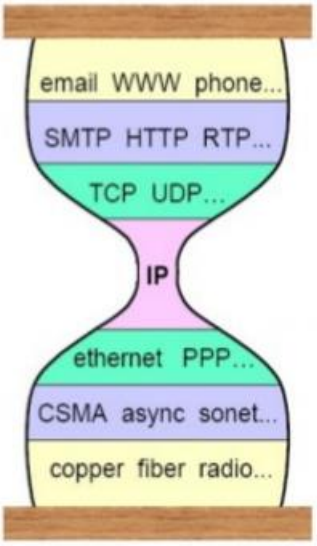
## 第三節 論文架構

第一章為研究動機與目的；第二章介紹相關背景知識與文獻探討；第三章為系統架構、網路架構與系統操作流程；第四章為各元件安裝說明和設定方式；第五章為實驗結果；最後為本論文之總結與未來展望。

# 第二章　背景知識

## 第一節　整合通訊協定的挑戰

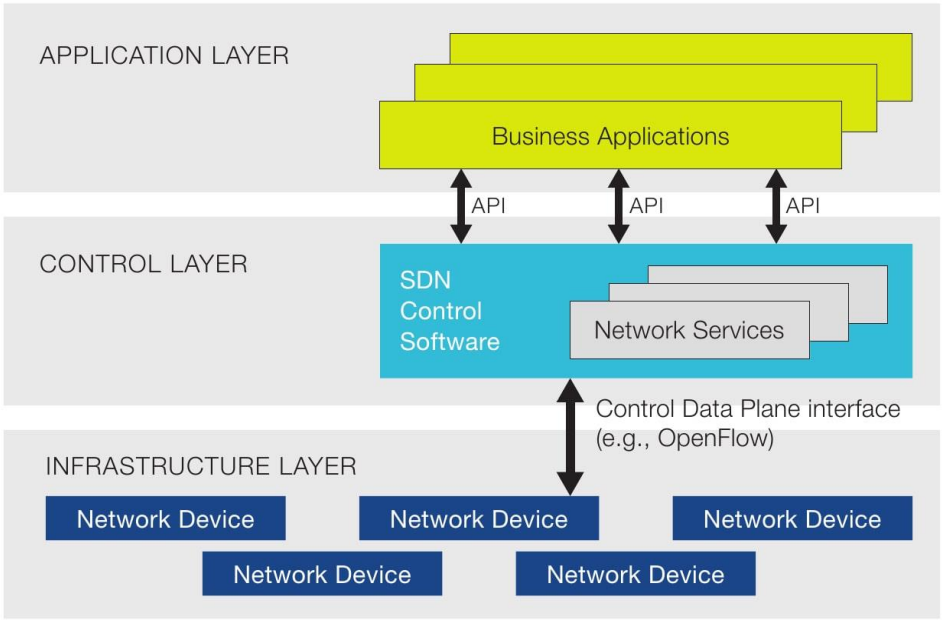
在網路迅速的擴張之下，以 TCP/IP（Transmission Control Protocol/Internet Protocol）為核心發展的網路系統架構呈現了一個沙漏狀的結構如圖五，使底層技術的差異性對於高層不可見，實現了「IP over everything」；而對於底層而言高層的協定封裝於 IP 中，達成「Everything over IP」，這種結構對於開發人員而言相當方便。但是在網路普遍的使用後也暴露出了一些問題，是當初設計時始料未及的。像是網路管理複雜、擴充性不足、安全性等問題，為了解決以上問題專家學者也不斷提出各種不同的新方案來修補這些問題，像是網際網路協定第六版（Internet Protocol version 6，IPv6）、虛擬區域網路（Virtual Local Area Network，VLAN）、多協定標籤交換（Multiprotocol Label Switching，MPLS）還有網際網路安全協定（Internet Protocol Security，IPSec），但是這種四處救火的方式很難將所有的網路問題從根本上解決。




圖五 Protocol Hourglass of TCP/IP [8]

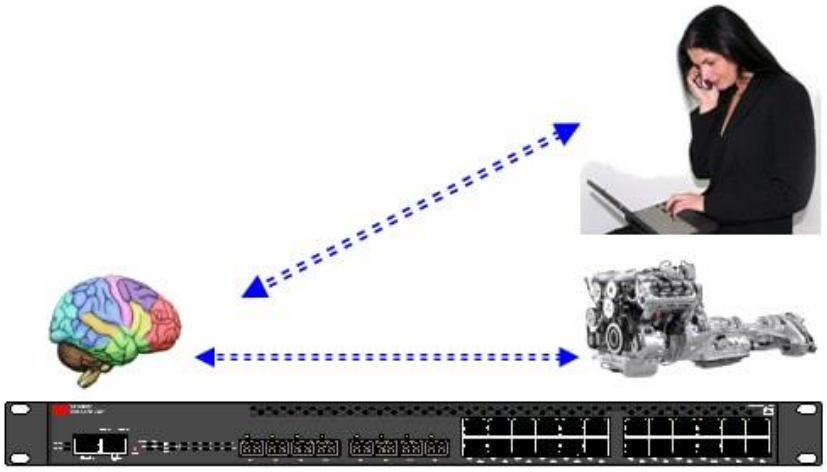
## 第二節　SDN

根據 ONF 所發佈的白皮書[9]，軟體定義網路的技術架構圖如圖六，控制層為應用提供可視化的管理介面，管理者就可以透過軟體從邏輯上來操作網路控制與網路服務。將網路設備分為控制層與基礎設施層兩個層，之間透過一個介面來溝通。網路基礎設施層接收控制層的指令完成轉發任務，其硬體規格的異質性對應用層而言均可被忽略。



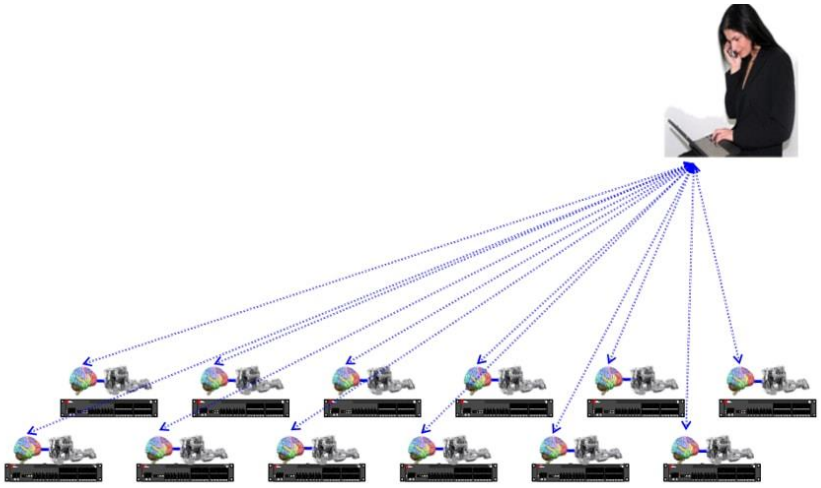

圖六　Software-defined network architecture[9]

傳統的網路設備是由一個控制平面(control plane)與一個資料平面(data plane)所組成如圖七[10]。控制平面就像是設備的大腦。在網路中這些控制平面互相溝通協調後建立網路拓樸。資料平面連接設備上的實體網路埠負責資料的交換。管理者透過設備的控制平面去操控資料平面。



圖七　The relation of manager and traditional network device[10]

當網路中設備很多時，管理者與網路設備的關係如圖八[10]。管理者必須熟知所有設備的語法，如果要滿足新的需求時，管理者就要逐一操作設備來完成新的設定。

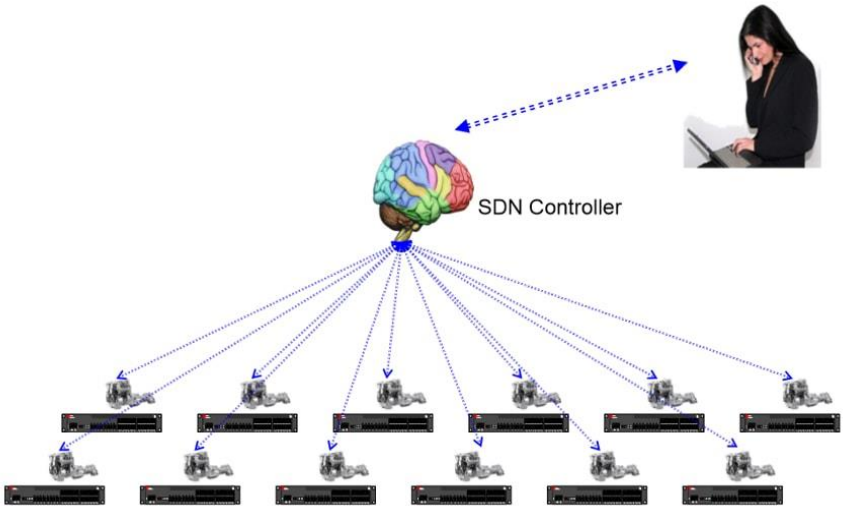


圖八　The relation of manager and traditional network architecture[10]

而軟體定義網路是一個新的架構，它將控制平面與資料平面分離，把網路中所有設備的控制平面集合起來，統一管理所有的資料平面如圖九[10]。利用集中式控制機制，由 SDN 控制器負責協調流程並促進其它資料平面進行互動的應用程式，使得網路功能可以透過程式化來實現，令資料平面根據控制器的訊息進行通訊。最後網路管理員根據應用程式的介面來操作抽象的網路功能，像是各種協定的運

作；符合這個協定的封包可在紀錄其內容後，繼續轉送封包、丟棄封包或改變封包欄位的內容。總結軟體定義網路的特徵如下：

1. 控制平面與資料平面分離。
2. 開放的可程式化界面。
3. 自動化控制網路服務的應用程式。



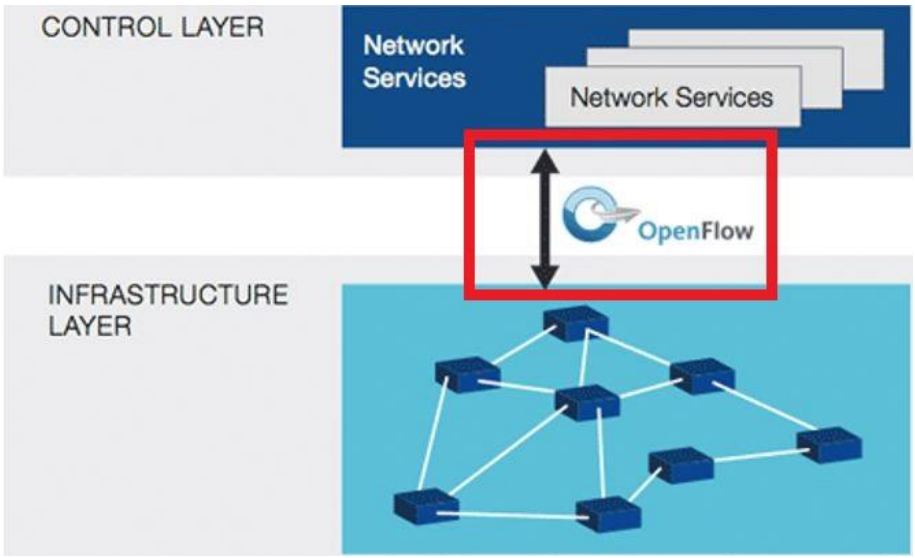

圖九　The relation of manager and SDN architecture[10]

在資料中心不斷發展的同時，網路設備似乎只能滿足速度和吞吐量，至於其他方面的功能性則缺乏彈性，像是在雲端網路中相當重要的網路虛擬化，相對於軟體定義網路將網路的管理層從硬體中抽離，而網路虛擬化將網路功能從網路設備中抽離出，兩者都以軟體來實現，所以整合起來會比傳統網路依賴設備製造商提供韌體更新來的方便。商用網路設備通常具有管理介面，通常網路管理員只要利用介面上的功能或語法去操作設備通常不會發生問題。如果想要嘗試新協定時，則需要更換設備或仰賴廠商推出支援其協定的韌體才行。軟體定義網路使得網路功能可程式化，網路功能將會變得更加彈性，就可以靈活的滿足各個組織的需求。像是一般的第二層網路設備可能無法辨識數據壅塞控制協定（Datagram Congestion Control Protocol，DCCP）的封包。而支援 SDN 的設備可以透過控制器來解析出數

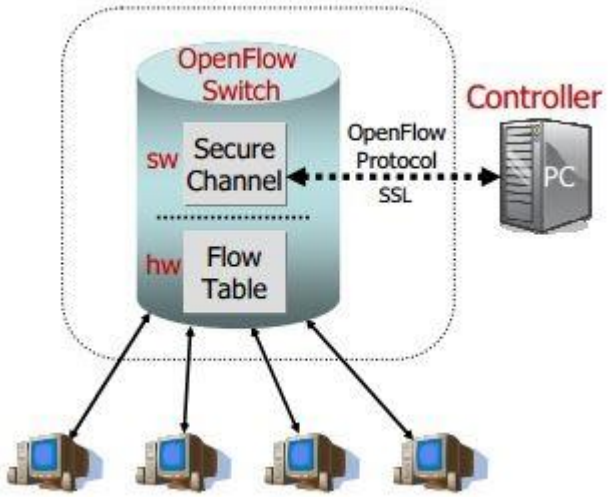
據壅塞控制協定的封包，進而針對此類型封包實行轉送或依照管理者的需求將數據壅塞控制協定的封包送回控制器再另行處理。雖然網路管理者的需求可以透過撰寫程式來滿足，但是若設備製造商持續根據自己的商業需求開發網路協定，網路管理員仍然缺乏一個統一的介面來管理這些分散在不同設備的網路協定。於是乎 OpenFlow 應運而生。

## 第三節　OpenFlow

由史丹佛大學所始創名為 OpenFlow[11]的協定就是專門為了只有資料平面的設備所設計的南向介面（如圖十中紅框位置所示）。這些設備被一個集中式的控制層所控制，實際上會有一台設備作為控制器（Controller），如圖十一。而這個控制器是這個網路中的控制平面，透過 OpenFlow 協定管理 OpenFlow 交換器中的流程表（Flow Table）。控制器負責管理網路中所有的 OpenFlow 交換器。這樣一個集中管理的概念很適合在網路中，同時管理許多網路設備來限制使用者存取控制，即便使用者移動到其他網路設備底下，仍然可以被管理者的配置所管理。而透過可程式化的介面可以因應使用者需求新增服務，同時由控制層對所有基礎設施層的網路設備統一管理網路設備上的存取控制清單，大大減少了管理者的麻煩，所以軟體定義網路搭配 OpenFlow 是管理存取控制的一個絕佳方案。

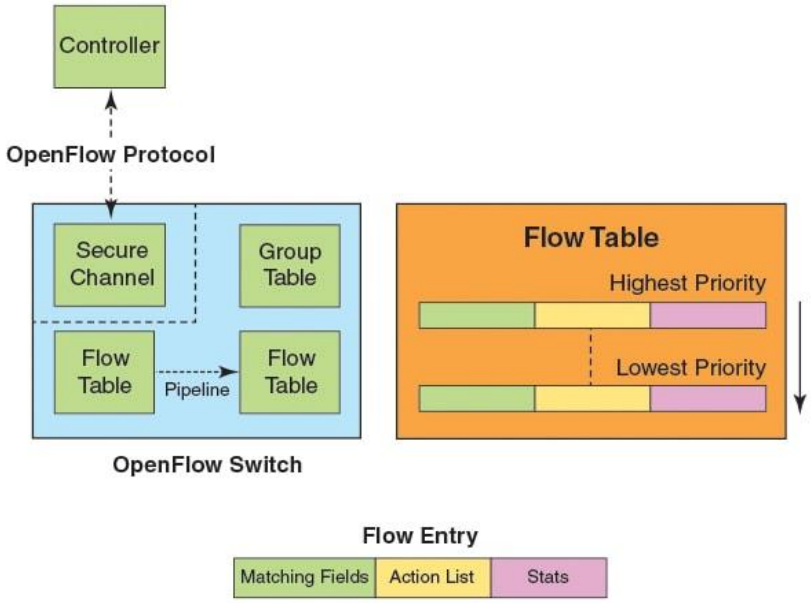


圖十　OpenFlow is one of must popular southbound interfaces standard[12]




圖十一 OpenFlow Controller[11]

OpenFlow 架構參考圖十二[13]，控制器做為軟體定義網路中控制層的角色，而基礎設施層中的OpenFlow交換器中，會有一個流程表；流程表中的條目稱作flow entry。由控制器透過 OpenFlow 對 OpenFlow 交換器新增、修改或刪除 flow entry，封包通過 OpenFlow 交換器時先與 flow entry 進行比對，來決定此封包接下來的 action 是什麼。




圖十二　OpenFlow architecture[14]

在 OpenFlow 規範之下，訊息種類分別如下：

1. Symmetric：當控制器與交換器建立溝通管道，雙方互相交換資訊的訊息。
2. Asynchronous：當交換器需要通知控制器或是需要控制器協助時所發出的訊息，稱為 Packet-In。
3. Controller-to-switch 則是提供控制器對交換器進行設定、新增、修改或刪除流程表等訊息，稱為 Packet-Out。

以實際情況舉例來說，如果有一封包需要經過交換器至其他主機，其處理流程如下：

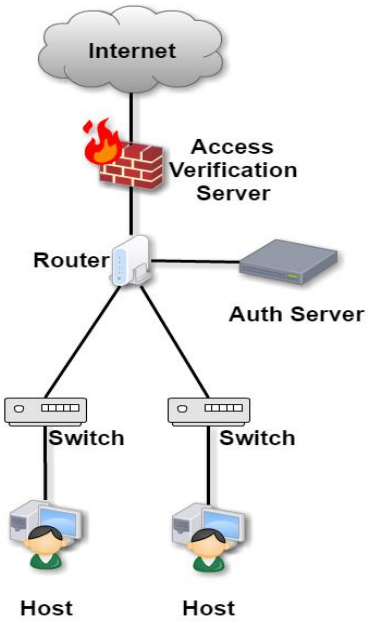
1. 首先控制器與交換器建立連線並交換所需的資訊。
2. 封包進入交換器時，交換器會查詢流程表是否有符合的 flow entry。
3. 當沒有符合的 flow entry 時，交換器發出 Packet-In 訊息給控制器，請控制器處理。此 Packet-In 訊息會包含這個沒有符合任何 flow entry 的封包資訊。
4. 當控制器收到這個 Packet-In 訊息後，以 Packet-Out 訊息告知交換器，應如何處理這個封包（如：將封包送到哪一個實體埠），同時透過 FLOW_MOD 在交換器上建立新的 flow entry，該 flow entry 包含此後該如何處理此類型的封包。
5. 交換器之後接收到此類型的封包即不需要再發送 Packet-In 請求控制器處理。

每部 OpenFlow 交換器會維護一個或多個流程表，而流程表是由一至多個 flow entry 所組成。其中 flow entry 的順序依照優先權決定，數字愈大優先權愈高，故比對到對應的 flow entry 後即不會再比對優先權較低的 flow entry。每筆 flow entry 所包含項目名稱及功能分別為：

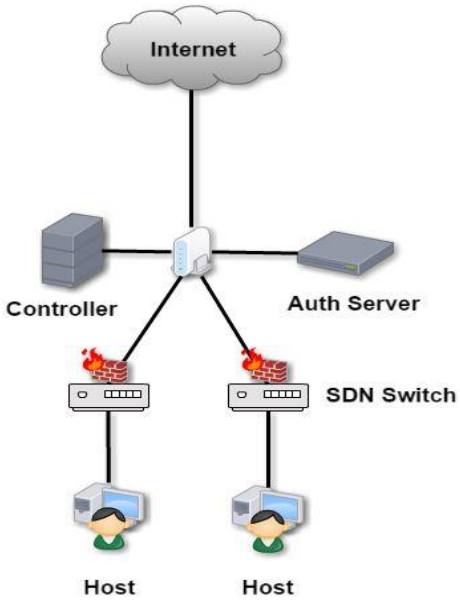
1. Matching fields：欲比對之封包欄位
2. Action list：針對此類型之封包的處理方式
3. Stats：統計封包數量

## 第四節　網頁認證

如果網路管理者開發一個軟體用於認證，而在使用者未認證之前就只能在管理者允許使用網路的範圍內使用，像是只能連接公司內部的伺服器；使用者必須由公司的檔案傳輸協定伺服器先下載軟體才能夠進行認證，然後才能連接至外部網路。這樣對於使用者相當不方便。所以通常會讓使用者透過網頁來進行認證。在使用者接上網路線或者是連上 Wi-Fi 後，開啟瀏覽器輸入網址；在連線時網址會經由 DNS 伺服器的解析將認證網站的 IP 位址回傳至使用者的電腦，再經由瀏覽器對認證網站索取網頁內容。最後呈現於使用者的瀏覽器上進行認證。這樣管理者即不用專門為了認證開發一隻程式，使用者也省去了下載與安裝的功夫。在經過網頁認證後，即能正常的上網。目前實作網頁認證的方式大多為，在網路對外出口設置訪問驗證伺服器（Access Verification Server ）如圖十三，透過認證機制來管理訪問驗證伺服器上的規則，來達到限制上網的效果。但如此一來也會面臨與存取控制清單設定位置同樣的問題，當距離訪問驗證伺服器越遠的設備無法及時阻擋流量，就會造成網路中額外的負擔。所以透過軟體定義網路的架構，每一台 OpenFlow 交換器本身即可以透過流程表達成阻擋流量的功能如圖十四，而認證使用的網頁服務也可以透過流程表導流機制完成。



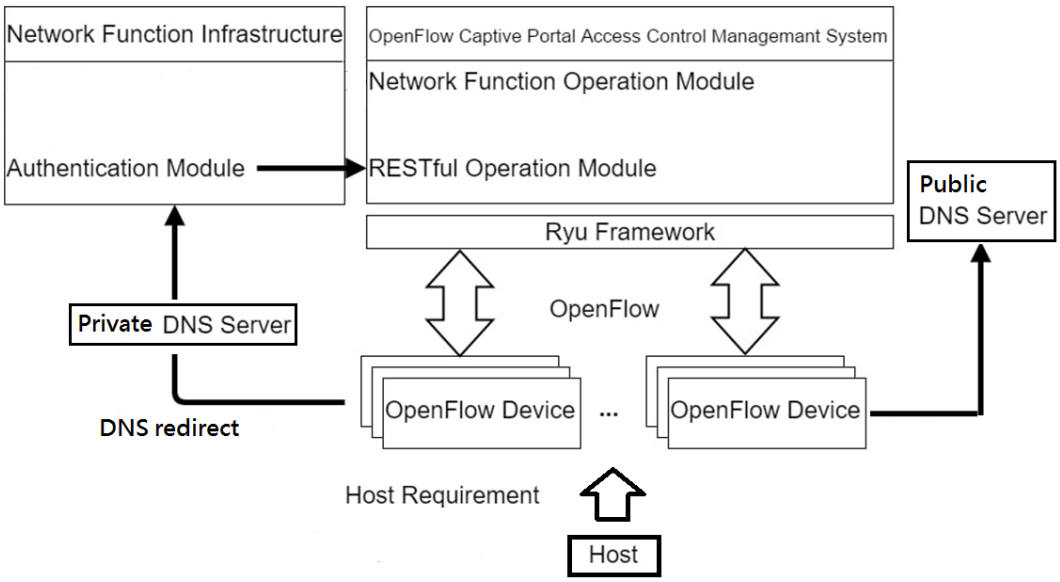

圖十三 Traditional captive portal




圖十四 OpenFlow captive portal

# 第三章　OpenFlow 網頁認證網路存取控制系統

## 第一節　系統架構

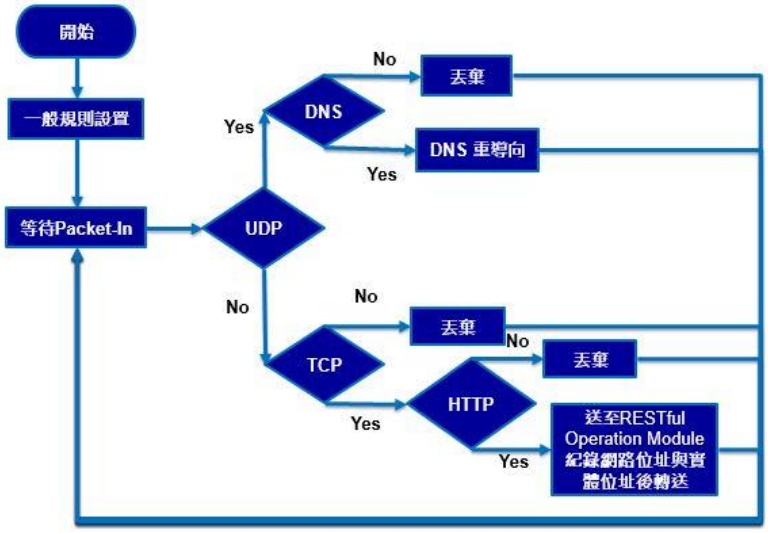



圖十五　System architecture

本文提出一個運作在 SDN 環境中的存取控制系統，提供管理者一個自動化的機制，以方便管理用戶使用網路，圖十五所示為系統架構。此架構中包含 Ryu Framework、Network Function Infrastructure、OpenFlow Device 與 OpenFlow Captive Portal Access Control Management System 以下簡稱 OCPAC。Ryu Framework 是一個以元件為基礎的 OpenFlow 軟體定義網路開發框架，透過 Ryu 所提供的介面，可以很輕鬆的去開發所需的網路管理功能。用 Ryu 的術語來描述，本系統為一個建構於 Ryu Framework 上的一個 Ryu 應用程式。在 Network Function Infrastructure 中，有維持基礎網路服務的元件，如動態主機設定協定（Dynamic Host Configuration Protocol，DHCP）伺服器、網域名稱系統（Domain Name System，DNS）伺服器與認證伺服器。OCPAC 系統接收來自於認證伺服器的資訊，據以合成 OpenFlow

設備中流程表裡每個規則的參數。OCPAC 系統中有兩個較為重要之模組，分別為 Network Function Operation Module 與 RESTful Operation Module。Network Function Operation Module 提供網路協定的規則，例如是否允許位址解析協定（Address Resolution Protocol，ARP）、動態主機設定協定（DHCP）、網域名稱系統（DNS）和超文字傳輸協定（HyperText Transfer Protocol，HTTP）等協定的封包傳送。而其中較為特殊的是，為了提供主機認證的機制，所以 OCPAC 加入網域名稱重導（DNS redirect）的技術在 Network Function Operation Module 中，使認證機制能夠順利進行。以下章節將會詳細介紹網域名稱系統重導向的步驟。

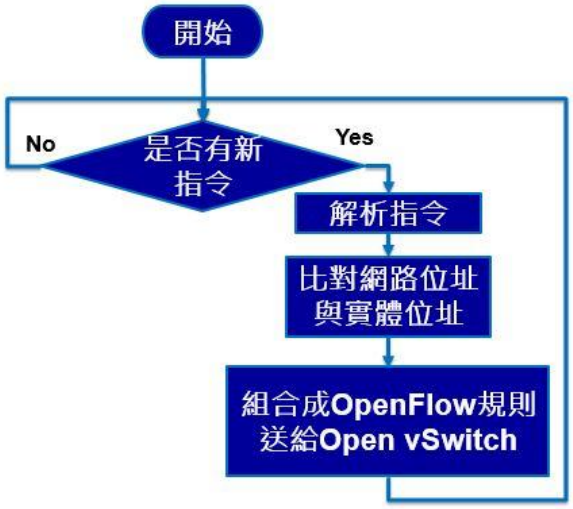
圖十六為 Network Function Operation Module 流程圖。首先設置網路運作時所需要的基本網路協定規則（ARP、DNS），接著等待來自於 OpenFlow 交換器的 Packet-In 訊息。接著判斷此訊息是否是 DNS 的封包還是 HTTP 的封包。如果是 DNS 封包即啟動 DNS 重導向之機制；如果是 HTTP 封包即送至 RESTful Operation Module 記錄實體位址與網路位址後轉送至 WWW 伺服器。




圖十六 Network Function Operation Module flow chart

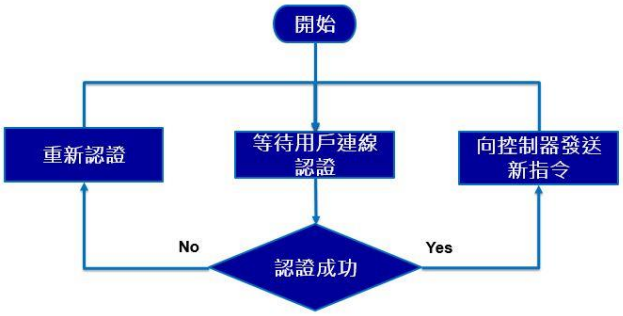
而 RESTful Operation Module 是一個網頁介面，讓管理者可以在網頁上修改 OpenFlow 交換器上的 flow entry。RESTful 操作一般包含 GET/POST/PUT/DELETE，本系統用到其中的 GET/POST/DELETE：GET 取得 OpenFlow 交換器 flow entry 的

列表；用 POST 新增 OpenFlow 交換器上的 flow entry；用 DELETE 刪除 OpenFlow 交換器上的 flow entry；管理者可以隨時透過 RESTful Operation Module 來更新規則，以應付網路中的各種情況。RESTful Operation Module 的流程如圖十七，接收來自 Authentication Module 的新指令後，合成 OpenFlow 的封包向所有的 OpenFlow 交換器發送新規則。



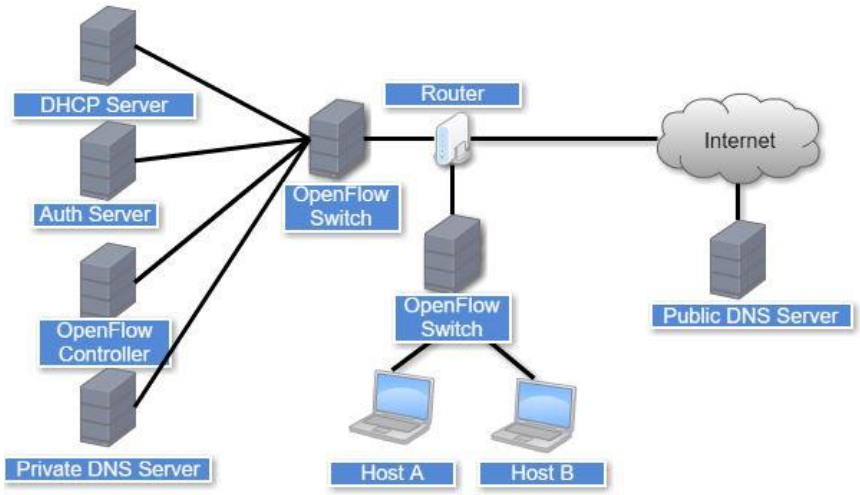

圖十七 RESTful Operation Module flow chart

另外 Authentication Module 是以 Python Flask 建構的認證伺服器，在主機(host)認證結束後，發送指令給 RESTful Operation Module。RESTful Operation Module 會將收到的參數組成新的規則加入 OpenFlow 設備的流程表，使主機可以順利上網。流程如圖十八，用戶認證成功即向控制器發送指令，以將新規則寫入交換器。




圖十八 Authentication Module flow chart

## 第二節 網路架構

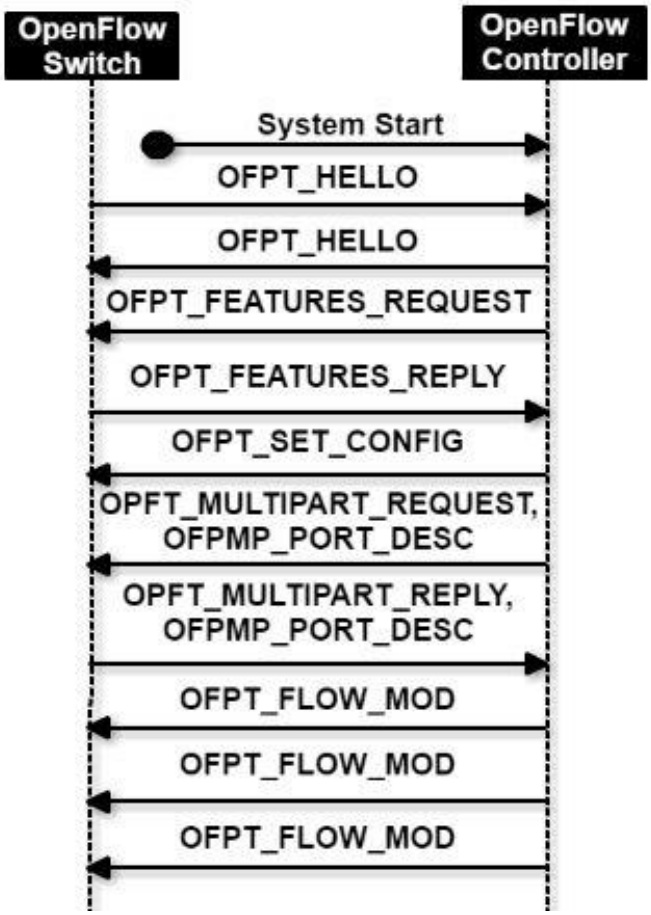



圖十九　Network architecture

圖十九所示為本系統的網路架構圖，與一般公用網路架構相似，有一個 DHCP 伺服器負責配置主機的網路位址、子網路遮罩、閘道器位址及 DNS 伺服器位址。所配置的 DNS 伺服器位址是公共（public）的 DNS 伺服器，像是 Google 的 DNS 伺服器 8.8.8.8、8.8.4.4[14]或暨南大學的 DNS 伺服器 163.22.2.1。而私有（private）的 DNS 伺服器是當主機的實體位址不在預設的授權清單中時，為了強迫這些主機必須要先通過認證才能順利上網，這些主機未經過認證之前只能夠透過私有的 DNS 伺服器所提供的轉址服務進行上網，而轉址的結果永遠都是網頁認證伺服器，直到透過網頁伺服器進行網頁認證成功後才能夠使用公共的 DNS 伺服器的服務。而其他沒有經過轉址服務的連線，例如直接輸入 IP 位址的封包會因為沒有比對到任何的規則而直接被系統阻斷，規則部分可以參考第四章第二節中之表二。而整個網路架構中，最重要的就是一部 SDN 控制器來做整個網路的核心，與能支援 OpenFlow 的交換器。

## 第三節　系統操作流程

在上一節對系統的架構有基礎的描述後，此節將會對於系統操作的流程進行講解。請參考圖二十。




圖二十　System initial signaling flow

整個系統於 OpenFlow 控制器的執行開始，以下將會逐步描述各訊息內容如下：

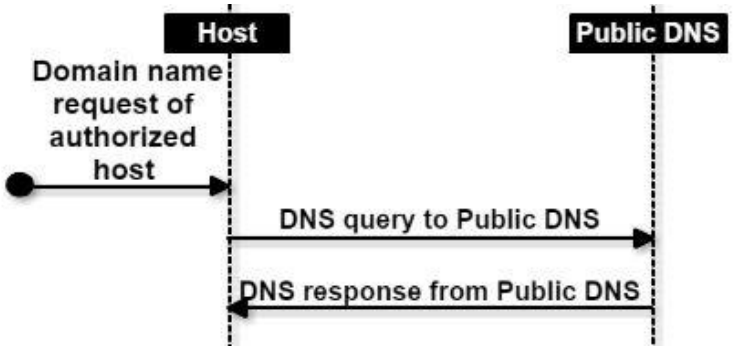
1. OFPT_HELLO：控制器或交換器啟動時，互發 Hello 訊息來通知對方自己已啟動。
2. OFPT_FEATURES_REQUEST/REPLY：詢問/回覆交換器的 OpenFlow 版

本與交換器支援的功能。

3. OFPT_SET_CONFIG：控制器用這個訊息去配置或查詢交換器參數設定。
4. OFPT_MULTIPART_REQUEST/REPLY, OFPMP_PORT_DESC：DESC 全名為 DESCRIPTION，控制器可以從這個訊息中，得知系統中所有對於支援 OpenFlow 的埠的狀況。
5. OFPT_FLOW_MOD：透過這個訊息來修改流程表。提供網路運作時所需要的規則都是在這個時候寫進流程表中的。

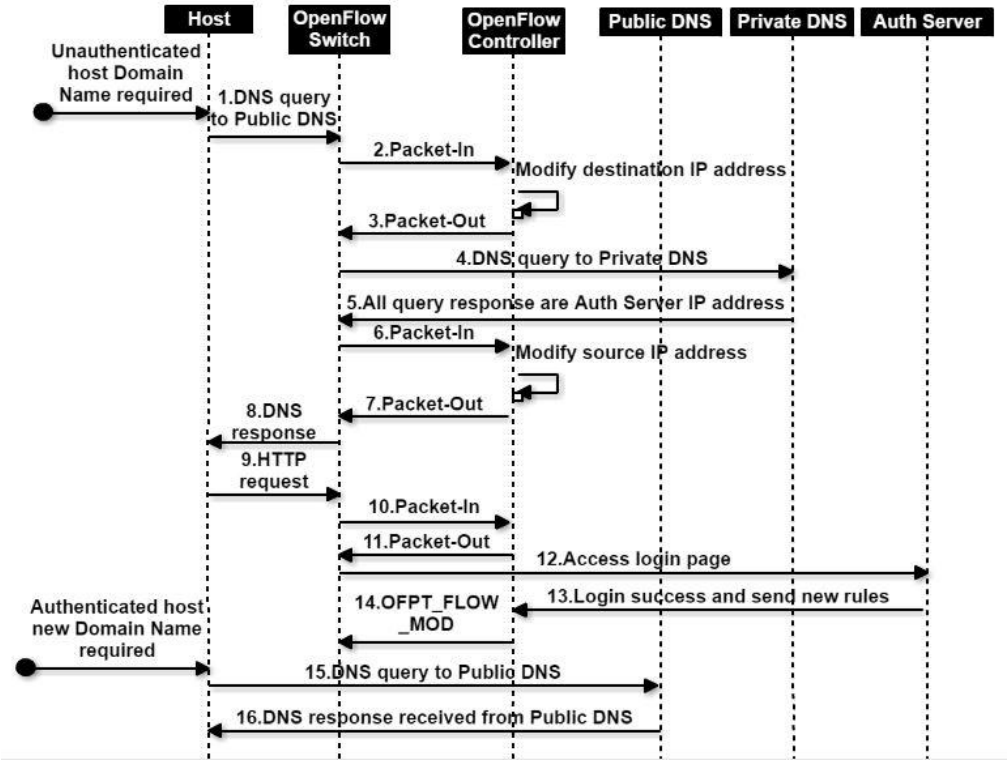
以上步驟結束後，系統處於一個被動的狀態，等待事件的發生。以下發生的事件皆由主機發起，分成以下三種情況：

1. 主機登錄網路時必須要取得 DHCP 的服務，而這個過程中將會觸發的規則，如允許 DHCP discovery 封包通過的規則，都已經於系統啟動時以 OFPT_FLOW_MOD 寫入流程表中了。所以，所有的主機連接上網路時，均可取得 IP 位址。
2. 已授權的主機，其 MAC 位址將被加到 OpenFlow 交換器上。之後該主機欲對外通訊時，即不需要經過 DNS 重導向的過程。所以，已認證的主機可以直接 query public DNS 伺服器如圖二十一。



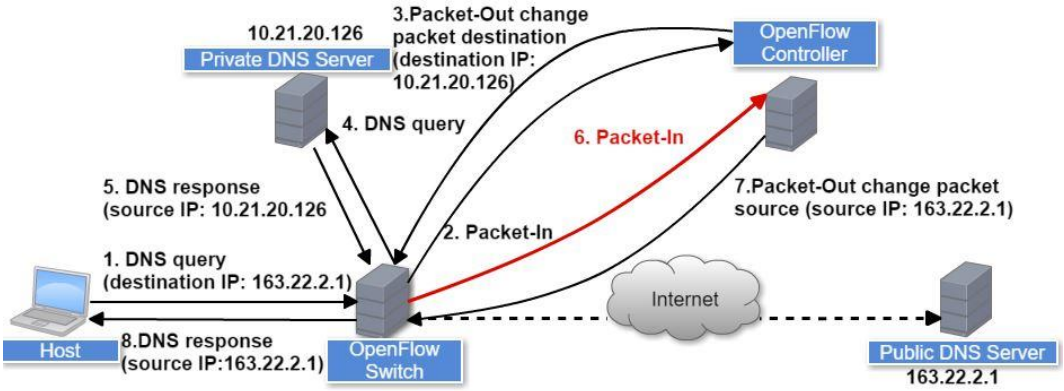

圖二十一　Authorized host signaling flow

3. 未授權主機必須要經過認證以獲得網路存取授權，如圖二十二所示，首先透過 DNS 重導向的方式（運作方式詳述於下一節）， 讓主機連接到認證主機。在授權後給予高優先權的規則，使主機能順利使用網路。




圖二十二　DNS redirect signaling flow

## 第四節　DNS 重導向

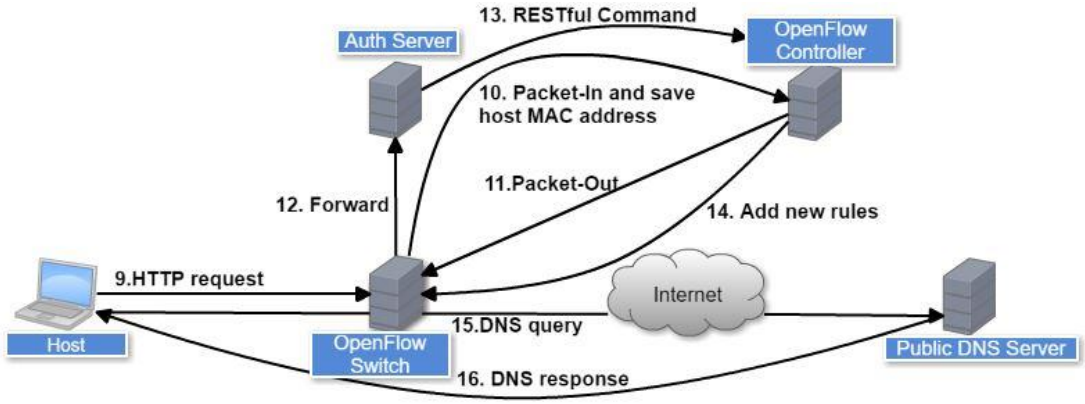



圖二十三　DNS redirect process

以下展示透過 OpenFlow 實作 DNS 重導向與主機認證之過程，過程 1 至 8 步

驟請參考圖二十三（Public DNS IP 位址為 163.22.2.1；Private DNS IP 位址為 10.21.20.126），步驟 9 至 16 過程請參考圖二十四，說明如下：

1. 由主機向 Public DNS 發起 DNS query。
2. OpenFlow 交換器收到 DNS query 的封包後，在流程表中沒有對應的規則，就向控制器發送 Packet-In；控制器收到 Packet-In 後比對是否為 DNS query。
3. 控制器透過 Packet-Out 修改 DNS query 中 destination IP 位址的欄位，將 destination IP 位址由 Public DNS 伺服器 IP 位址改成 Private DNS 伺服器 IP 位址，並將封包送出。
4. OpenFlow 交換器根據此封包的 IP 位址欄位將 DNS query 送至 Private DNS 伺服器。所以，DNS query 封包並沒有送至 Public DNS 伺服器。
5. Private DNS 伺服器發送 DNS response 回覆 DNS query 結果，為了將網頁服務強制導向認證伺服器，所以 query 結果皆為認證伺服器的 IP 位址。
6. 由於此時 DNS response 的 source IP 位址為 Private DNS 伺服器，與當初主機發出的 DNS query 的 destination IP 位址不同。所以，若封包直接送回主機將不會被接受。為了解決這個問題，OpenFlow 交換器收到 DNS response 的封包時，會向控制器發送 Packet-In。（為了與步驟 2 有所區隔，步驟 6 以紅色標示。）
7. 控制器透過 Packet-Out 修改 DNS response 中 source IP 位址的欄位。將 source IP 位址由 Private DNS 伺服器 IP 位址改成 Public DNS 伺服器 IP 位址，並將封包送出。
8. OpenFlow 交換器根據 IP 位址欄位將修改過的 DNS response 封包傳送至主機。



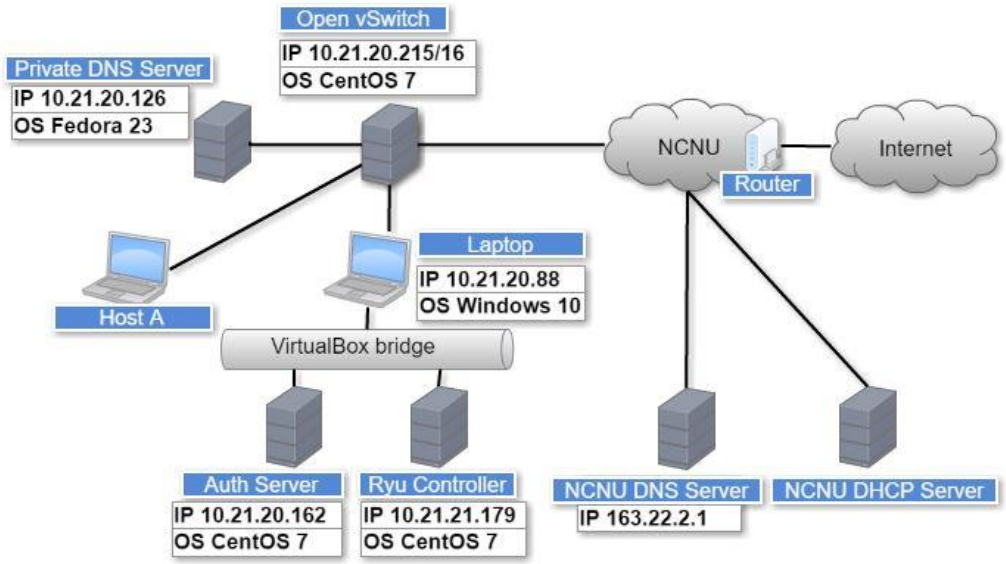

圖二十四　Host authentication process

9. 主機進行 HTTP 連線時，會根據 DNS query 的結果向認證伺服器進行連線。
10. OpenFlow 交換器比對沒有符合 HTTP 連線的規則，向控制器發送 Packet-In，控制器將其連線來源之實體位置與網路位址記錄起來，供新增規則時比對。
11. 控制器記錄完後向 OpenFlow 交換器發送 Packet-Out，並將封包送出。
12. OpenFlow 交換器將封包導向認證伺服器主機，即可透過網頁服務進行認證。
13. 如果認證成功，認證伺服器以 RESTful 指令通知 Ryu 控制器，由 RESTful Operation Module 負責接收。
14. 在 RESTful Operation Module 接受到通知後，合成 OpenFlow 的封包向所有的 OpenFlow 交換器發送新規則，使主機可以不用再進行認證，就可直接上網。新規則在一段時間（由管理者決定時間長短，本系統預設 86400 秒，即一天）未使用後就會自動消失，而使用者在規則消失之後就必須重新認證，此機制將會在下一節詳述。
15. 當下次主機發送新的 DNS query 時，因為 OpenFlow 交換器已經新增了該主機的存取規則，所以，就不會被導向 Private DNS 伺服器。
16. 主機取得正確的 DNS response 後，得以正常上網。

## 第五節 Timeout 機制

在 OpenFlow 控制器給予 flow entry 規則時，可以配置 timeout 的時間。而 timeout 分為兩種，一種是 hard timeout，即無論如何，時間一到規則即從 OpenFlow 交換器上消失；另一種 idle timeout，則是當規則已經持續一段時間沒有被比對到時，超過這個時間即消失。所以本系統在給予已認證之用戶規則時，設置 idle timeout。若用戶持續一段時間沒有使用網路，則規則消失，待用戶下次連接上網時，就必須重新認證。本系統預設 idle timeout 時間為一天 86400 秒。當用戶超過一天沒有使用網路，規則將會自動消失。

# 第四章　系統實作




圖二十五　OCPAC network architecture

本研究認為實驗的可複製性很重要，所以本章節會將實驗平台建置的步驟與設定的重點做詳細的介紹。如圖二十五網路架構圖所示，本系統可以分為以下網路元件，其中 Ryu 控制器與 Auth 伺服器透過同一台電腦的 VirtualBox 所架設：

1. Ryu 控制器：

   作為控制網路中所有支援 OpenFlow 設備的控制器，詳細設定見本章第一節；程式碼部分，Network Function Operation Module 參考附錄一；RESTful Operation Module 參考附錄二。

2. Open vSwitch：

   支援 OpenFlow，作為執行 ACL 與 DNS 重導向的設備，詳細設定見本章第二節。

3. Auth Server：

   作為提供認證服務的伺服器，詳細設定見本章第三節；程式碼部分參考附錄三。

4. DHCP Server：

作為提供 DHCP 服務的伺服器，本系統可直接使用暨南大學計算機網路中心所提供之現有 DHCP 伺服器，無須另行架設。

5. Private DNS Server：

作為提供 DNS query 結果為認證伺服器的 DNS 伺服器，詳細設定見本章第四節。

6. Public DNS Server：

作為提供正確的 DNS query 結果，選用公共的 DNS 伺服器。而本文之主機如果透過暨南大學計算機網路中心所提供之 DHCP 服務，則會取得暨南大學計算機網路中心所提供之 DNS 伺服器服務。

7. Router：

使用暨南大學計算機網路中心所提供之服務，在本系統中作為一個與外部網路隔離的邊界路由器。但不支援 OpenFlow。

8. Host：

作為有網路需求的用戶，要透過網路上網，主機隨機選用。

設備規格參照表一。

表一　System specification table

| Virtual Machine | vCPU | Memory | OS | Software |
|---|---|---|---|---|
| Ryu Controller | Intel® Core™ i5-4210U 2.40 GHz | 2G | CentOS 7 | Ryu 3.22 |
| Auth Server | Intel® Core™ i5-4210U 2.40 GHz | 2G | CentOS 7 | Flask Web Frameworks |
| Standalone Server | CPU | Memory | OS | Software |
| Open vSwitch | Intel®　Core™ | 8G | CentOS 7 | Open |

| | | | | |
|---|---|---|---|---|
| | i5-2400 3.10GHz | | | vSwitch 2.3.1 |
| Private DNS Server | Intel® Atom™ N280 1.66GHz | 2G | Fedora 23 | bind |

## 第一節 Ryu 控制器

此元件透過 VirtualBox 的虛擬機器運作。

### 第一小節 Ryu 安裝

1. 安裝環境所需要的套件。

   #yum install -y python-setuptools python-devel python-crypto python-simplejson autoconf python-pip python-devel gcc gcc-c++ libxml2-devel libxslt-devel

2. 安裝 pip，pip 是一個管理 Python 套件的工具。

   #easy_install pip

3. 透過 pip 安裝 ryu 套件。

   #pip install ryu

4. 安裝 git，由於 ryu 的專案發佈在 git 上，所以要透過 git 來下載。

   #yum install -y git

5. 複製 ryu 專案至本機。

   #git clone git://github.com/osrg/ryu.git

6. 安裝 ryu 專案

   #cd ryu; python ./setup.py install

### 第二小節 Network Function Operation Module 運作

以下程式以 simple_switch 13.py 為 Network Function Operation Module 示

範，本實驗在函式 switch_features_handler 中添加了 ARP、DHCP 等協定的規則，作為系統運作時維持網路運作的規則。並注意封包必為雙向的概念與規則優先權。若要禁止特定協定的封包也可以在此加入規則。接著在函式 _packet_in_handler 添加 DNS 重導向的規則。即完成此模組。

### 第三小節　RESTful Operation Module 運作

Ryu 有支援 WSGI（Web Server Gateway Interface）的網頁伺服器，透過這個機制來建立相關的 REST API 可以與其他系統或瀏覽器整合提供網頁服務。該模組在系統啟動後，Ryu 網頁伺服器也啟動，被動等待認證伺服器傳送 REST 命令給 Ryu 網頁伺服器。同時管理者也可以自行發送 REST 命令給 Ryu 網頁伺服器。

### 第四小節　本系統之流程表規劃

本系統將規則分為三類，第一類為通用規則，不論是已認證用戶或未認證用戶皆需要的規則，如 ARP、DHCP；第二類為已認證用戶之規則，實體位址為已事先註冊或已通過認證之實體位置即可順利使用網路；第三類為未認證用戶之規則，在任何規則都沒比對到後進行 Packet-In。而流程表的規則規劃參考表二。

表二　Flow table when a host is authorized

| Usage | Match | Priority | Action |
|---|---|---|---|
| General rules | | | |
| ARP | eth_type = 0x0806 | 600 | Normal |
| DHCP Discover | UDP_src = 67,UDP_dst = 68 | 400 | Normal |
| DHCP Offer | UDP_src = 68, UDP_dst = 67 | 400 | Normal |
| Authorized User | | | |
| Network access | eth address= Authorized Host | 500 | Normal |

| Unauthorized User | | | |
|---|---|---|---|
| | | 0 | Packet-In |

## 第二節 Open vSwitch

除了主機板上內建的一個網路埠，另外加裝了一張擁有四個網路埠的網路卡（型號：Intel® Ethernet Server Adapter I340-T4），讓此主機在安裝 Open vSwitch 軟體後作為一個 OpenFlow 交換器。以下將會詳細介紹 Open vSwitch 的安裝與設定。

### 第一小節　　Open vSwitch 安裝

在 CentOS 7 上安裝 Open vSwitch 2.3.1 步驟如下：

1. 安裝環境所需要的套件包。

   #yum -y install wget openssl-devel gcc make python-devel openssl-devel kernel-devel graphviz kernel-debug-devel autoconf automake rpm-build redhat-rpm-config libtool

2. 新增並登入使用者 ovs。

   #adduser ovs

   #su ovs

   #cd

3. 取得原始碼並於本地安裝。

   #mkdir -p ~/rpmbuild/SOURCES

   #wget http://openvswitch.org/releases/openvswitch-2.3.1.tar.gz

   #cp openvswitch-2.3.1.tar.gz ~/rpmbuild/SOURCES/

   #tar xfz openvswitch-2.3.1.tar.gz

   #sed 's/openvswitch-kmod, //g' openvswitch-2.3.1/rhel/openvswitch.spec > openvswitch-2.3.1/rhel/openvswitch_no_kmod.spec

4. 建立 RPM。

   # rpmbuild -bb --nocheck

   ~/openvswitch-2.3.1/rhel/openvswitch_no_kmod.spec

   #exit

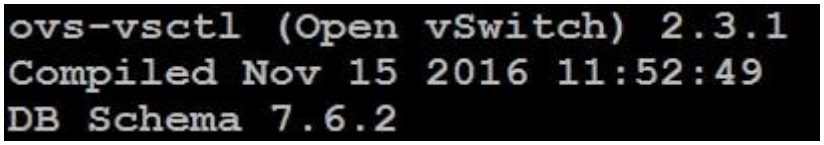
5. 本地安裝 Open vSwitch。

   #yum localinstall -y

   /home/ovs/rpmbuild/RPMS/x86_64/openvswitch-2.3.1-1.x86_64.rpm

6. 確認版本，版本如圖二十六。

   # ovs-vsctl -V




圖二十六　Show Open vSwitch version

7. SELinux 設定。

   #yum install -y policycoreutils-python

   #mkdir /etc/openvswitch

   # semanage fcontext -a -t openvswitch_rw_t "/etc/openvswitch(/.*)?"

   # restorecon -Rv /etc/openvswitch

8. 啟動服務（使用時需要先調整防火牆設定，開啟與控制器連接的埠號，如 TCP 埠：6633。）。

   #sudo systemctl start openvswitch.service

## 第二小節　Open vSwitch 設定

以下為本實驗 Open vSwitch 設定，設定結束後可以透過指令 ovs-vsctl show 觀察 Bridge。

1. 新增 Bridge 名稱為 ovs。

#ovs-vsctl add-br ovs

2. 啟動 Bridge ovs。

   #ifconfig ovs up

3. 清空 eth0 介面，此介面作為該 Open vSwitch 的 Uplink 使用。

   ifconfig eth0 0

4. 將 Bridge ovs 綁至 eth0 介面上。

   #ovs-vsctl add-port ovs eth0

5. Bridge ovs 就可以透過 eth0 取得 DHCP 的服務。

   #dhclient -r ovs && dhclient ovs

6. 啟動其他的介面作為 Downlink 使用。

   #ifconfig eth1 up

   #ifconfig eth2 up

   #ifconfig eth3 up

   #ifconfig eth4 up

7. 將這些介面綁至 Bridge ovs 上。

   #ovs-vsctl add-port ovs eth1

   #ovs-vsctl add-port ovs eth2

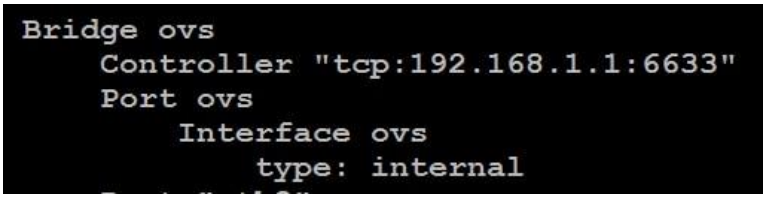
   #ovs-vsctl add-port ovs eth3

   #ovs-vsctl add-port ovs eth4

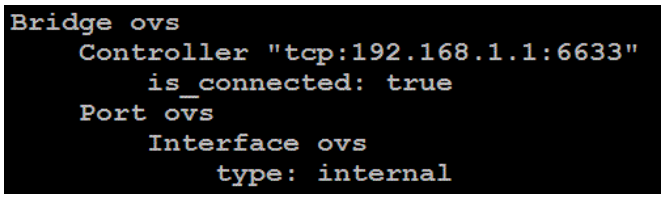
8. Bridge ovs 連結控制器，假設控制器 IP 位址為 192.168.1.1，且 TCP 埠號 6633。

   #ovs-vsctl set-controller ovs tcp:192.168.1.1:6633

連接控制器失敗畫面參考圖二十七，成功畫面參考圖二十八。




圖二十七　Open vSwitch fails to connect to Ryu controller




圖二十八　Open vSwitch connects to Ryu controller successfully

## 第三節 Auth Server

認證伺服器是透過 Flask 所建構起來的。Flask 是由 Python 所編寫的輕量級網頁應用框架，本研究會選用此架構是為了與 Ryu 控制器選用相同的語言撰寫以方便整合。

### 第一小節 Flask 安裝

請確認系統中存在 Python 2.6 以上之版本的 Python 。以下為 Flask 安裝步驟：

1. 安裝 pip。

   #easy install pip

2. 安裝 Flask。

   #pip install flask

待安裝完畢後，將整個 Flask 網站改為外部可登錄即可供其他使用者使用，請參考圖二十九與附錄三 Authentication Module 之程式碼，在執行網頁伺服器 app.run()當中加入 host='0.0.0.0'，本文為了擴充 IPv6 的服務，所以，將 host='0.0.0.0' 改為 host='::'，同時可以提供 IPv4 與 IPv6 的服務並指定埠號 80，詳細請參考附錄三。本研究撰寫一個前端網頁供使用者登入，前端網頁頁面請參考圖三十。以 Python 作為後端，根據使用者對前端網頁的登入資訊進行判斷，傳送 REST 命令給 RESTful Operation 模組。網站須注意可能有數個使用者同時登入網站，所以必須支援 threaded；支援 threaded 之部分程式碼請參考圖三十一與附錄三。

```
app.run(host='0.0.0.0')
```

圖二十九　Externally visible server


圖三十　Auth server web page

```
app.run(threaded=True)
```

圖三十一　Set flag to true to enable threading

## 第四節 Private DNS Server

由於 DNS 亦為常見之網路服務，以下討論其安裝與設定。

### 第一小節　　DNS 安裝

#yum -y install bind bind-utils

### 第二小節　　DNS 設定

設定檔/etc/named.conf 中需要特別注意的參數配置如下，其餘參數根據系統需求自行配置：

```
options {
    allow-query { any; };
};
zone ”.” IN {
    type master; file ”named.ca”;
};
```

1. allow-query { any; };允許所有其他機器向本機器進行 DNS query。
2. file ”named.ca”;所使用的 zone 檔案為 named.ca，下個小節會解釋此 zone 檔案該如何設定。

### 第三小節　　Zone 設定

設定檔/var/named/named.ca，所有查詢結果均為認證伺服器位址，；以本系統認證伺服器 IP 位址 10.21.20.162 為例，其設定如下：

```
$ORIGIN .
$TTL 1D
@    IN    SOA     @ none. ( 0 1D 1H 1W 3H );
     IN    NS      @
*    IN    A       10.21.20.162
```

# 第五章 實驗結果

經本研究環境佈建與實際測試，本系統與市面上常用之網頁認證方法由於架設位置之差異，即能觀察到對於整個網路的影響。而本研究提出的方法，藉由 SDN 的架構改善之後，可經由 Ryu 控制器統一發送規則，減少了設定的次數。由於設定存取控制的位置靠近末端的特性得以同時進行 IP 與 MAC 位址的管制，將匯集到 Firewall 的流量分散至 OpenFlow 交換器上，也可以避免 Firewall 成為網路瓶頸，達到良好的效果。

# 第六章 結論與未來展望

本研究提出一個適用於網路使用者流動性較高的環境中，透過 Ryu 集中控制 OpenFlow vSwitch 結合網頁認證技術，減少額外流量造成網路瓶頸形成之問題，達成網路管理存取控制之目的。

未來可以增加一個圖形介面的網頁操作介面，讓網路管理者可以透過更簡便的 Web 介面來檢視整個網路，讓繁瑣的更新存取控制清單能夠更有效率地完成，減少人為疏失，達到更好的效果。

# 參考文獻


[1] Gordon E Moore et al, “Progress in digital integrated electronics,” Electron Devices Meeting, volume 21, pages 11–13, 1975.

[2] Ravi S Sandhu and Pierangela Samarati, “Access control: principle and practice,” IEEE communications magazine, volume 32:40–48, 1994.

[3] Brocade, “Network OS 6.0 Troubleshooting Guide,” [Online]. Available:http://www.brocade.com/content/html/en/troubleshooting-guide/NOS_600_TSHOOT/GUID-632BEFC8-66F9-4FCF-BB83-092B8A85CD97.html.

[4] Starbucks, “星巴克無線上網約定條款,” [Online]. Available: https://www.wifly.com.tw/StarbucksFree/rule.aspx?DeviceKind=1&RTYPE=2.

[5] Guido Appenzeller, Mema Roussopoulos, and Mary Baker, “User-friendly access control for public network ports,” Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies, volume 2, pages 699–707, 1999.

[6] Floris, Andrea, and Luca Veltri, “Access control in IPv6-based roaming scenarios,” IEEE International Conference on Communications, volume 2, 2003.

[7] Thomas D Nadeau and Ken Gray, “SDN: software defined networks,” O’Reilly Media, 2013.

[8] Steve Deering, “Watching the Waist of the Protocol Hourglass,” [Online]. Available:https://www.iab.org/wp-content/IAB-uploads/2010/11/hourglass-london-ietf.pdf, 2001

[9] Open Networking Foundation, “Software-defined networking: The new norm for networks,”ONF White Paper, 2012.

[10]Tom Beggan, “Software defined networking (sdn) creating intelligent lan infrastructures,” [Online].Available:http://www.slideshare.net/adnettech/software-defined-networking-sdn-creating-intelligent-lan-infrastructures.

[11]Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner, “Openflow: enabling innovation in campus networks,” ACM SIGCOMM Computer Communication Review, volume 38:69–74, 2008

[12]Open Networking Foundation, “Software-Defined Networking (SDN) Definition,” [Online]. Available:https://www.opennetworking.org/sdn-resources/sdn-definition.

[13] Brocade, “FastIron Ethernet Switch Software Defined Networking (SDN) Configuration Guide,” [Online]. Available:http://www.brocade.com/content/html/en/configuration-guide/FI_08030_SDN/GUID-031030CA-62EC-4009-A516-5510238EF8F4.html.

[14]“Public DNS Server List,” [Online]. Available:http://public-dns.info/.

# 附錄一　Network Function Operation Module 之程式碼

以下為本論文 Network Function Operation Module 之程式碼，原始碼參考 simple_switch_13.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
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet
from ryu.lib.packet import ipv4
from ryu.lib.packet import ipv6
from ryu.lib.packet import udp
from ryu.lib.packet import tcp

mac_to_port_list={}
port_to_dns_list={}
class SimpleSwitch13(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super(SimpleSwitch13, self).__init__(*args, **kwargs)
        self.mac_to_port = {}

    @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
    def switch_features_handler(self, ev):
        datapath = ev.msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
        #parser_v1_0 = datapath.ofproto_v1_0_parser
        # install table-miss flow entry
        #
        # We specify NO BUFFER to max_len of the output
        # action due to
        # OVS bug. At this moment, if we specify a lesser number, e.g.,
        # 128, OVS will send Packet-In with invalid buffer_id and
        # truncated packet data. In that case, we cannot output packets
        # correctly.    The bug has been fixed in OVS
        # v2.1.0.
```

```
#match = parser.OFPMatch(eth_type = 0x0806)
#match = parser.OFPMatch()
#actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER)]
#self.add_flow(datapath, 0, match, actions)

#arp
match = parser.OFPMatch(eth_type = 0x0806)
actions = [parser.OFPActionOutput(ofproto.OFPP_NORMAL)]
self.add_flow(datapath, 600, match, actions)

#dhcp discovery
match = parser.OFPMatch(
                                    eth_type = 0x0800,
                                    ip_proto = 0x11,
                                    udp_src = 68,
                                    udp_dst = 67)
actions = [parser.OFPActionOutput(ofproto.OFPP_NORMAL)]
self.add_flow(datapath, 400, match, actions)

#dhcp offer
match = parser.OFPMatch(
                                    eth_type = 0x0800,
                                    ip_proto = 0x11,
                                    udp_src = 67,
                                    udp_dst = 68)
actions = [parser.OFPActionOutput(ofproto.OFPP_NORMAL)]
self.add_flow(datapath, 400, match, actions)

#ICMPv6    Router Solicitation
match = parser.OFPMatch(
                                    eth_type = 0x86dd,
                                    ip_proto = 0x3a,
                                    icmpv6_type = 133)
actions = [parser.OFPActionOutput(ofproto.OFPP_NORMAL)]
self.add_flow(datapath, 400, match, actions)
#Router Advertisement
match = parser.OFPMatch(
                                    eth_type = 0x86dd,
                                    ip_proto = 0x3a,
                                    icmpv6_type = 134)
actions = [parser.OFPActionOutput(ofproto.OFPP_NORMAL)]
self.add_flow(datapath, 400, match, actions)
#Neighbor Solicitation
match = parser.OFPMatch(
                                    eth_type = 0x86dd,
                                    ip_proto = 0x3a,
```

```
                                    icmpv6_type = 135)
actions = [parser.OFPActionOutput(ofproto.OFPP_NORMAL)]
self.add_flow(datapath, 400, match, actions)
#Neighbor Advertisement
match = parser.OFPMatch(
                                    eth_type = 0x86dd,
                                    ip_proto = 0x3a,
                                    icmpv6_type = 136)
actions = [parser.OFPActionOutput(ofproto.OFPP_NORMAL)]
self.add_flow(datapath, 400, match, actions)


#Version 2 Multicast Listener Report
match = parser.OFPMatch(
                                    eth_type = 0x86dd,
                                    ip_proto = 0x3a,
                                    icmpv6_type = 143)
actions = [parser.OFPActionOutput(ofproto.OFPP_NORMAL)]
self.add_flow(datapath, 400, match, actions)


#icmp block
match = parser.OFPMatch(
                                    eth_type = 0x0800,
                                    ip_proto = 0x01)
actions = []
self.add_flow(datapath, 100, match, actions)

#no match to controller
match = parser.OFPMatch()
actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER)]
self.add_flow(datapath, 0, match, actions)


#My PC
match = parser.OFPMatch(
                                    eth_type = 0x0800,
                                    eth_src = "9c:eb:e8:14:d0:45")
actions = [parser.OFPActionOutput(ofproto.OFPP_NORMAL)]
self.add_flow(datapath, 500, match, actions)
match = parser.OFPMatch(
                                    eth_type = 0x0800,
                                    eth_dst = "9c:eb:e8:14:d0:45")
actions = [parser.OFPActionOutput(ofproto.OFPP_NORMAL)]
self.add_flow(datapath, 500, match, actions)
#default rules for client value name c is
# client
```

```
def add_flow(self, datapath, priority, match, actions, buffer_id=None):
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser

    inst = [parser.OFPInstructionActions(
            ofproto.OFPIT_APPLY_ACTIONS,actions)]

    if buffer_id:
        mod = parser.OFPFlowMod(
                                        datapath=datapath,
                                        buffer_id=buffer_id,
                                        priority=priority,
                                        match=match,
                                        instructions=inst)
    else:
        mod = parser.OFPFlowMod(
                                        datapath=datapath,
                                        priority=priority,
                                        match=match,
                                        instructions=inst)
        datapath.send_msg(mod)


@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
    # If you hit this you might want to increase
    # the "miss_send_length" of your switch
    if ev.msg.msg_len < ev.msg.total_len:
        self.logger.debug(
                        "packet truncated: only %s of %s bytes",
                        ev.msg.msg_len, ev.msg.total_len)
    msg = ev.msg
    datapath = msg.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser
    in_port = msg.match['in_port']

    pkt = packet.Packet(msg.data)
    eth = pkt.get_protocols(ethernet.ethernet)[0]
    pkt_udp = pkt.get_protocol(udp.udp)
    pkt_tcp = pkt.get_protocol(tcp.tcp)
    pkt_ipv4 = pkt.get_protocol(ipv4.ipv4)
    pkt_ipv6 = pkt.get_protocol(ipv6.ipv6)

    dst = eth.dst
    src = eth.src
```

```
        dpid = datapath.id
        self.mac_to_port.setdefault(dpid, {})

        #self.logger.info("packet in %s | MacSrc:%s #MacDst:%s %s",
                          dpid, src, dst, in_port)


        # learn a mac address to avoid FLOOD next time.

        self.mac_to_port[dpid][src] = in_port


        if dst in self.mac_to_port[dpid]:
            out_port = self.mac_to_port[dpid][dst]
        else:
            out_port = ofproto.OFPP_FLOOD

        actions = [parser.OFPActionOutput(out_port)]
#DNS redir – dns query
        if pkt_udp:
            if pkt_udp.dst_port==53:
                if pkt_ipv6:
                    pass
                if pkt_ipv4:
                    print mac_to_port_list
                    print port_to_dns_list
                    if pkt_ipv4.dst != "10.21.20.126":
                        mac_to_port_list[eth.src] =
                        pkt_udp.src_port
                        port_to_dns_list[mac_to_port_list[eth.src]] =
                            pkt_ipv4.dst
                        print "DNS Query : "
                        print ("--------------------------------\n")
                        print "User MAC : "
                        print eth.src
                        print "User UDP port : "
                        print pkt_udp.src_port
                        print "User UDP port in list : "
                        print mac_to_port_list[eth.src]
                        print "DNS server : "
                        print pkt_ipv4.dst
                        print "DNS server in list : "
                        print port_to_dns_list[mac_to_port_list[eth.src]]
                        print ("--------------------------------\n")
                        match = parser.OFPMatch(
                            eth_type = 0x800,
```

```
                        eth_src = eth.src,
                        udp_dst = 53,
                        udp_src = mac_to_port_list[eth.src],
                        ipv4_dst =
                        port_to_dns_list[
                        mac_to_port_list[eth.src]],
                        ip_proto = 0x11)
                    actions = [
                            parser.OFPActionSetField(
                                ipv4_dst="10.21.20.126"),
                            parser.OFPActionSetField(
                                eth_dst="00:23:5a:ff:7a:01"),
                            parser.OFPActionOutput(
                                datapath.ofproto.OFPP_NORMAL)]
#back DNS redir – dns response
if pkt_udp.src_port==53:
    if pkt_ipv6:
        pass
    if pkt_ipv4:
        print mac_to_port_list
        print port_to_dns_list
        if pkt_ipv4.src == "10.21.20.126":
            print "DNS response : "
            print ("--------------------------------\n")
            print "User MAC : "
            print eth.dst
            print "User UDP port : "
            print pkt_udp.dst_port
            print "User UDP port in list : "
            print mac_to_port_list[eth.dst]
            print "DNS server : "
            print pkt_ipv4.src
            print "DNS server in list : "
            print port_to_dns_list[mac_to_port_list[eth.dst]]
            print ("--------------------------------\n")
            match = parser.OFPMatch(
                eth_type = 0x800,
                udp_src = 53,
                udp_dst = mac_to_port_list[eth.dst] ,
                eth_dst = eth.dst,
                ipv4_src = "10.21.20.126",
                ip_proto = 0x11)
            actions = [
                        parser.OFPActionSetField(
                            ipv4_src=port_to_dns_list[
                            mac_to_port_list[eth.dst]]),
                        parser.OFPActionOutput(
```

```
                                                    datapath.ofproto.OFPP_NORMAL)]
                                else:
                                    return
#http 封包 Packet-In 給 REST 模組紀錄後再以 Packet-Out 轉送
                    if pkt_tcp:
                        if pkt_tcp.dst_port == 80:
                            print "to_http_server_mac : " + src

                            match = parser.OFPMatch(
                                                    eth_type = 0x0800,
                                                    ip_proto = 0x06,
                                                    tcp_dst = 80)
                            actions = [
                                parser.OFPActionOutput(ofproto.OFPP_NORMAL)]

                            match = parser.OFPMatch(
                                                eth_type = 0x0800,
                                                ip_proto = 0x06,
                                                tcp_src = 80)
                            actions = [
                                parser.OFPActionOutput(ofproto.OFPP_NORMAL)]

                            #return
        # install a flow to avoid packet_in next time
        #              if out_port != ofproto.OFPP_FLOOD:
        #                  match = parser.OFPMatch(in_port=in_port,
        #                      eth_dst=dst)
        # verify if we have a valid buffer_id,
        # if yes avoid to send both
        # flow_mod & packet_out
        #                  if msg.buffer_id != ofproto.OFP_NO_BUFFER:
        #                      self.add_flow(datapath, 1, match,
        #                                    actions, msg.buffer_id)
        #                      return
        #                  else:
        #                      self.add_flow(datapath, 1, match,
        #                                    actions)
                    data = None
                    if msg.buffer_id == ofproto.OFP_NO_BUFFER:
                        data = msg.data
        #                self.add_flow(datapath, 10,
        #                              match_dns_src, actions)

                    out = parser.OFPPacketOut(
                                                datapath=datapath,
                                                buffer_id=msg.buffer_id,
                                                in_port=in_port,
```

```
actions=actions, data=data)
datapath.send_msg(out)
```

# 附錄二　RESTful Operation Module 之程式碼

以下為本論文 RESTful Operation Module 之程式碼，原始程式為 rest_firewall.py，紅色部分為我所添加或修改之部分（如對目錄結構需要詳細的解釋請參考 REST API 列表。）：

```
import logging
import json
from webob import Response
from ryu.app.wsgi import ControllerBase
from ryu.app.wsgi import WSGIApplication
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller import dpset
from ryu.controller.handler import MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.exception import OFPUnknownVersion
from ryu.lib import mac
from ryu.lib import dpid as dpid_lib
from ryu.lib import ofctl_v1_0
from ryu.lib import ofctl_v1_2
from ryu.lib import ofctl_v1_3
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet
from ryu.lib.packet import tcp
from ryu.lib.packet import ipv4
from ryu.lib.packet import ipv6
from ryu.ofproto import ether
from ryu.ofproto import inet
from ryu.ofproto import ofproto_v1_0
from ryu.ofproto import ofproto_v1_2
from ryu.ofproto import ofproto_v1_2_parser
from ryu.ofproto import ofproto_v1_3
from ryu.ofproto import ofproto_v1_3_parser

# ==============================
#              REST API
# ==============================
#
#   Note: specify switch and vlan group, as follows.
#     {switch-id} : 'all' or switchID
#     {vlan-id}      : 'all' or vlanID
#
#
```

```
# about Firewall status
#
# get status of all firewall switches
# GET /firewall/module/status
#
# set enable the firewall switches
# PUT /firewall/module/enable/{switch-id}
#
# set disable the firewall switches
# PUT /firewall/module/disable/{switch-id}
#

# about Firewall logs
#
# get log status of all firewall switches
# GET /firewall/log/status
#
# set log enable the firewall switches
# PUT /firewall/log/enable/{switch-id}
#
# set log disable the firewall switches
# PUT /firewall/log/disable/{switch-id}
#

# about Firewall rules
#
# get rules of the firewall switches
# * for no vlan
# GET /firewall/rules/{switch-id}
#
# * for specific vlan group
# GET /firewall/rules/{switch-id}/{vlan-id}
#
#
# set a rule to the firewall switches
# * for no vlan
# POST /firewall/rules/{switch-id}
#
# * for specific vlan group
# POST /firewall/rules/{switch-id}/{vlan-id}
#
#   request body format:
#     {"<field1>":"<value1>", "<field2>":"<value2>",...}
#
#         <field>    : <value>
#       "priority": "0 to 65533"
```

```
#       "in_port" : "<int>"
#       "dl_src"    : "<xx:xx:xx:xx:xx:xx>"
#       "dl_dst"    : "<xx:xx:xx:xx:xx:xx>"
#       "dl_type" : "<ARP or IPv4 or IPv6>"
#       "nw_src"    : "<A.B.C.D/M>"
#       "nw_dst"    : "<A.B.C.D/M>"
#       "ipv6_src": "<xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx/M>"
#       "ipv6_dst": "<xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx/M>"
#       "nw_proto": "<TCP or UDP or ICMP or ICMPv6>"
#       "tp_src"    : "<int>"
#       "tp_dst"    : "<int>"
#       "actions" : "<ALLOW or DENY>"
#
#     Note: specifying nw_src/nw_dst
#               without specifying dl-type as "ARP" or "IPv4"
#               will automatically set dl-type as "IPv4".
#
#     Note: specifying ipv6_src/ipv6_dst
#               without specifying dl-type as "IPv6"
#               will automatically set dl-type as "IPv6".
#
#     Note: When "priority" has not been set up,
#               "0" is set to "priority".
#
#     Note: When "actions" has not been set up,
#               "ALLOW" is set to "actions".
#
#
# delete a rule of the firewall switches from ruleID
# * for no vlan
# DELETE /firewall/rules/{switch-id}
#
# * for specific vlan group
# DELETE /firewall/rules/{switch-id}/{vlan-id}
#
#   request body format:
#     {"<field>":"<value>"}
#
#         <field>    : <value>
#       "rule_id" : "<int>" or "all"
#


SWITCHID_PATTERN = dpid_lib.DPID_PATTERN + r'|all'
VLANID_PATTERN = r'[0-9]{1,4}|all'

REST_ALL = 'all'
```

```
REST_SWITCHID = 'switch_id'
REST_VLANID = 'vlan_id'
REST_RULE_ID = 'rule_id'
REST_STATUS = 'status'
REST_LOG_STATUS = 'log_status'
REST_STATUS_ENABLE = 'enable'
REST_STATUS_DISABLE = 'disable'
REST_COMMAND_RESULT = 'command_result'
REST_ACL = 'access_control_list'
REST_RULES = 'rules'
REST_COOKIE = 'cookie'
REST_PRIORITY = 'priority'
REST_MATCH = 'match'
REST_IN_PORT = 'in_port'
REST_SRC_MAC = 'dl_src'
REST_DST_MAC = 'dl_dst'
REST_DL_TYPE = 'dl_type'
REST_DL_TYPE_ARP = 'ARP'
REST_DL_TYPE_IPV4 = 'IPv4'
REST_DL_TYPE_IPV6 = 'IPv6'
REST_DL_VLAN = 'dl_vlan'
REST_SRC_IP = 'nw_src'
REST_DST_IP = 'nw_dst'
REST_SRC_IPV6 = 'ipv6_src'
REST_DST_IPV6 = 'ipv6_dst'
REST_NW_PROTO = 'nw_proto'
REST_NW_PROTO_TCP = 'TCP'
REST_NW_PROTO_UDP = 'UDP'
REST_NW_PROTO_ICMP = 'ICMP'
REST_NW_PROTO_ICMPV6 = 'ICMPv6'
REST_TP_SRC = 'tp_src'
REST_TP_DST = 'tp_dst'
REST_ACTION = 'actions'
REST_ACTION_ALLOW = 'ALLOW'
REST_ACTION_DENY = 'DENY'
REST_ACTION_PACKETIN = 'PACKETIN'


STATUS_FLOW_PRIORITY = ofproto_v1_3_parser.UINT16_MAX
ARP_FLOW_PRIORITY = ofproto_v1_3_parser.UINT16_MAX-1
LOG_FLOW_PRIORITY = 0
ACL_FLOW_PRIORITY_MIN = LOG_FLOW_PRIORITY + 1
ACL_FLOW_PRIORITY_MAX = ofproto_v1_3_parser.UINT16_MAX-2

VLANID_NONE = 0
VLANID_MIN = 2
VLANID_MAX = 4094
```

```
COOKIE_SHIFT_VLANID = 32
#初始化紀錄表
mac_ip_list = {}

class RestFirewallAPI(app_manager.RyuApp):

    OFP_VERSIONS = [ofproto_v1_0.OFP_VERSION,
                    ofproto_v1_2.OFP_VERSION,
                    ofproto_v1_3.OFP_VERSION]

    _CONTEXTS = {'dpset': dpset.DPSet, 'wsgi': WSGIApplication}

    def __init__(self, *args, **kwargs):
        super(RestFirewallAPI, self).init(*args,**kwargs)

        # logger configure
        FirewallController.set_logger(self.logger)


        self.dpset = kwargs['dpset']
        wsgi = kwargs['wsgi']
        self.waiters = {}
        self.data = {}
        self.data['dpset'] = self.dpset
        self.data['waiters'] = self.waiters

        mapper = wsgi.mapper
        wsgi.registory['FirewallController'] = self.data
        path = '/firewall'
        requirements = {'switchid': SWITCHID_PATTERN,
                        'vlanid': VLANID_PATTERN}

        # for firewall status
        uri = path + '/module/status'
        mapper.connect('firewall', uri,
                       controller=FirewallController,
                       action='get_status',
                       conditions=dict(method=['GET']))

        uri = path + '/module/enable/{switchid}'
        mapper.connect('firewall', uri,
                       controller=FirewallController,
                       action='set_enable',
                       conditions=dict(method=['PUT']),
                       requirements=requirements)

        uri = path + '/module/disable/{switchid}'
```

```
mapper.connect('firewall', uri,
                        controller=FirewallController,
                        action='set_disable',
                        conditions=dict(method=['PUT']),
                        requirements=requirements)

# for firewall logs
uri = path + '/log/status'
mapper.connect('firewall', uri,
                        controller=FirewallController,
                        action='get_log_status',
                        conditions=dict(method=['GET']))

uri = path + '/log/enable/{switchid}'
mapper.connect('firewall', uri,
                        controller=FirewallController,
                        action='set_log_enable',
                        conditions=dict(method=['PUT']),
                        requirements=requirements)

uri = path + '/log/disable/{switchid}'
mapper.connect('firewall', uri,
                        controller=FirewallController,
                        action='set_log_disable',
                        conditions=dict(method=['PUT']),
                        requirements=requirements)

# for no VLAN data
uri = path + '/rules/{switchid}'
mapper.connect('firewall', uri,
                        controller=FirewallController,
                        action='get_rules',
                        conditions=dict(method=['GET']),
                        requirements=requirements)

mapper.connect('firewall', uri,
                        controller=FirewallController,
                        action='set_rule',
                        conditions=dict(method=['POST']),
                        requirements=requirements)

mapper.connect('firewall', uri,
                        controller=FirewallController,
                        action='delete_rule',
                        conditions=dict(method=['DELETE']),
                        requirements=requirements)
```

```
        # for VLAN data
        uri += '/{vlanid}'
        mapper.connect('firewall', uri,
                                controller=FirewallController,
                                action='get_vlan_rules',
                                conditions=dict(method=['GET']),
                                requirements=requirements)

        mapper.connect('firewall', uri,
                                controller=FirewallController,
                                action='set_vlan_rule',
                                conditions=dict(method=['POST']),
                                requirements=requirements)

        mapper.connect('firewall', uri,
                                controller=FirewallController,
                                action='delete_vlan_rule',
                                conditions=dict(method=['DELETE']),
                                requirements=requirements)

    def stats_reply_handler(self, ev):
        msg = ev.msg
        dp = msg.datapath

        if dp.id not in self.waiters:
            return
        if msg.xid not in self.waiters[dp.id]:
            return
        lock, msgs = self.waiters[dp.id][msg.xid]
        msgs.append(msg)

        flags = 0
        if dp.ofproto.OFP_VERSION == ofproto_v1_0.OFP_VERSION or \
                dp.ofproto.OFP_VERSION == ofproto_v1_2.OFP_VERSION:
            flags = dp.ofproto.OFPSF_REPLY_MORE
        elif dp.ofproto.OFP_VERSION == ofproto_v1_3.OFP_VERSION:
            flags = dp.ofproto.OFPMPF_REPLY_MORE

        if msg.flags & flags:
            return
        del self.waiters[dp.id][msg.xid]
        lock.set()

    @set_ev_cls(dpset.EventDP, dpset.DPSET_EV_DISPATCHER)
    def handler_datapath(self, ev):
        if ev.enter:
            FirewallController.regist_ofs(ev.dp)
```

```
            else:
                FirewallController.unregist_ofs(ev.dp)

        # for OpenFlow version1.0
        @set_ev_cls(ofp_event.EventOFPFlowStatsReply, MAIN_DISPATCHER)
        def stats_reply_handler_v1_0(self, ev):
            self.stats_reply_handler(ev)

        # for OpenFlow version1.2 or later
        @set_ev_cls(ofp_event.EventOFPStatsReply, MAIN_DISPATCHER)
        def stats_reply_handler_v1_2(self, ev):
            self.stats_reply_handler(ev)

        @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
        def packet_in_handler(self, ev):
            pkt = packet.Packet(ev.msg.data)
            eth = pkt.get_protocols(ethernet.ethernet)[0]
#將經過 OVS 封包網路位址的實體位址紀錄下來
            pkt_ipv4 = pkt.get_protocol(ipv4.ipv4)
            pkt_ipv6 = pkt.get_protocol(ipv6.ipv6)
            pkt_tcp = pkt.get_protocol(tcp.tcp)
            if pkt_tcp:
                if pkt_tcp.dst_port == 80:
                    if pkt_ipv4:
                        if pkt_ipv4.dst == "10.21.20.162":
                            print "ipv4 :" + pkt_ipv4.src
                            mac_ip_list[pkt_ipv4.src] = eth.src
                    if pkt_ipv6:
                        if pkt_ipv6.dst ==
                        "2001:e10:6840:21:a00:27ff:fe16:b5ba":
                            print "ipv6 :" + pkt_ipv6.src
                            mac_ip_list[pkt_ipv6.src] = eth.src
            FirewallController.packet_in_handler(ev.msg)


class FirewallOfsList(dict):
    def __init__(self):
        super(FirewallOfsList, self).__init__()

    def get_ofs(self, dp_id):
        if len(self) == 0:
            raise ValueError('firewall sw is not connected.')

        dps = {}
        if dp_id == REST_ALL:
            dps = self
        else:
```

```
                try:
                    dpid = dpid_lib.str_to_dpid(dp_id)
                except:
                    raise ValueError('Invalid switchID.')

                if dpid in self:
                    dps = {dpid: self[dpid]}
                else:
                    msg = 'firewall sw is not connected. :
                                switchID=%s' % dp_id
                    raise ValueError(msg)

            return dps


class FirewallController(ControllerBase):

    _OFS_LIST = FirewallOfsList()
    _LOGGER = None

    def __init__(self, req, link, data, **config):
        super(FirewallController, self).__init__(req, link, data, **config)
        self.dpset = data['dpset']
        self.waiters = data['waiters']

    @classmethod
    def set_logger(cls, logger):
        cls._LOGGER = logger
        cls._LOGGER.propagate = False
        hdlr = logging.StreamHandler()
        fmt_str = '[FW][%(levelname)s] %(message)s'
        hdlr.setFormatter(logging.Formatter(fmt_str))
        cls._LOGGER.addHandler(hdlr)

    @staticmethod
    def regist_ofs(dp):
        dpid_str = dpid_lib.dpid_to_str(dp.id)
        try:
            f_ofs = Firewall(dp)
        except OFPUnknownVersion as message:
            FirewallController._LOGGER.info(
                                        'dpid=%s: %s', dpid_str, message)
            return

        FirewallController._OFS_LIST.setdefault(dp.id, f_ofs)
#將不需要的規則註解掉 原系統中已有同樣規則
        #f_ofs.set_disable_flow()
```

```
        #f_ofs.set_arp_flow()
        f_ofs.set_log_enable()
        FirewallController._LOGGER.info('dpid=%s: Join as firewall.',dpid_str)

    @staticmethod
    def unregist_ofs(dp):
        if dp.id in FirewallController._OFS_LIST:
            del FirewallController._OFS_LIST[dp.id]
            FirewallController._LOGGER.info(
                'dpid=%s: Leave firewall.', dpid_lib.dpid_to_str(dp.id))

    # GET /firewall/module/status
    def get_status(self, req, **_kwargs):
        return self._access_module(REST_ALL,'get_status',waiters=self.waiters)

    # POST /firewall/module/enable/{switchid}
    def set_enable(self, req, switchid, **_kwargs):
        return self._access_module(switchid, 'set_enable_flow')

    # POST /firewall/module/disable/{switchid}
    def set_disable(self, req, switchid, **_kwargs):
        return self._access_module(switchid, 'set_disable_flow')

    # GET /firewall/log/status
    def get_log_status(self, dummy, **_kwargs):
        return self._access_module(
                                REST_ALL,'get_log_status',waiters=self.waiters)

    # PUT /firewall/log/enable/{switchid}
    def set_log_enable(self, dummy, switchid, **_kwargs):
        return self._access_module(
                                switchid,'set_log_enable',waiters=self.waiters)

    # PUT /firewall/log/disable/{switchid}
    def set_log_disable(self, dummy, switchid, **_kwargs):
        return self._access_module(switchid,
                                           'set_log_disable',
                                           waiters=self.waiters)

    def _access_module(self, switchid, func, waiters=None):
        try:
            dps = self._OFS_LIST.get_ofs(switchid)
        except ValueError as message:
            return Response(status=400, body=str(message))

        msgs = []
        for f_ofs in dps.values():
```

```
            function = getattr(f_ofs, func)
            msg = function() if waiters is None else function(waiters)
            msgs.append(msg)

        body = json.dumps(msgs)
        return Response(content_type='application/json', body=body)

    # GET /firewall/rules/{switchid}
    def get_rules(self, req, switchid, **_kwargs):
        return self._get_rules(switchid)

    # GET /firewall/rules/{switchid}/{vlanid}
    def get_vlan_rules(self, req, switchid, vlanid, **_kwargs):
        return self._get_rules(switchid, vlan_id=vlanid)

    # POST /firewall/rules/{switchid}
    def set_rule(self, req, switchid, **_kwargs):
        return self._set_rule(req, switchid)

    # POST /firewall/rules/{switchid}/{vlanid}
    def set_vlan_rule(self, req, switchid, vlanid, **_kwargs):
        return self._set_rule(req, switchid, vlan_id=vlanid)

    # DELETE /firewall/rules/{switchid}
    def delete_rule(self, req, switchid, **_kwargs):
        return self._delete_rule(req, switchid)

    # DELETE /firewall/rules/{switchid}/{vlanid}
    def delete_vlan_rule(self, req, switchid, vlanid, **_kwargs):
        return self._delete_rule(req, switchid, vlan_id=vlanid)

    def _get_rules(self, switchid, vlan_id=VLANID_NONE):
        try:
            dps = self._OFS_LIST.get_ofs(switchid)
            vid = FirewallController._conv_toint_vlanid(vlan_id)
        except ValueError as message:
            return Response(status=400, body=str(message))

        msgs = []
        for f_ofs in dps.values():
            rules = f_ofs.get_rules(self.waiters, vid)
            msgs.append(rules)

        body = json.dumps(msgs)
        return Response(content_type='application/json', body=body)

    def _set_rule(self, req, switchid, vlan_id=VLANID_NONE):
```

```
            try:
                rule = json.loads(req.body)
#Packet-In 時已將實體位址與網路位址紀錄
#此處查表將指令中的網路位置更改成實體位址
                print rule
                if "src" in rule:
                    rule['dl_src'] = mac_ip_list[rule['src']]
                    rule.pop('src')
                if "dst" in rule:
                    rule['dl_dst'] = mac_ip_list[rule['dst']]
                    rule.pop('dst')
            except SyntaxError:
                FirewallController._LOGGER.debug(
                        'invalid syntax %s', req.body)
                return Response(status=400)

            try:
                dps = self._OFS_LIST.get_ofs(switchid)
                vid = FirewallController._conv_toint_vlanid(vlan_id)
            except ValueError as message:
                return Response(status=400, body=str(message))

            msgs = []
            for f_ofs in dps.values():
                try:
#將規則的協定不管是 IPv4 or IPv6 都加進流程表中
                    if "dl_type" not in rule:
                        rule['dl_type'] = "IPv4"
                        msg = f_ofs.set_rule(rule, self.waiters, vid)
                        msgs.append(msg)
                        rule['dl_type'] = "IPv6"
                        msg = f_ofs.set_rule(rule, self.waiters, vid)
                        msgs.append(msg)
                    else:
                        msg = f_ofs.set_rule(rule, self.waiters, vid)
                        msgs.append(msg)
                except ValueError as message:
                    return Response(status=400, body=str(message))

            body = json.dumps(msgs)
            return Response(content_type='application/json', body=body)

        def _delete_rule(self, req, switchid, vlan_id=VLANID_NONE):
            try:
                ruleid = json.loads(req.body)
            except SyntaxError:
                FirewallController._LOGGER.debug('invalid syntax %s', req.body)
```

```
                return Response(status=400)

            try:
                dps = self._OFS_LIST.get_ofs(switchid)
                vid = FirewallController._conv_toint_vlanid(vlan_id)
            except ValueError as message:
                return Response(status=400, body=str(message))

            msgs = []
            for f_ofs in dps.values():
                try:
                    msg = f_ofs.delete_rule(ruleid, self.waiters, vid)
                    msgs.append(msg)
                except ValueError as message:
                    return Response(status=400, body=str(message))

            body = json.dumps(msgs)
            return Response(content_type='application/json', body=body)

        @staticmethod
        def _conv_toint_vlanid(vlan_id):
            if vlan_id != REST_ALL:
                vlan_id = int(vlan_id)
                if (vlan_id != VLANID_NONE and
                        (vlan_id < VLANID_MIN or
                        VLANID_MAX < vlan_id)):
                    msg = 'Invalid {vlan_id} value. Set [%d-%d]'
                        % (VLANID_MIN,
                        VLANID_MAX)
                    raise ValueError(msg)
            return vlan_id

        @staticmethod
        def packet_in_handler(msg):
            pkt = packet.Packet(msg.data)
            dpid_str = dpid_lib.dpid_to_str(msg.datapath.id)
            #FirewallController._LOGGER.info('dpid=%s:
            #                                 Blocked packet = %s',
            #                                 dpid_str, pkt)


class Firewall(object):

        _OFCTL = {ofproto_v1_0.OFP_VERSION: ofctl_v1_0,
                  ofproto_v1_2.OFP_VERSION: ofctl_v1_2,
                  ofproto_v1_3.OFP_VERSION: ofctl_v1_3}
```

```
def __init__(self, dp):
        super(Firewall, self).__init__()
        self.vlan_list = {}
        self.vlan_list[VLANID_NONE] = 0       # for VLAN=None
        self.dp = dp
        version = dp.ofproto.OFP_VERSION

        if version not in self._OFCTL:
            raise OFPUnknownVersion(version=version)

        self.ofctl = self._OFCTL[version]

def _update_vlan_list(self, vlan_list):
        for vlan_id in self.vlan_list.keys():
            if vlan_id is not VLANID_NONE and vlan_id not in vlan_list:
                del self.vlan_list[vlan_id]

def _get_cookie(self, vlan_id):
        if vlan_id == REST_ALL:
            vlan_ids = self.vlan_list.keys()
        else:
            vlan_ids = [vlan_id]

        cookie_list = []
        for vlan_id in vlan_ids:
            self.vlan_list.setdefault(vlan_id, 0)
            self.vlan_list[vlan_id] += 1
            self.vlan_list[vlan_id] &= ofproto_v1_3_parser.UINT32_MAX
            cookie = (vlan_id << COOKIE_SHIFT_VLANID) + \
                self.vlan_list[vlan_id]
            cookie_list.append([cookie, vlan_id])

        return cookie_list

@staticmethod
def _cookie_to_ruleid(cookie):
        return cookie & ofproto_v1_3_parser.UINT32_MAX

# REST command template
def rest_command(func):
        def _rest_command(*args, **kwargs):
            key, value = func(*args, **kwargs)
            switch_id = dpid_lib.dpid_to_str(args[0].dp.id)
            return {REST_SWITCHID: switch_id,
                        key: value}
        return _rest_command
```

```
@rest_command
def get_status(self, waiters):
    msgs = self.ofctl.get_flow_stats(self.dp, waiters)

    status = REST_STATUS_ENABLE
    if str(self.dp.id) in msgs:
        flow_stats = msgs[str(self.dp.id)]
        for flow_stat in flow_stats:
            if flow_stat['priority'] == STATUS_FLOW_PRIORITY:
                status = REST_STATUS_DISABLE

    return REST_STATUS, status

@rest_command
def set_disable_flow(self):
    cookie = 0
    priority = STATUS_FLOW_PRIORITY
    match = {}
    actions = []
    flow = self._to_of_flow(cookie=cookie, priority=priority,
                            match=match, actions=actions)

    cmd = self.dp.ofproto.OFPFC_ADD
    self.ofctl.mod_flow_entry(self.dp, flow, cmd)

    msg = {'result': 'success','details': 'firewall stopped.'}
    return REST_COMMAND_RESULT, msg

@rest_command
def set_enable_flow(self):
    cookie = 0
    priority = STATUS_FLOW_PRIORITY
    match = {}
    actions = []
    flow = self._to_of_flow(cookie=cookie, priority=priority,
                                    match=match, actions=actions)

    cmd = self.dp.ofproto.OFPFC_DELETE_STRICT
    self.ofctl.mod_flow_entry(self.dp, flow, cmd)

    msg = {'result': 'success', 'details': 'firewall running.'}
    return REST_COMMAND_RESULT, msg

@rest_command
def get_log_status(self, waiters):
    msgs = self.ofctl.get_flow_stats(self.dp, waiters)
```

```
        status = REST_STATUS_DISABLE
        if str(self.dp.id) in msgs:
            flow_stats = msgs[str(self.dp.id)]
            for flow_stat in flow_stats:
                if flow_stat['priority'] == LOG_FLOW_PRIORITY:
                    if flow_stat['actions']:
                        status = REST_STATUS_ENABLE


        return REST_LOG_STATUS, status


    @rest_command
    def set_log_disable(self, waiters=None):
        return self._set_log_status(False, waiters)

    @rest_command
    def set_log_enable(self, waiters=None):
        return self._set_log_status(True, waiters)

    def _set_log_status(self, is_enable, waiters):
        if is_enable:
            actions = Action.to_openflow(
                self.dp, {REST_ACTION: REST_ACTION_PACKETIN})
            details = 'Log collection started.'
        else:
            actions = []
            details = 'Log collection stopped.'

        cmd = self.dp.ofproto.OFPFC_ADD

        if waiters:
            msgs = self.ofctl.get_flow_stats(self.dp, waiters)

            if str(self.dp.id) in msgs:
                flow_stats = msgs[str(self.dp.id)]
                for flow_stat in flow_stats:
                    priority = flow_stat[REST_PRIORITY]
                    if (priority == STATUS_FLOW_PRIORITY
                            or priority == ARP_FLOW_PRIORITY):
                        continue
                    action = flow_stat[REST_ACTION]
                    if action == [
                            'OUTPUT:%d' % self.dp.ofproto.OFPP_NORMAL]:
                        continue

                    cookie = flow_stat[REST_COOKIE]
                    match = Match.to_mod_openflow(
                            flow_stat[REST_MATCH])
```

```
                        flow = self._to_of_flow(
                            cookie=cookie, priority=priority,
                            match=match, actions=actions)
                        self.ofctl.mod_flow_entry(self.dp, flow, cmd)
        else:
            # Initialize.
            flow = self._to_of_flow(cookie=0,
                        priority=LOG_FLOW_PRIORITY,
                        match={}, actions=actions)
            self.ofctl.mod_flow_entry(self.dp, flow, cmd)

        msg = {'result': 'success', 'details': details}
        return REST_COMMAND_RESULT, msg

    def set_arp_flow(self):
        cookie = 0
        priority = ARP_FLOW_PRIORITY
        match = {REST_DL_TYPE: ether.ETH_TYPE_ARP}
        action = {REST_ACTION: REST_ACTION_ALLOW}
        actions = Action.to_openflow(self.dp, action)
        flow = self._to_of_flow(cookie=cookie, priority=priority,
                                match=match, actions=actions)

        cmd = self.dp.ofproto.OFPFC_ADD
        self.ofctl.mod_flow_entry(self.dp, flow, cmd)

    @rest_command
    def set_rule(self, rest, waiters, vlan_id):
        msgs = []
        cookie_list = self._get_cookie(vlan_id)
        for cookie, vid in cookie_list:
            msg = self._set_rule(cookie, rest, waiters, vid)
            msgs.append(msg)
        return REST_COMMAND_RESULT, msgs

    def _set_rule(self, cookie, rest, waiters, vlan_id):
        priority = int(rest.get(REST_PRIORITY,
                                ACL_FLOW_PRIORITY_MIN))

        if (priority < ACL_FLOW_PRIORITY_MIN
                or ACL_FLOW_PRIORITY_MAX < priority):
            raise ValueError(
                'Invalid priority value. Set [%d-%d]'
                % (ACL_FLOW_PRIORITY_MIN,
                ACL_FLOW_PRIORITY_MAX))

        if vlan_id:
```

```
                rest[REST_DL_VLAN] = vlan_id

            match = Match.to_openflow(rest)
            if rest.get(REST_ACTION) == REST_ACTION_DENY:
                result = self.get_log_status(waiters)
                if result[REST_LOG_STATUS] == REST_STATUS_ENABLE:
                    rest[REST_ACTION] = REST_ACTION_PACKETIN
            actions = Action.to_openflow(self.dp, rest)
            flow = self._to_of_flow(cookie=cookie, priority=priority,
                                    match=match, actions=actions)

            cmd = self.dp.ofproto.OFPFC_ADD
            try:
                self.ofctl.mod_flow_entry(self.dp, flow, cmd)
            except:
                raise ValueError('Invalid rule parameter.')

            rule_id = Firewall._cookie_to_ruleid(cookie)
            msg = {'result': 'success',                    'details': 'Rule added. :
rule_id=%d' % rule_id}

            if vlan_id != VLANID_NONE:
                msg.setdefault(REST_VLANID, vlan_id)
            return msg

        @rest_command
        def get_rules(self, waiters, vlan_id):
            rules = {}
            msgs = self.ofctl.get_flow_stats(self.dp, waiters)

            if str(self.dp.id) in msgs:
                flow_stats = msgs[str(self.dp.id)]
                for flow_stat in flow_stats:
                    priority = flow_stat[REST_PRIORITY]
                    if (priority != STATUS_FLOW_PRIORITY
                            and priority != ARP_FLOW_PRIORITY
                            and priority != LOG_FLOW_PRIORITY):
                        vid = flow_stat[REST_MATCH].get(
                            REST_DL_VLAN, VLANID_NONE)
                        if vlan_id == REST_ALL or vlan_id == vid:
                            rule = self._to_rest_rule(flow_stat)
                            rules.setdefault(vid, [])
                            rules[vid].append(rule)

            get_data = []
            for vid, rule in rules.items():
                if vid == VLANID_NONE:
```

```
                vid_data = {REST_RULES: rule}
            else:
                vid_data = {REST_VLANID: vid, REST_RULES: rule}
            get_data.append(vid_data)

        return REST_ACL, get_data

    @rest_command
    def delete_rule(self, rest, waiters, vlan_id):
        try:
            if rest[REST_RULE_ID] == REST_ALL:
                rule_id = REST_ALL
            else:
                rule_id = int(rest[REST_RULE_ID])
        except:
            raise ValueError('Invalid ruleID.')

        vlan_list = []
        delete_list = []

        msgs = self.ofctl.get_flow_stats(self.dp, waiters)
        if str(self.dp.id) in msgs:
            flow_stats = msgs[str(self.dp.id)]
            for flow_stat in flow_stats:
                cookie = flow_stat[REST_COOKIE]
                ruleid = Firewall._cookie_to_ruleid(cookie)
                priority = flow_stat[REST_PRIORITY]
                dl_vlan = flow_stat[REST_MATCH].get(
                    REST_DL_VLAN, VLANID_NONE)

                if (priority != STATUS_FLOW_PRIORITY
                        and priority != ARP_FLOW_PRIORITY
                        and priority != LOG_FLOW_PRIORITY):
                    if ((rule_id == REST_ALL or rule_id == ruleid)
                    and (vlan_id == dl_vlan or vlan_id == REST_ALL)):
                        match = Match.to_mod_openflow(
                        flow_stat[REST_MATCH])
                        delete_list.append([cookie, priority, match])
                    else:
                        if dl_vlan not in vlan_list:
                            vlan_list.append(dl_vlan)

        self._update_vlan_list(vlan_list)

        if len(delete_list) == 0:
            msg_details = 'Rule is not exist.'
            if rule_id != REST_ALL:
```

```
                msg_details += ' : ruleID=%d' % rule_id
            msg = {'result': 'failure', 'details': msg_details}
        else:
            cmd = self.dp.ofproto.OFPFC_DELETE_STRICT
            actions = []
            delete_ids = {}
            for cookie, priority, match in delete_list:
                flow = self._to_of_flow(cookie=cookie, priority=priority,
                                        match=match, actions=actions)
                self.ofctl.mod_flow_entry(self.dp, flow, cmd)

                vid = match.get(REST_DL_VLAN, VLANID_NONE)
                rule_id = Firewall._cookie_to_ruleid(cookie)
                delete_ids.setdefault(vid, '')
                delete_ids[vid] += (
                    ('%d' if delete_ids[vid] == "else ',%d')
                     % rule_id)

            msg = []
            for vid, rule_ids in delete_ids.items():
                del_msg = {'result': 'success', 'details': 'Rule deleted. :
                    ruleID=%s' % rule_ids}
                if vid != VLANID_NONE:
                    del_msg.setdefault(REST_VLANID, vid)
                msg.append(del_msg)

        return REST_COMMAND_RESULT, msg

    def _to_of_flow(self, cookie, priority, match, actions):
        flow = {'cookie': cookie,
                'priority': priority,
                'flags': 0,
                'idle_timeout': 86400,
                'hard_timeout': 0,
                'match': match,
                'actions': actions}
        return flow

    def _to_rest_rule(self, flow):
        ruleid = Firewall._cookie_to_ruleid(flow[REST_COOKIE])
        rule = {REST_RULE_ID: ruleid}
        rule.update({REST_PRIORITY: flow[REST_PRIORITY]})
        rule.update(Match.to_rest(flow))
        rule.update(Action.to_rest(self.dp, flow))
        return rule
```

```
class Match(object):

    _CONVERT = {REST_DL_TYPE:
                    {REST_DL_TYPE_ARP: ether.ETH_TYPE_ARP,
                     REST_DL_TYPE_IPV4: ether.ETH_TYPE_IP,
                     REST_DL_TYPE_IPV6: ether.ETH_TYPE_IPV6},
                    REST_NW_PROTO:
                    {REST_NW_PROTO_TCP: inet.IPPROTO_TCP,
                     REST_NW_PROTO_UDP: inet.IPPROTO_UDP,
                     REST_NW_PROTO_ICMP: inet.IPPROTO_ICMP,
                     REST_NW_PROTO_ICMPV6: inet.IPPROTO_ICMPV6}}

    _MATCHES = [REST_IN_PORT,
                    REST_SRC_MAC,
                    REST_DST_MAC,
                    REST_DL_TYPE,
                    REST_DL_VLAN,
                    REST_SRC_IP,
                    REST_DST_IP,
                    REST_SRC_IPV6,
                    REST_DST_IPV6,
                    REST_NW_PROTO,
                    REST_TP_SRC,
                    REST_TP_DST]

    @staticmethod
    def to_openflow(rest):

        def __inv_combi(msg):
            raise ValueError('
                                Invalid combination: [%s]' % msg)

        def __inv_2and1(*args):
            __inv_combi('%s=%s and %s' % (args[0], args[1], args[2]))

        def __inv_2and2(*args):
            __inv_combi('%s=%s and %s=%s' % (
                args[0], args[1], args[2], args[3]))

        def __inv_1and1(*args):
            __inv_combi('%s and %s' % (args[0], args[1]))

        def __inv_1and2(*args):
            __inv_combi('%s and %s=%s' % (args[0], args[1], args[2]))

        match = {}
```

```
# error check
dl_type = rest.get(REST_DL_TYPE)
nw_proto = rest.get(REST_NW_PROTO)
if dl_type is not None:
    if dl_type == REST_DL_TYPE_ARP:
        if REST_SRC_IPV6 in rest:
            __inv_2and1(
                REST_DL_TYPE, REST_DL_TYPE_ARP,
                REST_SRC_IPV6)
        if REST_DST_IPV6 in rest:
            __inv_2and1(
                REST_DL_TYPE, REST_DL_TYPE_ARP,
                REST_DST_IPV6)
        if nw_proto:
            __inv_2and1(
                REST_DL_TYPE, REST_DL_TYPE_ARP,
                REST_NW_PROTO)
    elif dl_type == REST_DL_TYPE_IPV4:
        if REST_SRC_IPV6 in rest:
            __inv_2and1(
                REST_DL_TYPE, REST_DL_TYPE_IPV4,
                REST_SRC_IPV6)
        if REST_DST_IPV6 in rest:
            __inv_2and1(
                REST_DL_TYPE, REST_DL_TYPE_IPV4,
                REST_DST_IPV6)
        if nw_proto == REST_NW_PROTO_ICMPV6:
            __inv_2and2(
                REST_DL_TYPE, REST_DL_TYPE_IPV4,
                REST_NW_PROTO,
                REST_NW_PROTO_ICMPV6)
    elif dl_type == REST_DL_TYPE_IPV6:
        if REST_SRC_IP in rest:
            __inv_2and1(
                REST_DL_TYPE, REST_DL_TYPE_IPV6,
                REST_SRC_IP)
        if REST_DST_IP in rest:
            __inv_2and1(
                REST_DL_TYPE,
                REST_DL_TYPE_IPV6, REST_DST_IP)
        if nw_proto == REST_NW_PROTO_ICMP:
            __inv_2and2(
                REST_DL_TYPE, REST_DL_TYPE_IPV6,
                REST_NW_PROTO, REST_NW_PROTO_ICMP)
    else:
        raise ValueError(
                'Unknown dl_type : %s' % dl_type)
```

```
else:
    if REST_SRC_IP in rest:
        if REST_SRC_IPV6 in rest:
            __inv_1and1(REST_SRC_IP,
                                REST_SRC_IPV6)
        if REST_DST_IPV6 in rest:
            __inv_1and1(REST_SRC_IP,
                                REST_DST_IPV6)
        if nw_proto == REST_NW_PROTO_ICMPV6:
            __inv_1and2(
                REST_SRC_IP, REST_NW_PROTO,
                REST_NW_PROTO_ICMPV6)
        rest[REST_DL_TYPE] = REST_DL_TYPE_IPV4
    elif REST_DST_IP in rest:
        if REST_SRC_IPV6 in rest:
            __inv_1and1(REST_DST_IP,
                                REST_SRC_IPV6)
        if REST_DST_IPV6 in rest:
            __inv_1and1(REST_DST_IP,
                                REST_DST_IPV6)
        if nw_proto == REST_NW_PROTO_ICMPV6:
            __inv_1and2(
                REST_DST_IP, REST_NW_PROTO,
                REST_NW_PROTO_ICMPV6)
        rest[REST_DL_TYPE] = REST_DL_TYPE_IPV4
    elif REST_SRC_IPV6 in rest:
        if nw_proto == REST_NW_PROTO_ICMP:
            __inv_1and2(
                REST_SRC_IPV6, REST_NW_PROTO,
                REST_NW_PROTO_ICMP)
        rest[REST_DL_TYPE] = REST_DL_TYPE_IPV6
    elif REST_DST_IPV6 in rest:
        if nw_proto == REST_NW_PROTO_ICMP:
            __inv_1and2(
                REST_DST_IPV6, REST_NW_PROTO,
                REST_NW_PROTO_ICMP)
        rest[REST_DL_TYPE] = REST_DL_TYPE_IPV6
    else:
        if nw_proto == REST_NW_PROTO_ICMP:
            rest[REST_DL_TYPE] =
            REST_DL_TYPE_IPV4
        elif nw_proto == REST_NW_PROTO_ICMPV6:
            rest[REST_DL_TYPE] =
            REST_DL_TYPE_IPV6
        elif nw_proto == REST_NW_PROTO_TCP or \
                nw_proto == REST_NW_PROTO_UDP:
            raise ValueError(
```

```
                                'no dl_type was specified')
                    else:
                        raise ValueError(
                            'Unknown nw_proto: %s' % nw_proto)

        for key, value in rest.items():
            if key in Match._CONVERT:
                if value in Match._CONVERT[key]:
                    match.setdefault(key,
                        Match._CONVERT[key][value])
                else:
                    raise ValueError(
                                'Invalid rule parameter. :
                                key=%s' % key)
            elif key in Match._MATCHES:
                match.setdefault(key, value)

        return match

    @staticmethod
    def to_rest(openflow):
        of_match = openflow[REST_MATCH]

        mac_dontcare = mac.haddr_to_str(mac.DONTCARE)
        ip_dontcare = '0.0.0.0'
        ipv6_dontcare = '::'

        match = {}
        for key, value in of_match.items():
            if key == REST_SRC_MAC or key == REST_DST_MAC:
                if value == mac_dontcare:
                    continue
            elif key == REST_SRC_IP or key == REST_DST_IP:
                if value == ip_dontcare:
                    continue
            elif key == REST_SRC_IPV6 or key ==REST_DST_IPV6:
                if value == ipv6_dontcare:
                    continue
            elif value == 0:
                continue

            if key in Match._CONVERT:
                conv = Match._CONVERT[key]
                conv = dict((value, key) for key,
                value in conv.items())
                match.setdefault(key, conv[value])
            else:
```

```
                match.setdefault(key, value)

        return match

    @staticmethod
    def to_mod_openflow(of_match):
        mac_dontcare = mac.haddr_to_str(mac.DONTCARE)
        ip_dontcare = '0.0.0.0'
        ipv6_dontcare = '::'

        match = {}
        for key, value in of_match.items():
            if key == REST_SRC_MAC or key == REST_DST_MAC:
                if value == mac_dontcare:
                    continue
            elif key == REST_SRC_IP or key == REST_DST_IP:
                if value == ip_dontcare:
                    continue
            elif key == REST_SRC_IPV6 or key == REST_DST_IPV6:
                if value == ipv6_dontcare:
                    continue
            elif value == 0:
                continue

            match.setdefault(key, value)

        return match


class Action(object):

    @staticmethod
    def to_openflow(dp, rest):
        value = rest.get(REST_ACTION, REST_ACTION_ALLOW)

        if value == REST_ACTION_ALLOW:
            out_port = dp.ofproto.OFPP_NORMAL
            action = [{'type': 'OUTPUT',
                       'port': out_port}]
        elif value == REST_ACTION_DENY:
            action = []
        elif value == REST_ACTION_PACKETIN:
            out_port = dp.ofproto.OFPP_CONTROLLER
            action = [{'type': 'OUTPUT', 'port': out_port, 'max_len': 128}]
        else:
            raise ValueError('Invalid action type.')
```

```
        return action

    @staticmethod
    def to_rest(dp, openflow):
        if REST_ACTION in openflow:
            action_allow = 'OUTPUT:%d' % dp.ofproto.OFPP_NORMAL
            if openflow[REST_ACTION] == [action_allow]:
                action = {REST_ACTION: REST_ACTION_ALLOW}
            else:
                action = {REST_ACTION: REST_ACTION_DENY}
        else:
            action = {REST_ACTION: 'Unknown action type.'}

        return action
```

# 附錄三　Authentication Module 之程式碼

以下為本論文 Authentication Module 之程式碼：

```
from flask import Flask, render_template, redirect, url_for, request
from flask import jsonify
from uuid import getnode as get_mac
import pycurl, json
import time
import threading

app = Flask(__name__)

@app.route('/nowelcome')
def nowelcome():
	return render_template("nowelcome.html")

@app.route('/welcome')
def welcome():
	return render_template("welcome.html")

@app.route('/', methods=['GET', 'POST'])
def login():

	#user name and password
	user_acc = {"lab":"802.16", "solomon":"rfc3261", "jjchen":"rfc7668",
"ccyang":"rfc6334"}
	print ('IP:'+request.remote_addr)
	error = None
	if request.method == 'POST':
		if request.form['username'] in user_acc.keys():
			if request.form['password']==user_acc[request.form['username']]:
				print "ALL right"
				getIP(request.remote_addr)
				if "2001:e10" in request.remote_addr :
					addRule6(request.remote_addr)
					return redirect(url_for('welcome'))

				else:				addRule(request.remote_addr[7:])
					return redirect(url_for('welcome'))
			else:
				print "error password"
				error = ('Invalid    password. Please try again.')
		else:
			print "error acount"
			error = ('Invalid    username.
```

```
                                    Please try again.')
        return render_template('login.html', error=error)


#紀錄曾經登錄過認證伺服器的位址
def getIP(add):
        f = open('record.log','a')
        f.write(add+'\n')
        f.close
        print("IP:"+add+" has been written")


def addRule(add):
        # switch ID : 00001c6f65d30e24
        #如果要操控所有的 switch 可以使用 all 取代 switch ID
        # Example: http://10.21.21.179:8080/firewall/rules/all
        #以下為指定特定的 switch，switch IP 位址是：10.21.21.179
        ryu_ovs2_url = "http://10.21.21.179:8080/firewall/rules/00001c6f65d30e24"
        data_1 = json.dumps({"src": add, "priority": "500"})
        data_2 = json.dumps({"dst": add, "priority": "500"})
        c = pycurl.Curl()
        c.setopt(pycurl.URL, ryu_ovs2_url)
        c.setopt(pycurl.POST, 1)
        c.setopt(pycurl.POSTFIELDS, data_1)
        c.perform()
        print()
        c = pycurl.Curl()
        c.setopt(pycurl.URL, ryu_ovs2_url)
        c.setopt(pycurl.POST, 1)
        c.setopt(pycurl.POSTFIELDS, data_2)
        c.perform()
        print()


def addRule6(add):
        ryu_ovs2_url = "http://10.21.21.179:8080/firewall/rules/00001c6f65d30e24"
        data_1 = json.dumps({"src": add, "priority": "500"})
        data_2 = json.dumps({"dst": add, "priority": "500"})
        print data_1
        c = pycurl.Curl()
        c.setopt(pycurl.URL, ryu_ovs2_url)
        c.setopt(pycurl.POST, 1)
        c.setopt(pycurl.POSTFIELDS, data_1)
        c.perform()
        print()
        c = pycurl.Curl()
        c.setopt(pycurl.URL, ryu_ovs2_url)
```

```
        c.setopt(pycurl.POST, 1)
        c.setopt(pycurl.POSTFIELDS, data_2)
        c.perform()
        print()



if __name__ == '__main__':

        #app.run(host='0.0.0.0', port=80, threaded=True)
        app.run(host='::', port=80, threaded=True)
        if count_list[address_list.index(
        request.remote_addr)] > 4:
                redirect(url_for('nowelcome'))
```