國立暨南國際大學資訊工程學系

碩士論文

以軟體定義網路實現具有動態 VLAN 分配的網路隔離

Network Isolation with Dynamic VLAN Assignment using Software-Defined Networking

指導教授：吳坤熹博士

研究生：吳騰然

中華民國 112 年 6 月

doi:10.6837/ncnu202300216

國立暨南國際大學碩士論文考試審定書

資訊工程學系（研究所）
研究生 吳騰然 所提之論文

Network Isolation with Dynamic VLAN Assignment using Software-Defined Networking

以軟體定義網路實現具有動態 VLAN 分配的網路隔離

經本委員會審查，符合碩士學位論文標準。

學位考試委員會

劉一宇 委員兼召集人

張碩杰 委員

吳坤熹 委員

中華民國 112 年 6 月 30 日

doi:10.6837/ncnu202300216

致謝詞

兩年碩士班的生活時光飛逝，在碩士班生活中最感謝的人就是吳坤熹老師。最一開始的我連論文要看什麼、要怎麼找重點都還不知道的情況下，一路上都是老師給予我許多的幫助與鼓勵，為我講解如何找好的論文、怎麼樣去從一篇論文內找出重點，讓我觀看論文的熟練度飛快的提升，現在可以至少在一周內找到兩、三篇不錯的論文並掌握重點，也讓我體會到當研究生的過程與經歷。甚至在我碩二時期，在論文題目的構想階段遇到了瓶頸，老師也提供了許多的建議與想法，最後在與老師的積極討論下，才能成功訂下現在這個題目，這些都是老師細心指導下的成果。

在實驗室中的訓練，老師更是教會了我一個工程師除了技術強以外，表達能力也是非常重要的。所以在日常的 Meeting 上，不管是報告也好問問題也好，需要清楚地表達內容才能讓台下的聽眾能夠理解才是最重要的，這也是我當初選擇指導老師的初心。表達能力這方面也都讓我目前在中國信託實習時能夠展現出來，不管是Mentor、科長詢問我專案上的內容以及專業知識都可以清楚表達。

在體能上，老師也期望每個人都有可以擁有運動的習慣，不管是跑步、自行車或者是其它運動，這些運動都可以讓我們在研究之外，可以有其它地方可以放鬆自己的心情。這也讓我進入到中信實習後，每週會挑一天去健身房運動，也與同事去羽球場一起打球，同時也更了解同事之間的個性、想法…。

此外，我也感謝家人們，在聽到我選擇就讀研究所時，換來的是一個積極、正面的支持。在我拖著疲憊的身軀回家休息時，家人也總是支持我、鼓勵我，讓我能有動力繼續完成研究所的學業。

另外，我想告訴我自己，研究所畢業不是終點，而是一個新的開始，就像與老師最後一次路跑時跟我講的一句話 “人生就像馬拉松，永遠都是向前跑著，沒有終點”，這句話在我的心中印象深刻。也讓我有幸在實習期間體會到，即使在感覺自己已經學了許多的專業知識，但是在企業實際運作上仍然有許多是我們尚未知道的

doi:10.6837/ncnu202300216

知識，讓我知道自己必須一直往前，學習更多的知識。

doi:10.6837/ncnu202300216

論文名稱：以軟體定義網路實現具有動態 VLAN 分配的網路隔離
校院系：國立暨南國際大學科技學院資訊工程學系　　頁數：64
畢業時間：中華民國 112 年 6 月　　學位別：碩士
研究生：吳騰然　　指導教授：吳坤熹博士

摘要

傳統的網路交換機 (Switch) 若要設置 VLAN (Virtual Local Area Network)，網路管理者需要使用命令行 (Command Line) 或 Web Interface 去一台一台去設定，這樣會造成網路管理者無法集中管理 VLAN，以至於會花費許多時間去逐一觀看網路設備的 VLAN。而在使用者設備的部分，網路管理者通常是指定使用者座位的網路埠 (Port) 來分配 VLAN，若使用者移動至其它位置，必須隨之重新設置使用者的 VLAN，容易造成網路管理者因為複雜的手動配置而設定錯誤。

本篇論文將使用軟體定義網路 (Software-Defined Networking, SDN) 的架構，來解決上述的問題。VLAN 統一管理的問題，我們將使用 SDN 創建 List 表來去統一管理 VLAN ID，這樣使網路管理者能夠快速地在 List 表中找到每台設備的 VLAN ID。在動態 VLAN 分配的部分，採用辨識使用者電腦的網路卡 MAC (Media Access Control) 位址來分配 VLAN ID，這樣使用者不管移動到哪個房間，只要連上網路，即可分配到對應的 VLAN，自動達到網路隔離的效果。

關鍵字：動態 VLAN 分配、軟體定義網路、網路隔離、OpenFlow

doi:10.6837/ncnu202300216

Title of Thesis: Network Isolation with Dynamic VLAN Assignment using Software-Defined Networking

Name of Institute: Department of Computer Science and Information Engineering, College of Science and Technology, National Chi Nan University Pages: 64

Graduation Time : 06/2022 Degree: Master

Student Name: Teng-Jan Wu Advisor Name: Dr. Quincy Wu

Abstract

To configure VLAN (Virtual Local Area Network) on traditional network switches, network managers need to use command-line or web interface to configure those devices one by one, which makes it time-consuming for network managers to manage VLAN, because they spend a lot of time to configure individual network device. If the users always sit at a fixed location, the aforementioned process, although tedious, is still acceptable. However, if the user moves to another room and try to connect his computer through the network port in that room, network managers must assign the corresponding network port to a VLAN which the user belongs to. When lots of people are moving inside an organization, these tedious configuration tasks will ultimately become prone to error, due to complicated manual configuration.

In this thesis, we use the Software-Defined Networking (SDN) architecture to solve the above problem. For unified VLAN management, we use SDN to create a list table to manage VLAN ID uniformly, so that network managers can quickly find the VLAN ID of each device in the list table. For dynamic VLAN assignment, the MAC (Media Access Control) address of the user's computer is used to find the mapped VLAN ID. This ensures that the VLAN is automatically configured for the user's device whenever his device moves to a new location. Through this automatic configuration of VLANs, the effect of

doi:10.6837/ncnu202300216

network isolation is achieved.

Keywords: Dynamic VLAN Assignment, Network Isolation, OpenFlow, Software-Defined Networking

doi:10.6837/ncnu202300216

目次

doi:10.6837/ncnu202300216

doi:10.6837/ncnu202300216

表目次

doi:10.6837/ncnu202300216

圖目次

doi:10.6837/ncnu202300216

doi:10.6837/ncnu202300216

第一章 緒論

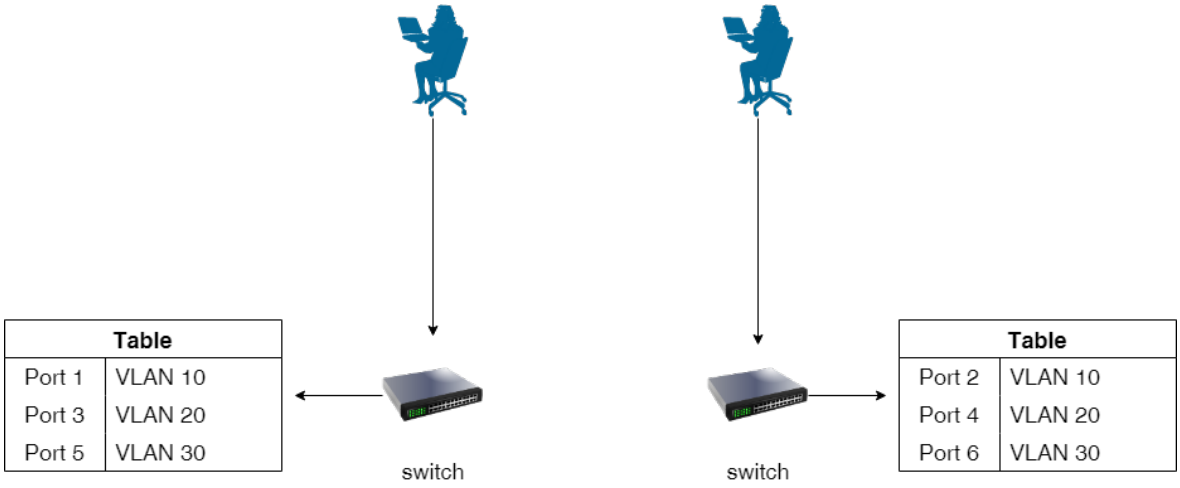
第一節 研究動機

自 1998 年 IEEE 組織定義了 802.1Q 的標準規範[1]，也就是 VLAN (Virtual Local Area Network) 這項技術，之後 VLAN 便作為可以提高網路效率和網路安全的方法而受到廣泛的使用。VLAN 技術將網路交換機 (Network Switch) 的每個網路埠設定一個 VLAN ID，並且只有與其相同的 VLAN ID 才可以互通，反之不同者則不通。VLAN 的技術可以用來將大型的 LAN (Local Area Network) 網路分割成更小的子網路，並進行邏輯網路的規劃、分區[2]。舉個例子來說：某一家企業的部門有 IT、HR 和 PM 部門，每個部門的伺服主機都接在網路交換機的網路埠上。網路管理者透過命令行 (Command Line) 或 網頁介面(Web Interface) 連線到網路交換機，並根據部門的主機網路埠設定 VLAN ID。例如，我們可以設定 IT 是 VLAN 10、HR 是 VLAN 20、PM 是 VLAN 30，這使得同個部門的主機之間可以互通，不同部門之間則會因為 VLAN ID 的不同而不互通。這樣不但可以隔離其它部門之間的流量，達到網路隔離 (Network Isolation) 的效果，也可以讓網路中的異常狀況（如 loop）限制在一個部門，而不致影響到整個企業網路。這也是為甚麼 VLAN 從發明至今為止已經過了 30 餘年，但依舊是企業、醫院…廣泛使用的技術[3]。

但從上述得知，網路管理者若要設定 VLAN 就必須要針對主機接上網路交換機的網路埠進行設定，這樣網路管理者會遇到一些在管理與設定上的問題。第一、因為網路交換機與主機數量眾多，網路管理者較難統一管理每台網路交換機 VLAN，若主機要更動時往往都需要花費大量時間去逐一查詢每台網路交換機 VLAN 資訊 (如圖一所示) 才可以找到該台主機對應的 VLAN。第二、若是每個部門的人員在不同時段需要進入某一間會議室使用網路，而在會議中需要連回自己的主機或網路硬碟去存取資料，這時會議室的網路埠就要配合不同時段使用會議室的

doi:10.6837/ncnu202300216

部門來做調整。如此一來，網路管理在設定上會非常的麻煩，常常會需要去手動修改，進而增加網路管理者配置錯誤的機率。



圖一 傳統網路交換機查詢/設定 VLAN 方式

doi:10.6837/ncnu202300216

第二節 研究目的

本篇論文將使用軟體定義網路 (Software-Defined Networking, SDN) 的架構來控制網路交換機，目的是為了來解決上述提到的 VLAN 統一管理、靈活性，以及為使用者分配 VLAN ID 的問題。論文中所提出的系統能達到以下成效：

1. 使用 SDN 創建一個列表 (List) 來統一管理 VLAN
2. 使用者的筆記型電腦（以下簡稱「筆電」）或主機接上網路交換機的網路埠，依據 MAC 位址分配 VLAN ID
3. 不須限定辦公位置，只須接入網路埠即可分配到對應的 VLAN ID

第一點是希望讓網路管理者藉由 SDN 即可管理網路交換機的優勢，達到統一管理 VLAN 的效果，這樣網路管理者只需要在 SDN 創建列表，之後就可以一目了然知道每台網路交換機 VLAN 的狀況，並且可以視需要直接更改 VLAN。第二、三點本篇論文將使用 SDN 的框架 Ryu 來撰寫動態 VLAN 分配的程式，來辨別使用者的筆電或主機 MAC 位址，據以分配 VLAN ID 的動作。這樣一來使用者就不用顧慮位置的問題，只需要接入網路交換機的網路埠，封包就會送往 SDN 來辨別筆電的 MAC 位址來分配對應的 VLAN ID，讓使用者帶著筆電無論身處何處，VLAN ID 都會自動根據筆電的 MAC 位址進行分配。

doi:10.6837/ncnu202300216

第三節 論文架構

本篇論文的內容將分為七章，第一章是研究動機與研究目的；第二章為本篇論文的研究背景；第三章為文獻回顧；第四章為網路架構與系統流程；第五章為實驗所需要安裝的套件說明與執行方式；第六章為實驗的結果；最後是本論文的總結與未來展望。

doi:10.6837/ncnu202300216

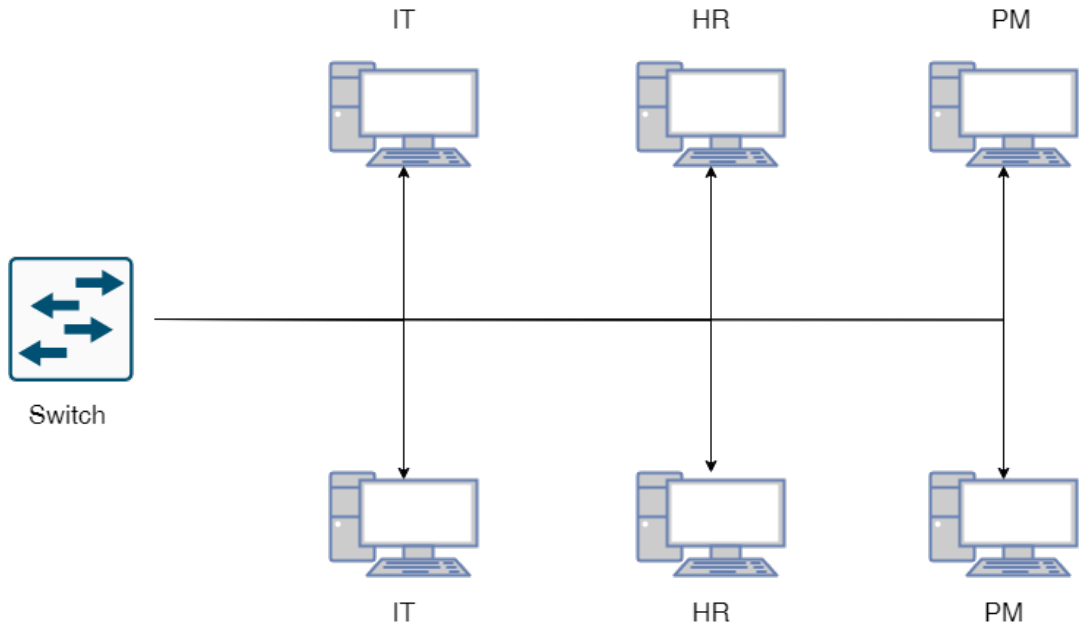
第二章 研究背景

本章將講述我們在實現 VLAN 統一管理、動態 VLAN 分配所需的技術。第一節介紹 VLAN，讓讀者可以充分地了解 VLAN 的基礎知識。第二節介紹 SDN，講述為何 SDN 可以控制多台網路交換機，其技術與原理都將在第二節講述。第三節介紹 OpenFlow，作為 SDN 與網路交換機間的通訊介面，我們會在本節講述有關於 OpenFlow 的基礎知識。

第一節 VLAN

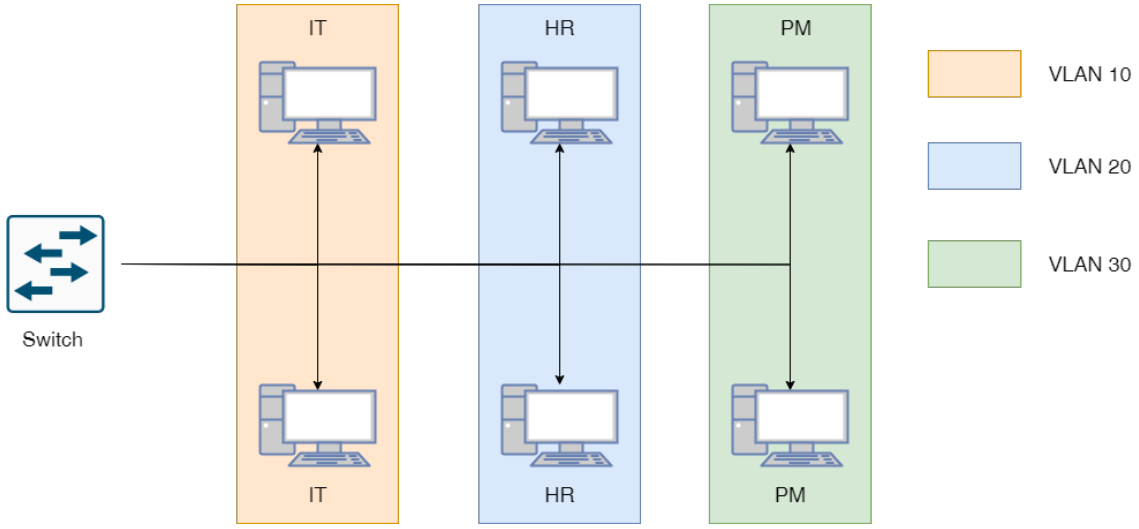
隨著網際網路的發展，人們可以容易取得 3C 產品電腦、筆記型電腦、手機…可以自由自在地上網，但隨之而來的是網路安全的問題。我們可能會遇到下載的檔案帶有病毒，又或者是信件收到釣魚郵件，引誘我們點擊連結或下載檔案，進而導致電腦受到病毒的感染。在企業內部的主機也不例外，某些企業內部的主機為了方便，網路交換機只設置預設 VLAN，讓所有公司的部門的設備都可以互相存取。以圖二為例，在未設置 VLAN 的情況下，HR、IT、PM 部門都可以互相存取，可以說是非常的方便。但這在網路安全上會產生一些問題，例如說：我們知道 IT 部門常因開發軟體、或是測試新系統的需求，需要進行有關網路的實驗。實驗過程中假如遇到使網路交換機產生 Loop 的狀況，形成廣播風暴（Broadcast Storm），這將會導致其它部門 (HR、PM) 的網路運作都受到影響，沒有辦法正常的進行作業。另一個情況，若很不幸地駭客利用社交工程、釣魚郵件的手法來針對公司的員工進行攻擊，其中若有一位員工點擊釣魚郵件或撿取來路不明的 USB 插入公司的筆電，這將使得駭客利用藏在裡面的病毒可以遠端控制該員工的電腦，並藉由此電腦存取其它部門的資源 (SMB、Database Server)以竊取資料[4]。

doi:10.6837/ncnu202300216



圖二 未設置 VLAN 的拓樸圖

為了解決上面的問題，就有了 VLAN 這項技術的存在。倘若網路交換機針對部門來做 VLAN 的設置，如圖三所示，將各個部門配置不同了 VLAN ID，例如 IT 是 VLAN 10、HR 是 VLAN 20、PM 是 VLAN 30。由於相同的 VLAN ID 才能夠相互存取，不同的 VLAN ID 要互相存取是不行的，這樣的設置將可達到「網路隔離」的效果，讓前述的 Loop、安全性問題限制在特定的部門內，而不致影響其他部門。



圖三 設置 VLAN 的拓樸圖

但是，在設備要進行 VLAN 設定時，傳統上網路管理者都是到設備前以

doi:10.6837/ncnu202300216

Console 模式連線，使用命令行（Command-Line Interface，簡稱 CLI）或者是 Web Interface 的功能，來達到設定或查詢 VLAN 的功能。雖然這種靜態設置的方式已經行之多年，是許多網路管理者的作法，但還是有諸多不便，例如:

1. VLAN 統一問題：在傳統網路設備設定上，一次只能設定一台設備所擁有的網路埠。若要查詢第二台、第三台的 VLAN，就必須要另行連線，逐一去對網路設備進行查詢。這樣子不但會花費網路管理者大量的時間去做查詢，還有可能會因為過多的設備忙中有錯，而導致 VLAN 配置錯誤。
2. VLAN 設置靈活度：在傳統網路設備若要進行設定 VLAN，需要逐一對網路設備改變 VLAN 設置，來針對部門、使用者的座位進行配置。但若需要更改設定時，例如：不同部門的人需要使用會議室的網路埠，這使得網路管理者就必須要到交換機前查詢網路埠的配置狀態，然後找到會議室的網路埠進行修改，這樣容易使得網路管理者可能會因為複雜的網路配置而導致錯誤配置。並且網路設備若是來自不同廠商，則設定 VLAN 的方式也會隨之不同，考驗著網路管理者對於不同廠商的網路設備設定的熟悉度。

綜合上面的幾點來說，這無疑是現在網路管理者使用靜態設置的缺點，為了克服上述靜態設置 VLAN 這些問題。本篇論文將會在下一節介紹軟體定義網路(Software-Defined Networking, SDN)，利用軟體定義網路的優勢來對上述提到靜態設置 VLAN 的問題，一一解決。下一小節介紹 VLAN Port 的類型，包含 Access Port 與 Trunk Port，說明網路交換機 Access Port 和 Trunk Port 的運作原理與功能。

第一小節 Access Port

Access Port 用於連接網路交換機與單一設備或單一 VLAN。Access Port 主要用於連接終端設備，如: 個人電腦、印表機或網路電話…。Access Port 只能將封包傳送到指定的 VLAN。當接在 Access Port 上的電腦發出一個封包時，交換機會將其轉

doi:10.6837/ncnu202300216

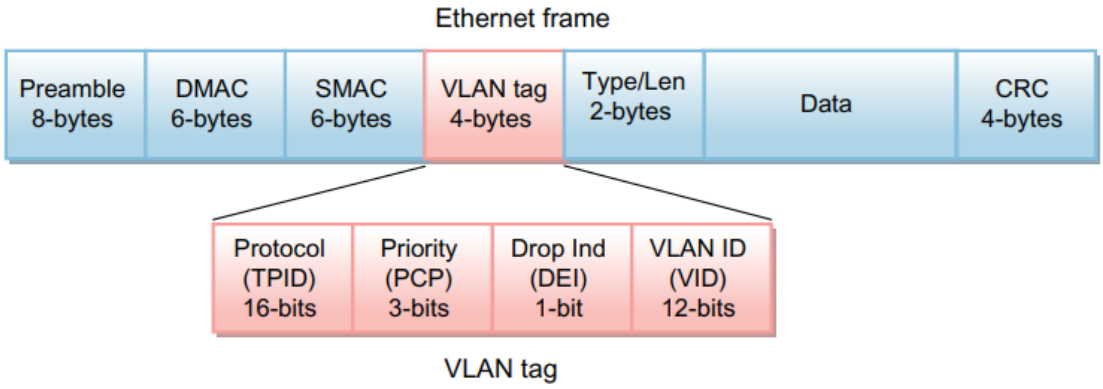
送到同一台交換機上屬於相同 VLAN ID 的其他 Access Port 上。

第二小節 Trunk Port

Trunk Port 用於網路交換機之間的連接。為了讓多個 VLAN 的資料封包可以共用一個 Trunk Port 傳輸，在封包的標頭中會新增一個四位元組的 VLAN Tag (如圖四所示)，VLAN Tag 欄位說明如下 ：

- TPID (Tag Protocol Identifier) : 存放內容為 0x8100，來表示這是一個 IEEE 802.1Q 標示的封包
- PCP (Priority Code Point) : 封包優先級別，預設為 0
- DEI (Drop Eligible Indicator) : 預設為 0，值為 1 時表示遇到壅塞時可以 Drop Frame
- VID (VLAN Identifier，簡稱 VLAN ID) : VLAN ID 最多為 4094 個 (1～4094)

VLAN ID 用於區分資料封包所屬的 VLAN，以便在不同網路設備之間進行 VLAN 隔離。目的地的網路交換機收到封包後，會移除 VLAN Tag，並轉送到對應該 VLAN ID 的所有 Access Port。若有對應此 VLAN ID 的其他 Trunk Port，則無須移除 VLAN Tag，直接轉送。

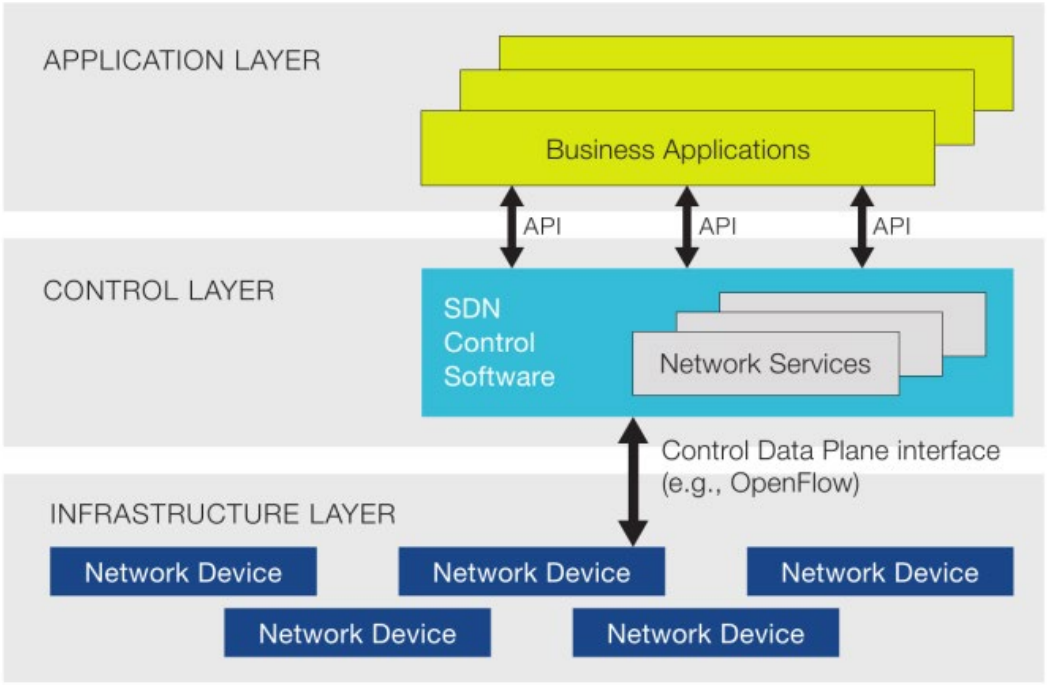


圖四 VLAN Tag 格式[5]

doi:10.6837/ncnu202300216

第二節 軟體定義網路 (SDN)

軟體定義網路（Software-Defined Networking），簡稱為 SDN。SDN 的網路架構如圖五所示，主要透過控制層 (Control Layer) 與基礎設施層 (Infrastructure Layer) 進行分離來實現[6]。控制層主要透過撰寫腳本（Script）來負責管理整個網路服務，控制層會與基礎設施層的網路設備（如 Switch、Router）進行通訊，並將網路設備接收到的封包傳送到控制層的 SDN Controller，然後根據我們所撰寫的腳本，來決定網路封包的去向。基礎設施層是一般所謂的網路設備，像是 Switch、Router 等。網路設備將網路封包送到 SDN Controller 去做判斷後，依判斷結果將封包送至目的地。

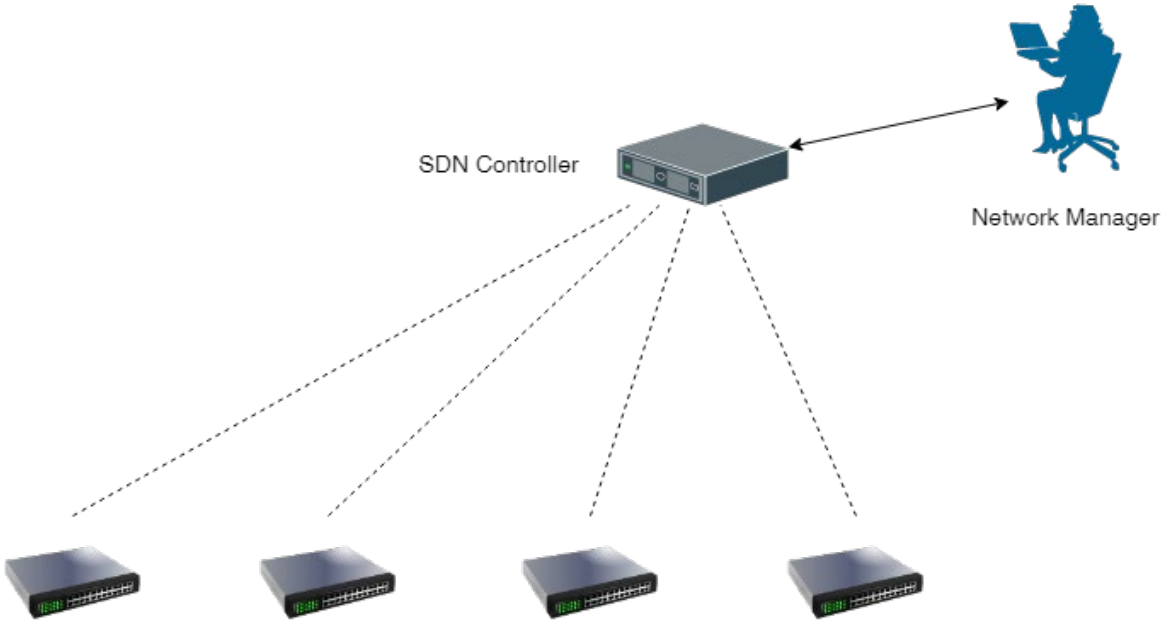


圖五 SDN 網路架構圖[7]

傳統網路設備與 SDN 相比，傳統網路設備的控制層和基礎設施層是結合在一起的[8]，在控制邏輯上都是硬體裝置。由於每個網路設備都是獨立運作，網路設備還因為不同廠商會有不同的指令格式，這也造成了網路管理者需要熟記每個廠牌的指令，還需要逐一去設定每台網路設備，配置錯誤的機率也會隨之上升。SDN 則是控

doi:10.6837/ncnu202300216

制層與基礎施設層分開的，控制層從硬體控制變成軟體化的方式來控制基礎施設層。如圖六所示，網路管理者可以透過撰寫軟體來控制 SDN。而 SDN Controller 像是一個指揮官一樣，可以管理底下的網路設備。這使得 SDN 可以擁有出色的掌控能力。傳統網路設備必須逐一去做設定，但 SDN 具有可以控制多台網路設備的能力，不同的廠牌的網路設備也能以相同的腳本去控制，以減輕網路管理者需要熟悉多廠牌網路設備的設定指令與發生錯誤配置的壓力。SDN 可以擁有自動化的優點，在傳統網路設備我們只能夠逐一手動設定，在 SDN 上由於是以軟體作為實現的，只需撰寫程式就可進行自動化的管理，讓網路管理者減少手動設定的需求。

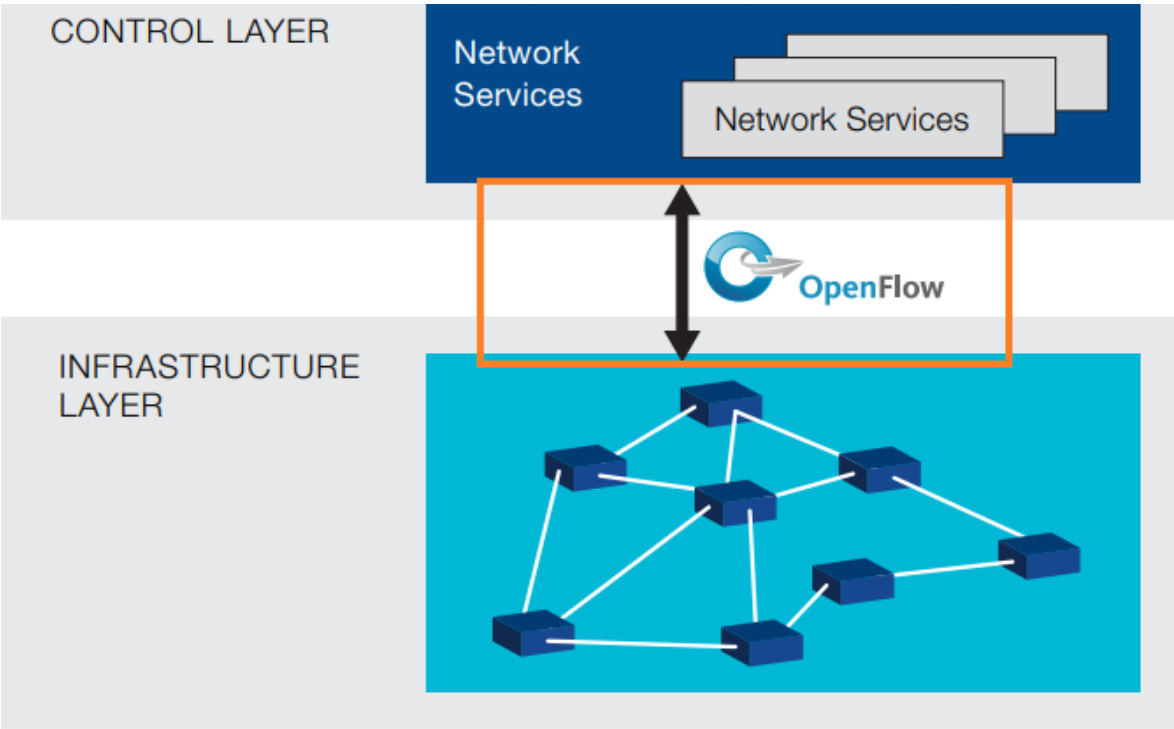


圖六 網路管理者使用 SDN Controller 控制網路設備示意圖

doi:10.6837/ncnu202300216

第三節 OpenFlow

OpenFlow 協定是由史丹佛大學 (Stanford University) 研究人員所提出[9]，OpenFlow 是為了實現 SDN 控制層與基礎設施層之間的溝通協定而設計出來的 (如圖七棕色框所示)。基礎設施層的網路設備收到封包後，透過 OpenFlow 協定送往給 SDN Controller。這使得 OpenFlow 協定可以實現統一管理基礎設施層的網路設備，而網路封包的處理規則由 SDN Controller 統一決定。



圖七 SDN architecture[10]

OpenFlow 作為現代學術、企業人員都喜歡用的協定，根據 Fei Hu[11]所述，相較於其他協定，與 OpenFlow 最接近 SDN 通訊協定標準的是 ForCES (Forwarding and Control Element Separation)。ForCES 是由 IETF 所定義的標準，ForCES 的架構是把控制層與基礎設施層分別獨立開來，兩個平面都有各自獨立的模型[12]，控制層主要負責網路的規則；基礎設施層主要負責網路設備轉發的規則，兩個平面需要各自撰寫程式、訂定規則。

OpenFlow 以集中式管理以及處理封包的靈活性相較之下略具優勢。在集中式管

doi:10.6837/ncnu202300216

理部分，SDN Controller 可以透過使用 OpenFlow 協定實現對基礎設施層的網路設備，達到集中式管理的功能。而處理封包的靈活性則是網路管理者能夠統一在控制層進行撰寫規則，對封包進行轉發與處理。其缺點為擴展性以及單點故障問題，當網路設備增加，處理的封包也就越多，這使得 SDN Controller 需要接收許多封包，而導致處理封包的速度變慢。單點故障則是因 OpenFlow 集中管理，如此在 SDN Controller 故障時，會使得基礎設施層的網路設備停止服務。

ForCES 在優勢上面則是分散式控制與可擴展性高，ForCES 協定採用分散式控制架構，封包轉發功能分散在網路設備中，可以降低 SDN Controller 的負擔和單點故障的風險。在擴展性部分，ForCES 協定支援較大規模的網路架構，因為封包轉發功能分散在各個設備中，能夠更好地應對擴展性需求。但其缺點為複雜度高與設備硬體需求，ForCES 需要各自撰寫控制層與基礎設施層的程式碼，這顯然要考驗網路管理者的程式設計能力，同時具有開發、維護成本高的問題。在網路設備需求部分，由於封包轉發功能分散在各個設備中，對於設備的需求(記憶體、CPU)會顯得更高。

綜合上述所敘，雖然 ForCES 可以支援大型網路架構是非常吸引企業使用的方案，但 ForCES 現在還尚未有明確的框架、控制層、基礎設施層的規則，使用的難度比起 OpenFlow 相對較複雜，因為控制層與基礎設施層都需要各自撰寫程式。導致 ForCES 到至今還尚未被學術、企業廣泛使用，這也是為什麼現在 OpenFlow 依然是最熱門的首選。

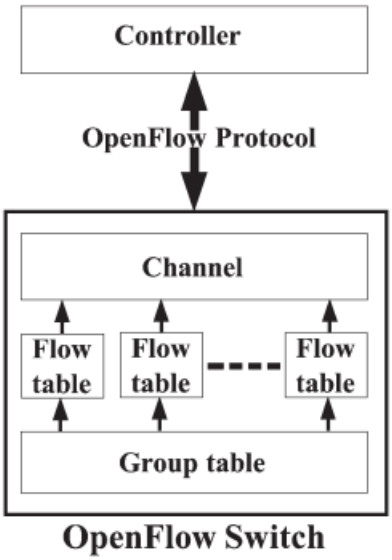
OpenFlow 的架構如圖八所示，OpenFlow 交換機的架構為以下部分 :

1. Group Table : Group Table 內包含多個 Flow Table 規則。
2. Flow Table : 當接收到封包時，用於判斷 OpenFlow 交換機封包轉發至哪個埠口。
3. Channel : 主要是將 OpenFlow 交換機連接到 Controller，並可以實現遠端連線至 Controller。

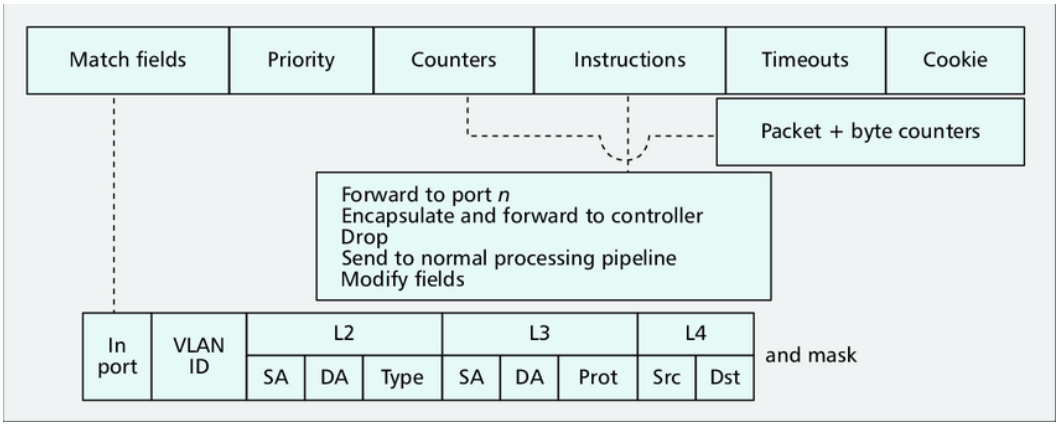
doi:10.6837/ncnu202300216

4. OpenFlow Protocol : Controller 與 OpenFlow 交換機之間的通訊協定。

舉例來說 : 當有封包送到 OpenFlow 交換機時，先檢查該封包是否有對應的 Flow Entry (如圖九所示)，如果沒有則送往 Controller。此時的 Controller 就會接收到 Packet-In 的資訊，該封包的資訊主要有 in port、Source MAC Address、Destination MAC Address 等。再來當 Controller 判斷完該封包的去向後，就會發送 Packet-Out 的訊息給 OpenFlow 交換機，指示如何處理該封包 (如: 封包送往哪個埠口)。在此同時將該封包的 Flow Entry 規則新增給 OpenFlow 交換機。這樣一來，OpenFlow 交換機下一次接收到同樣封包並且有對應到 Flow Entry 的規則時，就可以馬上處理封包轉發的路徑，不必再送往 Controller 判斷。



圖八 OpenFlow model[11]



圖九 Flow Entry[13]

doi:10.6837/ncnu202300216

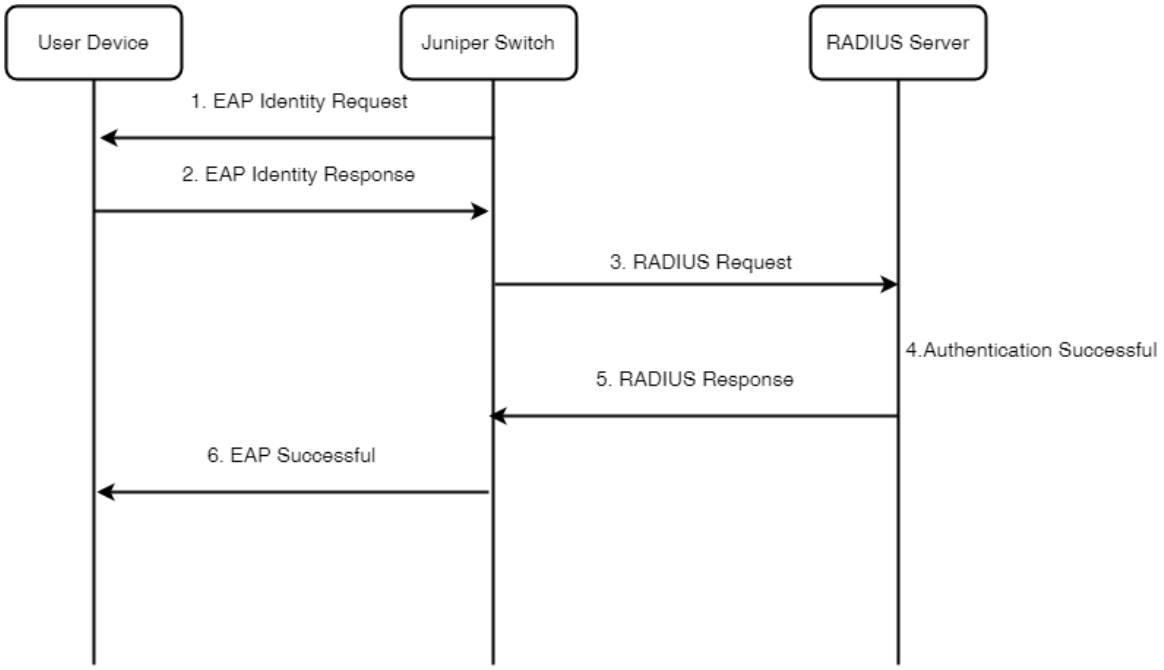
第三章 文獻探討

第一節 動態 VLAN 分配

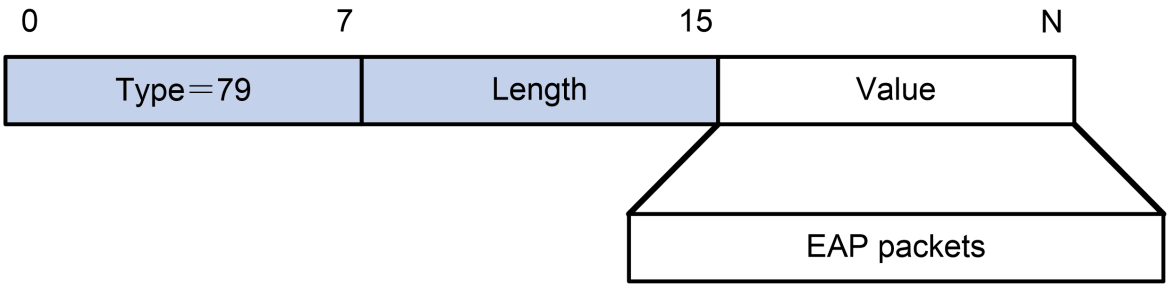
與靜態指定網路埠的 VLAN 設置相比，所謂「動態 VLAN」 是根據使用者主機的 MAC 或 IP 位址進行 VLAN ID 的分配。這使得當使用者移動到不同的位置時，只需連接到網路埠，就即可分配到對應的 VLAN，確保使用者在各個位置都能夠獲得所需的網路設定，進一步實現了網路隔離的效果。

在網路設備商 Juniper 的手冊中提供了 RADIUS (Remote Authentication Dial-In User Service) 搭配 Juniper 交換機動態 VLAN 分配的服務[14]。RADIUS 是一種網路協定，用於提供網路設備 (如:交換機、NAS…) 的身分驗證服務，並架設資料庫伺服器 (RADIUS Server) 集中管理使用者登入資訊。RADIUS 伺服器搭配 Juniper 交換機的驗證使用者機制流程如圖十所示，當使用者設備接入 Juniper 交換機的網路埠，Juniper 交換機發送 EAP (Extensible Authentication Protocol) 驗證請求給使用者設備。當使用者輸入完帳戶與密碼，驗證資訊傳回至 Juniper 交換機。接下來，Juniper 交換機透過 RADIUS 協定向 RADIUS Server 發送請求驗證使用者資訊，驗證資訊就會添加至 RADIUS 協定封包內 (如圖十一所示)，發送至 RADIUS Server。如果驗證成功，則會將驗證成功的封包傳回到 Juniper 交換機，接著再由 Juniper 交換機向使用者回應驗證結果，使用者設備就可以連上內部網路[15]。

doi:10.6837/ncnu202300216



圖十 RADIUS EAP 流程



圖十一 EAP Over RADIUS Format[16]

因此，藉由架設 RADIUS 伺服器驗證使用者的資訊分配 VLAN，以使用者名稱、密碼、Tunnel-Type (VLAN 類型)、Tunnel-Medium-Type (VLAN 標準值)、Tunnel-Private-Group-ID (VLAN ID) 為主要的設定，為使用者提供動態 VLAN 分配的服務。但是其缺點為需要輸入大量的指令，例如：若想要指定兩台設備 (印表機、伺服器) 使用 RADIUS 進行認證，兩台的 VLAN ID 分別為 10、20，則需要在網路設備與 RADIUS Server 連線、建立 VLAN、網路埠驗證類型、添加使用者資訊至 RADIUS Server，詳細的指令如圖十二。

本論文參考 Juniper 的動態 VLAN 分配架構，將 RADIUS Server 變成 SDN Controller、Juniper 交換機變成 Open vSwitch，在 SDN Controller 上撰寫一個動態

doi:10.6837/ncnu202300216

VLAN 分配服務。如此一來，網路管理者只需輸入 Open vSwitch 的 dpid 編號、使用者設備的 MAC 位址、VLAN ID 等組態資訊，在 SDN 進行簡易的設置，即可提供動態 VLAN 分配的服務，不需要有複雜的設置，也不必逐一連線至網路設備進行設置。

```
# 建立與 RADIUS Server 的連線
user@switch# set access radius-server 10.0.0.100 secret juniper
user@switch# set access profile profile1 authentication-order radius
user@switch# set access profile profile1 radius authentication-server 10.0.0.100
# 建立 VLAN
user@switch# set vlans vlan10 vlan-id 10
user@switch# set vlans vlan20 vlan-id 20
# 設定使用者接入網路埠的 VLAN
user@switch# set interfaces ge-0/0/19 unit 0 family ethernet-switching vlan
members vlan10
user@switch# set interfaces ge-0/0/20 unit 0 family ethernet-switching vlan
members vlan20
# 設定使用者網路埠的驗證類型
user@switch# set protocols dot1x authenticator interface ge-0/0/19 mac-radius
user@switch# set protocols dot1x authenticator interface ge-0/0/20 mac-radius
# 連線至 RADISU Server 設定使用者資訊
user@switch# ssh 10.0.0.100
user@radius# edit /etc/raddb
user@radius# vi users
      00040ffdacfe Auth-type:=EAP, User-Password = "00040ffdacfe"
            Tunnel-Type : VLAN
            Tunnel-Medium-Type : IEEE-802
            Tunnel-Private-Group-ID : 10
      0004aecd235f Auth-type:=EAP, User-Password = "0004aecd235f"
            Tunnel-Type : VLAN
            Tunnel-Medium-Type : IEEE-802
            Tunnel-Private-Group-ID : 20
```

圖十二 Juniper 交換機設置動態 VLAN 服務[14]

doi:10.6837/ncnu202300216

第二節 網路隔離

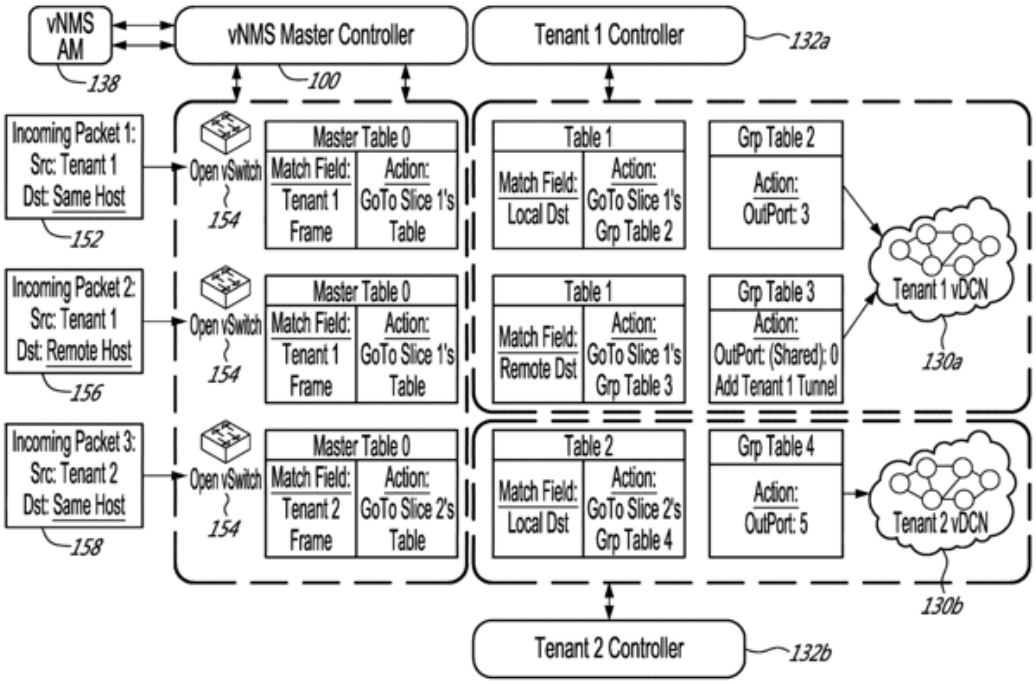
在[17]所述，此篇專利為了解決雲端上多租戶(Multi-Tenant)網路隔離的問題，多租戶通常會運用在雲端系統中，意指多位使用者可以共用雲端的資源和基礎設施，同時需要為每位使用者之間保持隔離，確保不會存取到其他使用者的資源。為了使每位使用者之間不會因為共享基礎設施而存取到彼此資源，此篇專利除了解決網路隔離的問題，在設計上採用 OpenFlow 多層式架構方式實現網路隔離。而 OpenFlow 多層式架構為 Flow Table 分成多個層次，每層都具有不同的規則。當交換機接收到封包時，依經過不同層進行規則對照和處理。每層的 Flow Table 可以根據特定的規則選擇執行不同的動作，也可以將封包轉發給更高優先層次進行處理。

在技術上該專利的控制層選擇使用 SDN、基礎設施層使用 Open vSwitch、控制層與基礎設施層之間的溝通協定使用 OpenFlow。因此，使用 OpenFlow 設計多層式的 Flow Table 架構 (如圖十三所示)，其流程為 : 當 Tenant 1 使用者封包進入 Open vSwitch，第一層的 Flow Table(Table 0)判斷該使用者封包的目的地 IP 位址應該要去哪一層 Flow Table 處理。接下來根據 Flow Table 所設計的規則將封包轉發到其它層的 Flow Table，判斷該封包是否還需要轉發到其它層 Flow Table 或是直接送至指定的網路埠。藉由轉發到其它層的 Flow Table，讓封包能夠與其它使用者的網路流量分開，進而達到為使用者提供網路隔離的服務。其 OpenFlow 多層式 Flow Table 優勢在可以對不同的 Flow Table 層進行處理，將處理封包的效能增加。在針對多位使用者時，藉由對 Flow Table 層進行撰寫規則，可以輕鬆地為每位使用者封包處理轉發至相對應的 Flow Table 層，以提供網路管理者對使用者封包更細膩的控制。

但其缺點就是當使用者越來越多時，Flow Table 的層數就相對的要多好幾層，這可能會讓開發 SDN 的網路管理者需要新增或調整 Flow Table 的動向，且後面的維護成本也相對的高。因此本論文參考此專利的網路隔離設計，使用 VLAN 進行網路隔離，在 Flow Table 設計的部分只需要一層，在單一 Flow Table 層新增四個規則，判斷

doi:10.6837/ncnu202300216

封包的 VLAN ID 是多少，再往對應該 VLAN ID 的網路埠送出，以提供使用者封包進入 Flow Table 層時對照的規則。如此一來，即可達到不用設計多層式的 Flow Table 增加維護成本，只需要判斷 VLAN ID 就可以為使用者提供網路隔離的服務。

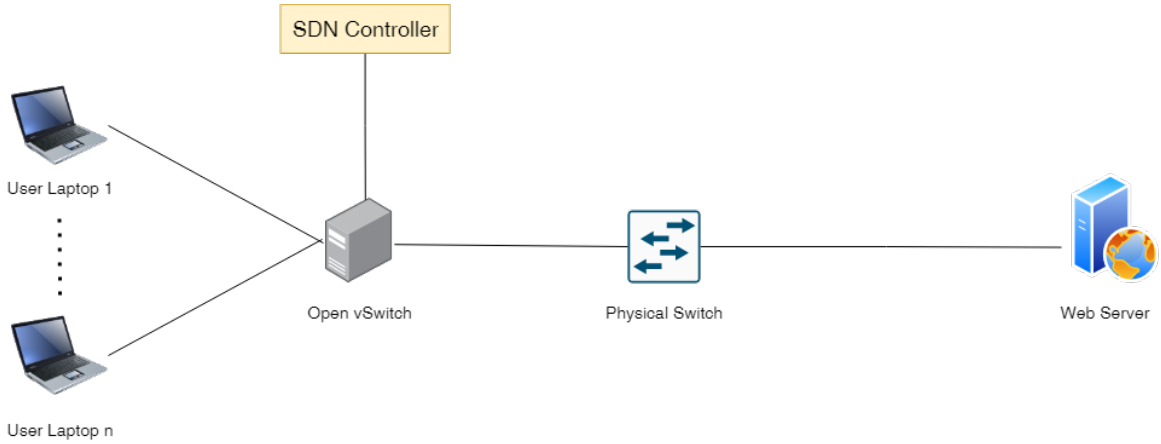


圖十三 An example of table re-direction [17]

doi:10.6837/ncnu202300216

第四章 SDN 動態 VLAN 分配系統

第一節 系統架構



圖十四 系統架構圖

本文提出的方法是使用 SDN 撰寫動態 VLAN 分配，判斷使用者筆電的 MAC Address 來分配 VLAN ID，這樣不管使用者的筆電接到哪個網路埠，該網路埠的 VLAN 都會自動被設定為與筆電一致。統一管理 VLAN 的問題利用 SDN 軟體化優勢，來創建列表統一管理 VLAN，並且可以快速的找尋 VLAN ID。如圖十四所示，首先看到 Open vSwitch 的主機， 這只須使用一部普通電腦安裝 Open vSwitch 軟體，由於 Open vSwitch 支援 OpenFlow[18]，這使得 SDN 可以管理 Open vSwitch，在實驗上可以節省購買實體 OpenFlow 交換機的費用。SDN 的 Framework 使用的是 Ryu，這是一個使用 Python 所撰寫的 SDN Framework[19]，本文將使用 Ryu 撰寫動態 VLAN 分配程式。實體交換機則用以連接 Open vSwitch 主機與 Web Server，將實體交換機連接 Open vSwitch 主機與 Web Server 的網路埠都設定成 Trunk Port，以使帶有 VLAN Tag 的封包可以通過。Web Server 使用了 Virtual VLAN Interface 來創建多個不同網段的 Virtual VLAN Interface，以達到網路隔離的效果。下一節將會詳細介紹在本文所規劃的 SDN 流程。

doi:10.6837/ncnu202300216

第二節 SDN 流程圖

本節將詳細介紹 SDN 動態 VLAN 的流程，這邊講述 SDN 主要的 Function，分別是 Packet-In Function、Flood_Packet Function、Forward_Packet Function 來完成本文 SDN 動態 VLAN 分配的程式撰寫。

第一小節 Packet-In Function

首先說明 Packet-In Function 的流程圖。Packet-In Function 主要是接收來自不知道該如何處理的封包，進而轉送到 SDN Controller 進行判斷。Packet-In 流程圖如圖十五所示:

1 本文的 Access Port 是以 MAC Address 來判斷分配 VLAN ID，一開始先設定使用者的 MAC Address、VLAN ID，前置作業完成後，執行 SDN 程式。

2 SDN Controller 接收到來自網路設備不知道該如何處理的封包。

3 首先判斷該封包是否有 VLAN Tag :

3.1 如果有。紀錄該封包的 VLAN ID，使用變數標記 (tag = 1) 該封包是有 VLAN ID 的。

3.2 如果沒有。使用變數標記 (tag = 0) 該封包是沒有 VLAN ID 的，並驗證該封包的 Source MAC Address 或 Destination MAC Address 是不是有在網路管理者所建立的管制名單內(使用者電腦的 MAC Address) :

3.2.1 如果沒有，則將封包丟棄。

3.2.2 如果有，則記錄該封包的 MAC Address 對應網路管理者所建立名單的 VLAN ID。

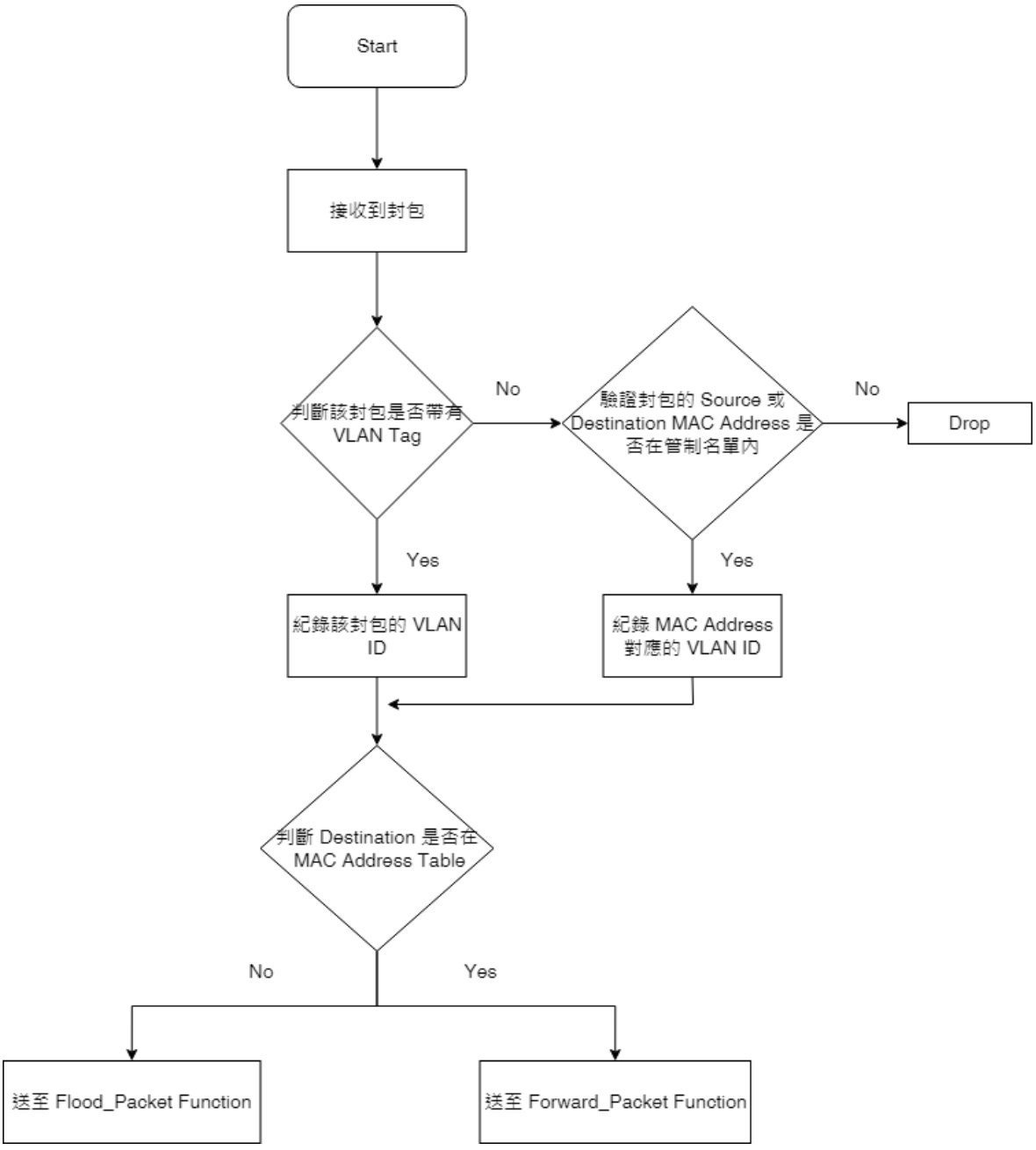
4 接下來判斷該封包的 Destination MAC Address 是否已學習記錄在 MAC Address Table 內 :

4.1 如果沒有學習記錄在 MAC Address Table，將封包的資訊 VLAN ID、tag、Source MAC Address、Destination MAC Address，送往

doi:10.6837/ncnu202300216

Flood_Packet Function。

4.2 如果有學習記錄在 MAC Address Table，將封包的資訊 VLAN ID、Out Port(封包送往的 Port)、tag、Source MAC Address、Destination MAC Address，送往 Forward_Packet Function。



圖十五 Packet-In Function Flowchart

doi:10.6837/ncnu202300216

第二小節 Flood_Packet Function

Flood_Packet Function 主要是因為交換器還尚未學習封包轉發的埠口，不知道若要到達 Destination MAC Address 應轉發至哪個指定的埠口上才可以抵達。這邊使用 ARP (Address Resolution Protocol) [20] 廣播封包的方式去找尋正確的路徑，當 ARP Request 送出後，就會廣播至區域網路內。屆時在所屬網段內的主機都會收到 ARP Request 的廣播封包，但只有與其 Destination IP Address 相同的主機會進行 ARP Response 回覆。所回覆的 MAC 位址會被對方儲存在 ARP Table 中，完成 ARP 的通訊流程。

接下來將詳細解說 Flood_Packet Function，如圖十六所示:

1 接收到來自於 Packet-In Function 的封包資訊。該封包是沒有儲存在 MAC Address Table 內的，故接收到此封包。

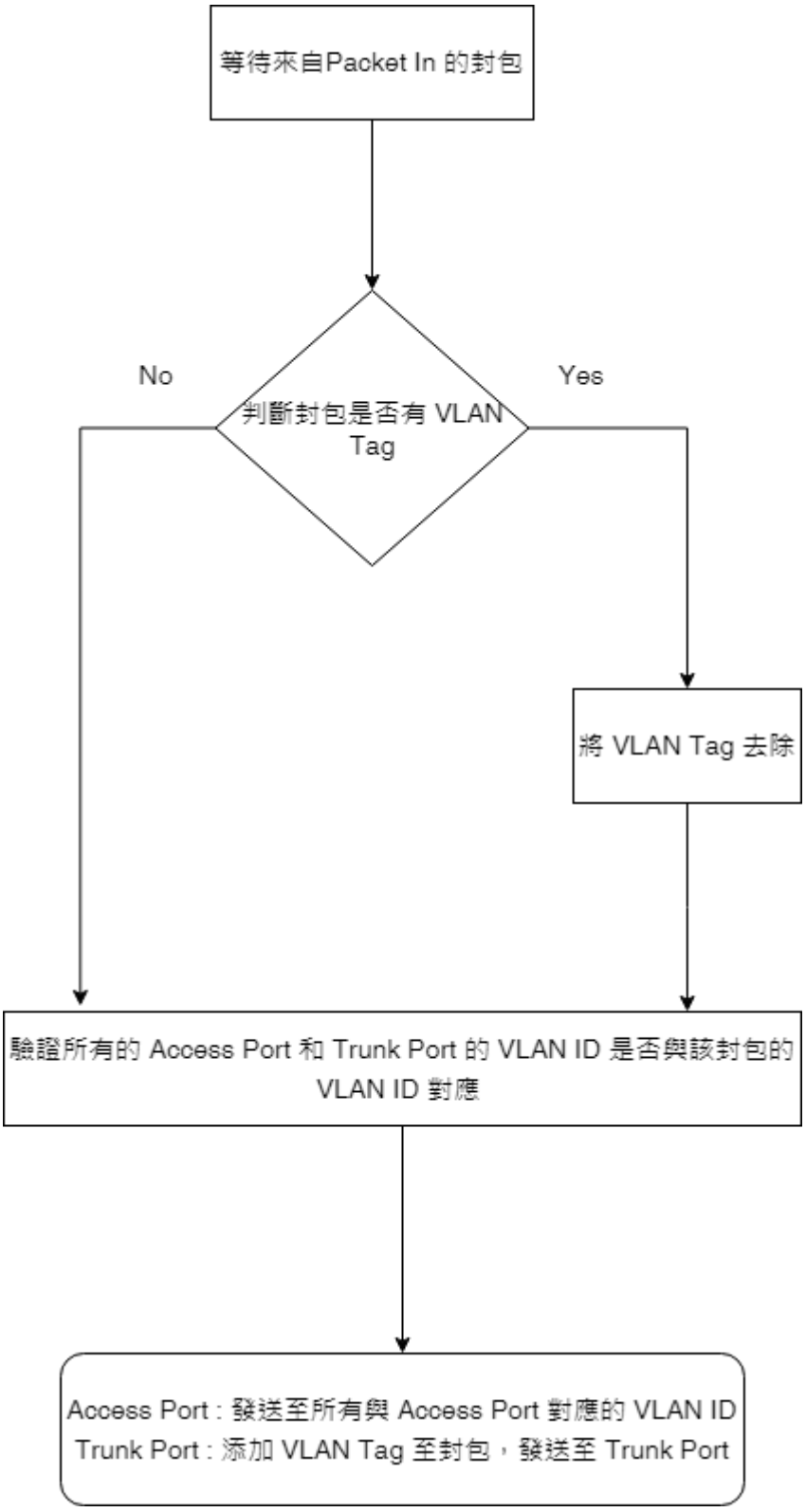
2 判斷該封包是否有 VLAN Tag (可以從第一小節 Packet-In Function 使用的變數 tag 是 1 還是 0 去做判斷) :

2.1 如果沒有 VLAN Tag，則找出對應於這個 Access Port 的 VLAN ID。

2.2 如果有 VLAN Tag，則將封包 VLAN Tag 去除。

3. 轉送到所有與此 VLAN ID 相同的 Access Port，以及添加 VLAN Tag 後，將封包送往允許該 VLAN ID 通過的 Trunk Port。

doi:10.6837/ncnu202300216



圖十六 Flood_Packet Function Flowchart

doi:10.6837/ncnu202300216

第三小節 Forward_Packet Function

最後是 Forward_Packet Function 流程圖。Forward_Packet Function 主要是當封包的 Destination MAC Address 已經學習過，並存在於 MAC Address Table 內，才會將封包的資訊送到 Forward_Packet Function。接下來，會先判斷封包的 in port (Source MAC Address 的 Port)、out port (根據 MAC Address Table 內封包轉發出去指定的 Port) 是 Access Port 或是 Trunk Port，再去做後續處理封包與添加 Flow Entry 的動作。

以下將詳細介紹圖十七的流程圖 ：

1 接收到來自 Packet-In Function 封包的資訊。

2 首先，判斷如果 in port 是 Access Port 和 out port 是 Access Port，且是同樣的 VLAN ID :

2.1 如果是。根據 MAC Address Table 學習的紀錄，將封包直接送往指定的 port，並添加 Flow Entry 給 Open vSwitch。

3 如果不是。判斷 in port 是 Access Port 和 out port 是 Trunk Port，且 Trunk Port 是允許該 VLAN ID 通過 ：

3.1 如果是。根據 MAC Address Table 學習的紀錄，先將封包添加 VLAN Tag，之後送往指定的 port，並添加 Flow Entry 給 Open vSwitch。

4 如果不是。判斷 in port 是 Trunk Port 和 out port 是 Access Port，且封包的 VLAN ID 是跟 Access Port 的 VLAN ID 是相同的 ：

4.1 如果是。根據 MAC Address Table 學習的紀錄，先將封包移除 VLAN Tag，之後送往指定的 port，並添加 Flow Entry 給 Open vSwitch。

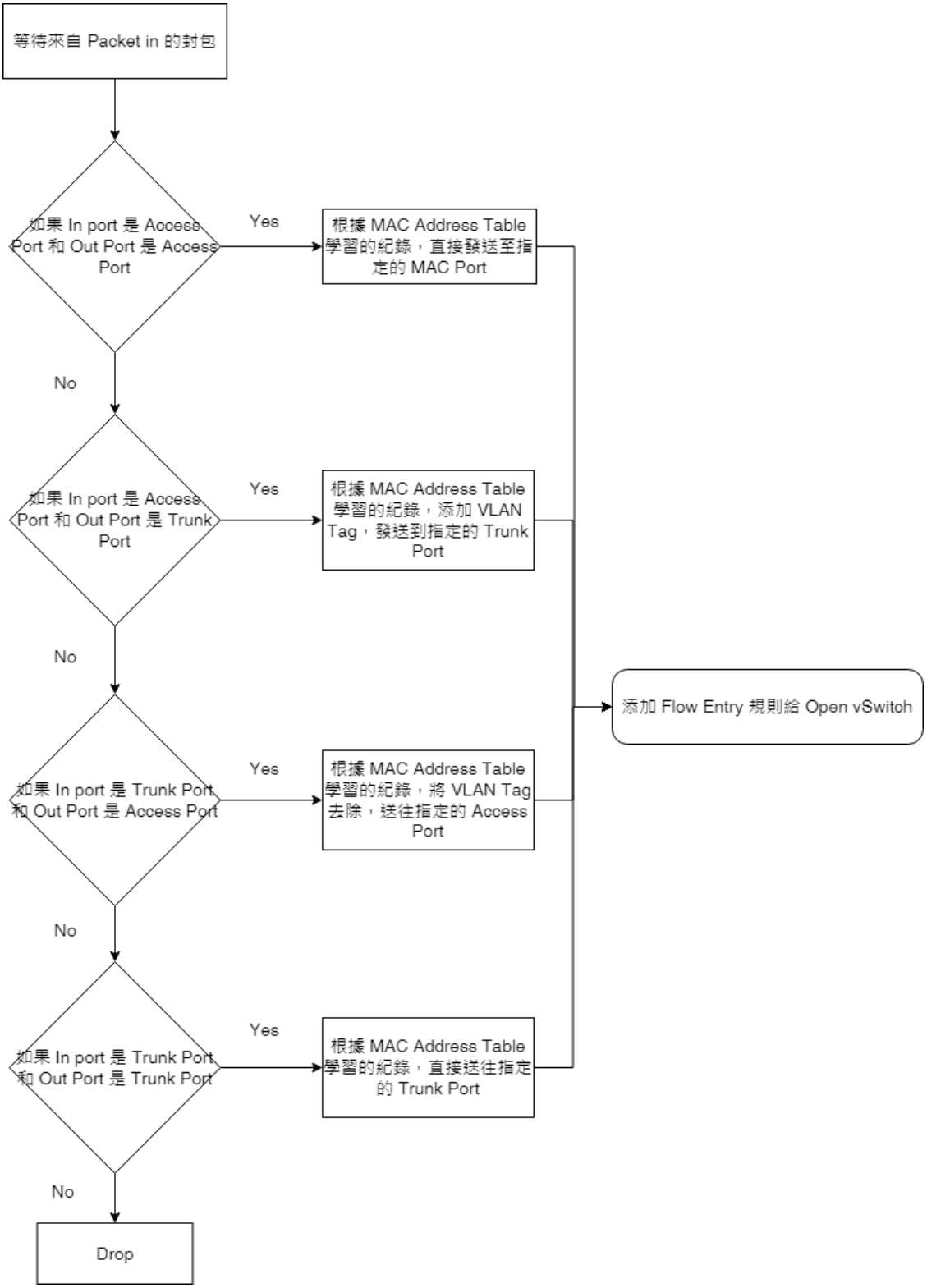
5 如果不是。判斷 in port 是 Trunk Port 和 out port 是 Trunk Port，且封包的 VLAN ID 是跟 Trunk Port 的 VLAN ID 是相同的 ：

5.1 如果是。根據 MAC Address Table 學習的紀錄，封包直接送往指定的

doi:10.6837/ncnu202300216

port，並添加 Flow Entry 給 Open vSwitch。

6　如果不是，代表找不到轉發的 Port，則將封包丟棄。



圖十七 Forward_Packet Function Flowchart

doi:10.6837/ncnu202300216

第三節 Flow Table 設計

本節的 Flow Table 的設計如表一所示。在圖十七時有講述當確認封包轉發出口後，SDN Controller 根據封包的內容 (如：VLAN ID、In Port、Source MAC Address、Destination MAC Address) 來撰寫 Flow Entry 規則，然後透過 OpenFlow 協定傳送規則給 Open vSwitch[21]。當有符合 Flow Entry 的封包時，就可以直接針對封包進行轉發至指定的埠口，這樣封包就可以不必每次都送往 SDN Controller 做判斷。

表一 Flow Table Design

Rule Name	Match	Action
Access Port to Access Port	In Port Destination MAC Address Source MAC Address	封包不必加上 VLAN Tag，直接送往指定的 out port
Access Port to Trunk Port	In Port Destination MAC Address Source MAC Address	封包加上 VLAN Tag，送往指定 out port
Trunk Port to Access Port	In Port VLAN ID Destination MAC Address Source MAC Address	封包去除 VLAN Tag，送往指定的 out port
Trunk Port to Trunk Port	In Port VLAN ID Destination MAC Address Source MAC Address	封包不必移除 VLAN Tag，直接送往指定的 out port

doi:10.6837/ncnu202300216

第四節 系統流程

在上一節詳細介紹 SDN 動態 VLAN 的流程圖與 Flow Table 後，本節將介紹系統是如何進行運作。參考圖十八當一臺筆電接上 Open vSwitch 的網路埠，使用者筆電想要訪問 Web Server VLAN 10 的虛擬網卡，使用筆電到 Web Server 之間的 Request、Response 運作方式，以下將會分成兩小節來做敘述。

第一小節 HTTP Request

如圖十八所示，首先講解使用者的筆電想要連線到 Web Server 的 Request 整個流程：

1 使用者筆電接上 Open vSwitch 主機的網路埠，使用者筆電欲連線到 Web Server 的 VLAN 10 的 Interface。封包送往 Open vSwitch。

2 Open vSwitch 收到封包後，不知道該如何處理該封包，送往 SDN Controller (Packet-In)。

3 SDN Controller 判斷使用者筆電的 MAC 地址，Open vSwitch 與實體交換機的網路埠都是設置 Trunk VLAN 10，在這邊分配 VLAN 10。

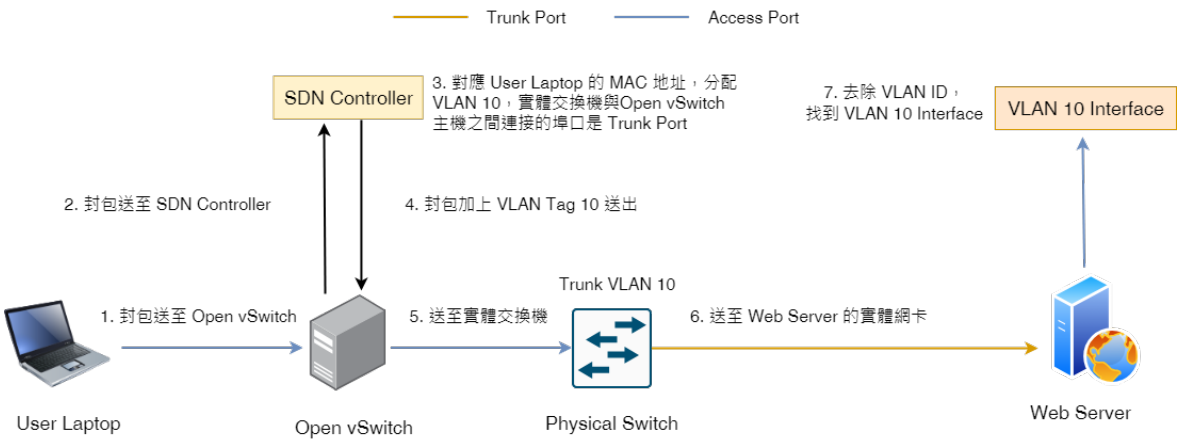
4 封包添加 VLAN Tag 並將封包送出至實體交換機 (Packet-Out)。

5 封包送往實體交換機，實體交換機與 Open vSwitch 主機之間連接的埠口是 Trunk Port，並允許該封包 VLAN 10 通過。

6 實體交換機與 Web Server 的網路埠同樣也是設置 Trunk VLAN 10，封包直接送往 Web Server 實體網卡。由於 Web Server 的 Virtual VLAN Interface 是基於實體網卡來去做創建，故 Web Server 實體網卡就像是 Trunk Port，而 Virtual VLAN Interface 是 Access Port。送到 Web Server 實體網卡後，先找尋封包的 VLAN ID 是否有對應到 VLAN 10 Interface，有的話將會進行去除 VLAN ID。

7 找到 VLAN 10 Interface，將封包送往 VLAN 10 Interface。

doi:10.6837/ncnu202300216



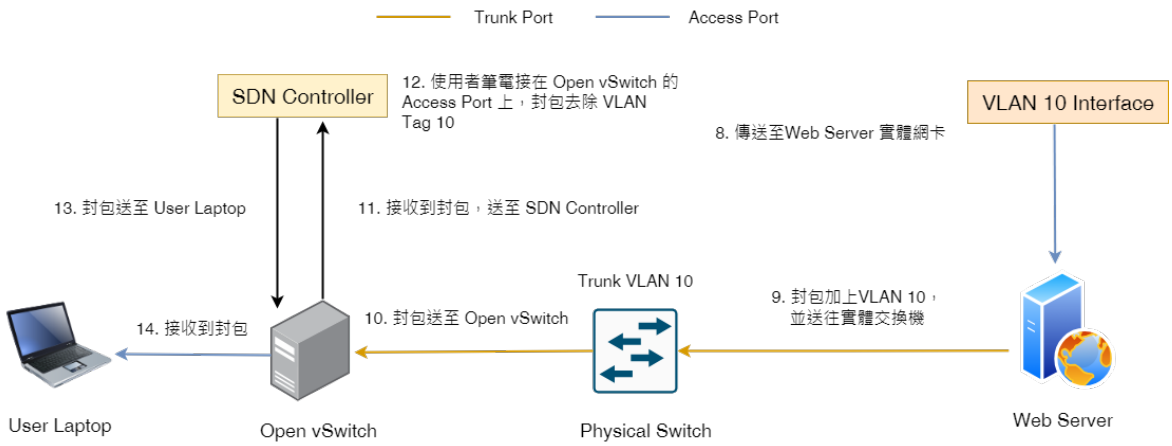
圖十八 HTTP Request

第二小節 HTTP Response

如圖十九所示，當使用者筆電 Request 送達 Web Server VLAN 10 Interface 後，接下來就是傳送回覆：

8. Web Server VLAN 10 Interface 將封包傳送至 Web Server 實體網卡。
9. 封包送至實體網卡後，封包添加 VLAN Tag 並送往實體交換機。
10. 實體交換機連接 Open vSwitch 的埠口是 Trunk VLAN 10，故直接將封包送往 Open vSwitch。
11. Open vSwitch 接收到不知道如何處理的封包，送至 SDN Controller (Packet-In)。
12. 使用者筆電接在 Open vSwitch 的 Access Port 上，該封包需要去除 VLAN Tag。
13. 去除完後的封包發送至使用者筆電 (Packet Out)。
14. 使用者筆電接收到封包。

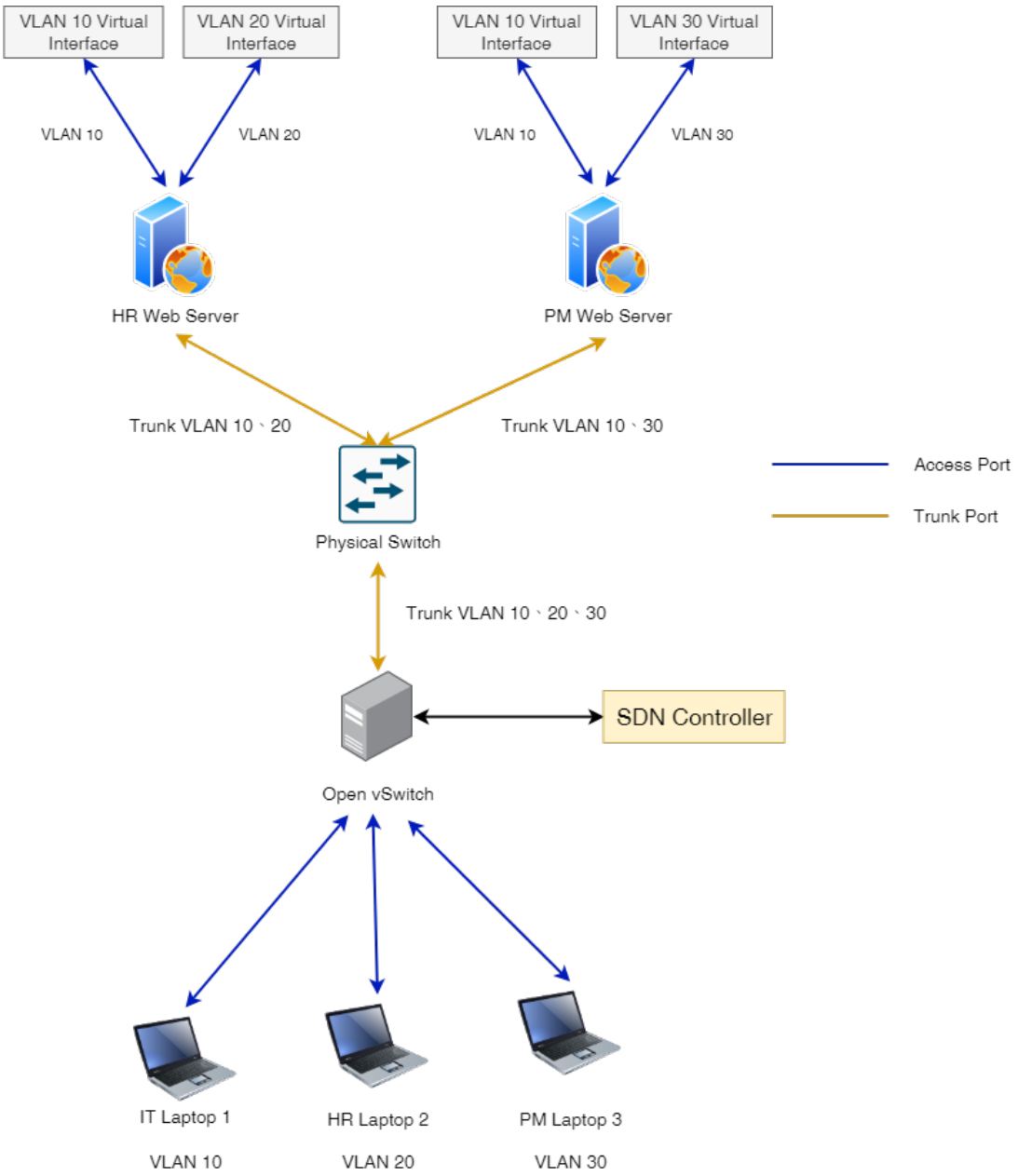
doi:10.6837/ncnu202300216



圖十九 HTTP Response

doi:10.6837/ncnu202300216

第五章 系統實作



圖二十 系統實作架構圖

在第四章的部分講述了系統的架構、SDN 流程圖的規劃，本章將著重在於將系統內容進行實作，包含系統實作架構圖、實驗環境架設等。本章的系統實作架構圖如圖二十所示，這邊以列表方式來介紹系統實作架構中各個主機所扮演的角色，順序由下往上講述：

doi:10.6837/ncnu202300216

1. User Laptop : User Laptop 在系統架構中分別為 IT Laptop、HR Laptop、PM Laptop，目的是為了模擬三個不同部門的設備接上 Open vSwitch 網路埠後，藉由 SDN Controller 判斷 MAC Address 來分配適當的 VLAN ID。
2. Open vSwitch : 開源的虛擬交換機軟體，支援 OpenFlow 協定。安裝在一台實體主機上 (這邊統一稱為 Open vSwitch 主機，如圖二十一所示)。Open vSwitch 中的橋接器(Bridge)負責處理網路流量，我們將實體網卡添加至橋接器，以進行網路封包的轉發。紫色是與實體交換機連接的部分，棕色則是給使用者筆電筆電連接用的網路埠。
3. SDN Controller : SDN 使用的 Framework 為 Ryu，與 Open vSwitch 安裝在同一台主機上 (如圖二十一所示)，並管制 Open vSwitch 網卡的封包傳送。程式碼如附錄一。

圖二十一 Open vSwitch 主機

doi:10.6837/ncnu202300216

4. Physical Switch : 實體交換機使用 ZYXEL GS1200-5，5 個網路埠的 Switch。實體交換機作為連接 Open vSwitch 主機與 Web Server 埠口的管道。實體交換機如圖二十二所示。

圖二十二 ZYXEL GS1200-5 Switch

5. Web Server : Web Server 採用 Nginx 作為網頁伺服器。在兩台 Web Server 上都建立了不同網段 Virtual VLAN Interface，達到網路隔離的目的。

主機的規格表如表二所示 ：

表二 主機規格配置表

Host	CPU	Memory	OS	Software
Open vSwitch	Intel® Core™ i5-11400 2.60 GHz	16G	Debian 11	Open vSwitch 2.15.0 Ryu 4.22
IT Web Server	Intel® Core™ i7-11700 2.50 GHz	64G	Debian 11	Nginx 1.18.0
HR Web Server	Intel® Core™ i5-9400 2.90 GHz	32G	Debian 11	Nginx 1.18.0

doi:10.6837/ncnu202300216

第一節 Open vSwitch

本節將講述如何安裝 Open vSwitch、將實體網卡加入 Open vSwitch 這兩個部分，建立起實作環節的環境：

第一小節 Open vSwitch 安裝

此小節講述安裝 Open vSwitch 步驟，以下將會以步驟的方式講述：

1 先將 apt 更新至最新版本。

#sudo apt update && sudo apt upgrade

2 使用 apt 直接下載 Open vSwitch。

#sudo apt install openvswitch-switch-dpdk -y

3 安裝完成後觀看 Open vSwitch 版本

#sudo ovs-vsctl --version

第二小節 Open vSwitch 設定

此小節講述使用 Open vSwitch 將實體網卡加入，並

1 新增一個 Bridge，名字叫做 ovs。

#sudo ovs-vsctl add-br ovs

2 將實體網卡 eno1 加入至 ovs 的 Bridge。

#sudo ovs-vsctl add-port ovs eno1

3 將 eno1 的網路介面清空，讓 Open vSwitch 可以藉由 eno1 將封包送往出去，並將網路資訊 (IP Address、Netmask) 指派給 ovs

#sudo ifconfig eno1 0

#sudo ifconfig ovs up

#sudo ifconfig ovs 192.168.0.50 netmask 255.255.255.0

4 將其餘實體網卡加入至 ovs 的 Bridge，提供使用者設備接入網路埠使用動態 VLAN 分配服務。

doi:10.6837/ncnu202300216

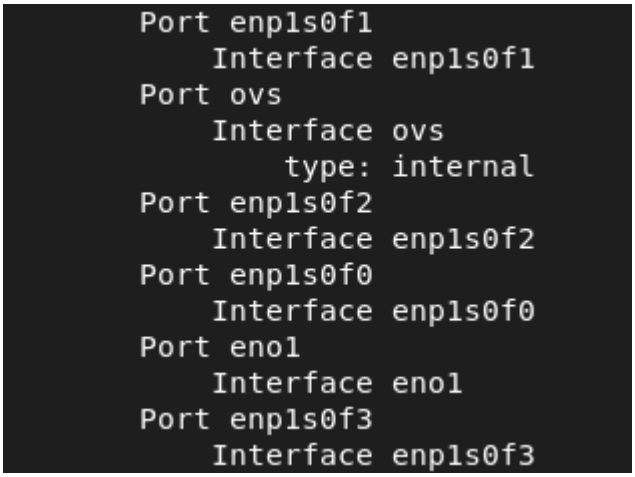
#sudo ovs-vsctl add-port ovs enp1s0f0

#sudo ovs-vsctl add-port ovs enp1s0f1

#sudo ovs-vsctl add-port ovs enp1s0f2

#sudo ovs-vsctl add-port ovs enp1s0f3

完成加入後輸入 sudo ovs-vsctl show，顯示畫面如圖二十三所示。



圖二十三 實體網卡加入 Open vSwitch

doi:10.6837/ncnu202300216

第二節 SDN Controller

本節主要講述安裝 Ryu Framework 的過程、Open vSwitch 以及執行前的前置設定：

第一小節 Ryu 安裝

以下步驟為安裝 Ryu 環境的過程：

1 安裝 Python 相關工具

#sudo apt-get install python-pip

#sudo apt-get install python-setuptools

2 使用 pip 安裝 ryu

#pip install ryu

3 使用 git 抓取 ryu 原始碼

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

4 切換到 ryu 目錄，執行安裝

#cd ryu/

sudo python3 ./setup.py install

第二小節 設定 Ryu Controller

此小節的設定接續圖二十三，以下為在 Open vSwitch 設定 SDN Controller 的步驟：

1 在 Open vSwitch 預設情況下是指定 OpenFlow 1.0 版本，這邊手動更改指定 OpenFlow 1.3 版本。

#sudo ovs-vsctl set bridge ovs protocols=OpenFlow13

2 在 Open vSwitch 設置與 SDN Controller 控制器連接的 IP 地址，以下指定使用本機 IP 地址 127.0.0.1，TCP 的埠號為 6633。

#sudo ovs-vsctl set-controller ovs tcp:127.0.0.1:6633

doi:10.6837/ncnu202300216

3　使用指令測試 SDN Controller 是否成功與 Open vSwitch 連接。

#sudo ryu-manager --verbose

3.1　成功的畫面如圖二十四所示，"is_connected: true"代表 SDN Controller 成功與 Open vSwitch 的 Bridge 成功連接。

圖二十四 SDN Controller 連接成功畫面

4　設定 Open vSwitch Dpid 的編號與找尋 Trunk Port Number，以提供下一小節所需要用到的設定。

#sudo ovs-vsctl set bridge ovs other_config:datapath-id=0000000000000001

#sudo ovs-vsctl -- --columns=name,ofport list interface eno1

第三小節 前置設定

在執行程式之前需要為使用者的 MAC Address 分配 VLAN，以及 Trunk Port 允許哪些 VLAN 通過。配置的範例如圖二十五，首先會有兩個變數 "access_port_data" 和 "trunk_port_data" 都是使用字典 (Dictionary) 的方式來存放資料。變數 "access_port_data"紀錄使用者筆電的 MAC Address，並為使用者筆電分配 VLAN ID。圖二十五顯示字典結構組成分別為 Dpid (Open vSwitch 的 ID)、MAC Address、VLAN ID 所組成。變數"trunk_port_data"控制哪些 VLAN ID 可以通過，該字典結構組成分別為 Dpid、Port Number、VLAN ID 所組成。

doi:10.6837/ncnu202300216

```
# { Dpid : {MAC Address: VLAN ID, ...},....}
access_port_data = {
    1:{"00:00:00:00:00:01":10,"00:00:00:00:00:02":20,"00:00:00:00:00:03":30}
}

# { Dpid : {Port Number: [VLAN ID 1, VLAN ID 2, VLAN ID 3], ...},....}
trunk_port_data = {
    1:{1:[10,20,30]}
}
```

圖二十五 VLAN 前置設定

doi:10.6837/ncnu202300216

第三節 實體交換機架設

實體交換機作為連接 Open vSwitch 與 Web Server 主機之間的埠口，本節主要講述實體交換機設置 VLAN 的過程。

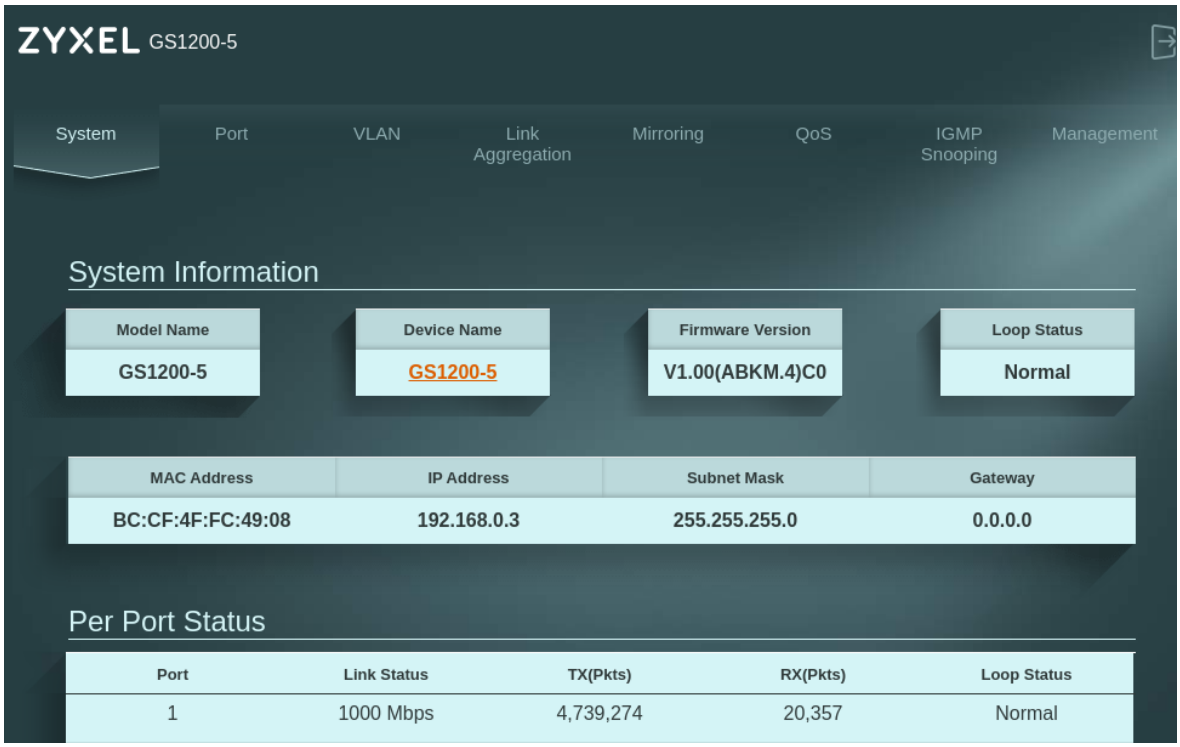
第一小節 實體交換機設定 VLAN

將實體交換機的電源接上後，就可以開啟 Web 頁面。在系統實作上只有網路管理者才可以進行操作，也就是擁有 SDN Controller 的主機，故這邊使用的是 Open vSwitch 主機來進行操作。

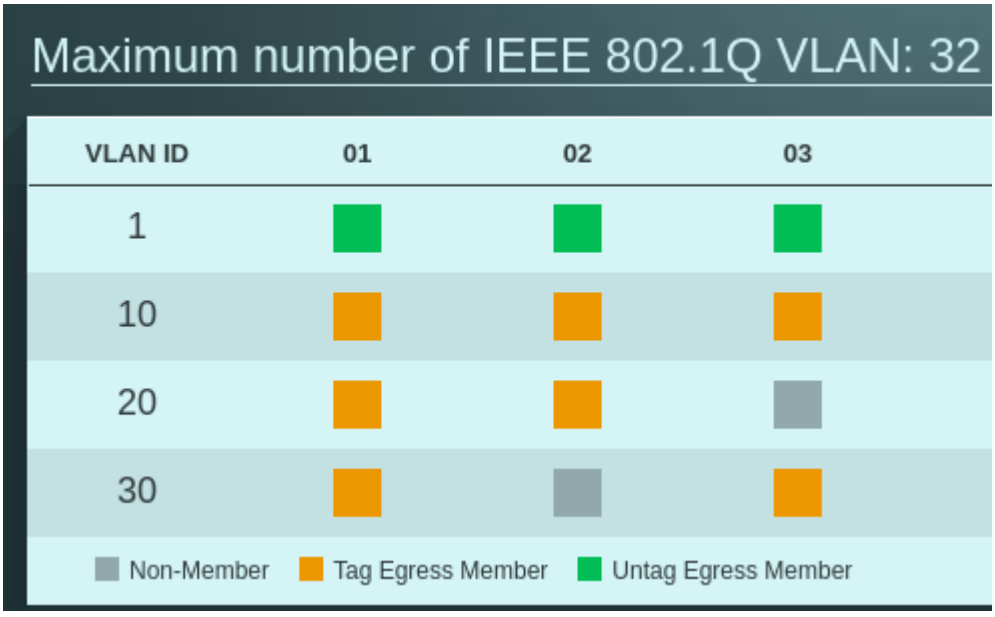
先在 Open vSwitch 主機上開啟 Firefox，輸入 "http://192.168.0.3"，管理畫面如圖二十六所示。接下來在管理介面選擇 "VLAN" 的頁面，進入到該頁面後，就可以開始新增本次系統實作的 VLAN ID (VLAN 10、VLAN 20、VLAN 30)，新增好的畫面如圖二十七所示。灰色表示該埠口沒有設置 VLAN ID。橘色表示該埠口為 Trunk Port，也就是允許哪些 VLAN ID 經過此埠口。綠色則是 Access Port，若有 VLAN ID 的封包經過該埠口時，將會進行去除 VLAN ID，送至所連接的主機。

綜合上述所敘，Port 1 連接的主機是 Open vSwitch 主機，允許 VLAN 10、20、30 通過。Port 2 連接的主機是 HR Web Server 主機，允許 VLAN 10、20 通過。Port 3 連接的主機是 PM Web Server 主機，允許 VLAN 10、30 通過。

doi:10.6837/ncnu202300216



圖二十六 ZYXEL GS1200-5 Switch Web 管理介面



圖二十七 ZYXEL GS1200-5 Switch VLAN 設置

doi:10.6837/ncnu202300216

第四節 Web Server 架設

本節主要講述 Web Server 的 Nginx 與 Virtual VLAN Interface 是如何安裝與架設：

第一小節 Nginx 安裝

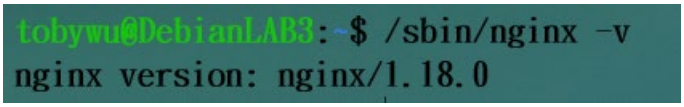
本小節講述安裝 Nginx 的過程[22]，以及啟動 Nginx 網頁。

1 使用 apt 安裝 Nginx。

#sudo apt install nginx

2 查看 Nginx 版本以及啟動

#/sbin/nginx -v



圖二十八 Nginx Version

#sudo systemctl start nginx.serivce

3 開啟 Firefox 輸入本地位址 (http://127.0.0.1) 查看 Nginx 網頁是否已經成功啟動，成功的畫面如圖二十九所示。

圖二十九 Nginx Web 畫面

第二小節 Virtual VLAN Interface 載入模組&架設

Virtual VLAN Interface 在一般的 Linux 主機的 Kernel 就已經具備，啟動的步驟

doi:10.6837/ncnu202300216

是將此模組載入和設定，以下將以步驟的方式來講述：

1 載入 Virtual VLAN Interface 模組 (802.1Q)

#modprobe 8021q && echo "8021q" >> /etc/modules

2 確認是否已經載入

#sudo lsmod | grep -i 8021q

3 至/etc/network/interface 網路介面卡修改設定，如圖三十所示

```
auto enp3s0
iface enp3s0 inet manual
        pre-up ifconfig $IFACE up

auto enp3s0.20
iface enp3s0.10 inet static
        vlan-raw-device enp3s0
        address 192.168.2.20
        netmask 255.255.255.0

auto enp3s0.30
iface enp3s0.30 inet static
        vlan-raw-device enp3s0
        address 192.168.3.31
        netmask 255.255.255.0
```

圖三十 Virtual VLAN Interface 設定

doi:10.6837/ncnu202300216

第六章 實驗結果

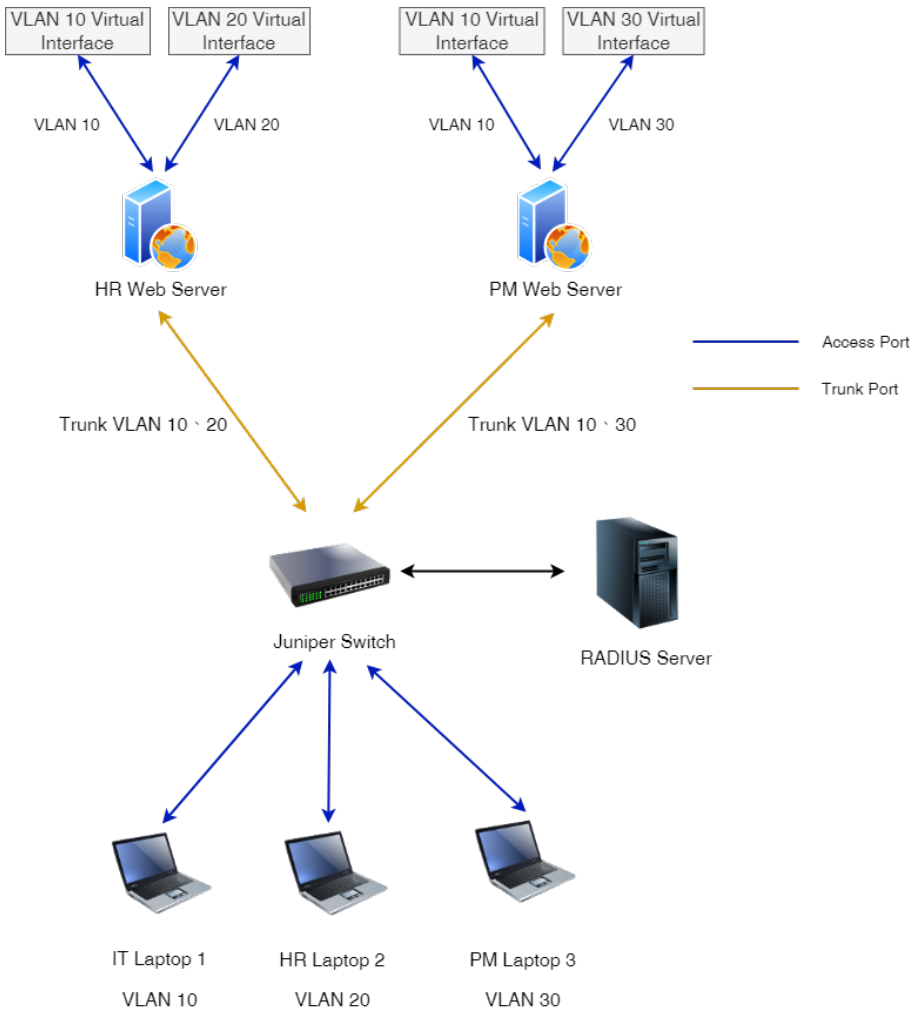
第一節 SDN Controller 與 Juniper 動態 VLAN 分配服務

經過本系統環境架設與測試，模擬使用三台不同部門的使用者筆電分別插入 Open vSwitch 的網路埠，三台筆電皆屬於不同的 VLAN，都可以分配到適當的 VLAN ID。模擬實際三台筆電由原本的網路埠，接入與原本不同的網路埠中，皆可以正確地分配到對應的 VLAN ID，此效果為本篇論文理想的情況。

在分配 VLAN ID 的時間上，本篇論文 SDN Controller 動態 VLAN 分配服務與文獻回顧的 Juniper 所提供的動態 VLAN 分配服務進行比較。Juniper 所提供的動態 VLAN 分配實驗環境架構如圖三十一，有三台的使用者設備、Juniper 交換機、RADIUS Server。使用者設備接入到 Juniper 交換機的網路埠後，會發送 EAP 驗證給使用者設備，使用者必須輸入帳戶與密碼。屆時輸入完成後，Juniper 交換機會把使用者資訊送給 RADIUS Server，驗證沒有問題後，將驗證成功的訊息傳回到 Juniper 交換機，再經由 Juniper 交換機發送成功的訊息給使用者設備，這時使用者設備就成功分配到相對應的 VLAN ID，並且能夠成功的連線至自己的部門。

在測試為使用者設備分配 VLAN ID 的時間方式，SDN Controller 接收到封包後開始計算時間，而後針對該封包的 Source MAC Address 或 Destination MAC Address 對列表進行查詢是否有相對應的 VLAN ID，返回結果後計算時間結束。在 Juniper 交換機與 RADIUS Server 分配 VLAN ID 的時間，則是使用者輸入完驗證訊息後，等候 RADIUS Server 驗證成功的消息，使用 Wireshark 檢測該封包回傳成功驗證的秒數。

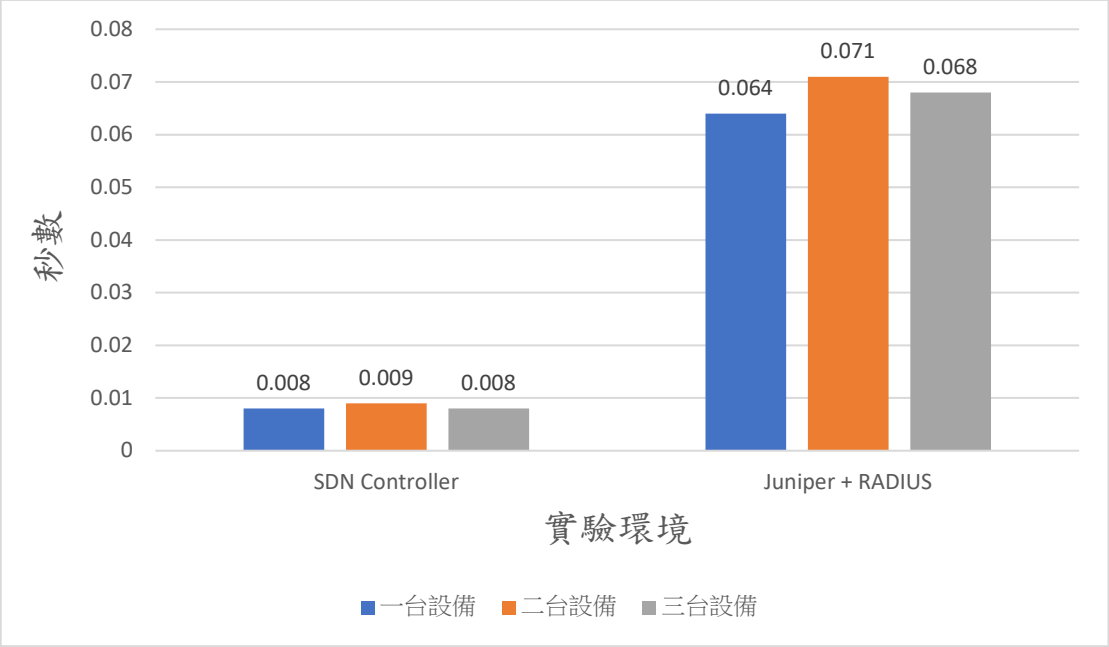
doi:10.6837/ncnu202300216



圖三十一 Juniper 交換機與 RADIUS 實驗環境

檢測的結果如圖三十二所示，在 SDN Controller 的部分在同時一 ~ 三台接入網路埠時，分配到 VLAN ID 的時間為 0.008 ~ 0.009 秒。而 Juniper 交換機與 RADIUS Server 部分同樣也是一 ~ 三台接入網路埠，分配 VLAN ID 時間平均為 0.06 ~ 0.08 秒之間。SDN Controller 在分配 VLAN ID 上比起 Juniper 提供的服務快上了 0.07 秒，這顯示本篇論文所提出的 SDN Controller 動態 VLAN 分配服務是明顯佔據了優勢。下一節，將測試對 SDN Controller 發送 1~1000 個不同 MAC 位址的封包，來檢測在多個封包的壓力下，是否還可以繼續擁有良好的分配 VLAN ID 的表現。

doi:10.6837/ncnu202300216

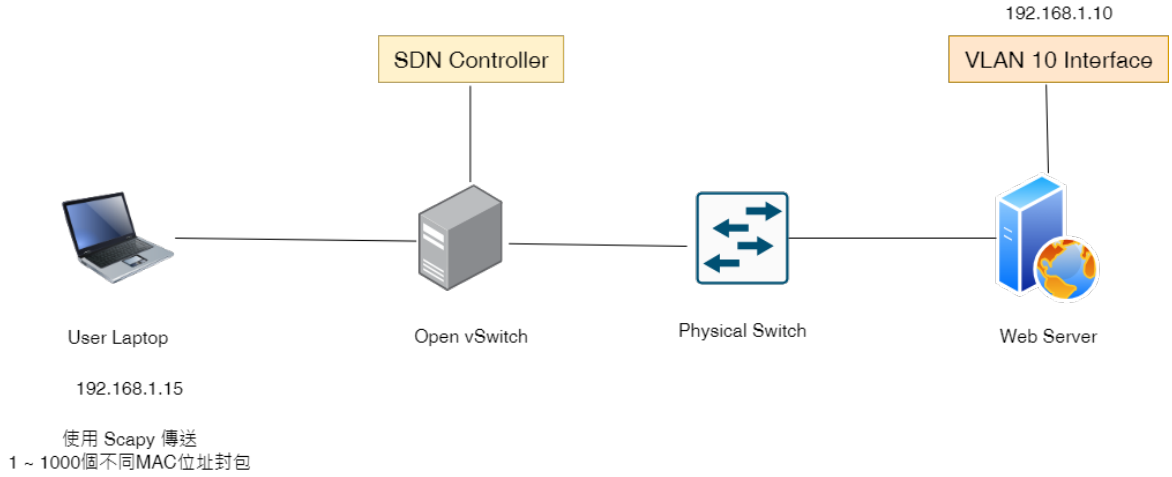


圖三十二 SDN Controller 與 Juniper + RADIUS Server 比較分配 VLAN 時間

doi:10.6837/ncnu202300216

第二節 SDN Controller 動態 VLAN 分配效能測試

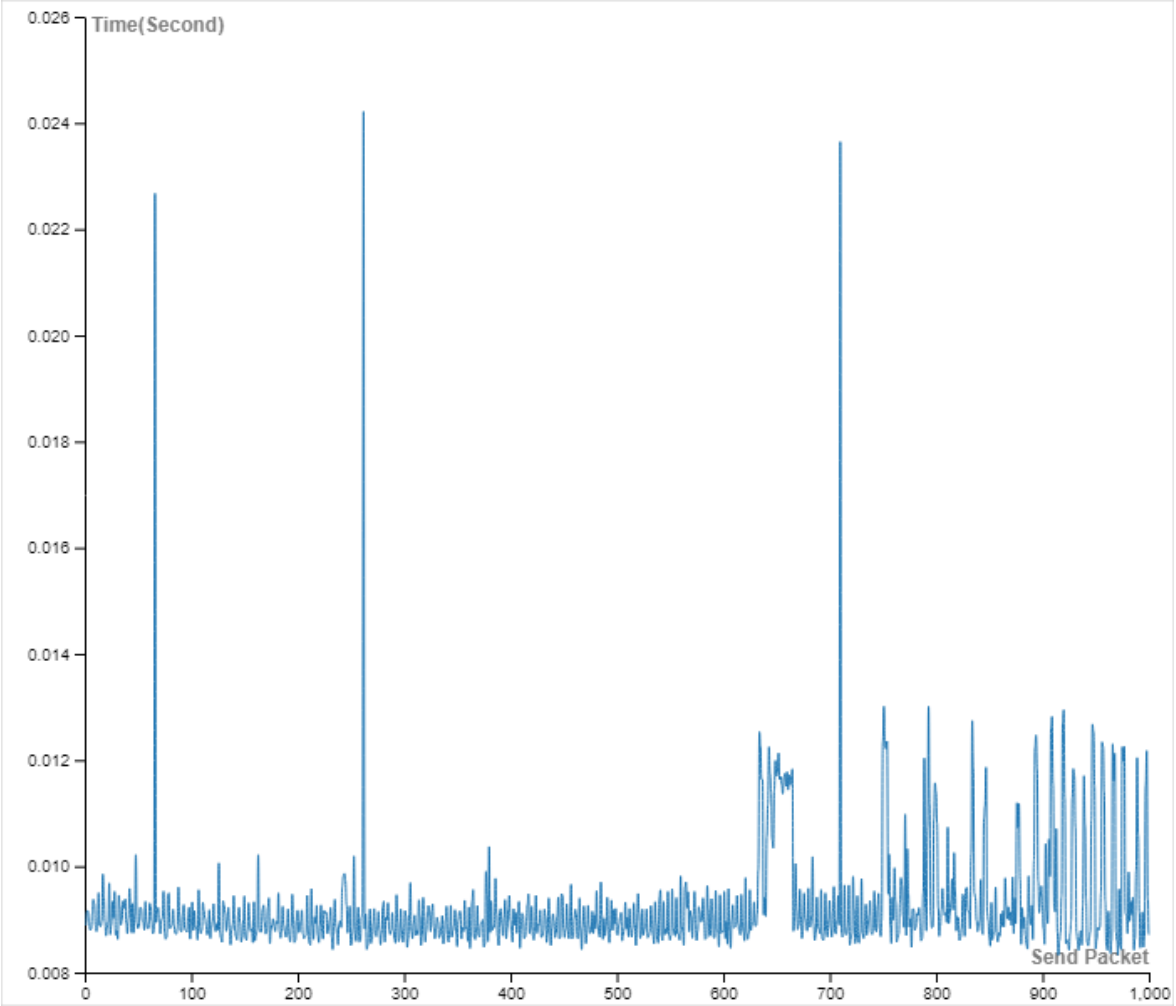
本節講述 SDN Controller 在動態 VLAN 分配的效能，以了解處理大量 MAC 位址和分配 VLAN 的時間。本次實驗的環境如圖三十三所示，使用者筆電 (192.168.1.15) 使用 Scapy 工具偽造 1~1000 不同的 MAC 位址封包傳送到 Web Server 的 VLAN 10 Interface (192.168.1.10)，偽造 MAC 的封包到 Open vSwitch 後，都會由 OpenFlow 協定送至 SDN Controller 判斷，而實體交換機則是在與 Open vSwitch 主機、Web Server 之間的網路埠設置 Trunk Port，允許 VLAN 10 通過。



圖三十三 SDN Controller 效能測試環境

經由傳送 1~1000 個不同 MAC 位址的封包，SDN Controller 動態 VLAN 分配，所得到的結果如圖三十四所示，觀察到在傳送封包數量達到 600 以前，每 100 個封包只會出現 1 ~ 3 個分配 VLAN 時間是超過 0.01 秒，其餘封包都是在 0.008 ~ 0.009 秒。而 600 – 10000 之間的封包有一半以上分配時間超過 0.01 秒。由此可之封包數量越多，SDN Controller 分配 VLAN 處理的時間隨之增長。但是，這不是一個大問題，當封包數量來到第 1000 個時，秒數為 0.012 秒，比起第一個封包 (0.008 秒) 只增加了 0.004 秒，這表示當 SDN Controller 處理多個不同 MAC 位址的封包都還可以保有適當的秒數，讓使用者能夠不用因為使用人數一多而有延遲分配到 VLAN ID 的感受。

doi:10.6837/ncnu202300216



圖三十四 SDN Controller 效能量測

doi:10.6837/ncnu202300216

第七章 結論與未來展望

本篇論文提出了使用 SDN 的框架 Ryu 撰寫動態 VLAN 分配與統一管理 VLAN 的方式，且透過實體主機來模擬使用者筆電連至任意網路埠的情況，讓使用者不必為了指定座位而煩惱，也使得網路管理者能夠不必花費大量時間查找每一台網路設備的 VLAN，管理程序更加簡便，變更使用者的 VLAN 也顯得靈活許多。

未來希望可以將純文字的設定檔變成 Web 管理頁面，讓網路管理者能夠使用網頁來查找使用者的 VLAN 狀況，並且把新增、移除、修改、搜尋 VLAN 的功能也加到網頁介面，讓網路管理者可以更容易即時查找，以達到更好的管理效果。

doi:10.6837/ncnu202300216

參考文獻

[1] G. Leischner and C. Tews, "Security Through VLAN Segmentation : Isolating and Securing Critical Assets Without Loss of Usability," 2007.

[2] Z. Xiyang and C. Chuanqing, "Research on VLAN Technology in L3 Switch," in *2009 Third International Symposium on Intelligent Information Technology Application*, 21-22 Nov. 2009 2009, vol. 3, pp. 722-725, doi: 10.1109/IITA.2009.498.

[3] V. G. Nguyen and Y. H. Kim, "SDN-based enterprise and campus networks: A case of VLAN management," *Journal of Information Processing Systems,* vol. 12, pp. 511-524, 01/01 2016, doi: 10.3745/JIPS.03.0039.

[4] H. Altunbasak, S. Krasser, H. L. Owen, J. Grimminger, H.-P. Huth, and J. Sokol, "Securing Layer 2 in Local Area Networks," Berlin, Heidelberg, 2005: Springer Berlin Heidelberg, in Networking - ICN 2005, pp. 699-706, doi: https://doi.org/10.1007/978-3-540-31957-3_79.

[5] G. Lee, "Chapter 5 - Data Center Networking Standards," in *Cloud Networking*, G. Lee Ed. Boston: Morgan Kaufmann, 2014, pp. 87-102.

[6] G. Pujolle, *SDN (Software-Defined Networking)*. 2020.

[7] M. E.-G. H. Khalil, A. Ibrahim, M. E. A. Kanona, and A. E. Ahmed, "Evaluation of Load Balancing Performance in Software Defined Network vs Classical Network," presented at the SCCSIT7: 7th International Conference of Computer Science and Information Technology, International University of Africa. Khartoum, Sudan, 2020.

[8] M. Awais, M. Asif, M. B. Ahmad, T. Mahmood, and S. Munir, "Comparative Analysis of Traditional and Software Defined Networks," in *2021 Mohammad Ali Jinnah University International Conference on Computing (MAJICC)*, 15-17 July 2021 2021, pp. 1-6, doi: 10.1109/MAJICC53071.2021.9526236.

[9] N. McKeown *et al.*, "OpenFlow: Enabling Innovation in Campus Networks," *SIGCOMM Comput. Commun. Rev.,* vol. 38, no. 2, pp. 69-74, 2008, doi: 10.1145/1355734.1355746.

[10] L. Ong, V. Shukla, M. Vissers, K. Sethuraman, and G. Cazzaniga. "OpenFlow-enabled Transport SDN." https://opennetworking.org/wp-content/uploads/2013/05/sb-of-enabled-transport-sdn.pdf (accessed on 2023-06-29).

[11] F. Hu, Q. Hao, and K. Bao, "A Survey on Software-Defined Network and OpenFlow: From Concept to Implementation," *IEEE Communications Surveys & Tutorials,* vol. 16, no. 4, pp. 2181-2206, 2014, doi: 10.1109/comst.2014.2326417.

doi:10.6837/ncnu202300216

[12] L. Yang, R. Dantu, T. Anderson, and R. Gopal. "Forwarding and Control Element Separation (ForCES) Framework." https://www.rfc-editor.org/info/rfc3746 (accessed on 2023-06-29).

[13] R. Jain and S. Paul, "Network virtualization and software defined networking for cloud computing: a survey," *IEEE Communications Magazine,* vol. 51, no. 11, pp. 24-31, 2013, doi: 10.1109/MCOM.2013.6658648.

[14] "Understanding Dynamic VLAN Assignment Using RADIUS Attributes." https://www.juniper.net/documentation/us/en/software/junos/user-access/topics/topic-map/802-1x-authentication-switching-devices.html#id-understanding-dynamic-vlan-assignment-using-radius-attributes (accessed on 2023-06-29).

[15] P. C. B. Aboba. "RADIUS (Remote Authentication Dial In User Service) Support For Extensible Authentication Protocol (EAP)." https://www.rfc-editor.org/info/rfc3579 (accessed on 2023-06-29).

[16] "EAP over RADIUS." https://techhub.hpe.com/eginfolib/networking/docs/switches/5130ei/5200-3946_security_cg/content/485048061.htm (accessed on 2023-06-29).

[17] M. Pourzandi, M. F. Ahmed, M. Cheriet, and C. Talhi, "Multi-tenant isolation in a cloud environment using software defined networking," 2018. [Online]. Available: https://patents.google.com/patent/US9912582B2/en

[18] "OpenFlow Support in Open vSwitch." https://docs.openvswitch.org/en/latest/topics/openflow/ (accessed on 2023-06-29).

[19] "Ryu-SDN." https://ryu-sdn.org/ (accessed on 2023-06-29).

[20] D. C. Plummer. "An Ethernet Address Resolution Protocol." https://www.rfc-editor.org/rfc/rfc826 (accessed on 2023-06-29).

[21] B. Pfaff *et al.*, "The Design and Implementation of Open vSwitch," in *Symposium on Networked Systems Design and Implementation*, 2015.

[22] "Nginx." https://nginx.org/en/ (accessed on 2023-06-29).

doi:10.6837/ncnu202300216

附錄

附錄一 SDN Controller 動態 VLAN 程式碼

```
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 ether_types
from ryu.lib.packet import vlan
from ryu.lib import hub
import requests


access_port_url = "access_port_data"
trunk_port_url = "trunk_port_data"

# VLAN Definition class

class VlanDefinition:
    def __init__(self, *args):
        pass

    def To_Get_Access_MAC(self, dpid, vlan_id):
        mac = []
        access_port_data_url_1 = f"http://localhost:5000/{access_port_url}/{dpid}"
        url_read_1 = requests.get(access_port_data_url_1)
        access_port_data_1 = url_read_1.json()

        for port in access_port_data_1:
            access_port_data_url_2 =
f"http://localhost:5000/{access_port_url}/{dpid}/{port}"
            url_read_2 = requests.get(access_port_data_url_2)
```


doi:10.6837/ncnu202300216

```
                access_port_data_2 = url_read_2.json()
                if access_port_data_2 == vlan_id:
                    mac.append(port)
        return mac

    def To_Get_Trunk_Ports(self, dpid, vlan_id):
        ports = []
        trunk_port_data_url_1 = f"http://localhost:5000/{trunk_port_url}/{dpid}"
        url_read_1 = requests.get(trunk_port_data_url_1)
        trunk_port_data_1 = url_read_1.json()

        for port in trunk_port_data_1:
            trunk_port_data_url_2 =
f"http://localhost:5000/{trunk_port_url}/{dpid}/{port}"
            url_read_2 = requests.get(trunk_port_data_url_2)
            trunk_port_data_2 = url_read_2.json()
            if vlan_id in trunk_port_data_2:
                ports.append(int(port))
        return ports

    def This_Is_Access_MAC(self, dpid, vlan_id, port):
        access_port_data_url_1 = f"http://localhost:5000/{access_port_url}/{dpid}"

        url_read_1 = requests.get(access_port_data_url_1)

        access_port_data_1 = url_read_1.json()
        if port in access_port_data_1:
            access_port_data_url_2 =
f"http://localhost:5000/{access_port_url}/{dpid}/{port}"
            url_read_2 = requests.get(access_port_data_url_2)
            access_port_data_2 = url_read_2.json()
            if vlan_id == access_port_data_2:
                return True
        return False

    def This_Is_Trunk_Port(self, dpid, vlan_id, port):
        trunk_port_data_url_1 = f"http://localhost:5000/{trunk_port_url}/{dpid}"
```


doi:10.6837/ncnu202300216

```
            url_read_1 = requests.get(trunk_port_data_url_1)
            trunk_port_data_1 = url_read_1.json()

            if port in trunk_port_data_1:
                trunk_port_data_url_2 =
f"http://localhost:5000/{trunk_port_url}/{dpid}/{port}"
                url_read_2 = requests.get(trunk_port_data_url_2)
                trunk_port_data_2 = url_read_2.json()
                if vlan_id in trunk_port_data_2:
                    return True
            return False

Vlan_Definition = VlanDefinition()


class L2_VLANSwitch13(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super(L2_VLANSwitch13, self).__init__(*args, **kwargs)
        # Initialize mac address table.
        self.mac_to_port = {}
        self.datapaths = {}
        self.old_config = {}
        self.monitor_thread = hub.spawn(self._monitor)

    @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
    def switch_features_handler(self, ev):
        datapath = ev.msg.datapath
        self.datapath_2 = ev.msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
        self.datapaths = datapath

        # Install the table-miss flow entry.
        match = parser.OFPMatch()
        actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
```


doi:10.6837/ncnu202300216

```
                                                    ofproto.OFPCML_NO_BUFFER)]
        self.add_flow(datapath, 0,0, match, actions)

    def _monitor(self):
        while True:
            hub.sleep(3)

            try:
                access_port_data_url_1 = f"http://localhost:5000/{access_port_url}"
                url_read_1 = requests.get(access_port_data_url_1)
                access_port_all = url_read_1.json()
            except requests.exceptions.RequestException as e:
                self.logger.error('Failed ! %s',e)

            new_config = {}
            #inner_key_diff = set()
            inner_key_diff = {}
            diff = {}
            if not self.old_config:
                self.old_config = access_port_all
                print("--------------------------------")
                print("OLD_CONFIG : ", self.old_config)
                print("--------------------------------")
            else:
                new_config = access_port_all
                print("--------------------------------")
                print("NEW_CONFIG : ", new_config)
                print("--------------------------------")
                if self.old_config == new_config:
                    print("--------------------------------")
                    print("Same config")
                    print("--------------------------------")
                    pass
                else:
                    for key in self.old_config:
                        if key not in new_config:
                            diff[key] = self.old_config[key]
```


doi:10.6837/ncnu202300216

```
                else:
                    for inner_key in self.old_config[key]:
                        if inner_key not in new_config[key] or
new_config[key][inner_key] != self.old_config[key][inner_key]:
                            inner_key_diff = inner_key
                            if key not in diff:
                                diff[key] = {}
                            diff[key][inner_key] =
self.old_config[key][inner_key]

                self.old_config = new_config
                del_mac = None
                print("--------------------------------")
                print("Config has change")
                print(self.old_config)
                print(inner_key_diff)
                print("--------------------------------")

                print("--------------------------------")
                print("Go test function delete flow entry.")
                self.Delete_Change_Config_File(self.datapaths, inner_key_diff)
                print("--------------------------------")

    def Delete_Change_Config_File(self, datapath, inner_key):
        #ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        del_match = parser.OFPMatch(eth_src=inner_key)
        self.del_flow(datapath=datapath, match=del_match, table_id=0)
        del_mac = inner_key
        print("--------------------------------")
        print("Success delete flow entry !!!")
        print("--------------------------------")
        check = isinstance(datapath.id, str)
        print("Is the string ? ",check)

        if del_mac != None:
```


doi:10.6837/ncnu202300216

```
                del self.mac_to_port[datapath.id][del_mac]
                print("--------------------------------")
                print("Success delete flow table !!!")
                print("--------------------------------")
                pass
            else:
                pass


    def add_flow(self, datapath, priority, table_id, 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, table_id=0, match=match,
            instructions=inst)
        else:
            mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
            match=match, table_id=0, instructions=inst)
        datapath.send_msg(mod)

    def del_flow(self, datapath, match, table_id):
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        mod = parser.OFPFlowMod(datapath=datapath,
command=ofproto.OFPFC_DELETE, out_port=ofproto.OFPP_ANY,
out_group=ofproto.OFPG_ANY, match=match,table_id=table_id)
        datapath.send_msg(mod)

    def Flood_Packet_To_Vlan_Ports(self, msg, vlan_id, tagged,src,dst,out_port):
        datapath = msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
```


doi:10.6837/ncnu202300216

```
            in_port = msg.match['in_port']
            actions = []

            # If packet have VLAN ID, untag first.
            if tagged == 1:
                actions.append(parser.OFPActionPopVlan())



            dpid = datapath.id
            access_ports = Vlan_Definition.To_Get_Access_MAC(dpid, vlan_id)
            trunk_ports = Vlan_Definition.To_Get_Trunk_Ports(dpid, vlan_id)
            self.logger.info("flood packet: dpid ({}) vlan_id ({}) access ports ({}) trunk
ports ({}) ".format(dpid,vlan_id,access_ports,trunk_ports))

            # Judgment src or dst in access port
            if src in access_ports:
                actions.append(parser.OFPActionOutput(out_port))
            elif dst in access_ports:
                actions.append(parser.OFPActionOutput(out_port))
            else:
                pass

            actions.append(parser.OFPActionPushVlan(0x8100))
            vid = vlan_id | ofproto_v1_3.OFPVID_PRESENT
            actions.append(parser.OFPActionSetField(vlan_vid=vid))

            # Output to trunk port
            for port in trunk_ports:
                if port != in_port:
                    actions.append(parser.OFPActionOutput(port))

            data = None
            if msg.buffer_id == ofproto.OFP_NO_BUFFER:
                data = msg.data
            out = parser.OFPPacketOut(datapath=datapath, buffer_id=msg.buffer_id,
in_port=in_port, actions=actions, data=data)
            datapath.send_msg(out)
```


doi:10.6837/ncnu202300216

```
    def Forward_Packet_To_Vlan_Port(self, msg, vlan_id, out_port, dst, src, tagged):
        self.logger.info("forward packet to a port {}".format(out_port))
        datapath = msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
        in_port = msg.match['in_port']
        dpid = datapath.id
        actions = []

        # in port == access port, out port == access port
        if (Vlan_Definition.This_Is_Access_MAC(dpid, vlan_id, src) and
        Vlan_Definition.This_Is_Access_MAC(dpid, vlan_id, dst)):
            self.logger.info("Access port {} to access port {} : VLAN
{}".format(in_port,out_port,vlan_id))

            actions.append(parser.OFPActionOutput(out_port))
            match = parser.OFPMatch(in_port=in_port, eth_dst=dst, eth_src=src)
            if msg.buffer_id != ofproto.OFP_NO_BUFFER:
                self.add_flow(datapath, 1, 0, match, actions, msg.buffer_id)
                return
            else:
                self.add_flow(datapath, 1, 0, match, actions)

        # in port == access port, out port == trunk port
        elif (Vlan_Definition.This_Is_Access_MAC(dpid, vlan_id, src) and
        Vlan_Definition.This_Is_Trunk_Port(dpid, vlan_id, out_port)):
            self.logger.info("Access port {} to trunk port {} : VLAN
{}".format(in_port,out_port,vlan_id))

            actions.append(parser.OFPActionPushVlan(0x8100))
            vid = vlan_id | ofproto_v1_3.OFPVID_PRESENT
            actions.append(parser.OFPActionSetField(vlan_vid=vid))
            actions.append(parser.OFPActionOutput(out_port))
            match = parser.OFPMatch(in_port=in_port, eth_dst=dst, eth_src=src)
            if msg.buffer_id != ofproto.OFP_NO_BUFFER:
                self.add_flow(datapath, 1, 0, match, actions, msg.buffer_id)
```


doi:10.6837/ncnu202300216

```
                return
            else:
                self.add_flow(datapath, 1, 0, match, actions)

        # in port == trunk port, out port == access port
        elif (Vlan_Definition.This_Is_Trunk_Port(dpid, vlan_id, in_port) and
        Vlan_Definition.This_Is_Access_MAC(dpid, vlan_id, dst)):
            self.logger.info("Trunk port {} to access port {} : VLAN
{}".format(in_port,out_port,vlan_id))

            actions.append(parser.OFPActionPopVlan())
            actions.append(parser.OFPActionOutput(out_port))
            vid = vlan_id | ofproto_v1_3.OFPVID_PRESENT
            match = parser.OFPMatch(in_port=in_port, vlan_vid=vid, eth_dst=dst,
eth_src=src)
            if msg.buffer_id != ofproto.OFP_NO_BUFFER:
                self.add_flow(datapath, 1, 0, match, actions, msg.buffer_id)
                return
            else:
                self.add_flow(datapath, 1, 0, match, actions)

        # in port == trunk port, out port == trunk port
        elif (Vlan_Definition.This_Is_Trunk_Port(dpid, vlan_id, in_port) and
        Vlan_Definition.This_Is_Trunk_Port(dpid, vlan_id, out_port)):
            self.logger.info("Trunk port {} to trunk port {} : VLAN
{}".format(in_port,out_port,vlan_id))

            actions.append(parser.OFPActionOutput(out_port))
            vid = vlan_id | ofproto_v1_3.OFPVID_PRESENT
            match = parser.OFPMatch(in_port=in_port, vlan_vid=vid, eth_dst=dst,
eth_src=src)
            if msg.buffer_id != ofproto.OFP_NO_BUFFER:
                self.add_flow(datapath, 1, 0, match, actions, msg.buffer_id)
                return
            else:
                self.add_flow(datapath, 1, 0, match, actions)
        else:
```


doi:10.6837/ncnu202300216

```
                self.logger.info("Unknown event: in port {} to out port {} : VLAN
{}".format(in_port,out_port,vlan_id))

            data = None
            if msg.buffer_id == ofproto.OFP_NO_BUFFER:
                data = msg.data
            out = parser.OFPPacketOut(datapath=datapath, buffer_id=msg.buffer_id,
            in_port=in_port, actions=actions, data=data)
            datapath.send_msg(out)

        @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
        def _packet_in_handler(self, ev):
            msg = ev.msg
            datapath = msg.datapath
            ofproto = datapath.ofproto
            in_port = msg.match['in_port']
            dpid = datapath.id

            pkt = packet.Packet(msg.data)
            eth = pkt.get_protocols(ethernet.ethernet)[0]

            vlan_id = 0
            tagged = 0
            dst = eth.dst
            src = eth.src

            access_port_data_src_url_1 = f"http://localhost:5000/{access_port_url}/{dpid}"
            access_port_data_src_url_2 =
f"http://localhost:5000/{access_port_url}/{dpid}/{src}"
            url_read_src_1 = requests.get(access_port_data_src_url_1)
            url_read_src_2 = requests.get(access_port_data_src_url_2)
            access_port_data_src_1 = url_read_src_1.json()
            access_port_data_src_2 = url_read_src_2.json()

            access_port_data_dst_url_1 = f"http://localhost:5000/{access_port_url}/{dpid}"
            access_port_data_dst_url_2 =
f"http://localhost:5000/{access_port_url}/{dpid}/{dst}"
```


doi:10.6837/ncnu202300216

```
            url_read_dst_1 = requests.get(access_port_data_dst_url_1)
            url_read_dst_2 = requests.get(access_port_data_dst_url_2)
            access_port_data_dst_1 = url_read_dst_1.json()
            access_port_data_dst_2 = url_read_dst_2.json()

            # Fliter LLDP
            if eth.ethertype == ether_types.ETH_TYPE_LLDP:
                return
            # If packet have VLAN ID, record VLAN ID
            elif eth.ethertype == ether_types.ETH_TYPE_8021Q:
                vlan_hdr = pkt.get_protocols(vlan.vlan)[0]
                vlan_id = vlan_hdr.vid
                tagged = 1
            else:
                # If packet have't VLAN ID, confirm the src or dst mac address whether it is
in the control list

                if src in access_port_data_src_1:
                    vlan_id = access_port_data_src_2
                elif dst in access_port_data_dst_1:
                    vlan_id = access_port_data_dst_2
                else:
                    pass

            # Mac Address Table
            self.mac_to_port.setdefault(dpid, {})
            self.mac_to_port[dpid][src] = in_port

            print(self.mac_to_port)
            if dst in self.mac_to_port[dpid]:
                out_port = self.mac_to_port[dpid][dst]
                self.Forward_Packet_To_Vlan_Port(msg, vlan_id, out_port, dst, src,
tagged)
            else:
                out_port = ofproto.OFPP_FLOOD
                self.Flood_Packet_To_Vlan_Ports(msg, vlan_id, tagged,src,dst,out_port)
```


doi:10.6837/ncnu202300216

```
@set_ev_cls(ofp_event.EventOFPPortStateChange, MAIN_DISPATCHER)
def port_state_change_handler(self, ev):
    datapath = ev.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser
    change_port = ev.port_no
    del_mac = None

    for host in self.mac_to_port[datapath.id]:
        if self.mac_to_port[datapath.id][host] == change_port:
            del_match = parser.OFPMatch(eth_src=host)
            self.del_flow(datapath=datapath, match=del_match, table_id=0)
            del_mac = host
            break

    if del_mac != None:
        del self.mac_to_port[datapath.id][del_mac]
```


doi:10.6837/ncnu202300216

附錄二 VLAN 前置設定檔

設定使用者筆電的 MAC Address、VLAN ID，以及 Trunk Port 允許哪些 VLAN ID 通過，可隨時依據當時情境來更改。以下的設定檔為範例 ：

```
from flask import Flask, jsonify, request


app = Flask(__name__)


# 紀錄設定檔的內容
access_port_data = {
    1:{
        "00:00:00:00:00:01":10,
        "00:00:00:00:00:02":20,
        "00:00:00:00:00:03":30,
        }
}


trunk_port_data = {
    1:{
        1:[10,20,30]}
}


@app.route('/access_port_data', methods=['GET'])
def get_access_port_all():
    try:
        result = access_port_data
```


doi:10.6837/ncnu202300216

```
    except KeyError:
        result = "Not Found dpid"

    return jsonify(result)


@app.route('/access_port_data/<int:dpid>', methods=['GET'])
def get_Access_MAC_and_VLANID(dpid):
    try:
        result = access_port_data[dpid]
    except KeyError:
        result = "Not Found dpid"

    return jsonify(result)

@app.route('/access_port_data/<int:dpid>/<string:mac>', methods=['GET'])
def get_Access_VLANID(dpid,mac):
    try:
        result = access_port_data[dpid][mac]
    except KeyError:
        result = "Not Found VLAN_ID"

    return jsonify(result)

@app.route('/trunk_port_data/<int:dpid>',methods = ['GET'] )
```


doi:10.6837/ncnu202300216

```
def get_trunk_port_and_VLANID(dpid):
    try:
        result = trunk_port_data[dpid]
    except KeyError:
        result = "Not Found dpid"

    return jsonify(result)

@app.route('/trunk_port_data/<int:dpid>/<int:port_number>',methods = ['GET'] )
def get_trunk_vlanID(dpid,port_number):
    try:
        result = trunk_port_data[dpid][port_number]
    except KeyError:
        result = "Not Found VLAN_ID"

    return jsonify(result)

if __name__ == '__main__':
    app.run(debug = True)
```


doi:10.6837/ncnu202300216