基於 OpenFlow 交換機之跨雲端安全管理機制研究

陳奕明　　朱永彤
國立中央大學資訊管理學系
cym@cc.ncu.edu.tw
100423052@cc.ncu.edu.tw

摘要

在跨雲端環境，使用者及雲端供應商可以對不同雲端上的資訊進行存取，然而隨之而來的則是必須面對資訊洩露和資訊安全政策等更多的挑戰，跨雲端環境下，封包轉送須經過網際網路，若缺少妥善的安全機制，很可能遭到攻擊者竊聽封包，另外也須確保使用者所接收資訊符合自訂安全政策。現有網路設備及協定在跨雲端來臨下已顯不敷使用，也因此 OpenFlow 成為研究重點。OpenFlow 於跨雲端安全管理之相關研究尚未多見，本篇希望以 OpenFlow 實現跨雲端安全管理機制，以 OpenFlow 建構出跨雲端運算的環境；結合 Secure in-packet Bloom Filter，將轉送路徑資訊儲存於跨雲端銜接的路徑節點及封包中，在進行跨雲端傳送時，得以隱藏雲端內部資訊，避免資訊洩漏；同時也可利用 OpenFlow 達到使用者自訂的安全控制，符合跨雲端運算的安全需求。

關鍵詞：跨雲端, OpenFlow, Secure in-packet Bloom Filter

1. 緒論

雲端運算的蓬勃發展，目前已經有許多服務及技術在其上成形，許多的雲端供應商則以各自的技術提供了多樣的服務選擇[1]。根據美國國家標準技術研究所(National Institute of Standards and Technology)[2]的定義，依照雲端運算環境的部署方式將其分為三種：私有雲、公有雲及混合雲。混合雲結合了公有雲及私有雲的好處，透過將兩個或以上的雲端設施互相連結，並根據其相關技術及協議達到彼此的互通。在這種架構下，企業可以將一部份資訊工作交給其他雲端，而保留核心部分在組織內部。如此，企業使用者可能會在不同的雲端上存取資訊，例如奇摩(Yahoo) 會去租用新聞雲、氣象雲的資料，因此跨雲端運算的環境及服務已逐漸成形。在跨雲端環境中，必須讓使用者及雲端供應商將不同雲端上的資訊互相存取和使用，然而，隨之而來的可能是資訊洩露及資訊安全控制的挑戰。隨著雲端環境互動的增加，將會帶給雲端使用者和服務供應商在安全上的挑戰[3][4]。

隨著雲端環境轉型為跨雲端環境，在安全性的考量也成為我們必須要注意的焦點，如圖 1 所示為跨雲端運算的安全需求[3][4]。在跨雲端環境的傳輸，必須要能確保發送端到目的端所經過的網際網路(Internet)的安全性，因為一般封包路由都是基於 TCP/IP 協定技術，難保會遭到有心人士擷取封包導致網路攻擊發生。

使用者的傳輸資料，為了要都能符合合格的資訊安全政策，必須允許使用者控制雲端運算安全機制的運作，例如使用者的安全政策之一是要防止分散式阻斷服務攻擊(Distributed Denial of Service, DDoS)，對於接收的資訊則必須加上額外的過濾器做檢查，以免雲端內之機器遭到攻擊而導致服務中斷。在此情況下，系統必須允許使用者可在資訊流的路徑上加入 MiddleBox (例如入侵偵測系統、防火牆等)，對接收資訊進行過濾或檢查。

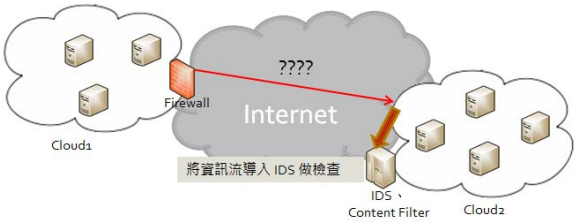

圖 1 跨雲端安全問題

除了上述的安全需求，跨雲端運算的環境中，也有效能上的需求[5]，下面為相關的描述：

(1) 流量需求的增加(Traffic between clouds)：

要能夠應付跨雲端之間增加的流量需求，因為在未來跨雲端運算環境的通訊流量會不斷地增加。

(2) 最小延遲(Minimum delay)：

隨著跨雲端環境擴大，其也會包含各種不同的雲端設施及服務，而有些服務可能會需要即時(real time)的溝通，所以即時通訊可能是必不可少的。

(3) 流量控制(Traffic control)：

跨雲端環境中包含有許多雲端設施，因此在傳輸封包時，不屬於通信兩端之網通設備、實體機器及使用者，就不應該收到封包，以免造成流量上的浪費或是安全疑慮。

近年來，面對不斷快速增高的網路需求，現有網路設備已瀕臨硬體傳輸效能的極限[6]，在跨雲端運算環境更顯不敷使用。因此，OpenFlow 已逐漸成為近年研究的重點項目。美國史丹佛大學所開發的 OpenFlow，已成功幫助 Google 處理急遽升高的網路流量需求，Google 將其佈署在跨雲端資料中心網路後，資訊傳輸流量不斷被擴大及優化[7]。本研究將使用 OpenFlow Switch 搭建出跨雲端環境，然而 OpenFlow 並未強調其安全性，因此我們將針對前述兩項安全需求，結合 OpenFlow 和 Secure in-packet Bloom Filter 方法，希望可以建構出跨雲端

環境的安全管理機制，同時利用 OpenFlow 來滿足跨雲端高速運算需求。

本篇後續章節如下：第二節探討本篇選用背景技術及相關研究；第三節描述系統的設計與架構；第四節為實驗與討論；第五節為結論及未來方向。

2. 相關研究

本節針對本篇所使用之相關技術進行探討，分別介紹 OpenFlow 專案與適用於跨雲端封包傳輸之 LIPSIN 和 Secure iBF 方法。

2.1 OpenFlow Switch

OpenFlow 可以讓研究人員測試自己設計的協定或政策，它是軟體定義網路的實現，將過去統一由設備執行的路徑控制與封包傳輸功能區分，讓路徑規劃以安裝 OpenFlow 控制軟體的控制伺服器（Controller）來進行，而封包傳輸則由 OpenFlow 交換器來執行。其以開放式網路管理架構來提高網路效能與彈性應用需求，協助網路管理員或服務供應商更為精確的網路管理能力。

OpenFlow 主要是透過 Controller 去管理底下的每台電腦的封包傳遞，OpenFlow Switch 主要包含了 Flow Table 以及 OpenFlow 相關軟體，Flow Table 負責封包查詢(Lookup)以及轉送(Forwarding)，其中的每個條目(Flow entry)都由表頭(Header)、統計資訊(Counter)以及對應動作(Action)所組成，表頭是用來比對封包用，包含有十個欄位。統計資訊用來維持此 Flow 的統計資訊。對應動作則指出符合此 Flow 的封包所應該做的動作；而 OpenFlow 相關軟體則負責與控制器(Controller)利用安全性通道(Secure Channel)互相交換資訊。

2.2 NOX – An OpenFlow Controller

NOX[8]為 OpenFlow 的控制器，當 OpenFlow Switch 面對未知流量時，NOX 控制器決定該流量的對應處理方式。

圖 2 為 NOX 網路組件圖，包含一台伺服器上安裝 NOX 軟體以及連接數台 OpenFlow Switch，使用者和網路管理者可以在 NOX 控制器上對底下任何一台 Switch 設定封包流向或是管理規則；其之上的管理元件(APP)，可以讓使用者自行選用或添加，根據需求設計控管策略或加入需要的功能或協定。

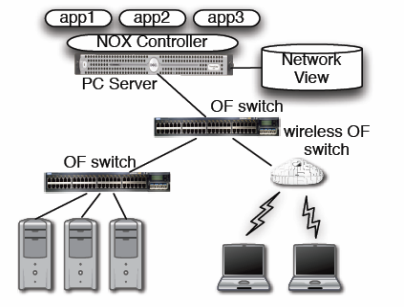

圖 2 NOX 網路組件圖

2.3 LIPSIN

LIPSIN(Line Speed Publish/Subscribe Inter-Networking)[9]為一個新興的多播轉發架構，很適合用在跨雲端傳輸中，它的運作機制是使用識別鏈結(Link)來取代節點，首先利用布隆過濾器的格式對每一個鍊結賦予一獨特鏈結編號(LinkID)，長度為 m 個位元(bit)長且其中的 k 個位元設為 1。LIPSIN 拓樸系統利用這些 LinkID 及可連接性的資訊，建立網路拓樸，如圖 3 示意。

圖 3 LIPSIN 網路拓樸及封包轉發

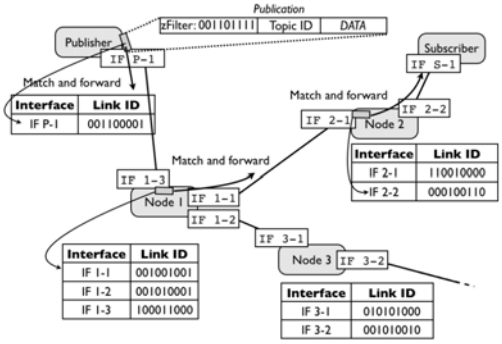

圖 3 為轉發示意，此情境中 Publisher 欲發送封包至 Subscriber，因此拓樸元件會生成轉送封包所需的 zFilter，而轉送路徑為透過 IFP-1、IF1-1 及 IF2-2 轉送，因此將路徑上 LinkID 利用二進制 OR 合併成為此轉送路徑的 zFilter。當各個轉送節點接收到封包時，便透過二進制 AND 比較 zFilter 和 LinkID 決定其傳出介面，演算法如圖 4 所示。

圖 4 LIPSIN 轉發演算法

Algorithm 1: Forwarding method of LIPSIN

Input: Link IDs of the outgoing links; zFilter in the packet header

foreach *Link ID of outgoing interface* **do**
  if *zFilter & Link ID == Link ID* **then**
    Forward packet on the link
  end
end

2.4 Secure in-packet Bloom Filter

LIPSIN 不具有管理機制，LinkID 一經決定好後就不會再更新，每個轉送節點都會紀錄一個固定的對映表。如此，攻擊者可能就可以偽造合法地 zFilter 對特定路徑注入流量或是以暴力法去分析破解 zFilter 進行攻擊。為了避免上述安全疑慮，Ghani [10][11] 提出 Secure in-packet Bloom Filter(SiBF)方法，又稱為 z-Formation。其方法設計上皆與 LIPSIN 的概念一致，唯其加入了加解密金鑰來動態計算 LinkID 和 zFilter，如圖 5 所示。

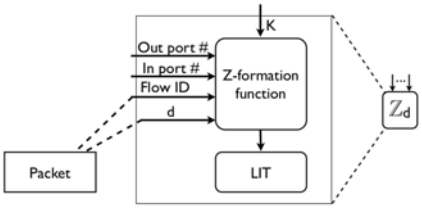

圖 5 Secure iBF function

比 LIPSIN 方法安全的原因是因為解密金鑰是由發送端發送的，只有指定的轉送點才有解密金鑰，當金鑰改變時，計算出來的資訊就會不同，也就是說 LinkID 是會變動的，zFilter 值也會跟著不同(依據金鑰不同而改變)，這種方式提供了更安全的保障(隱匿性)。

3. 系統設計與架構

3.1 目標

本篇目標為建立一適用於跨雲端的安全管理機制，因此針對以下兩點作為目標：1. 資訊加密：對封包進行加密，將封包內儲存之雲端內部資訊、位址加以隱蔽，避免成為攻擊者目標，使攻擊者僅能獲取有限的資訊，難以竊聽封包獲得雲端內部之相關資訊。2. 安全控制：讓使用者參與安全控制，允許使用者在雲端出入口加入 MiddleBox，在跨雲端傳送時將資訊流轉向做檢查，檢查後的正常資訊才可進入雲端內部。

3.2 系統架構

現有的研究方法中，於跨雲端運算皆有其不足之處，本篇提出的架構希望以截長補短的方式建立適用於跨雲端運算的系統架構。NetFPGA 搭建的 OpenFlow Switch 可以加速 Flow Table 查詢；透過控制器，方便建立跨雲端的網路拓樸及決定封包轉送路徑；金鑰則可透過控制器管理及向轉送路徑上的 OpenFlow Switch 進行分派；資訊加密方面，可利用 zFilter 將雲端內部及轉送資訊進行加密；安全控制方面，使用者則可以透過控制器，依需求對 OpenFlow Switch 制定安全政策。

依據上面描述，本篇之系統將以 OpenFlow 作為基底，圖 6 為 OpenFlow 網路示意圖，在各個雲端出入口皆放置一台 OpenFlow Switch，彼此之間透過網際網路連接，系統將結合 OpenFlow 和 SiBF 方法以達到資訊加密，保障在跨雲端傳輸時雲端內部之安全。而所有 OpenFlow Switch 皆透過 OpenFlow Protocol 接受一 NOX 控制器集中管理。

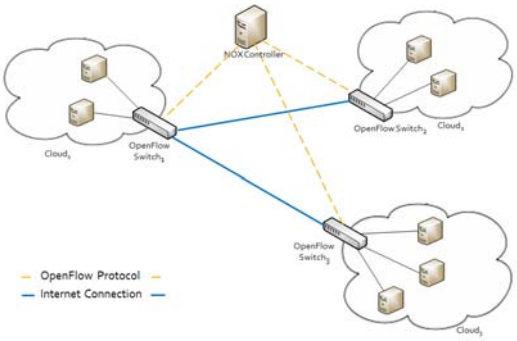

圖 6 OpenFlow 跨雲端網路示意

3.3 NOX 控制器

文獻中 zFilter 的生成倚靠的是其拓樸元件，在 OpenFlow 網路是透過 NOX 控制器集中管理底下所有的 OpenFlow Switch，因此適合用以維護網路拓樸、計算轉送路徑和產生 zFilter 及 LinkID，下面為相關功能描述：

(1) 建立 OpenFlow 網路拓樸：網路拓樸資訊用以作為封包轉送路徑的計算，NOX 控制器可透過發送封包探勘底下網路設備之連接資訊，建立 OpenFlow 網路拓樸。

(2) 找出封包轉送路徑：封包轉送路徑可藉由 NOX 控制器預設的 OSPF 路徑演算法進行計算。

(3) 產生 LinkID、zFilter：前項動作完成後，將路徑上所有 LinkID 進行二進制或(OR)，再以加密金鑰 K 進行 XOR 加密後作為轉送 zFilter。

(4) 回傳 LinkID、zFilter：LinkID 與 zFilter 的產生與回傳，皆是由 OpenFlow Protocol 來傳遞。NOX 發送命令寫入 LinkID 和 Key 至轉送節點上，以及將 zFilter 經由 OpenFlow Switch 再轉交至使用者。

3.3 OpenFlow 交換機

為了使 OpenFlow Switch 符合本篇之設計，能適用於跨雲端運算環境之中，下述為新增的功能：

(1) 增加 OpenFlow Switch 對 zFilter 的判斷：在接收到封包時，先提取封包標頭，將 zFilter 進行 XOR 運算解密後，再進行 Flow Table 查找，將儲存的 LinkID 與 zFilter 進行二進制 AND 後再依據結果對應之 actions 處理封包。

(2) 依據 Flow Table 查找結果進行對應動作：根據封包內 zFilter 與 LinkID 進行比對，比對後將封包透過指定介面傳出。

3.4 LinkID 與 zFilter

LinkID 與 zFilter 都在 NOX 控制器中完成，LinkID 產生方式為將一 64 位元變數，0 至 31 位元儲存來源端 IP 位址，32 至 63 位元儲存目的端 IP 位址，再用雜湊函數計算儲存位址，將變數中儲存位元設為 1 做為轉送路徑 LinkID。

zFilter 產生方式為將轉送路徑上所有的 LinkID 進行二進制或(OR)，再與金鑰(Key)進行邏輯異或(XOR)合併產生。

3.5 Flow Table

Flow Table 其上儲存的條目(Flow entry)比對項目僅為 LinkID 而行動(Action)為指定封包傳出介面。表 1 為 OpenFlow Switch 的 Flow Table 示意。

表 1 Flow Table 結構表

LinkID	Action
	Output:interface
10000100000001010101010010101010	1
10000100000001010101010011111111	2
00000100101000101110111010101101	3
00000000100000000001000000000000	4

3.6 系統運作流程

本節以情境闡述本篇系統運作流程。

3.6.1 網路拓樸更新

於文獻中，zFilter 的生成需仰賴拓樸系統，而在 OpenFlow 網路中，可利用集中控制的網路控制器來定期更新網路拓樸，以取得最新的封包轉送路徑。圖 7 為此情境之示意圖。其運作步驟如下：1.NOX 控制器對網路底下的 OpenFlow 設備送出 LLDP (Link Layer Discovery Protocol)探勘訊息。2.OpenFlow Switch 接收到訊息後會在傳遞此訊息至所連接設備，取得其連接情形。3.探勘完畢後，OpenFlow Switch 將探勘資訊回傳至 NOX 控制器，由控制器建立目前最新的網路拓樸。

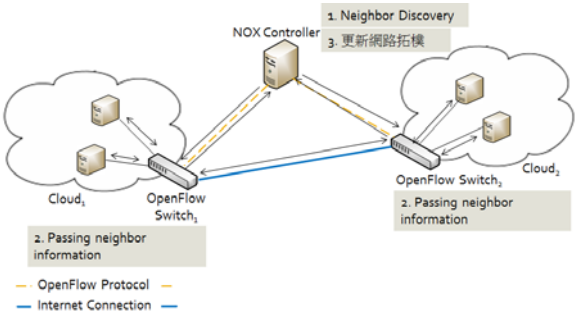

圖 7 OpenFlow 網路拓樸更新流程

3.6.2 LinkID 和 zFilter 值產生

本節假設於 OpenFlow 網路中，NOX 控制器已取得了最新 OpenFlow 網路拓樸，而使用者 $User_1$ 與 $Host_{1,1}$ 需取得彼此之間通訊所使用的 zFilter，圖 8 為此情境示意圖。步驟如下：1. $User_1$ 透過 OF $Switch_2$ 提出至 $Host_{1,1}$ 之間進行傳輸的 zFilter 要求。2. OF $Switch_2$ 將此要求轉送至 NOX 控制器。3. NOX 控制器根據網路拓樸找出可用之封包傳輸路徑，擇定具最小網路時延之 OpenFlow Switch 作為轉送端，於此情境中選擇了 OFS_2、OFS_1 作為轉送路徑。4. NOX 控制器將針對 OFS_2、OFS_1 更新其 Flow Table，生成 LinkID 及 zFilter($Host_{1,1}$,$User_1$)及指定 OF Switch 該使用哪個介面傳出封包。5. NOX 控制器計算出 LinkID，將路徑上所有 LinkID 進行二進制或(OR)，再與密鑰 K 進行邏輯異或(XOR) 生成 zFilter，將 K、LinkID 寫入至 OFS_1、OFS_2，並將 zFilter 經由 OFS_1 轉交至 $Host_{1,1}$ 完成 zFilter 產生流程。

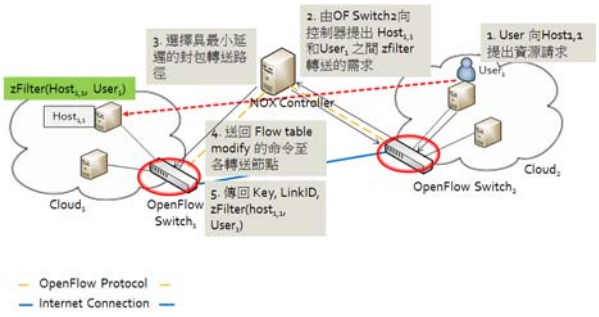

圖 8 zFilter 值產生流程圖

3.6.3 轉送封包

完成 zFilter 產生流程之後，此時 $Host_{1,1}$ 已可使用 zFilter 來完成與 $User_1$ 之間的通訊需求。圖 9 為此情境示意圖。步驟如下：1.$Host_{1,1}$ 將 zFilter 插入封包標頭中，傳送至 OFS_1。2.在此前 NOX 控制器已先將密鑰 K 及 LinkID 寫入至各轉送節點，OF $Switch_1$ 接收到封包後，提取封包 zFilter 值並與 K 進行邏輯異或(XOR)運算解密，接著進行 Flow Table 查詢，將儲存之 LinkID 與 zFilter 進行二進制 AND 運算後，依據結果依對應 actions 由指定介面傳出。3.OF $Switch_2$ 接收到封包後，同樣進行 zFilter 解密、Flow Table 查詢，比對 LinkID 與 zFilter 結果後依對應 actions 由指定介面傳出。4. $User_1$ 接收到封包，完成封包傳送。

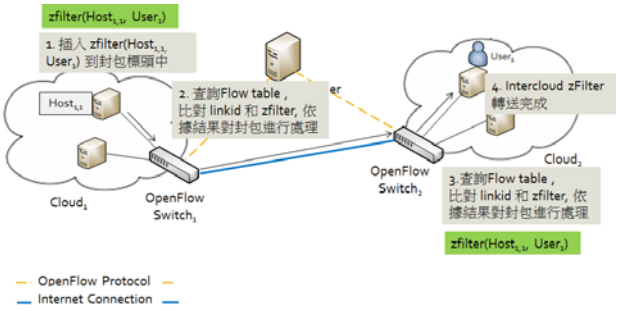

圖 9 封包轉送流程

3.6.4 安全規則制訂

使用者或公司行號於跨雲端環境進行運時，可能會面臨到不同的雲端彼此安全政策不一的形，使用者將無法確保自己的個人資訊，不會在雲端通訊時不經意的洩露。因此對於安全政策的統整管理可以利用 NOX 控制器對各個雲端出入口之 OpenFlow Switch 增設安全政策管理機制，使用者可依據需求去添加 MiddleBox 以符合安全規則。圖 10 為此情境示意圖。步驟如下：1.由控制器寫入規則指定接收封包的處理，須先由 Middle Box 做檢查。2.$Host_{1,1}$ 發送封包，OF $Switch_2$ 提取封包 zFilter 值並與 K 進行邏輯異或(XOR)解密，接著進行 Flow Table 查詢，將儲存的 LinkID 與 zFilter 進行二進制 AND，依據結果依對應 actions 由指定介面傳出。3.OF $Switch_1$ 接收封包後，依據 Flow Table 規則，將封包導向至 Middle Box，依據安全規則檢查封包，如果是不合法的內容則阻擋掉。4.正常封包再回傳至 OF $Switch_1$，再由 OF $Switch_1$ 轉送至目的端。

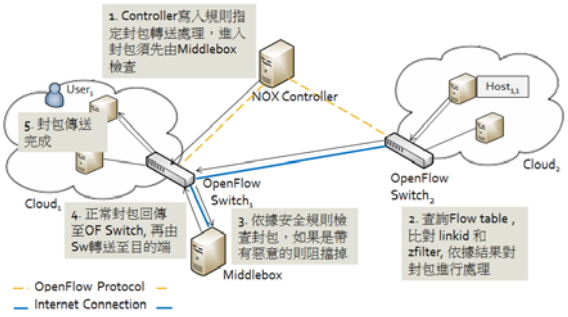

圖 10 自訂安全規則流程

4. 實驗與討論

本節以 SiBF 封包轉送、封包轉向至 MiddleBox 檢查兩項實驗評估系統可行性，來進行探討。

4.1 實驗環境

實驗環境如圖 11，依據第三章的系統架構及流程，分別架設了以下伺服器，兩台 Switch 皆連接到相同的控制器，OpenFlow Switch$_1$ 底下連接了一台 MiddleBox 主機用作封包轉向和過濾的測試。

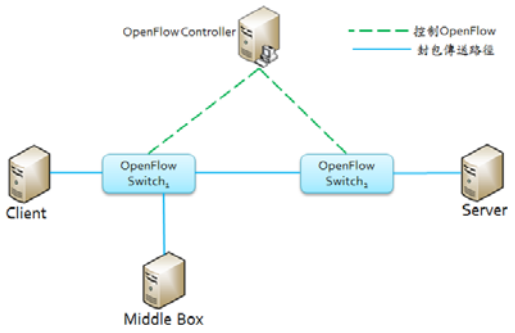

圖 11 實驗系統架構

4.2 系統架設及操作

OpenFlow Switch 及 NOX Controller 依照官方教學架設完成後，即可建立連線及開始測試，如圖 12。

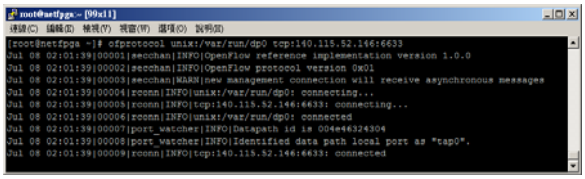

圖 12 OpenFlow 與控制器建立連線

4.3 封包轉送

4.3.1 NOX 程式流程圖

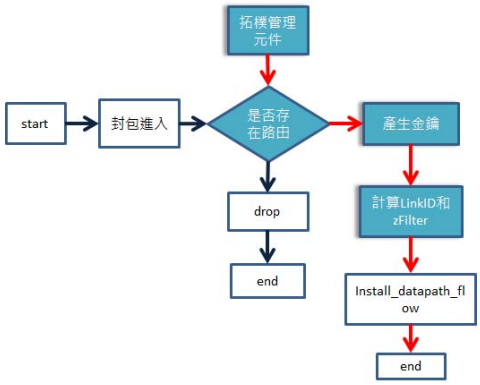

圖 13 程式流程圖

我們的目的是透過 LinkID 和 zFilter 來進行封包轉送，由 NOX 控制器負責計算和寫回 OpenFlow Switch，由 OpenFlow Switch 判讀接收的封包及轉送。圖 13 為 NOX 程式流程圖。白色的方塊是原有的程式，藍色的部分是新增上去的。新增的功能為管理及維護網路拓樸，當有封包進入後，會先找出封包轉送路徑，再產生加密金鑰計算 LinkID 和 zFilter，完成後透過 install_datapath_flow 將資訊寫回 Switch 上。而金鑰管理的部分，我們亦加入了時間參數，每 60 分鐘將進行金鑰的更新。

4.3.2 於 OpenFlow 轉送端上的處理

在 OpenFlow Switch 上要新增的功能為，查找 Flow Table 時，將 zFilter 值與 Key 作邏輯異或(XOR)，以及 zFilter 和 LinkID 做 AND 運算比較，此部份我們將修改原有的程式來達成此功能。

4.3.3 將封包轉向

要使封包轉向我們可以利用 OpenFlow Switch 提供的 dpctl 工具來新增、修改 Flow。我們要使得接收所有封包皆導向至連接的 MiddleBox 進行過濾，設定的指令範例如下：dpctl add-flow nl:0 in_port=1, actions=output:2。使得只要是來自這個網卡的封包，一律由 port2 出去，而 port2 底下所連接的是 MiddleBox，可以檢查封包。封包檢查完後會再回傳至 Open Flow Switch 上，由 Switch 進行 Flow Table 查找後再轉送封包至正確的目的端。

4.4 實驗一 SiBF 封包轉送

此實驗目的為針對 OpenFlow 結合 SiBF 方法後對封包轉送的功能進行測試，實驗中將運行兩台 OpenFlow Switch 和 NOX 控制器並進行連接，再透過 Server 向 Client 發送測試封包，並觀察其結果。

實驗採用的測試工具是 Colasoft 的 Packet Builder。Packet Builder 可以製造出 TCP、IP、ARP 封包，並且可以設定封包播送間隔及數量。我們以 Packet Builder 發送 1000 個封包，設定為 1 秒發送一次。發送完封包後，在 OpenFlow Switch 主機端打入指令查看封包統計，驗證封包是否正確從 Switch 轉送出去，以及使用 Wireshark 監聽封包轉送情形。

如圖 14 為 Switch 端之封包統計，而接收的封包數量不止 1000 個的原因是因為在 NOX 控制器上運行了 of_monitor 元件，該元件會定期地發送 LLDP 封包對底下連接的設備進行探勘，以保持最新網路拓樸。所以在 Switch 端查看封包統計時會有小量的額外封包。

圖 14 Switch 端封包統計

除了查看 Flow Table 的封包流量統計外，我們在以 Wireshark 查看封包轉送時的情形，以下為 Wireshark 監聽到的封包，tap0 即為 Switch 端，封包在轉送時會以 zFilter 取代封包 IP 資訊作為轉送的判讀資訊，可看到 IP 欄位是空的，如圖 15。

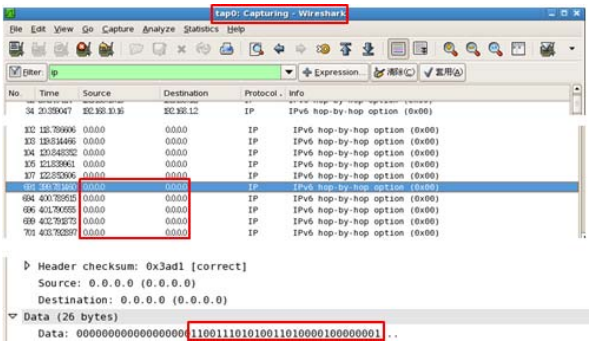

圖 15 Wireshark 監聽封包的情形

4.5 實驗二封包轉向至 MiddleBox

我們要讓封包在進入時被導到 MiddleBox 做檢查，在 MiddleBox 主機上同時有 eth1 和 eth2 兩張網卡，由 eth1 網卡去接收來自 OpenFlow Switch 的封包，eth2 則將過濾後的封包轉送回 Switch。MiddleBox 以安裝 Snort 及安全分析引擎（Basic Analysis and Security Engine, BASE）進行測試，而 Snort 主機為了考量安全性，避免遭受外部攻擊，因此以橋接(bridge)的方式將 eth1 和 eth2 連結，並不指定 IP，如此可避免 IP 資訊洩漏所導致的安全疑慮。接著由 PC2 使用 Nmap 掃描工具對 PC1 進行掃描，驗證是否可將封包流轉向至 Snort，並且進行攻擊來源主機的阻擋。當掃描完成後，如圖 16 查看 Snort 主機上的安全分析引擎可看到確實有封包進入。

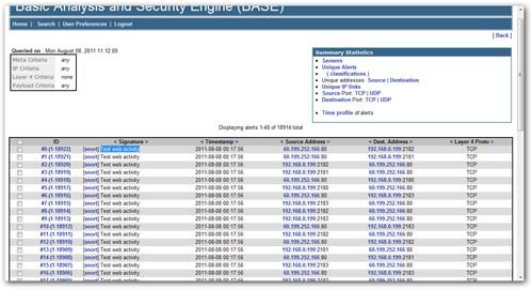

圖 16 封包紀錄至 BASE 資料庫

另外 Snort 也會記錄此 IP，並把 IP 傳遞至 iptables 的規則內，寫入一條新規則阻擋此來源位址的封包，如圖 17 所示。

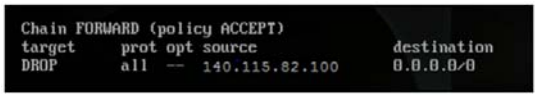

圖 17 iptables 阻擋攻擊主機

4.6 實驗結果討論

透過實驗證實了我們可以透過 NOX 控制器和 OpenFlow Switch 來控制封包的流動，我們可以利用 NOX 控制器找出轉送路徑及計算 zFilter 和 LinkID 並寫回 OpenFlow Switch 上，而在 OpenFlow Switch 也可以正確地透過 zFilter 轉送封包。也可以利用 OpenFlow Switch 將封包導向至 MiddleBox，達到封包過濾以及阻擋攻擊，保護雲端內部安全的目的。

5. 結論及未來研究

本篇所設計的系統結合了 SiBF 方法增強了 OpenFlow 於跨雲端上安全性，透過 zFilter，將封包轉送資訊加密避免雲端內部資訊洩漏，避免竊聽的攻擊。而對於使用者安全政策的統整管理則是利用 NOX 控制器對各個雲端出入口之 OpenFlow Switch 增設安全政策管理機制。使本篇系統更適於跨雲端環境之應用。

未來研究主要可往下列幾個方向：1.使用者會依不同需求配置不同的 MiddleBox，因此之後可測試更多的 MiddleBox 主機，評估是否適用於 OpenFlow 搭建的跨雲端網路中。2.只有一台控制器控管底下所有 OpenFlow Switch 可能會有效能上的問題，因此可將系統架構改為每一雲端皆有一台控制器控管雲內部，而所有控制器再交由一台集中控制器來掌管，負責對路徑、流量進行調配。未來若要整合不同地區的雲端設施，可嘗試將其改良為此種架構。3.對系統進行壓力測試，探討在高流量的運算需求下，是否能達到良好的效能表現，以及將封包轉向至 MiddleBox 檢查時，會不會因流量過大導致誤報或是遺失封包的情形，這是未來研究目標。

參考文獻

[1] 如何建構企業專屬的雲端運算平台,2010 年取自http://www.ringline.com.tw/

[2] Mell, Peter, Timothy Grance, "The NIST Definition of Cloud Computing (draft).", NIST special publication 800: 145, 2011

[3] Chen, Shiping, Suya Nepal, Ren Liu. "Secure Connectivity for Intra-Cloud and Inter-Cloud Communication", Parallel Processing Workshops (ICPPW), 40th IEEE International Conference on, 2011

[4] Kretzschmar, Michael, Mario Golling, Sebastian Hanigk. "Security Management Areas in the Inter-Cloud.", Cloud Computing (CLOUD), IEEE International Conference on, 2011

[5] Dayananda, M. S., Ashwin Kurmar. "Architecture for inter-cloud services using IPsec VPN." Advanced Computing & Communication Technologies (ACCT), 2012 Second International Conference on. IEEE, 2012.

[6] McKeown, Nick, et al., "OpenFlow: enabling Innovation in Campus Networks.", ACM SIGCOMM Computer Communication Review 38(2): 69-74, 2008

[7] Google Sets New Internet Traffic Record, 2010 年取自 http://ddos.arbornetworks.com/2010/10/google-breaks-traffic-record/

[8] Gude, Natasha, et al., "NOX: towards an operating system for networks.", ACM SIGCOMM Computer Communication Review 38(3): 105-110, 2008

[9] Jokela, Petri, et al., "LIPSIN: line speed publish/subscribe inter-networking.", ACM SIGCOMM Computer Communication Review, 2009

[10] Ghani, A. and P. Nikander, "Secure in-packet Bloom Filter forwarding on the NetFPGA.", Proceedings of the European NetFPGA Developers Workshop, 2011

[11] Rothenberg, Christian Esteve, et al., "Self-routing Denial-of-service resistant capabilities using in-packet Bloom filters.", Computer Network Defense (EC2ND), 2009 European Conference on, IEEE 2009