國立中山大學資訊工程系

碩士論文

Department of Computer Science and Engineering

National Sun Yat-sen University

Master Thesis

基於 SDN 環境中具動態閥值之 DDoS 防禦設計

Dynamic Threshold for DDoS Mitigation in SDN Environment

研究生：洪國智

Guo-Chih Hong

指導教授：李宗南 博士

Dr. Chung-Nan Lee

中華民國　108 年 7 月

July 2019

論文審定書

國立中山大學研究生學位論文審定書

本校資訊工程學系碩士班

研究生洪國智（學號：M053040014）所提論文

基於SDN環境中具動態閥值之DDoS防禦設計
Dynamic Threshold for DDoS Mitigation in SDN Environment

於中華民國 108 年 7 月 26 日經本委員會審查並舉行口試，符合碩士學位論文標準。

學位考試委員簽章：

召集人 李漢銘 ____　　委 員 李宗南 ____

委 員 賴威光 ____　　委 員 王友群 ____

委 員 龔旭陽 ____　　委 員 ____

指導教授(李宗南) ____ (簽名)

摘要

第五世代行動網路通訊系統(5th generation mobile networks)之發展正在進行中，除了無線網路設備、演算法之進步外，有線連線之軟體定義網路(SDN)發展也同步進行,本論文目的在於探討當今受矚目之 SDN 環境中可能面臨之(分散式)服務阻斷攻擊((D)DoS)的問題，由於 SDN 與以往網路之運作模式不同，具有中央控管之能力，因此能做出一些相較於現行網路架構下，除了原先可實現之手段外，新興的應對手段得以開發。本論文基於流量、Entropy 並藉由 SDN 之特性，提出可行之(D)DoS 防禦手法，並且透過平均數以及標準差來動態地獲得適合的流量、Entropy 判斷閥值，以正確地判斷攻擊。此外，高流量情況下，若非攻擊也將提供 Load Balance 功能來平衡各 Switches 之負載，降低環境傳輸延遲。結果顯示本研究所提出之方法能有效阻擋以高流量灌輸來阻斷服務的流量(flood)類(D)DoS 以及偽造 IP 之(D)DoS 攻擊，並且能夠平衡環境的負載。

關鍵字：軟體定義網路、資訊安全、服務阻斷攻擊、熵、負載平衡

Abstract

Software-Defined Networking (SDN) is one of the key technologies of 5th generation mobile networks (5G). However, like the traditional network architecture, SDN is also vulnerable to the Distributed Denial of Service (DDoS) attack. This thesis explores the dynamic threshold for DDoS attack in the SDN environment. Through the characteristics of SDN, we propose a feasible DDoS detection and defense mechanism. The proposed mechanism calculates the entropy of the network environment by the collected traffic status, and derives a dynamic threshold according to the network conditions to determine whether the environment is subject to DDoS attacks. In the event of a DDoS attack, our mechanism discards the traffic from the malicious nodes to the victim nodes with a flow entry. In addition, if no DDoS attacks occur in the environment, the proposed mechanism can disperse the traffic of the SDN switch, thereby balance the traffic load in the environment. The simulation results show that the method proposed in this thesis can mitigate the high traffic (D)DoS and fake IP (D)DoS, and also balance the load in the environment.

Keywords：SDN, Security, DDoS, Entropy, Load Balance

目錄

圖目錄

表目錄

第一章　簡介

隨著時間的演進，網路愈趨發達與成熟，如今 5G 網路即將面世，目前雖尚未有確切的標準制定，但依照「新世代行動網路聯盟（Next Generation Mobile Networks Alliance）」對於 5G 網路所定義的要求來看，5G 網路相比於 4G 網路將在各方面將有顯著之進步，勢必在未來能成為推動科技發展的一大助力。而當前 5G 網路發展分為兩部分，一為利用大量細胞網路(Cell)來組成整個網路系統，目的為提升網路覆蓋率及網路密度，並以小細胞(Small Cell)完成部分服務，減少大型基站(Base Station)處理過多流量的負載問題；另一則為利用軟體定義網路(Software-Defined Networking，SDN)來將封包轉送過程優化，其原理為利用遠端控制器(Remote Controller)控制 Switch，使其以特定或最佳路徑轉送封包，減少封包冗贅的轉送過程，達到提升轉送效率之作用。

資訊安全一直是各資訊領域中的重要課題，隨著科技越來越發達，安全性問題也將變得越來越不容忽視，其中 SDN 本身即存在一些安全性問題，在將來 5G 啟用後，倘若這些安全性問題尚未獲得解決，此問題將會在 5G 環境中延續，並衍伸出更多的資安問題，小自 IoT，大至雲端伺服器，都將可能因此造成無法挽回之損害。

1.1、 論文概述

本研究將把研究重心著重在 SDN 上之安全問題，由於 5G 通訊系統之標準尚未明確制定，因此 5G 之安全性研究相比於其他領域相對較少，但安全性問題向來是不可忽視之問題，尤其 SDN 依賴中央控制器以及新的通訊協議，可預期

的是，5G 將面臨更新且更嚴苛的資訊安全問題，雖然已有部分組織對於 SDN 網路架構提出安全願景及安全問題的預期處理方針，但也因 5G 網路尚未發展成熟而無法有較具體的處理方法。目前已有不少開源之 SDN 控制器，如 ONOS、OpenDaylight、Ryu、Floodlight 等，當中又以 ONOS 以及 OpenDaylight 較為廣泛被使用、推廣，本研究選用其中一種控制器進行研究，並透過該控制器來進行 SDN 之(D)DoS 相關研究。

1.2、 論文貢獻

本論文的貢獻主要有以下幾點：

1. 在相似的研究中，閥值之考量上有分固定閥值與動態閥值，其中固定閥值大多需要依照環境需求做調整，無法自動適應所有環境。動態閥值設計上，為了適應環境變化，會隨著環境改變而變動，通常依賴平均值或標準差來設計機制，但在環境流量穩態時，有誤判的可能，因此本論文也提出穩態時的解決方案，降低在穩態時誤判的可能性。

2. 考量 Entropy 能夠在大多數時候準確判斷是否有攻擊發生，但倘若僅考量 Entropy 則在低流量時，流量之些微浮動對於環境影響比高流量時大，也可能會有誤判之風險，因此本論文除了 Entropy 外，同時也考量到了流量，並動態的定義 Entropy 閥值。

3. 對於非(D)DoS 攻擊或網路流量高峰期之大流量，能以負載平衡方式，平衡環境中網路裝置之負載，使得硬體資源能更有效率地利用，也降低環境中之傳輸延遲。

4. 本論文提出之負載平衡方式同時考量路徑所經過之 hop 數與路徑之流量使

用率，相較於僅考量以上某一要素之方式，能夠選擇出更適合路徑，且能避免流量些微誤差所導致之不適當之路徑切換。

1.3、 論文架構

以下說明本論文組成架構，第二章為文獻探討，說明有關此論文所會提及到之技術、名詞；第三章為研究方法，主要說明本論文提出之系統如何實現；第四章為系統建置與實驗結果，說明本論文系統之硬體要求、工具等實驗環境相關事務，並展示本系統執行之成果後加以分析說明；第五章為結論，講述本論文所提出之方法、架構，以及成果是否合乎預想。

第二章　文獻探討

2.1、 第五世代移動通訊系統(5G)

第五世代移動通訊系統(5th generation mobile networks)，簡稱 5G，指的是移動通訊技術第五代，也是 4G (4th generation mobile networks)之後的延伸。「新世代行動網路聯盟」(Next Generation Mobile Networks Alliance，NGMM Alliance)在其 5G 白皮書[1]中提到的系統表現需求中，說明到資源與訊號效率須提升，並且要注意到未來 IoT 或是 MTC(Machine-type Communication)之應用會大幅提升訊號傳輸量，需要一個方法來解決此問題。而 Huawei 的 5G 架構白皮書[2]也提到，5G 須滿足三大需求，更高的帶寬(Enhanced Mobile Broad Band，eMBB)以達到更高的傳輸速度、更高的覆蓋率(Massive Machine Type Communication，mMTC)以滿足更多元的網路服務，並且有更低的延遲(Ultra-Reliable and Low-Latency Communications，uRLLC)來提升服務的可靠度，而此三項需求也將成為不同服務作網路切片(Network slicing)時所會用到的參考依據。滿足了此三大需求，則許多服務也將得以實現，像是需要高覆蓋率的智慧家庭、智慧城市，或是需要高可靠度低延遲的物聯網(Internet of Things，IoT)、觸覺互聯網(Tactile Internet)或是邊緣運算(Edge Computing)，這些或許是以往即有的服務概念，但礙於網路因素而難以完美實現，透過 5G 網路，這些服務將大有進展。

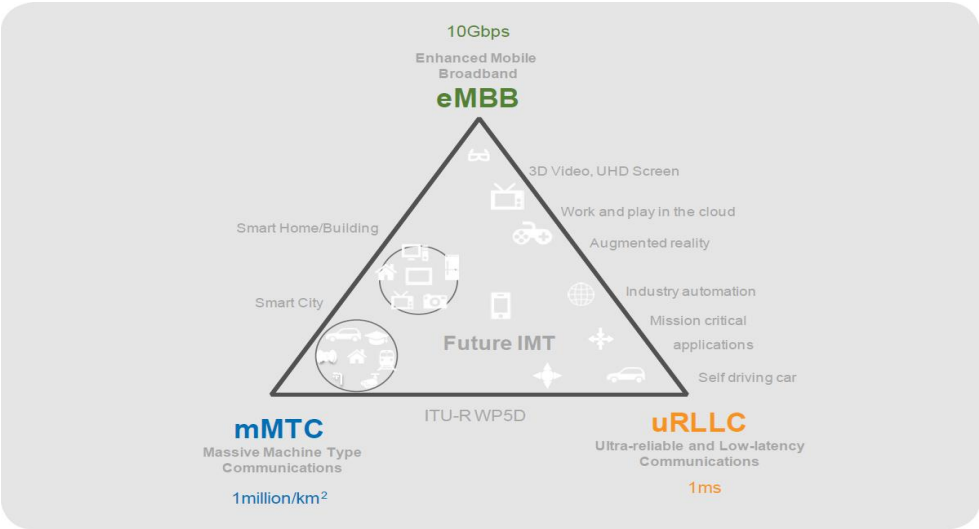

圖 2-1、5G 滿足三大需求[1]

從 5G 須滿足的需求來看，mMTC 之需求可以透過大量的細胞網路(Cell)所組成的網路架構來達成高覆蓋率之條件，此外，由於高速網路可以透過將訊號傳輸頻率提升來增加傳輸之訊號量，以換取每秒傳輸量之提升，但此舉將使得傳輸距離縮短，而大量的小細胞網路能解決此問題。

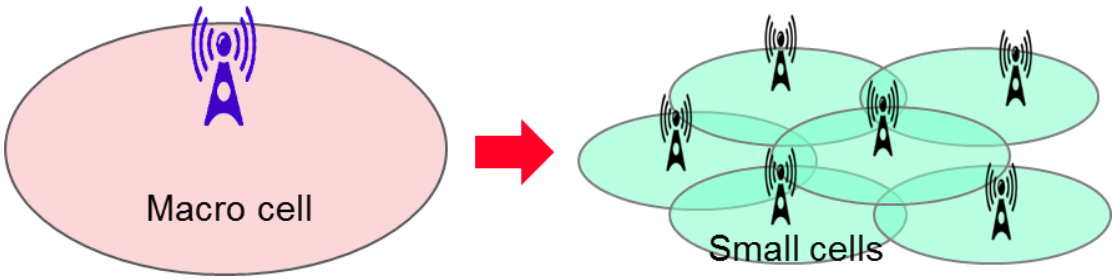

圖 2-2、以小細胞(Small cells)取代傳統大型基地台[3]

此外，提升網路速度的方式不僅僅只有以細胞網路提升覆蓋率的方式，也有藉由改善封包傳輸協議來提升封包轉送效率的方式，從而提升網路傳輸能力。

2.2、 軟體定義網路(SDN)

軟體定義網路(Software-defined networking，SDN)，是一種新的網路連線架

構，其具備快速、方便維護等特性，在未來 5G 網路追求更高網速、更低延遲的進程上，勢必能成為一大助力。SDN 不同於以往網路架構，其將路由器之控制層(Control plane)與資料層(Data plane，或稱 Infrastructure plane)分離，且以軟體方式實作控制層，以方便維護及部署。控制層分離後之 SDN 網路，透過遠端 SDN 控制器，網路管理者可以快速、方便地決定與執行位於轉送層的底層交換器(Switch)或路由器(Router)，讓它們知道該如何處理、傳送封包。

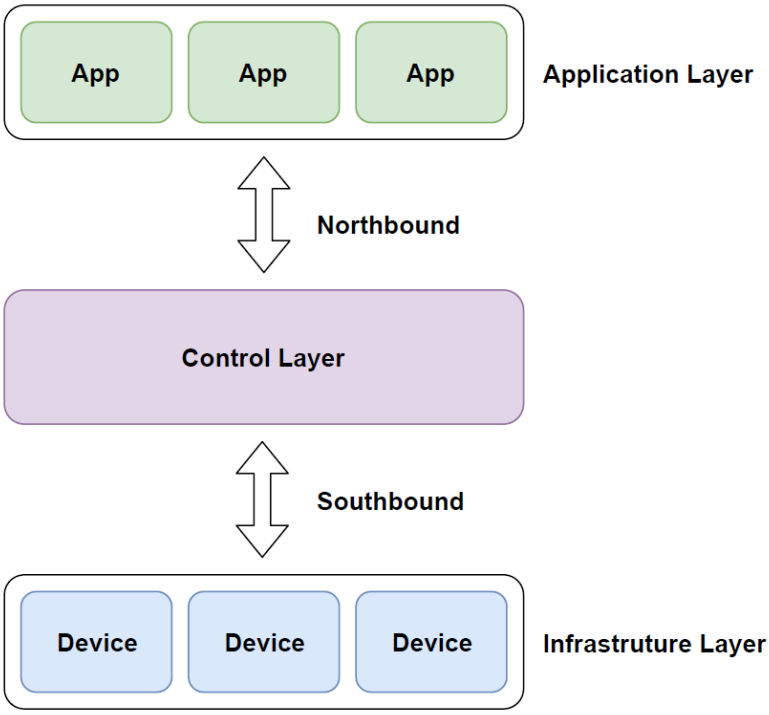

圖 2-3、SDN 運作模型

在 SDN 網路中，目前控制器與交換器之間溝通最常被使用的協定是 OpenFlow，並透過應用程式介面(Application Programmatic Interfaces，API)來進行，通常這樣的應用程式介面，稱為南向 API (Southbound API)。SDN 控制器除了透過南向 API 與底層的網路交換器互動，對於更上層的網路服務與應用程式的執行，也可透過開放的 API 來支援，這些 API 通常稱為北向 API (Northbound API)，可促進創新應用，並讓服務提供商以及設備供應商能夠更有效率地合作。

SDN 架構可以讓網路管理員或是開發者，在不更動硬體裝置的前提下，以

中央控制方式，用程式重新規劃網路，為控制網路流量、封包傳輸方式、轉送方式以及部署方式提供了更有效率的方法，也提供了核心網路及應用開發的良好平台。

2.2.1、 OpenFlow

OpenFlow [4]是一種網路協定，由 Open Network Foundation (ONF[5])制定，作用於網路七層模型的資料連結層(Data Link Layer)，被認為是實現 SDN 的重要協定之一。此協定目的是為了由遠端控制交換機或路由器的轉送平面(Forwarding Plane)，改變網路封包轉送之表格，進而從遠端設定網路封包的傳輸路徑。

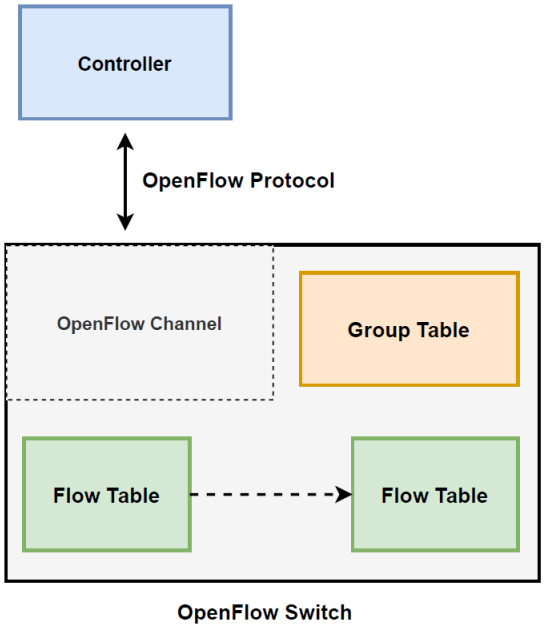

圖 2-4、OpenFlow Switch 與 Controller

如圖 2-4 所示，一個 switch 中包含一個 Group Table、一個 OpenFlow Channel 以及多個 Flow Table，並且一個 Flow Table 含有多個 Flow Entry，而每個 Flow Entry 皆如圖 2-5 所示，皆包含有 Match Fields、Priority、Counter、Instructions、Timeouts 以及 Cookie，而各區塊的功能如表 2-1 示。

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie	Flags

圖 2-5、Flow Entry

表 2-1、Flow Entry 各區塊功能

區塊	功能
Match Fields	與封包作匹配(Match)的部份，包含了送進來的 port、封包標頭、或可選擇加入一些 Metadata 作為匹配依據
Priority	該 Flow Entry 匹配的優先權
Counter	當有封包匹配成功時會更新，以做紀錄
Instructions	匹配之後的處理動作指令
Timeouts	Flow Entry 的最大存活時間或閒置時間
Cookie	供 Controller 使用，用於過濾流量統計、流量修改，以及流量偵測，此欄在處理封包時不會用到
Flag	Flag 的改變會更改此 Flow Entry 的管理方式

Group Table 功能與 Flow Table 相似，也和 Flow Table 一樣，包含許多 Group Entry，其中每個 Group Entry 都包含 Group Identifier、Group Type、Counters 以及 Action Buckets List，其功能如表 2-2 所示。

表 2-2、Group Entry 各區塊功能

區塊	功能
Group Identifier	32 位元的非負整數(Unsigned Integer)辨識碼，用以辨識屬於哪個 Group 用
Group Type	辨識屬於何種 Group，需要執行那些 Action Bucket
Counters	當 Group Entry 處理封包的時候，便會更新以做紀錄
Action Buckets List	由 Action Buckets 組成的清單，且有先後順序，每個 Action Bucket 包含多個執行動作(Action)

而 OpenFlow Channel (或稱 Secure Channel)是與遠端控制器溝通的橋樑，

Switch 與遠端控制器先建立 TLS (Transport Layer Security)通道後，才以 TCP (Transmission Control Protocol)的方式透過此通道交換訊息，此通道中的訊息皆為監控訊息以及狀態回報訊息，因此不須再像一般封包一樣匹配。

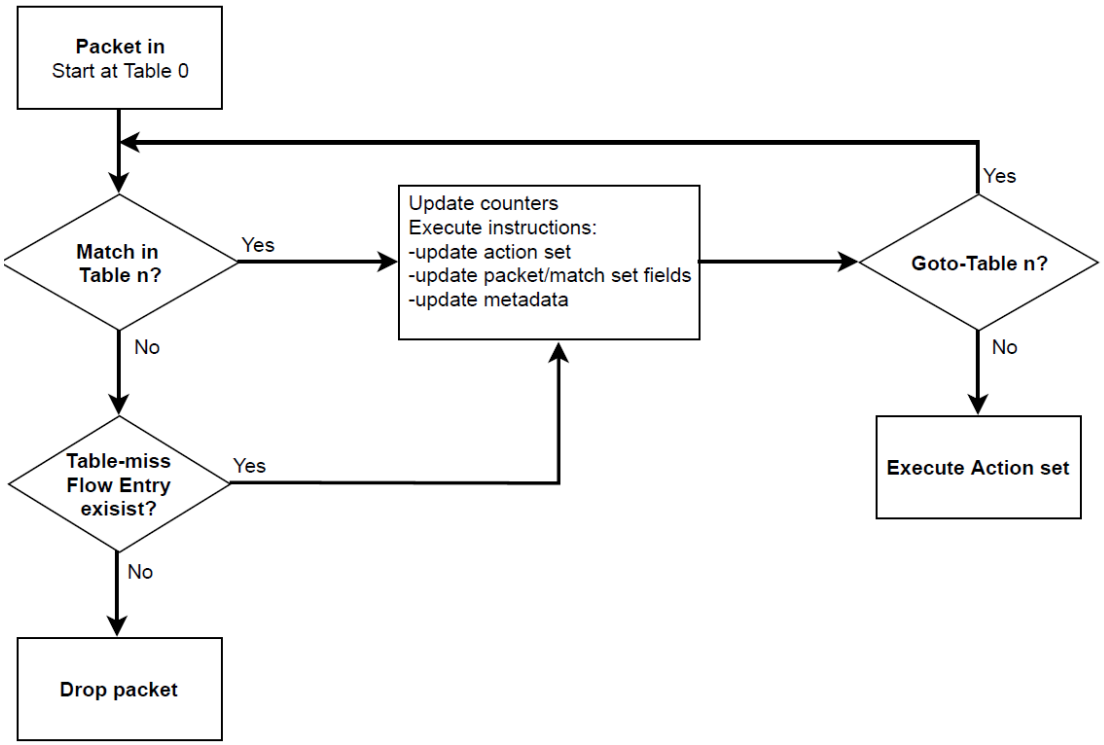

圖 2-6、封包匹配流程圖

當封包交由 OpenFlow 交換機轉送時，會根據交換機的 Flow Table 內容作匹配，如表----之內容，Flow Table 中的每個 Flow Entry 都有其優先權(Priority)。而各 Flow Table 間彼此也有優先順序，封包依照這些優些順序來跟 Flow Table 以及 Flow Entry 一一嘗試匹配。匹配過程如圖----所示，每一個 Flow Entry 都會有相對應的動作指令(Instructions)，封包如果匹配成功，則會執行相對應的動作指令。指令內容依照是否有 Goto-Table 動作，來區分為一般動作(Action)或是 Pipeline Processing。一般動作(Action)可能是轉發網路封包至另一個交換機，或是修改該封包內容，而 Pipeline Processing 代表的是有 Goto-Table 動作，該封包送到下一個 Flow Table 繼續嘗試匹配，而該封包相關的資訊也都會被傳到下一個 Flow Table。如果沒有 Goto-Table 動作，代表該封包不會再丟到下一個 Flow Table 繼

續匹配，此時通常都會直接針對這個網路封包做處理。處理動作可能是把網路封包直接捨棄、轉送至 Group Table 處理，或是透過 OpenFlow Channel 報告並交由 Controller 處理。一個 Group 可以包含多個複雜的處理過程，這些處理過程可能是 Flooding、Multi-path Forwarding、Fast Reroute，或是 Link Aggregation 等等。Group 甚至可以將多個不同的 Flow Entry 透過同樣的處理過程來對網路封包做相同的處理動作。

此外，一個 Flow Table 中可能會有 Table-miss Flow Entry，基本上是一個經過特殊設定的 Flow Entry，其特性與一般 Flow Entry 無異，並作用在封包無法與該 Flow Table 中之任何其他 Flow Entry 匹配時，最後由此 Table-miss Flow Entry 做處理，常見的 Table-miss Flow Entry 動作指令為將封包轉送至下一個 Flow Table 繼續匹配。但若一個 Flow Table 不存在 Table-miss Flow Entry，則無法成功於此 Flow Table 中匹配的封包將被捨棄。

2.2.2、 Open Network Operating System (ONOS)

Open Network Operating System [6]，簡稱 ONOS，是一款開源的 SDN 網路控制器，目標是提供 SDN 網路的 Control Plane，功能為管理交換機、連線等，並且此控制器提供軟體程序或模組運作的能力。

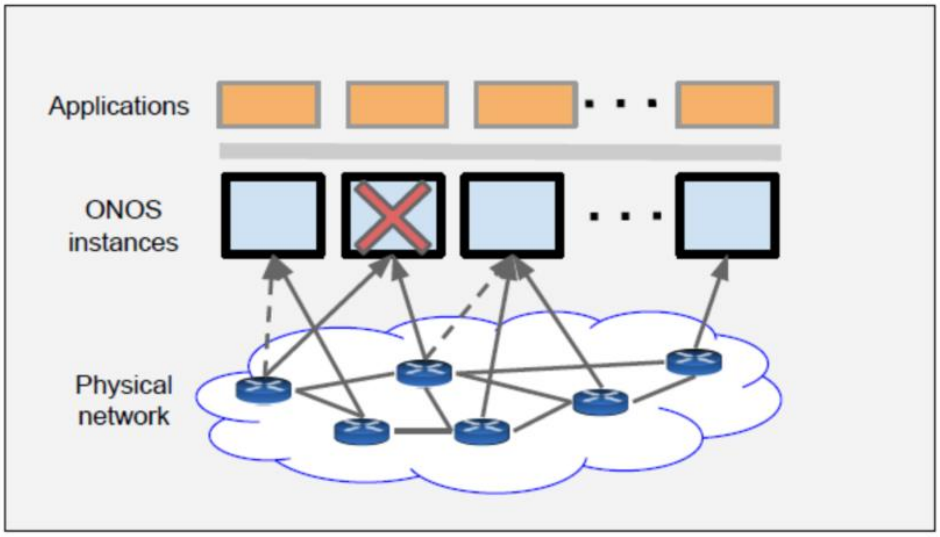

圖 2-7、ONOS 分散式核心[6]

如圖 2-7，ONOS 擁有能將多台 ONOS 控制器組成一個 cluster 的功能，以獲取更好的容錯能力，使效能及可靠性提昇，並且 cluster 內的各個控制器都同步進行相同的設定，而 cluster 內的各控制器不同於常見的 Master-Slave 運作模式，而是各網路設備對 cluster 內的控制器標示 Master 以及 Slave，當 Master 控制器失去連線則會從 Slave 挑選一個作為新的 Master 控制器。此外，ONOS 也可以在不中斷的條件下擴充 ONOS 控制器叢集，解決了擴充性問題。而 ONOS 上開發並運作之應用程式以及底下之網路設備也不需要得知此控制器是如何組成的，整個 ONOS 控制器叢集從外部角度來看就是一個 ONOS 控制器，而實際上 ONOS 控制器可隨時擴充或縮減其運算及儲存能力。

此外，ONOS 控制器還有一大特徵，軟體模組化，這是 ONOS 開發團隊非常重視的特徵，ONOS 是使用 OSGi [7]架構建置的，積極地將控制器的各功能模組化，目的是為了讓開發者得以方便地操作及配置 ONOS 中的功能，並且能夠簡單快速地進行開發，第三方開發者可透過引用各 ONOS 軟體模組或功能來快速開發其軟體，使其有快速拓展之特性，同時也能降低日後維護 ONOS 之難度。

2.2.3、 OpenDaylight

OpenDaylight [8]，為另一款開源的 SDN 網路控制器，是 Linux 基金會所提出之開源計畫，主要開發語言為 JAVA，能部署在任何支援 JAVA 的平台上，並且計畫成立主旨包含了簡化網路管理，故與 ONOS 同樣有模組化之特色，在維護上也屬方便。

此外，自從 Lithium 版本後，OpenDaylgiht 也能夠將多個控制器組成一個 cluster，提昇效能即可靠性。cluster 內的各控制器運作模式為常見的 Master-Slave 形式，cluster 內會挑選出一個控制器作為 Master，其餘的則分擔運算量，達到負載平衡作用，當 Master 失去作用時，也會從 Slave 中再次選出一個控制器作為新的 Master。

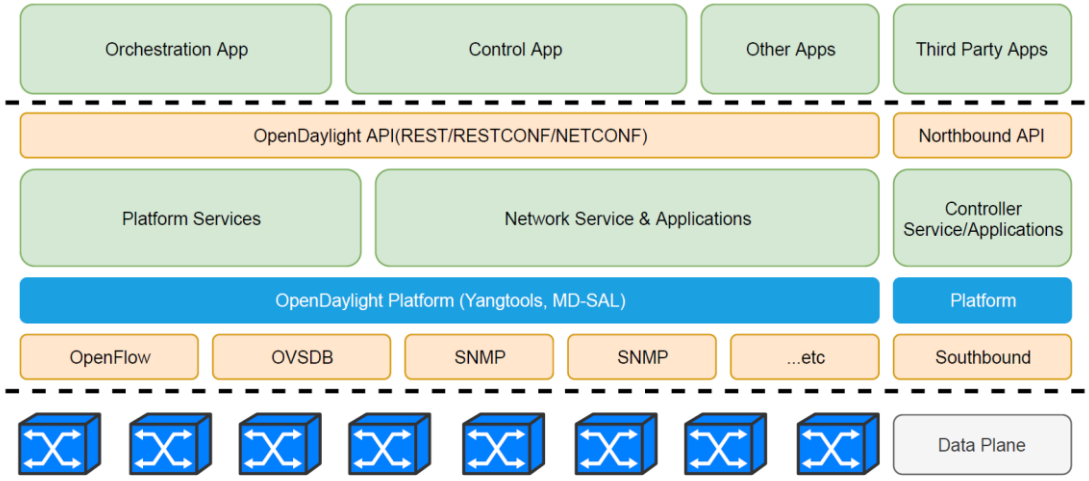

圖 2-8、OpenDaylight 架構

2.3、 阻斷服務攻擊

阻斷服務攻擊(DoS)為一種網路攻擊，目的是耗盡攻擊目標的網路或系統資源，使其所進行中的服務中斷或停止。而分散式阻斷服務攻擊(DDoS) 與 DoS 目的相同，皆為使目標的網路或系統資源耗盡，進而導致進行中的服務中斷或停止，

而不同的地方在於，此類攻擊為攻擊者使用多台主機作阻斷服務攻擊，可能用的是攻擊者自己的伺服器群，但較常見的是藉由網路上的殭屍電腦(Zombie)來共同發動攻擊。在論文[9]中說明到，(D)DoS 之防範重點在於辨別攻擊，這同時也是(D)DoS 難以防範的原因，欲採取相應對策前，需要能區分出流量是否正常、當前是否受到攻擊，故辨別攻擊一直都是防範(D)DoS 之關鍵，目前尚未有能完美辨別攻擊的偵測方式，故為熱門的安全性問題。

(D)DoS 並未能因為 5G 之網路速度變快，服務響應速度及能力獲得提升，而得到大幅改善，因為(D)DoS 為了達到阻礙服務之目的，攻擊手法不僅僅只有單純的堵塞頻寬，還有藉由協議漏洞來進行資源消耗之攻擊，因此在未來 5G 仍需擔心此問題。

(D)DoS 攻擊常使用到殭屍電腦(Zombie computer，或稱 botnet)，也稱作「肉雞」，指的是受到惡意軟體感染後，受惡意攻擊者控制的電腦，隨時依照攻擊者之控制而發動攻擊，並且通常受感染的電腦之擁有者沒有察覺自己的電腦已被感染。而(D)DoS 攻擊類型主要可分為兩類，分別為頻寬消耗型及資源消耗型，頻寬消耗型目的在於消耗攻擊目標之網路頻寬，使目標之網路頻寬塞滿，進而導致其無法進行其他連線而無法服務；資源消耗型之目的在於使攻擊目標之硬體或系統資源耗盡，使其系統難以處理其他工作，導致其無法穩定服務。常見的攻擊手法有 UDP Flood、TCP SYN Flood、反射放大攻擊…等，以下分別詳述各類攻擊手法。

a. UDP Flood

攻擊者故意傳送隨機 port 的 UDP 封包給攻擊目標，使目標必須尋找能在該 port 上處理該封包的程式，通常是沒有的，故回應 ICMP 封包給來源 IP，但攻擊者通常使用假造的來源位址，因此回應的 ICMP 封包則會在網路中傳遞，造成網路堵塞，並且攻擊者持續傳送此類封包，造成攻擊目標持續

尋找可處理之程式並回傳 ICMP 封包的忙碌狀態，導致其無法處理其他服務及連線，此為 DoS 攻擊的一種。針對 UDP Flood 攻擊，防火牆可設定 UDP 封包每秒允許通過數目的上限來抑制。

b. TCP SYN Flood

利用 TCP 協議中三向交握(Three-way Handshake)的協議漏洞來進行攻擊，惡意攻擊者藉由故意送出大量 TCP SYN 封包給攻擊目標，使目標回傳 TCP SYN/ACK 封包後，攻擊者故意不再回傳 ACK 封包給目標，使整個交握無法完成，而無法完成之交握內容會因為要進行重送，而暫時占用攻擊目標主機之運算資源直到交握的 Timeout 為止才釋放出這些運算資源，又由於攻擊者大量發出此類的交握，因此大量資源被佔據，攻擊目標將難以有足夠的資源穩定處理其他服務。此為 DDoS 攻擊的一種。針對此種攻擊，可降低 Timeout 的時間長度，以減緩此類攻擊的影響。

另外，分散式反射(Distributed Reflection Denial of Service Attack)攻擊，其原理與 SYN flood 相似，不同點在於攻擊者是藉由傳送 TCP SYN 封包給眾多實質存在的主機，並將來源位址假冒成攻擊目標，使得這些收到 TCP SYN 封包的主機回傳 TCP ACK/SYN 封包給攻擊目標，造成攻擊目標之網路資源及運算資源被耗盡，進而無法進行其他連線及服務。

c. Smurf

主要使用 Ping，利用廣播位址(子網路的每個 bit 皆為 1)來進行攻擊，攻擊者假冒攻擊目標之 IP，向某一網段的 router 發出 broadcast 的 ICMP Echo request 封包，並且因為是目的位址為廣播位址，router 會對該區網內所有電腦廣播此 ICMP 封包，此時所有區網內之主機會由於攻擊者之假冒位址而向攻擊目標送出 ICMP Echo reply 封包，導致這些 ICMP 封包在短時間內送往

攻擊目標，除了造成攻擊目標的網路堵塞外，也因為處理這些 ICMP reply 封包造成系統當機、暫停服務。並且由於這些封包傳輸路徑的關係，這些 ICMP 封包路過的路由器，其網域下的主機也會因為這些封包造成的網路堵塞而成為受害者。此類攻擊是很有效的利用體積小的網路封包造成大量封包流量的方式，並且同時能對系統及網路資源作出消耗，是非常巧妙的攻擊方式。但此攻擊目前已被 Router 擋下，帶有廣播位址的封包無法向外送出，只能在同一網段內發送。

d. Teardrop

攻擊原理是使用 IP 層定義封包分割重組之規則漏洞，攻擊者刻意製造不正常的封包序列，使封包在重組過程中發生問題，甚至可能導致部分系統當機而中斷服務。

e. Ping of Death

利用多數系統中常見之“Ping”功能，因為此功能是利用 ICMP 的 Echo Request/Reply 機制實現，又此類封包格式有 Option Data，其大小非固定，但規定上需小於 65535bytes，而攻擊者故意造出超過此容量之 Echo Request 封包，當送至目標主機時，作業系統因無法處理此種非法封包，進而造成當機。由於此類漏洞是系統上對於 ICMP 的處理漏洞，在一些新版的系統都已為此問題作修復。

f. DNS 反射放大

利用 DNS 查詢的漏洞進行攻擊，攻擊者假冒攻擊目標之 IP，向 DNS Server 送出域名查詢封包，之後 DNS Server 會回應給攻擊目標，並且 DNS 查詢之回應封包通常會比查詢封包大，少則 1~2 倍，多則 100 倍，故此攻擊又稱 DNS 放大攻擊。此攻擊通常以 DDoS 之方式進行。

由上述之描述並綜觀 5G 網路架構來思考 5G 之(D)DoS 問題，即可知道(D)DoS 並非完全是網路頻寬之問題，更有借助協議漏洞之攻擊方式，故未來 5G 網路仍需擔心(D)DoS，且由於 SDN 使用 flow rules 來控制封包轉送，因此控制器的封包流容易被識別，也使得 5G 之 SDN 控制器或許將成為(D)DoS 之熱門攻擊目標，且由於中央化的 SDN 網路控制，使得控制器之重要性大幅上升，一但癱瘓中央 SDN 控制器，整體網路之癱瘓也將無可避免，再者，SDN 控制器通常包含許多附屬之應用程式(Applications)，這些應用程式之安全漏洞也將造成不同層面之安全問題，可能將成為攻擊者之攻擊途徑。

2.4、 文獻探討

目前已有些機構以及論文在分析 SDN 的安全性問題，並分別提出安全願景以及預想的處理方式，在 Arbettu 等人[10]當中，分析了幾個目前較常被使用的 SDN 控制器，其中各個控制器當中都提供不同的安全性機制，使其各有能防治不同安全問題的能力，也因為機制的不同，各控制器所棘手的安全問題種類也相異，但這些 SDN 控制器共通點是，面對(D)DoS 攻擊之對策仍有待加強，並且從一些論文[11][12]可得知目前(D)DoS 尚屬 SDN 當中較為棘手之安全性問題，也因為攻擊手法多樣、透過殭屍電腦攻擊、難以辨別是否遭受攻擊，以及難以區分惡意流量等原因，導致目前尚無法完全防治，僅能透過減緩其對於整個系統及服務之損害來防禦，對於 SDN 之(D)DoS 應對上，也已有人開始著手研究，以下說明一些相關論文所提出之方法或系統。

Thomas 與 James [13]之主要應用場景為，協助於 SDN 環境中服務之 Server，於 Server 建立該論文所提及之防護機制。論文主要偵測方式為使用 iftop 作為流量偵測，直接在環境中以 iftop 蒐集流量，比較端點對端點傳輸吞吐量(req/resp)，

若比較值超過 1.5 則判斷該請求為惡意攻擊，而此論文對於攻擊之防禦機制為直接下達 drop 該流量之 Flow Entry 來達到防火牆之作用。該論文主要考量為流量，以流量比值作為判斷依據，且僅對 server 進行保護，倘若要保護多個對象則須分別為不同的對象架設這樣的防禦機制。

Lawal 與 Nuray [14]於 SDN 環境中，使用 sFlow 蒐集整個網路環境之流量，而偵測攻擊之方式為，當傳輸之流量大於預先定義好之閥值，則判斷此流量為攻擊，則下達 Flow Rule 以阻斷攻擊。此機制主要考量為流量，並且同時對於多個 Switch 進行偵測，能夠對整個環境進行評估，故也能保護多個對象。

Pande 等人[15]所提出之方法為利用封包轉送時交由控制器判斷如何安排路徑時，檢驗封包特徵(封包長度、種類等)，若為預先設定之可疑的特定種類封包則直接將其丟棄，ttl (Time To Live)錯誤也直接捨棄，封包長度大於預先設定之閥值也直接捨棄，若為 ARP 封包則判斷詢問 MAC 是否已在紀錄 table 中，若無則捨棄封包，通過以上過濾機制後，最後對環境之流量進行判斷，若流量高於閥值，則依照是否與受害者同區網做不同的防禦，先追溯攻擊來源，在判斷是否同區網，是則利用 Flow Entry 降低其傳輸速率，否則丟棄該流量之封包。此機制主要考量封包特徵與流量，由於為控制器原有之機制，理論上能對所有 Switch 進行偵測，故能保護多個對象。

Gkountis 等人[16]所提出之方法透過封包在 Switch 中尚未發現符合的 Flow Entry 則會先將其以 Packet_In 之方式送往 Controller 的特性，檢驗送往 Controller 之封包封包流量、大小，與閥值比對，進而得知是否當前有可能受到(D)DoS，依照狀況調整下發 Flow Entry 之 Hard_timeout 與 Idle_timeout，減少送往 Controller 查詢之封包流。此機制考量為封包特徵，並且主要保護對象微控制器。

Saifei 等人[17]所提出的方法是 SDN 控制器對於 Packet_In 機制以及送至控制器之封包的處理方法進行改善，以降低 DDoS 對於控制器的效能影響。更改控

制器對於送來之封包的處理方式，由一般的論詢法改為依照等待時間、序列(Queue)的長度以及未被攻擊的序列比例來算出權重值，在依照權重值來決定先處理那些封包。

Celesova 等人[18]所提出的方法分成兩部分，控制層的 DDoS 對策同樣是改善對於 Packet_In 與送至控制器封包的處理方法，每個請求者皆會有信賴度，並且有總請求數上限，若超過上限閥值則直接刪除請求者的請求，並更新在拒絕清單中，此外當信賴度低於惡意請求者的判斷閥值時，也會被判斷為攻擊者，並通知 Switch 捨棄掉該請求者的任何封包，而資料層部分則是以神經網路來判斷是否有攻擊，判斷攻擊後則會阻止惡意請求者。

You 等人[19]提出之方法也是透過 Switch 傳送 Packet_In 封包至 Controller 之特性來觀察環境流量狀態，蒐集各個 Packet_In 封包的來源 IP、目標 IP 以及目標 port 來計算 Entropy，並用常態分佈的經驗法則來得出閥值進行後續偵測。該論文藉由控制器原有之機制作為偵測方式，因此理論上也能對所有 Switch 進行偵測，而機制考量上為流量之 Entropy，但僅用 Entropy 判斷，在系統流量過低的狀況下可能會因為些微的流量浮動而有誤判的可能，且倘若整個環境流量狀況趨近穩態，也可能會有誤判的風險。

表 2-3、DDoS 相關文獻比較

方法	偵測媒介	判斷依據	閥值	防禦手法
R. M. Thomas and D. James [13]	iftop	輸入輸出之流量比值	「伺服器收到流量」除以「伺服器回傳流量」大於等於 1.5	阻斷該流量
B. H. Lawal and A. T. Nuray [14]	sFlow	流量	固定值	阻斷該流量
B. Pande et al. [15]	封包第一次送往控制器時檢	封包特徵(大小、速率、種類、ttl 等)	固定值	溯源並依照來源與目同或不同區網來採取

	驗			不同的防禦
C. Gkountis et al. [16]	控制器所收到之封包	流量	平均值	依照不同狀態修改 Flow Rule 的 timeout
L. Saifei et al. [17]	控制器所收到之封包	自定義之權重值	無(僅改善控制器封包處理之先後順序)	改善封包處理演算法以降低控制器所受之影響
B. Celesova et al. [18]	控制器所收到之封包	自定義之信賴度	固定值	將惡意用戶列為拒絕名單並阻斷該用戶
X. You et al. [19]	封包第一次送往控制器時檢驗	entropy	平均值 + 2 倍標準差	僅偵測
Proposed method	sFlow	流量、entropy	平均值 + 2 倍標準差，並訂定出最低差異值	溯源並下發 Drop 該流量的 Flow Entry

而 Load Balance 一直是資訊領域方面相當熱門且重要的一項議題，在高負載的環境尤其重要，故此方面之論文也已有不少研究，以下也說明一些 Load Balance 相關論文之方法。

Attarha 等人[20]所提出之 Load Balance 方式為判斷路徑中之負載是否超過預先定義之最大容忍值，其容忍值定義為帶寬的 0.7 倍，當負載超過帶寬的 0.7 倍即代表該條路徑出現壅塞，須尋找更加適合之路徑，而路徑蒐尋方式為找出所有可以自來源通往目標之路徑，並找出還能負載該流量之路徑，將流量引導至該路徑上。

Zakia 與 Yedder[21]所提出之 Load Balance 方法則是先以 Dijkstra's 最短路徑搜尋法找出自來源至目標的最短路徑，並從最短路徑中算出各路徑的最大使用率，選擇最大使用率最低的作為最佳路徑，將流量引導至該路徑上。

Nkosi 等人[22]所提出之 Load Balance 同樣以 Dijkstra's 最短路徑搜尋法找出自來源至目標的最短路徑，並在最短路徑中比較負載，負載低的路徑作為最佳路徑，將流量引導至該路徑上。

表 2-4、Load Balance 相關文獻比較

方法	判斷路徑切換條件	路徑尋找	判斷是否適合
Attarha et al. [20]	路徑負載超過 0.7 倍的帶寬(Capacity)	找出所有自來源至目標之路徑	找到之路徑尚有足夠的 Capacity 能負擔，若無適合路徑則不作任何更動
Zakia and Yedder [21]	找到路徑使用率(utilization)更低的路徑	以 Dijkstra's 演算法找出來源至目標之最短路徑	無判斷
Nkosi et al. [22]	找到負載更低的路徑	以 Dijkstra's 演算法找出來源至目標之最短路徑	無判斷
Proposed method	流量超過閥值，並且判斷非攻擊	計算路徑分數以選出適合路徑	算出當前路徑分數，並比較流量引導至新路徑後是否能讓新路徑分數比原路徑更好

第三章　研究方法

本論文主要針對流量(flood)類攻擊之系統，例如 UDP flood、ICMP flood 等會造成巨大流量之 DDoS 攻擊，並且對於偽造 IP 進行的攻擊也能應對。

本論文之系統主要利用 Entropy 發現是否環境中有(D)DoS 攻擊正在進行，若有，則追溯其可能來源，將攻擊來源往目標傳輸之流量，以下達 OpenFlow 之方式將其捨棄。此外，倘若環境中並未發生攻擊，但特定 OpenFlow Switch 負載過高，或是有尚未能夠察覺之攻擊發生，此系統也能進行負載平衡，將集中於特定 OpenFlow Switch 中之流量分散掉，進而平衡環境中之流量負載，減少傳輸延遲，以下詳細介紹本論文之方法與系統架構。

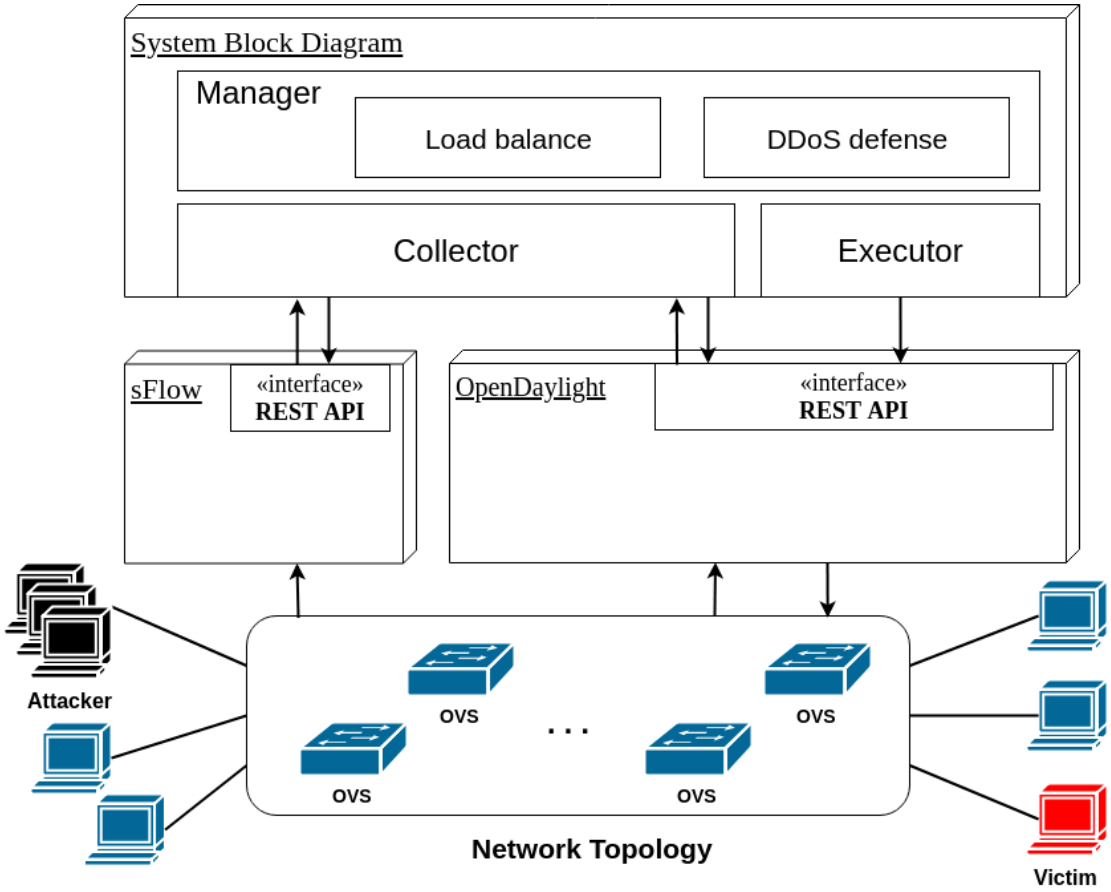

圖 3-1、架構實現示意圖

本研究所提出之(D)DoS 偵測防禦架構將包含三個模組，圖 3-1 所示即為本

研究所提出之架構，如圖 3-1 所示，本研究之程式屬於 Application 層，透過 REST API 向下與 sFlow、OpenDaylight 溝通，獲取拓樸及流量資訊，並透過 OpenDaylight 對 SDN 網路環境進行控管。

表 3-1、方程式參數定義

參數	定義
H	Entropy
n	流量來源之總個數
i	第 i 個流量來源
P_i	流量自第 i 個來源流向偵測目標之機率
f_i	偵測目標所收到來自第 i 個來源的流量大小(bytes)
f_t	偵測目標所收到之總流量大小(bytes)
μ	一段時間內之流量平均值
σ	一段時間內之流量標準差
$T_{traffic}$	流量閥值
$TL_{traffic}$	最低流量閥值
T_{Entropy}	偵測 Entropy 異常之閥值
H_{avg}	一段時間內之 Entropy 平均值
σ_{H}	一段時間內之 Entropy 標準差

3.1、 DDoS 偵測防禦

此小節將說明本論文所提出之系統偵測(D)DoS 之機制、主要參考依據，以及閥值如何決定。

3.2.1、 參考值

本論文之系統以抓取 Switch 之流量作為初步檢測，以 sFlow 觀測整個網路環境之各流量變化，並且從 OpenDaylight 抓取各 Switch 之總流量用以計算出定時流量，必要時能透過 sFlow 觀察更多內容，比如協議、流量來源，或是流量目標等細節，綜合以上這些蒐集值進行處理、分析，最後由系統產生決策，並作用於環境中。

3.2.2、 偽造 IP 偵測

本論文提出之系統包含偽造 IP 偵測功能，在系統初始化蒐集拓樸完成後，會參考拓樸資訊,將連接 Host 的 Switch 找出來,並對這些 Switch 個別開啟 thread 進行虛假 IP 偵測，偵測是否有 Host 以假的 Source IP 傳輸封包，目的為防止以假 IP 進行的 DDoS 攻擊。偵測的虛擬碼如圖 3-2 所示。

```
Function CheckFakeIP():
  1. ports ← get all ports connected to host
  2. get and record all IPs of the hosts connected to ports
  // continuously check if there is any host send packet with fake source IP
  3. while no exit do:
     a. flow_data ← get traffic data
     // check if there is any host send packet with fake source IP
     b. for all data in flow_data do:
        if source port of data in ports do:
           if the source IP isn't same to the recorded host IP do:
              drop the fake IP traffic
```

圖 3-2、偽造 IP 偵測虛擬碼

3.2.3、 流量比對

本論文所提出之系統，會將蒐集到之流量值與閥值進行比較，倘若流量值低於閥值，則代表環境正常，無發生任何的(D)DoS 攻擊或是高負載，而若流量值高過閥值，則代表環境可能發生異常，過多的負載可能代表著環境中正發生攻擊。

3.2.4、 Entropy

熵(Entropy)為一種熱力學的計算指標，原為計算能量退化以及一個系統失序的程度，但此計算方式也可套用在資訊領域，而 DDoS 攻擊的檢測即可以此指標作為判斷依據，分析整體系統環境受 DDoS 攻擊之可能性，以下說明 DDoS 攻擊如何以 Entropy 來檢測。

Entropy 可藉由 sFlow 以及 OpenDaylight 所蒐集到之流量狀態來計算得出，來判斷整個環境中是否受到 DDoS 攻擊，透過整個系統中流量的分布狀況來分析發生 DDoS 的可能性，公式(1)即為常見的 Entropy 計算公式。

$$H = -\sum_{i=1}^{n} P_i log_2 P_i \tag{1}$$

$$P_i = \frac{f_i}{f_t} \tag{2}$$

$$f_t = \sum_{i=1}^{n} f_i \tag{3}$$

在 DDoS 攻擊檢測中，公式(1)之 H 即為最終與閥值比較之值，其意義為不確定性之程度，即整個系統中流量分布的隨機程度；n 為偵測目標之來源總數(或目標總數)；P_i代表流量自來源 i 流向偵測目標之機率(或是偵測目標流向目標 i 的機率)，其定義如公式(2)所示，其中的 f_i即代表偵測目標所收到來自來源i的(或是偵測目標送往目標 i 的)流量大小(bytes)，f_t代表偵測目標所收到(或所送出)之總流量大小(bytes)。

此外，為了方便判斷，會將原 Entropy 計算公式在除上$\log_2 n$來歸一化，以將得出之值控制在 0~1 之間，因此實際上使用的計算式如式(4)所示。

$$H = -\sum_{i=1}^{n} P_i log_2 P_i \,/\, \log_2 n \tag{4}$$

公式(4)計算出的 H 之值越接近 0 即代表系統中的流量越集中來自某一來源

(或越集中送往某一目標)，也說明整個系統當中，有某些特定目標可能正遭受 DDoS 攻擊；反之，若 H 值越大，則可能代表系統中之流量流向越隨機，流量分布較平均，代表發生 DDoS 之可能性較低。

Entropy 所能使用的參考依據很多，來源 port、目標 port、來源 IP 以及目標 IP 等等，都可以作為 Entropy 參考資料，本研究將以 OpenDaylight 控制器，搭配 sFlow 持續蒐集各 Open vSwitch 之流量及流向，並定時計算出 Entropy 值，以判斷整個 OpenDaylight 控制器所控管之系統中是否有(D)DoS 正在發生，由於 OpenDaylight 控制器可以對於每個 Open vSwitch 之封包流向進行監控，可隨時得知當前的流量情形及整體流量之流向，依照流量流向、封包類型、是否特定特徵之封包數量上升，或是送至 OpenDaylight Controller 進行例外處理之封包特徵等資訊，判斷出目前可能正遭受何種(D)DoS 攻擊，依照可能的攻擊種類採取相對應之防禦手段，比如設定該特定流在一定時間內之流量上限、嘗試新增特殊 Flow Rule 來將惡意增加之流量引導至其餘地方或捨棄。

3.2.5、 閥值定義

本研究對於流量檢測之閥值並非固定值，而是隨著環境不同而自動調整，為動態閥值，閥值之定義方式為，儲存一段時間內之流量數據，並會持續更新，透過這些流量數據來計算出流量平均值μ以及流量標準差σ，計算方式如下式(5)及式(6)所示，N 表示所設定之參考流量值筆數，f_i代表第 i 筆紀錄之流量值。

$$\mu = \sum_{i=1}^{N} \frac{f_i}{N} \tag{5}$$

$$\sigma = \sqrt{\frac{\sum_{i=1}^{N} f_i^2 - \mu^2}{N}} \tag{6}$$

而常態分布[23]之分布狀況如圖 3-3 所示，利用常態分佈經驗法則：「一般情況下，會有 95%之值處於平均值加上兩個標準差之內」之特性，並且過低流量($\mu - 2\sigma$以下之流量值)不會對整個網路造成負擔，因此倘若流量皆為正常使用狀況，則將有 97.7%之流量值維持在兩個標準差內，故閥值T_{traffic}由下式(7)得出。

$$T_{\text{traffic}} = \mu + 2 * \sigma \tag{7}$$

倘若流量值超過T_{traffic}，即代表可能發生不正常之流量變化，並進一步計算 Entropy 來加以判斷，若判斷並非攻擊，則會進行負載平衡，並且同時系統會繼續儲存流量數據做為接下來的閥值計算用，換句話說，在確認了高流量並非惡意流量後，閥值也會因流量值之變動而上修，得出新的閥值。

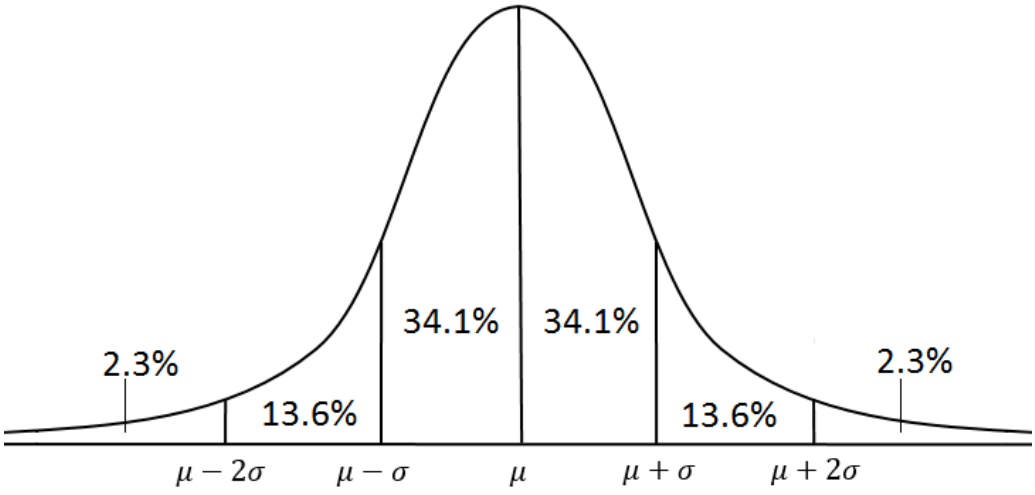

圖 3-3、常態分布

此外，流量閥值若完全依賴流量狀況，則在網路流量極低或無的情況下，初次接收較高的一般流量將導致系統誤判，若無做額外的修正，則將可能導致系統機制頻繁地被觸發而降低系統效能，因此閥值定義上還會再定義一個最低閥值TL_{traffic}，使T_{traffic}之計算式如式(8)所示。

$$T_{\text{traffic}} = \max\left(TL_{\text{traffic}}, \mu + 2 * \sigma\right) \tag{8}$$

於本研究之系統中，定義此最低閥值為 100K bps，理由為此閥值以下之流

量對於整個網路環境之效能並不造成嚴重影響。此外，由於機制上需蒐集固定時間內之流量數據，因此在系統初期初始化後，將有一段時間處於初步蒐集環境流量之狀態，此狀況下將不進行任何的流量判斷。

而本研究中判斷 Entropy 用之閥值(T_{Entropy})也為動態定義，同樣利用常態分佈的特性，以兩倍標準差來獲得，因此閥值之獲得方式如式(9)所示。

$$T_{\text{Entropy}} = H_{\text{avg}} + 2 * \sigma_{\text{H}} \tag{9}$$

由於T_{Entropy}隨著環境流量狀態而改變，因此在整個拓樸中之流量逐漸穩定而進入穩態的時候，將可能發生閥值無限趨近於平均 Entropy (H_{avg})的狀況，此乃 Entropy 標準差(σ_{H})逐漸趨近於 0 所導致，這將導致在穩態環境下，若 Entropy 有些微的變動，可能導致誤判的狀況，因此系統在初始化階段中，除了蒐集環境的流量狀態來獲得流量與 Entropy 閥值外，也需要一個最小的 Entropy 差值(d_{Lowest})，以幫助判斷穩態時的 Entropy 變化，目前依照實驗之經驗法則，將d_{Lowest}訂為 0.005。

3.2、 負載平衡

本論文所提出之系統中，也包含了負載平衡功能，此功能作用在環境流量大幅提升，並且檢測結果為無攻擊發生或是未偵測出攻擊之場景之中，期望能有效平衡環境中網路設備之負載，以達更佳的硬體資源使用效率，降低網路環境之傳輸延遲。此功能之虛擬碼如圖 3-4 所示。

```
Function LoadBalance():
  1. IP_pairs ← get all IP pairs that are transmitting traffic in the topology
  // load balance for all IP pair in the topology
  2. for all IP_pair in IP_pairs do:
    a. paths ← get all path from the source of IP_pair to the destination of IP_pair by networkx
    b. calculate the scores of paths
    c. opt_score ← the lowest score
    d. opt_path ← the path with the lowest score
    e. current_score ← get the score of current path of IP_pair
    f. threshold ← calculate the threshold to evaluate if the path change is necessary
    // evaluate if the path change is necessary
    g. if current_score – opt_score > threshold do:
        change path of IP_pair to opt_path
```

圖 3-4、Load Balance 虛擬碼

負載平衡功能作用於發現高流量，但經由個特徵判斷後，判斷並非攻擊，則進行負載平衡功能，本論文所提出之 Load Balance 機制為對 Controller 所控制之 switches 進行負載平衡，系統會偵測當前存在於拓樸中之所有 IP flow，對於所有的 IP flow，透過 networkx[24]先找出該 source IP 至 destination IP 之所有路徑，並估算各路徑之分數($Score_{\text{route}}$)，分數越低者越適合，其中 *Capacity* 為路徑之帶寬，為依照拓樸中的連線能力而預先設定之值，並且$Traffic_{\text{largest}}$代表路徑內之所有 Switch 中所偵測到最高的流量值；在找出分數最低之路徑後，計算將該 IP flow 之流量自原路徑移動至新路徑後，新路徑之分數與該流量在原路徑之分數相比是否更低，且低過於閥值，若是則將流量引導至新路徑，若否則該 IP flow 流量維持在原路徑傳輸。如此則可以找出最適合路徑，且避免因「些微流量差別導致之負載平衡流量轉移」造成負載更加失衡之狀況。

$$Utilization_{\text{route}} = Traffic_{\text{largest}} \,/\, Capacity \tag{10}$$

$$Score_{\text{route}} = \alpha \times \frac{hop_{\text{route}}}{hop_{\text{longest}}} + (1-\alpha) \times Utilization_{\text{route}} \tag{11}$$

$$0 \le \alpha \le 1 \tag{12}$$

計算路徑分數之主要算式如式(11)所示，而當中$Utilization_{\text{route}}$求法如式(10)所示，此計算方式之設計主要考量到路徑所經過之 hop(或稱 switch)數以及路徑

之使用率。考量 hop 數的原因為「當經過的 hop 數越多則 packet 所需之處理時間也會越多」，包含 switch 之負載狀態、packet 所需 match 之 Flow table 數量上升等等，皆可能造成更多的延遲，故經過之 hop 數必然是需要考量的；而使用率越高的路徑代表其負載狀態越壅塞，而如式(10)所示，在計算路徑使用率($Utilization_{\text{route}}$)時，$Traffic_{\text{largest}}$代表選用路徑所經之所有 switches 中所觀察到之最高流量作為代表來計算，此值所算出之使用率才能真正反映出整條路徑的負載狀態。並且式(11)中的hop_{route}代表該路徑所經過的 hop 數，而hop_{longest}代表所有路徑中，最長路徑所經過的 hop 數。而式(11)中之α值為一可變參數，使用此方法時，可隨使用場景、條件不同而更動此值，此值範圍為 0~1 之間，作用在於調適主要參考之條件之比例，路徑選擇少且不同路徑所經過之 hop 數量差異大之情況則建議使用較小之α值，反之則可選擇使用較大之α值，目前使用之α值為 0.05。

是否需要將流量導至新發現的最佳路徑上，則需要比對該流量在新路徑與原路徑的分數來得知是否轉移之後更好，因此抓出將被轉移的流量，評估該流量放入新路徑後，新路徑所得到的新分數是否高過於閥值，來確認是否需要將該流量移往新路徑，閥值部分如式(13)所示，$Utilization_{\text{new route}}$之定義方式如(15)所示，其中$Traffic_{\text{specific}}$代表預計將被轉移之流量，$Capacity$為路徑之帶寬，如此可以獲得預計被轉移之流量導向新的路徑後所造成之分數提升量，在該流量($Traffic_{\text{specific}}$)轉移至新路徑後，新路徑之分數相比於該流量($Traffic_{\text{specific}}$)在原路徑時之分數還低，且低過一定的比例才認為該流量($Traffic_{\text{specific}}$)移往新路徑之分數會比原路徑好，否則代表該流量($Traffic_{\text{specific}}$)切換至新路徑後效果差不多或更差，在此將提升量的 1/β 作為閥值，其中 β 為可調參數，值大於 0，本實驗中將 β 設定為 10，當原路徑之分數減掉流量轉移至新路徑後之新路徑分數小於$threshold_{\text{score}}$，則代表該流量移動至新路徑後，不比原路徑適合，或是效果差不多，故不做任何路徑更動。

$$threshold_{\text{score}} = ((1-\alpha) \times Utilization_{\text{new route}})/\beta \quad (13)$$

$$\beta > 0 \quad (14)$$

$$Utilization_{\text{new route}} = Traffic_{\text{specific}} / Capacity \quad (15)$$

3.3、 系統架構

此論文所提出之架構可以主要分為三個模組，分別為 Collector、Manager 以及 Executor，Collector 將會持續蒐集各 switches 所接收並傳遞之封包情形，並且將蒐集結果定期傳送給 Manager，而為了使管理者維護方便、Collector 還會保留一定時間內所蒐集到之流量紀錄，方便管理者觀察，協助修改安全策略；Manager 則為重要之決策模組，主要工作為依照 Collector 所提供之蒐集資料來判斷是否整個環境受到了(D)DoS 攻擊；而 Executor 則是對 OpenDaylight 進行控制之模組，依照 Manager 之指示，對 OpenDaylgiht 控制器進行控制，使其再往下對 OpenFlow switch 作用，保護整個網路環境。

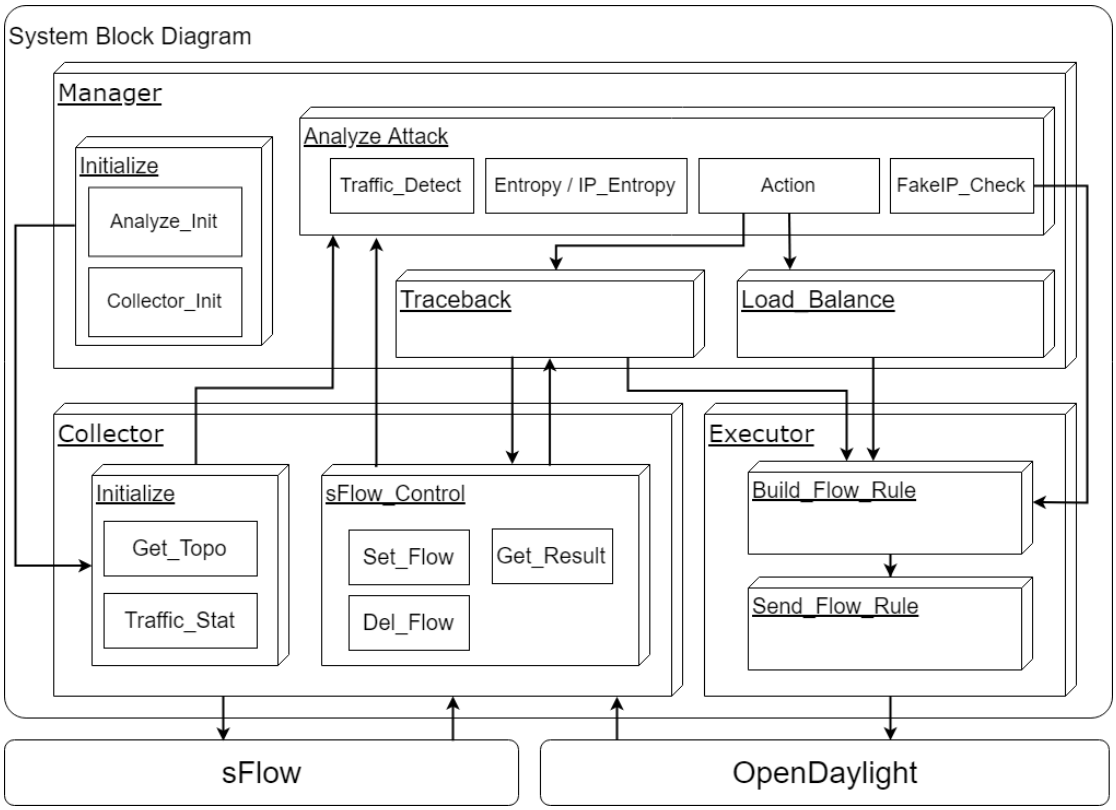

圖 3-5、程式細部規劃

圖 3-5 為本研究之程式細部規劃圖，Collector 之工作內容為初始化時建立整個 SDN 中之拓樸資訊，並於之後持續向 OpenDaylight 以及 sFlow 蒐集即時的封包轉送資料，定期將網路流量狀態之資料傳送給 Manager，使 Manager 得以判斷環境之安全狀態，並且也可藉由接收到 Manager 之主動要求而上傳目前所蒐集之封包轉送資料，因此 Collector 整體之運作流程圖如圖 3-6 所示。

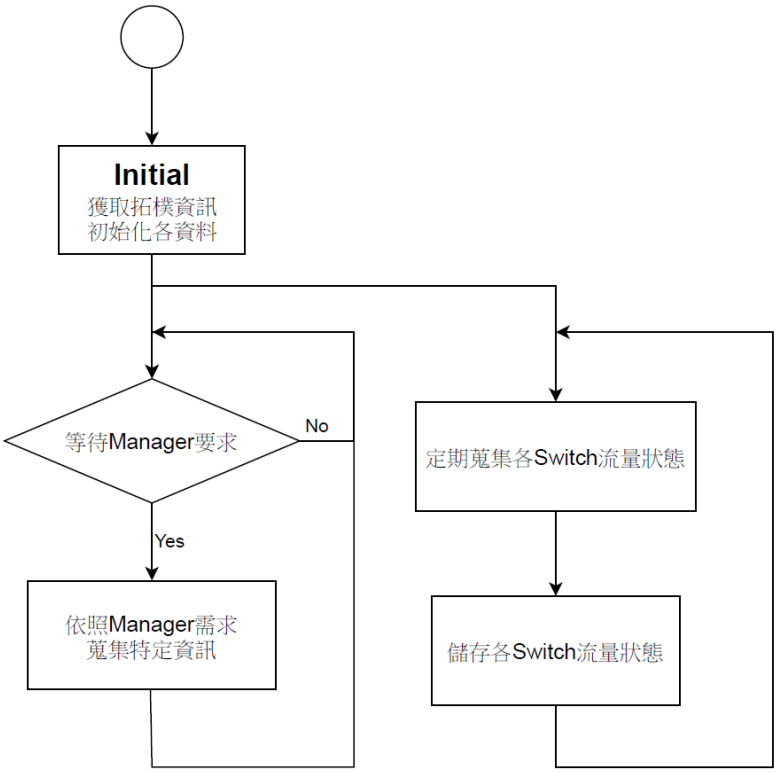

圖 3-6、Collector 流程圖

而圖 3-5 中之 Manager 則為此架構中，主要產生決策的模組，其開啟作為主要偵測之 Analyze 執行緒虛擬碼如圖 3-7 所示，以 Collector 上傳之封包轉送資料，配合 Entropy 計算來判斷是否可能有(D)DoS 攻擊正在發生，若有，則對可能正遭受攻擊之 Switch 要求接收之封包資料，並依照可能的攻擊種類，產生策略，並將策略傳送給 Executor，再由 Executor 依照策略控制及設定 Switch，若發生高流量，卻並未發現任何攻擊，則對網路環境進行負載平衡。

```
Function Analyze():
  // to initialize all parameters and learn the environment traffic state
  1. Initialize
    a. declare and initialize parameters
    b. collect and record normal traffic in the environment
    c. calculate and record normal entropy in the environment
  // start to analyze continuously if there is any malicious attack
  2. while not exit do:
    a. collect and record new traffic
    b. calculate and record new entropy
    c. calculate the threshold of traffic, entropy of source port, entropy of source IP, and entropy of destination IP
    // compare the new traffic and new entropy  with the thresholds
    d. if new traffic > threshold then:
        if difference between new entropy and threshold > lowest entropy difference then:
          traceback the attack, and drop the traffic
      else:
          load balance
```

圖 3-7、Manager 主要執行緒之虛擬碼

本系統之負載平衡方式為，對整個 Controller 所控制的網路環境內之所有 IP flow 進行負載平衡，對每一組來源 IP 至目標 IP，找出所有能夠自來源 IP 通往目標 IP 之路徑，以式(11)算出所有路徑之分數，找出分數最低者作為最適合之預選路徑，再比對流量轉移至新路徑是否比原路徑更適合，是則轉移，否則維持原路徑。

第四章　系統環境與測試結果

本章節講述本研究之模擬環境軟硬體需求、模擬環境以及測試結果。測試上將分為正常狀態以及模擬惡意攻擊來進行，在正常狀態下，測試系統是否會發生誤判，並且在高流量且未發生攻擊時，能否進行負載平衡；而惡意攻擊環境中，首先鋪設底層正常流量，並於一段時間後注入惡意攻擊，測試系統是否能正確抓出攻擊，並成功保留原先鋪設之正常流量，以驗證系統機制設計是否成功。

4.1、　軟硬體需求

本研究之實驗以模擬的方式進行，因此僅在單一主機內即可架起整個模擬環境，本研究使用到之主機規格如表 4-1 所示。

表 4-1、模擬所用之主機規格

項目	規格
系統	Ubuntu 16.04
核心	Intel Core i7-6700M (4 cores)
RAM	16GB
ROM	750GB

本系統之實現環境，對於控制器之需求僅需能夠下達 OpenFlow 控制 Open vSwitch[25](以下簡稱 OVS)以及藉由 REST API 進行溝通與控制，而本實驗最終決定選用 OpenDaylight 作為實現用之控制器，理由為相較於 ONOS，OpenDaylight 些許輕量化，能依照自己需求安裝需要的模組與套件，ONOS 則是將所有模組套件都安裝在內，安裝過程雖較為簡便，但也因此多了更多與實驗過程無關的模組。雖本實驗最終選用 OpenDaylight，但如上所述，本研究僅需要能透過 REST API 溝通與控制，並支援 OpenFlow，基本上即可用於實現本系統，故常見的控制器如 Ryu、Floodlight 等也是可行的。

表 4-2、OpenDaylight 硬體需求

項目	最低規格	建議規格
核心	2 Cores CPU	4~8 Cores CPU
RAM	2GB	8GB
HDD	10GB	64GB

而 OpenDaylight 之硬體需求如表 4-2 所示，除了此硬體需求之外，由於 OpenDaylight 控制器之安裝過程需求，以及使用模擬實驗時，常搭配一些軟體共同使用，因此對於實驗環境中軟體部分也有一些基本要求，表 4-3 即為本研究之系統環境建構需求，而這些環境需求之軟體安裝方式已有許多教學，故安裝過程在此不贅述。

表 4-3、系統環境建構需求

項目	版本
Java	8 以上(執行環境需求，較建議 Oracle1.8)
Karaf	任一版本(CLI 以及安裝需求)
Maven	任一版本(模擬時常搭配使用)
Mininet	3.2 以上(使用時常搭配使用，建議 3.3 以上)

4.2、 模擬環境

本研究選用 OpenDaylgiht 控制器，並搭配 Mininet[26] 3.3 版本，因 Mininet 在 3.2 版本以上之版本中，已可提供可運作 OpenFlow 協議之 OVS，因此以 Mininet 作為模擬網路拓樸之工具，如圖 4-1 所示，利用多個 OVS 與虛擬機(Virtual Machine)進行拓樸，OVS 底下連接數台虛擬機，並將這整個拓樸的控制交由 OpenDaylight Controller 來進行，由 OpenDaylight 來對整個網路的 Data Plane 做監測及控制，並以底下之虛擬機進行封包發送與接收來進行整個研究模擬。

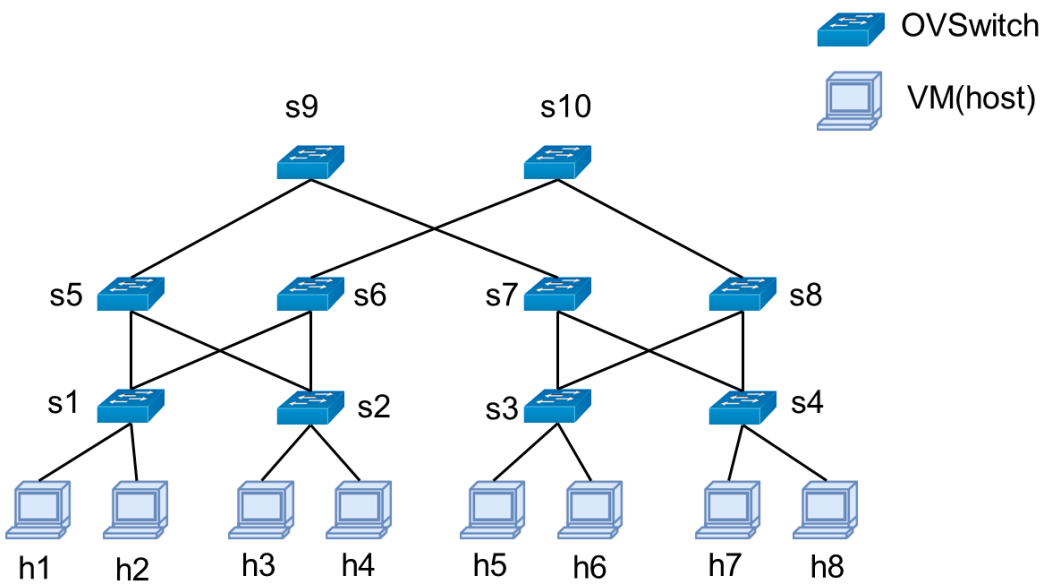

圖 4-1、研究模擬用之拓樸

圖 4-1 所示為本研究所模擬之拓樸，為使用 Mininet 所建立之拓樸，其中 h*i* 表示主機 *i*，s*j* 表示 OpenFlow 交換器 *j*，並且 h1~h8 之 IP 位址分別為 192.0.0.1~192.0.0.8，物理位址為拓樸產生時隨機產生，並且每條連線都設定 1GBps 為最大的傳輸速率，從此圖可以看到本研究所使用之拓樸大致上屬於樹狀拓樸，較為不同的地方是，連接之 OVS 多了一組備用 OVS (e.g. s1→s5→s9→s7→s4 對比 s1→s6→s10→s8→s4)，此為因應本系統中之負載平衡功能之需求，證明本系統之負載平衡功能是否有實現而設計。

而實驗中所提到之封包流量傳輸，乃透過 python 之函示庫 Scapy[27]來撰寫腳本，Scapy 為網路封包處理工具，可以用來偽造、模擬封包傳輸，或是自訂義協議進行傳輸，為網路流量相關實驗常用到之工具。而攻擊之模擬上，除了使用 Scapy 之外，也有使用 TFN2k[28]作為工具，TFN2k 為開源之 DDoS 工具，內含許多種攻擊模式

4.3、 測試結果

本論文之所有模擬皆使用圖 4-1 所示之模擬拓樸環境中進行，在此章節中，將會分別驗證本論文之系統各功能之實現成果，並一一說明各模擬成果之細節，以及其所代表之意義。

4.3.1、 正常狀態

本小節說明在未受攻擊之正常流量狀況本系統之執行結果，此小節之模擬環境如圖 4-1 所示，以 Scapy 作為流量產生工具，並分別以 h1 對 h8、h2 對 h3、h5 對 h4 以及 h6 對 h7 進行每秒 100KB 之封包傳輸，並以系統抓出各 Switch 中流量負載情形。

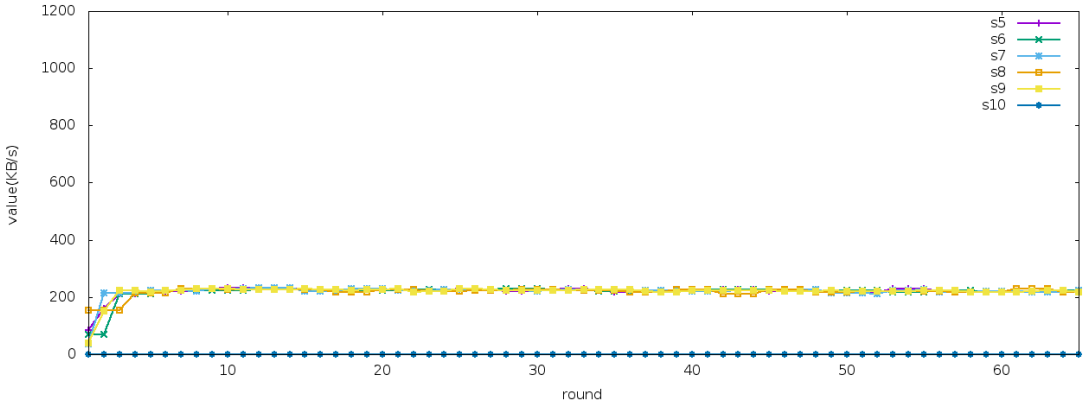

圖 4-2、正常流量下系統未啟動

在圖 4-2 為鋪設上述流量後，系統未處理狀況下之情形，圖中顯示不同 Switchs (s5~s10)之流量變化圖，Y 軸為流量負載，X 軸為系統抓取流量的週期(round)，設定為每 3 秒抓取一次。此外，由於 s1~s4 為最底部直接與 VMs 相連的 Switch，雖仍有抓取其流量負載值，但由於本小節之模擬場景中，系統機制並未對其有任何影響，故不特別顯示。可以看到所有 switch 皆有流量，僅 s10 完全無負載，因為 Controller 預設下發的封包傳遞路徑不包含 s10，所有在 s5、s6 與 s7、s8 之流量皆經過 s9，所以 s10 直至最後皆沒有負載。

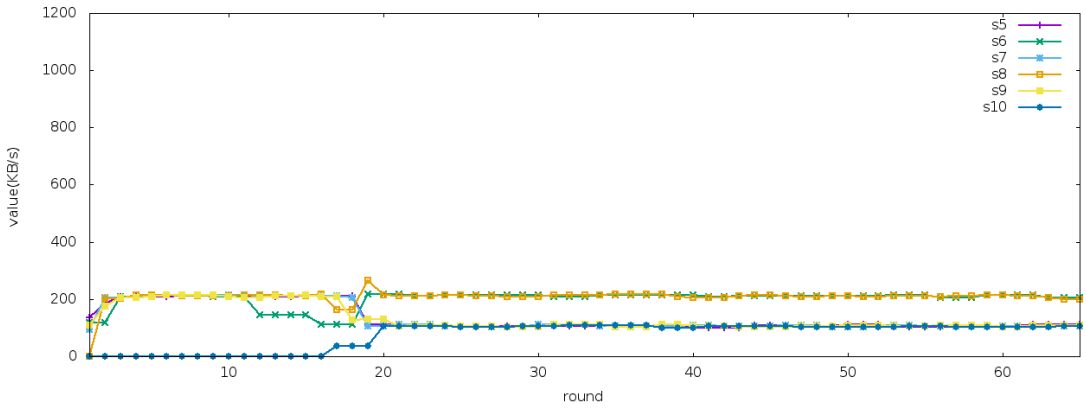

圖 4-3、正常流量下且系統啟動

圖 4-3 與圖 4-2 為同樣的模擬場景，但不同的是系統開始作用，圖中可以看到在 x 值 11 以前，負載情形跟圖 4-2 相同，因為目前系統設定上，前 10 round 屬於初始化階段，此階段工作為蒐集環境狀態，因此未有任何機制啟動，在 10 round 之後系統開始運作，並在 11 round 左右啟動負載平衡，並在接近 20 round 左右完成負載平衡，進入穩狀態，可以看到部分流量自 s9 轉移至 s10，s9 之負載因此下降，而 s10 之負載也上升。此後系統也有觸發負載平衡機制的情形發生，但系統的負載平衡評估下，各 switch 之負載為平衡狀態，因此未做任何路徑更動。

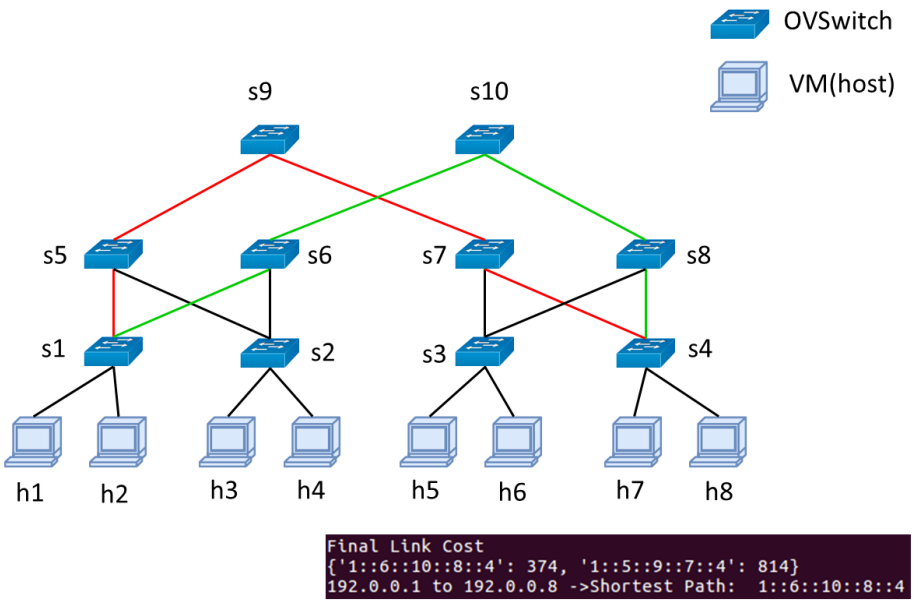

圖 4-4、負載平衡路徑安排

而圖 4-4 所示即為系統所判斷 h1 (192.0.0.1)傳輸封包至 h8 (192.0.0.8)之最適合路徑，系統求出 h1 連線至 h8 所需之實體連線最短路徑有兩條，並進一步比對此二路徑所經過之 Switches 負載情形，而此其中 s1→s6→s10→s8→s4 相較 s1→s5→s9→s7→s4 有更低的負載，因此選擇 s1→s6→s10→s8→s4 之路徑，並向路徑上之 Switches 下發雙向之 Flow Entry，使 Switch 能依照最短路徑傳輸，達到負載平衡之效果。

4.3.2、 惡意攻擊

本小節之系統測試中，同樣以 Scapy 之作為工具，且使用圖 4-1 做為模擬用之拓樸環境，選取其中之兩台 VM 對其餘之某一 VM 進行惡意攻擊模擬測試。

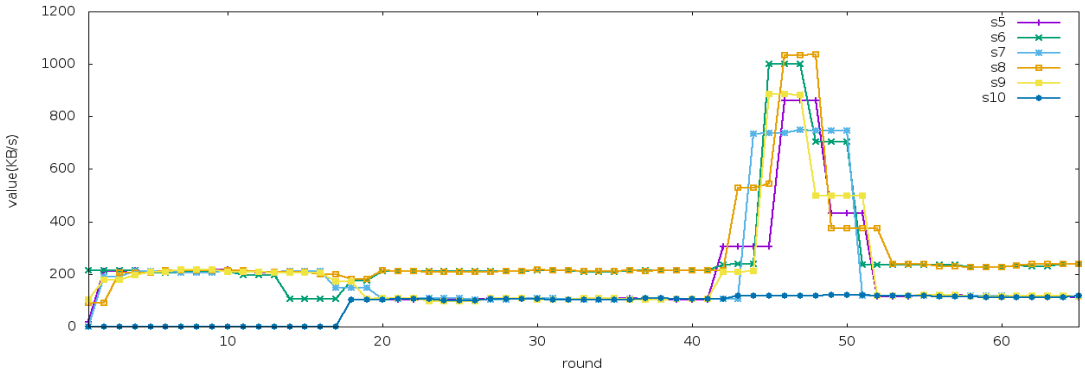

圖 4-5、模擬攻擊測試結果

圖 4-5 所示為模擬攻擊測試結果，Y 軸為流量負載，X 軸為系統抓取流量的週期(round)，並仍只顯示 s5~s10 之流量負載情形，因 s1~s4 為直接連接 VM 之 Switches，系統下發將攻擊流量捨棄之 Flow Entry 雖能使 s1~s4 之 Switch 將封包捨棄，但其仍會持續接收該流量，也因此流量負載並無任何變化，故不特別顯示。

圖 4-5 的 40 round 以前所顯示之各 Switch 流量為同樣以 Scapy 所鋪設之底層一般流量，目的是為了模擬網路環境中，一般狀態下之流量變化，所鋪設之模

擬一般流量分別為h1對h8、h2對h3、h5對h4，以及h6對h7分別進行每秒100KB/s之封包傳輸，並且在40 round時從h7與h8分別往h3注入每秒400KB/s之流量來模擬攻擊流量。

圖4-5中可看到在10 round以前，流量負載不平均，因為系統處於初始化階段，蒐集當前環境狀態，並在12 round左右開始負載平衡，在20 round後負載平衡完成，直至40 round前皆維持穩定狀態，在43 round左右時由於攻擊流量的注入導致部分switch流量飆升，而圖4-6與圖4-7分別為s2與s6之source port Entropy以及source IP Entropy之變化圖，可以看到在47 round左右時，不論s2或s6皆同時有Entropy低於threshold之現象，因此系統下發阻擋攻擊流量之Flow Entry，因此圖4-5在50 round之後的流量回到與原本所鋪設之一般流量相同，代表成功將惡意流量阻擋，並且該環境中之一般流量也持續傳輸。

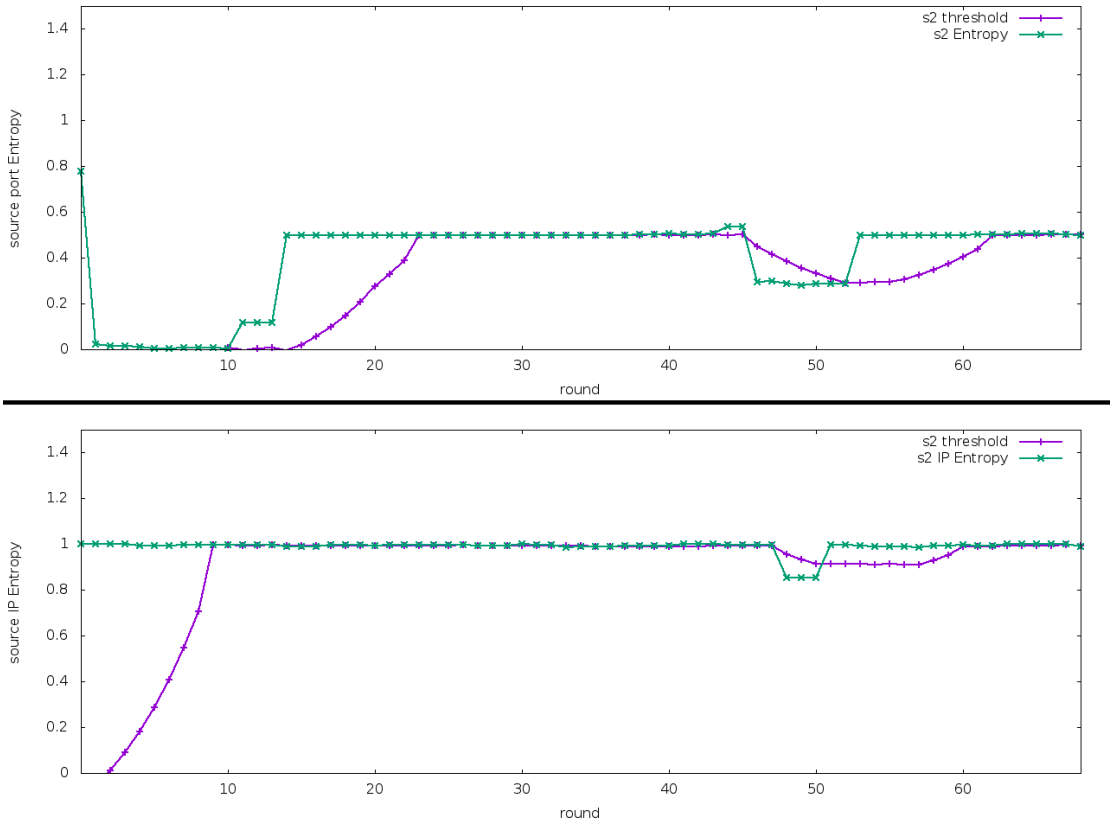

圖 4-6、s2之Entropy變化

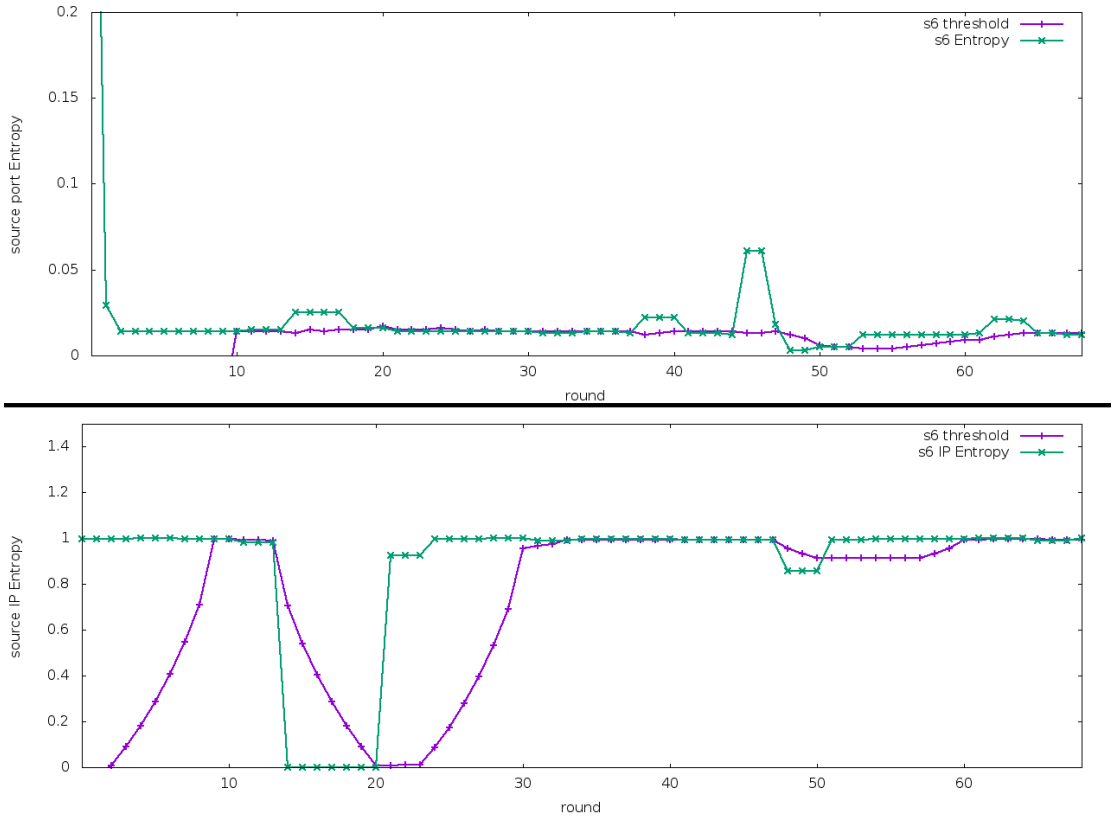

圖 4-7、s6 之 Entropy 變化

此外，在更多 Hosts 的情況下也能夠正確的偵測出攻擊，圖 4-8 為新拓樸在 OpenDaylight 中所讀取並產生的圖，可以看到相較於圖 4-1，增加了更多的 hosts。

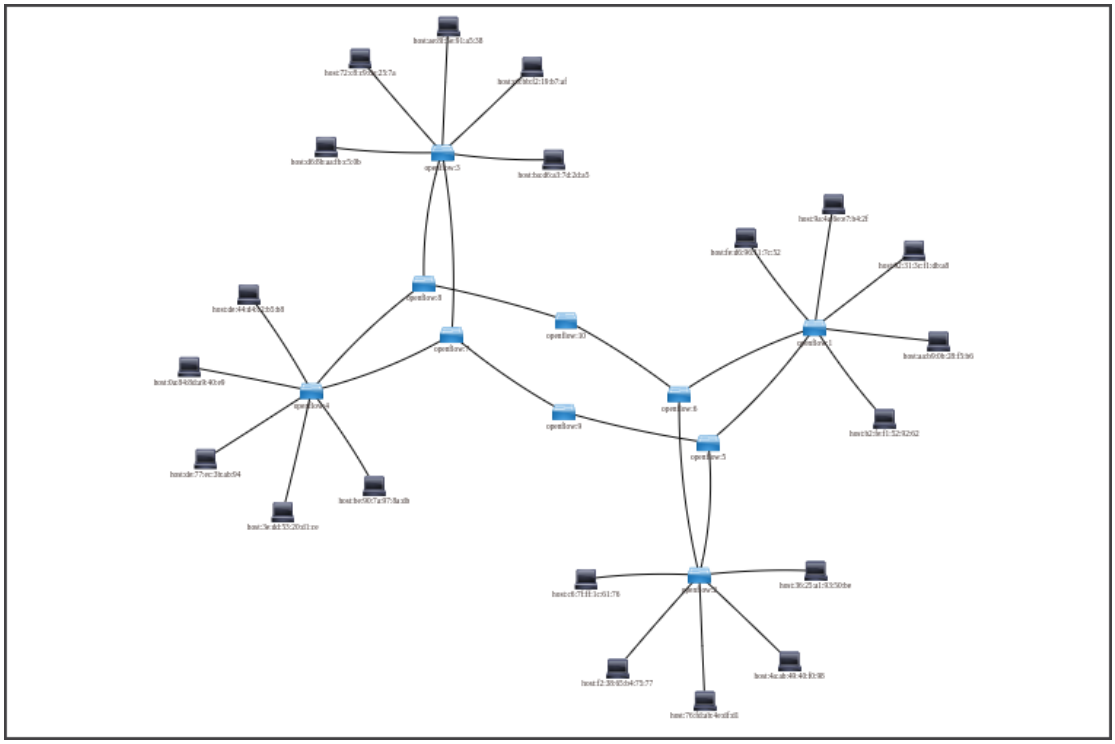

圖 4-8、在 OpenDaylight 所讀取到更多 hosts 之拓樸

在實驗上，以相同的方式鋪設基礎流量，並以 DDoS 工具 TFN2k 進行攻擊測試，以下說明模擬測試以及模擬之流量變化圖，圖 4-9 為以真實 IP 進行 flood 的模擬，並且圖 4-10 為偽造 IP 進行攻擊的模擬。

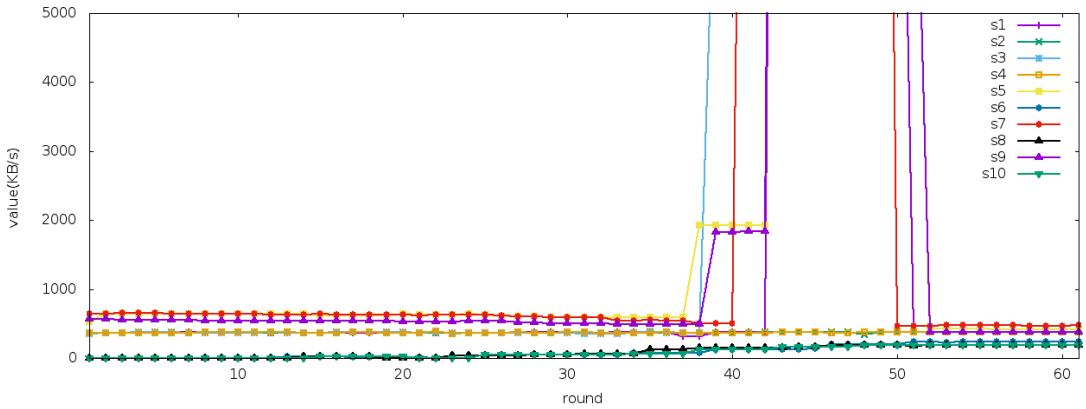

圖 4-9、實體 IP 之 flood 攻擊

圖 4-9 之模擬中，一般流量鋪設與前述模擬差異不大，因此不再贅述，並在 35 round 左右選擇 h14 灌輸流量至 h3，導致傳輸路徑中之 Switch 流量大幅上升，由於 sflow 之偵測延遲，系統約在 37~39 round 之間透過 sflow 蒐集之資料，偵測

到流量異常上升，並開始確認是否為攻擊，並於 40 round 偵測攻擊並溯源，接著下發 Drop 該流量之 FlowEntry 至最靠近攻擊來源的 Switch，最終在 50 round 左右各 Switches 流量大幅下降至原本流量，可看出攻擊確實被擋下。

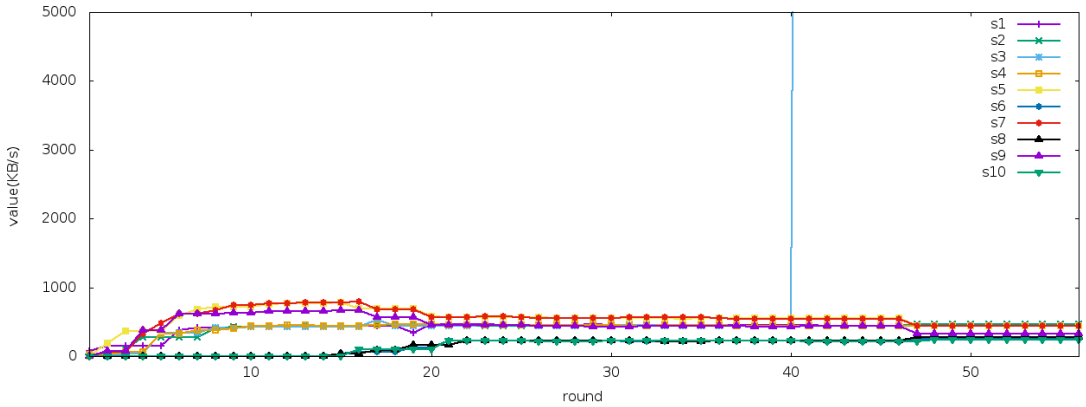

圖 4-10、偽造 IP 之 flood 攻擊

圖 4-10 之模擬中，一般流量之鋪設也與前述相同，因此也不再贅述。此次實驗則是在 35 round 左右同樣以 h14 對 h3 進行攻擊，但這次的攻擊為偽造 IP，可以看到攻擊流量約在 40 round 時被 sflow 蒐集到，並由偽造 IP 偵測模組確認有偽造 IP 的情形發生，因此馬上下發 Drop 的 FlowEntry，阻擋從該 port 傳出之封包，可看到其餘 Switch 之流量皆未改變，持續正常的傳輸。

第五章　結論

在本論文中，我們提出能夠應對流量(flood)類(D)DoS 之系統機制，其中，本研究之(D)DoS 之偵測方法為利用自適應之流量閥值配合熵(Entropy)來對於整個 SDN 網路環境之流量持續分析、評估，並利用 SDN 之特性，利用與目前網路架構不同之優勢來應對(D)DoS。

本論文提出之系統機制於發現異常流量後，利用 Entropy 確認是否為(D)DoS 攻擊，若是，則追溯流量來源，並於來源 Switch 中下達阻擋之 Flow Entry，若否，則進行 Load Balance，找出封包之最短路徑，於路徑所經過之各 Switch 內下達雙向之 Flow Entry 以引導流量至最適合路徑，降低環境傳輸延遲，若偵測到偽造 IP 之行為，也能立刻進行應對，阻絕該流量。並且在模擬測試中使用 Scapy 所撰寫之攻擊腳本以及 DDoS 工具 TFN2k 來進行攻擊，結果顯示系統機制確實能夠阻隔巨量流量的(D)DOS。

參考文獻

[1] R. E. Hattachi and J. Erfanian, “5G white paper.” NGMN white paper, 2015.

[2] Huawei, “5G network architecture a high-level perspective.” 2016.

[3] Available at: http://www.fujitsu.com/global/about/resources/news/press-releases/2015/0904-01.html

[4] OpenFlow switch specification, available at: http://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf

[5] Open network foundation, available at: https://www.opennetworking.org/

[6] ONOS white paper, available at: http://onosproject.org/wp-content/uploads/2014/11/Whitepaper-ONOS-final.pdf

[7] OSGi official website, available at: https://www.osgi.org/

[8] OpenDaylight official website, available at: https://www.opendaylight.org

[9] P. Kamboj, M. C. Trivedi, V. K. Yadav, and V. K. Singh, “Detection techniques of DDoS attacks: a survey,” *2017 4th IEEE Uttar Pradesh Section International Conference on Electrical, Computer and Electronics (UPCON)*, pp.675-679, 2017.

[10] R. K. Arbettu, R. Khondoker, K. Bayarou and F. Weber, “Security analysis of OpenDaylight, ONOS, Rosemary and Ryu SDN Controllers,” *2016 17th International Telecommunications Network Strategy and Planning Symposium (Networks)*, pp.37-44, 2016.

[11] I. Ahmad, T. Kumar, M.Liyanage, J. Okwuibe and M. Ylianttila, “5G security: analysis of threats and solutions,” *2017 IEEE Conference on Standards for*

Communications and Networking (CSCN), pp.193-199, 2017.

[12] K. K. Karmakar, V. Varadharajan and U. Tupakula, “Mitigating attacks in software defined network(SDN), ” *2017 Fourth International Conference on Software Defined Systems (SDS)*, pp.112-117, 2017.

[13] R. M. Thomas and D. James, “DDoS detection and denial using third party application in SDN,” *2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS)*, pp.3892-3897, 2017.

[14] B. H. Lawal and A. T. Nuray, “Real-time detection and mitigation of distributed denial of service (DDoS) attacks in software defined networking (SDN),” *2018 26th Signal Processing and Communications Applications Conference (SIU)*, pp.1-4, 2018.

[15] B. Pande, G. Bhagat, S. Priya and H. Agrawal, “Detection and mitigation of DDoS in SDN,” *2018 Eleventh International Conference on Contemporary Computing (IC3)*, pp.1-3, 2018.

[16] C. Gkountis, M. Tahab, J. Lloret and G. Kambourakis, “Lightweight algorithm for protecting SDN controller against DDoS attacks,” *2017 10th IFIP Wireless and Mobile Networking Conference (WMNC)*, pp.1-6, 2017.

[17] L. Saifei, C. Yunhe, N. Yongfeng and Y. Lianshan, “An effective SDN controller scheduling method to defence DDoS attacks,” *2019 Chinese Journal of Electronics*, pp.404-407, 2019.

[18] B. Celesova, J. Val'ko, R. Grezo and P. Helebrandt, “Enhancing security of SDN focusing on control plane and data plane,” *2019 15th International Wireless Communications & Mobile Computing Conference (IWCMC)*, pp.90-95, 2019.

[19] X. You, Y. Feng and K. Sakurai, “Packet_In message based DDoS attack detection in SDN network using OpenFlow,” *2017 Fifth International Symposium on Computing and Networking (CANDAR)*, pp.522-528, 2017.

[20] S. Attarha, K. H. Hosseiny, G. Mirjalily and K. Mizanian, “A load balanced congestion aware routing mechanism for software defined networks,” *2017 Iranian Conference on Electrical Engineering (ICEE)*, pp.2206-2210, 2017.

[21] U. Zakia and H. B. Yedder, “Dynamic load balancing in SDN-based data center networks,” *2017 8th IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, pp.242-247, 2017.

[22] M. C. Nkosi, A. A. Lysko and S. Dlamini, “Multi-path load balancing for SDN data plane,” *2018 International Conference on Intelligent and Innovative Computing Applications (ICONIC)*, pp.229-234, 2018.

[23] Avail at: https://en.wikipedia.org/wiki/Normal_distribution

[24] Networkx, available at: https://networkx.github.io/

[25] Open vSwitch, available at: http://www.openvswitch.org/

[26] Mininet, available at: http://mininet.org/

[27] Scapy, available at: https://scapy.net/

[28] TFN2k, available at: https://github.com/poornigga/tfn2k