隨資料流的 OpenFlow 控制訊息機制

沈上翔
臺灣科技大學資訊工程系
sshen@csie.ntust.edu.tw

摘要

在未來的 5G 網路之中，軟體定義網路將被廣泛地運用。然而，軟體定義網路仰賴中央控制器去對網路交換機下達命令來更新其轉發規則。在更新的過程之中，網路還是會有正在運行的封包。規則更新在不同交換機被應用的時間點和順序，會對這些正在運行的封包造成不同的影響。也可能造成不同於我們預期的結果而出現安全上的問題，因此如何確保每個封包被正確的交換機規則處理變成軟體定義網路中很重要的問題。本篇論文中提出了一個隨資料流路徑做規則更新的機制。更新規則的控制封包將延著資料封包的路徑跟資料封包一起做轉發，並對延途的交換機做規則的更新。我們方法可以確保封包全程使用新的規則或是全程使用舊的規則，來保證其正確性以避免造成網路安全性問題。本篇論文中，也設計藉由交換機規則的安排，達到控制封包隨著資料封包的路徑做更新的目的。

關鍵詞：資安通訊、軟體定義網路、控制層

Data Flow Aware OpenFlow Control Signaling System

Shan-Hsiang Shen
Dept. of Computer Science & Information Engineering,
National Taiwan University of Science & Technology
sshen@csie.ntust.edu.tw

Abstract

In the next generation 5G networks, software-defined networking will be widely used to manage ISP networks. The new SDN architecture arises new network security issues. To apply new policies, a central controller sends control messages to switches to update their forwarding rules. During the rule updates, there are on-going packets in switches, so the sequence of new rules applied in the switches is crucial for policy correctness. However, the varying transmission latency between the controller and switches makes it difficult to guarantee all packets follow either a new policy or an old policy. To address this issue, we propose a novel data flow aware OpenFlow control signaling system (DOC). In DOC, SDN control messages for rule updates are forwarded with the same route as data packets and update rules in the sequence of the

switches along the path. DOC can guarantee the policy correctness to avoid security issues during policy updates.

Keywords: Security; Software-defined networking; Control plane

壹、前言

在未來的 5G 網路之中，各網路單元如 S-GW 和 P-GW 將被虛擬化放入雲端之中運行。虛擬化後，可更有效地使用資源，及更有彈性地部署這些網路單元。例如我們可以根據不同的需求，將這些網路單元放在最適當的位置，來減少網路的延遲。此外，新的服務也可以快速而且低成本地進入市場運行。為了管理這些虛擬化的網路單元，軟體定義網路以及網路功能虛擬化將被導入。網路功能虛擬化平台可以用來管理這些虛擬化的網路單元，使之可以被任意地部署和搬移。而軟體定義網路利用中央控制器來導向網路流量。無論這些網路單元被部署的位置為何，中央控制器皆可將相對應的網路流量導入正確的網路元件。網路管理人員是服務的提供者可以在軟體定義網路和網路功能虛擬化的架構上，撰寫相關的程式來控制服務和網路流。因此，在軟體定義網路和網路功能虛擬化的協同運作之下，未來 5G 的網路架構在管理上會更加具有彈，也能更有利用網路資源。

軟體定義網路和網路功能虛擬化可以帶來管理上的許多好處。然而，新的網路架構也將會帶來有別於傳統網路在網路安全上的一些問題。這些新的網路安全問題值得我們好好思考，來減少未來 5G 網路遭受到網路攻擊的機會。有別於一般傳統網路，軟體定義網路仰賴中央控制器對交換機下達命令，來對網路流量做導向。這些命令是由中央控制器和交換機之間的控制頻道來做傳送。包含了在交換機中加入新的規則或是修改舊有的規則，來對應新的網路服務。交換機在收到命令之後，會立即對轉發表(forwarding table)做修改。

當網管人員對網路系統的政策做修改時，往往需要對數個交換機修改規則來對應新的網路政策。例如，我們要對防火牆(firewall)做負載平衡(load balance)時。為了將網路流量導向到新開啟的防火牆，中央控制器需對路徑上的交換機修改其轉發表內的規則。然而，如果路徑中這些規則的修改順序不對，可能會造成某些封包沒有按照網管人員所設定的政策被處理。例如在規則修改的過程中，網路中有正在傳送的封包，有些封包可能因為規則修改順序的問題不會經過防火牆。這種因為規則修改時間差所造成的錯誤，造成網路安全上的問題。在 OpenFlow 協定之中，當交換機完成規則的修改時，並不會告知中央控制器。因此，中央控制器難以掌控這些規則被修改的順序，也難以確保所有封包都按照網管人員所設定的政策被處理，而造成安全上的問題。

為了解決因規則修改順序所造成的問題，除了要計算正確不會產生錯誤的規則修改

順序之外，也要能確保在修改的過程能按照此順序來完成。中央控制器到不同的交換機延遲時間並不相同，而且是多變難以預測的。最簡單的解決方式是當中央控制器發出規則修改命令給第一台交換機之後，要等待一定的時間才會發出第二台交換機修改的命令。用以確保第一台交換機修改完成後，才會進行第二台的修改，維持正確的修改順序。這個等待的時間會拖長整個網路政策修改的過程。此外，如何決定等待的時間也是個問題。如果等待時間過長，可能前一個規則修改還來不及完成，會造成順序上的問題。而如果等待的時間過長，會使整個政策修改的時間會被拉長。漫長的政策修改時間也會成為網管人員的問題，因為新的政策更新可能會是重要安全更新，會影響到整個網路的運作。

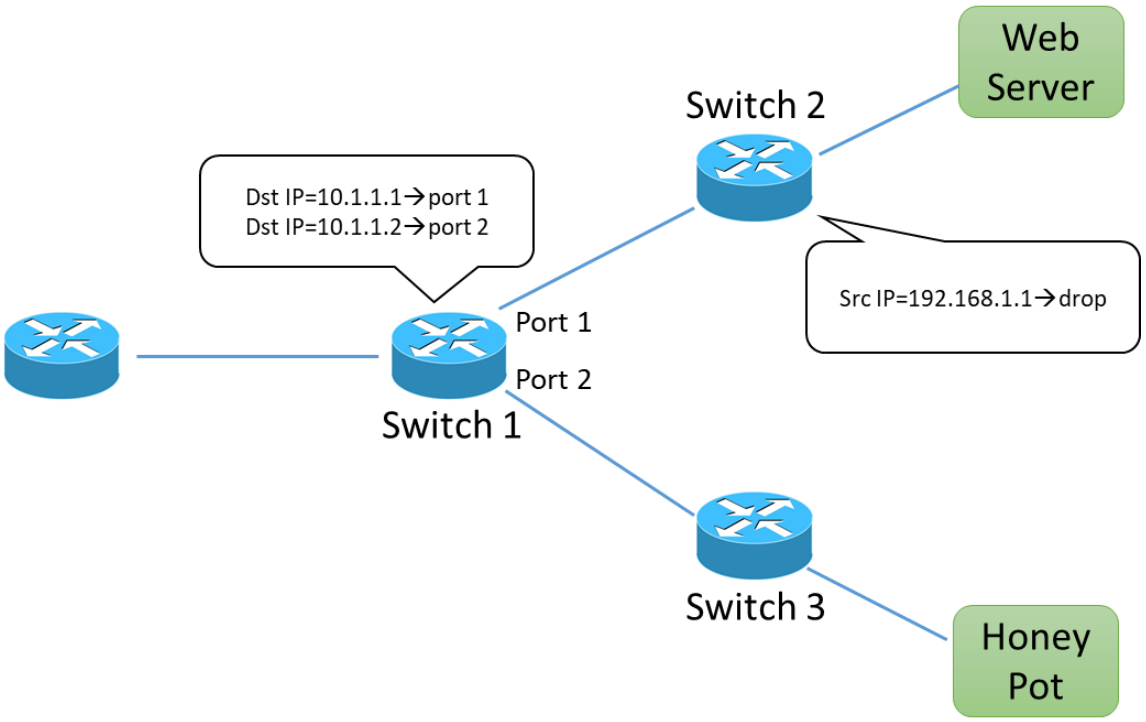
為了解決軟體定義網路中規則更新順序以及延遲的問題，相關研究主要分為三類：(1)用中央控制器去計算正確的更新順序，再依照這個順序更新交換機的規則。如 Ludwig et al.[5]以及 McClurg et al.[7]，(2)在封包中加入標籤，並在交換機中保留更新前和更新後的規則，利用不同的標籤來決定使用新或舊規則。Jin et al.[2]，Katta et al.[3]，以及 Reitblatt et al.[9]，(3)綜合(1)和(2)兩種方法，當找得到適當的更新順序時，則照此順序更新。如無適當順序，則在封包中加入標籤。然而，方法(1)會有無解的情況，而方法(2)和(3)會在封包中加入標籤。此標籤一方面增加封包的大小，此外因為更動了資料封包，會對軟體定義網路的使用者產生一些限制。如不能用相同的籤標在他們所控制的封包內，而減少了軟體定義網路的使用彈性。

為解決上述議題，本篇論文提出了一個新的規則更新方式。我們方式的原則是如果封包在政策更新的過程中，套用的是舊政策的規則，重進入網路到離開皆是套用舊政策的規則。相反地，如套用新政策的規則，全程都將套用新政策的規則。不會有任何一個封包套用到不同政策的規則，避免錯誤的發生。因此我們提出了隨資料流的 OpenFlow 控制訊息(Data flow aware OpenFlow control messages, DOC)。有別於傳統 OpenFlow 中資料和控制流量是分開考慮流向，DOC 重新考慮了控制訊息導向的問題。控制訊息的導向需要同時考慮資料流的方向，來解決因更新順序所造成的正確性上的問題。我們的方法是讓更新用的控制訊息沿著資料流的方向來傳送，當交換機收到控制訊息後，可隨即更新其轉發表。在交換機中，控制訊息的封包會和資料封包進入相同的隊列(queue)。因此，在整個路徑之中，這些封包的順序不變。在控制訊息封包之前的封包會全程套用舊的規則，而在控制訊息封包之後的封包會全程套用新的規則，而不會有正確性的問題。在模擬實驗之中，在真實網路拓樸和大型人造拓樸中比較了 DOC 與傳統更新方式。在 DOC 中，控制封包所經過的 hop 數只需要傳統更新方式的一半以下。在大型網路中更是明顯。在有 2000 台交換機的拓樸之中，DOC 相較於傳統更新方式更只需要八分之一的 Hop 數。因此，DOC 可以有效地減少網路規則更新的延遲。

貳、文獻探討

目前有許多相關的研究在探討軟體定義網路在規則更新順序上所造成的問題以及其嚴重性。SDNRacer[8]中提到了軟體定義網路有狀態同步上的問題，在某個時間點中央控制器和交換機的規則會有差異。不同事件如交換機中的規則更新真實發生的順序可能會造成無法預期的錯誤。為了找出這類錯誤，SDNRacer 為一個動態同步問題分析器。BigBug[6]設計了一個系統可以自動地去分析這些同步上所產生的問題，並且辨別這些問題的類型和產生原因。Attendre[10]也提到軟體定義網路中，規則更新順序的問題，會造成不必要的控制訊號，延遲，以及中央控制器額外的負擔。為了解決此問題，Attendre 設計了一個系統去偵測並移除這些因訊息順序所產生的多餘控制訊息。MED[13]利用了模擬的方式找出軟體定義網路中可能發生的錯誤。這些錯誤包含了路徑迴圈(loop)，可到達性(reachability)，競爭條件(race condition)，以及轉發表的錯誤。OFRewind[12]幫助了網路的管理人員去找出會造成網路異常的事件。他們會記錄所有的網路事件，並在其它非運作中的網路中重現這些事件，來找出造成異常的事件。SDN-RADAR[1]使用了分散式的網路監控系統去收集網路流量資訊做為 debug 之用。HSA[4]指出在網路中運行的許多不同的協定(如 MPLS，OSPF，以及 BGP)，在單獨存在時可能不會出現任何問題。然而，當這些協定同時在網路上運行時，卻可能會發生問題。這種情況也造成網管人員的困擾。HSA 設計了一個模型去分析封包的表頭在不同協定和不同交換機中的交互關係，來找出問題的所在。以上的相關研究多為找出軟體定義網路上，不同事件交互運作所產生的問題。但是並沒有提到該如何更新軟體定義網路中交換機規則，來減少因為更新事件發生順序所產生的問題。

在軟體定義網路中，也有一些論文在探討規則的更新。Ludwig et al.[5]以及 McClurg et al.[7]在中央控制器先計算軟體定義網路安全的規則更新順序。如這個順序是存在的，則會按著這個順序對交換機發出規則更新的控制訊息。他們所使用的方法不能保證找得到一個安全的順序，這也大大地限制了其方法的可用範圍。而其它的相關研究則在封包之中加入標籤，並在交換機中保留了新舊規則，利用標籤來決定要應用新規則或是舊規則。使封包會在整個路徑之中按照同樣的策略來被處理，如 Jin et al.[2]，Katta et al.[3]，以及 Reitblatt et al.[9]都是利用標籤來確保網路的正確性。然而使用標籤會使得封包整體的長度增加，而造成額外的頻寬占用。為了減少標籤的使用，FLIP[11]綜合了預先計算更新順序和使用標籤這兩種更新方式。他們先計算正確的順序，當有些更新無法找出正確順序時，則改用標籤。使得此方法在任何情況都可以適用，並能確保更新的正確性(所有的封包在整個路徑都被相同策略的規則來處理)。使用標籤來當作一種控制網路流量的方式在傳統網路中是合理的。然而，在軟體定義網路中會造成一些麻煩。因為軟體定義網路讓使用者可以很有彈性的控制封包，包含封包表頭的控制。網路平台不應該利用封包表頭(包含標籤)而影響到使用者控制網路的自由度。例如，使用者可能會去使用標籤來做封包比對，可能因此誤比對了平台所加入用來控制規則更新的標籤，而造成錯誤。因此 DOC 將不會在資料封包中加入任何的標籤。



圖一: 實例一：網路初期設定

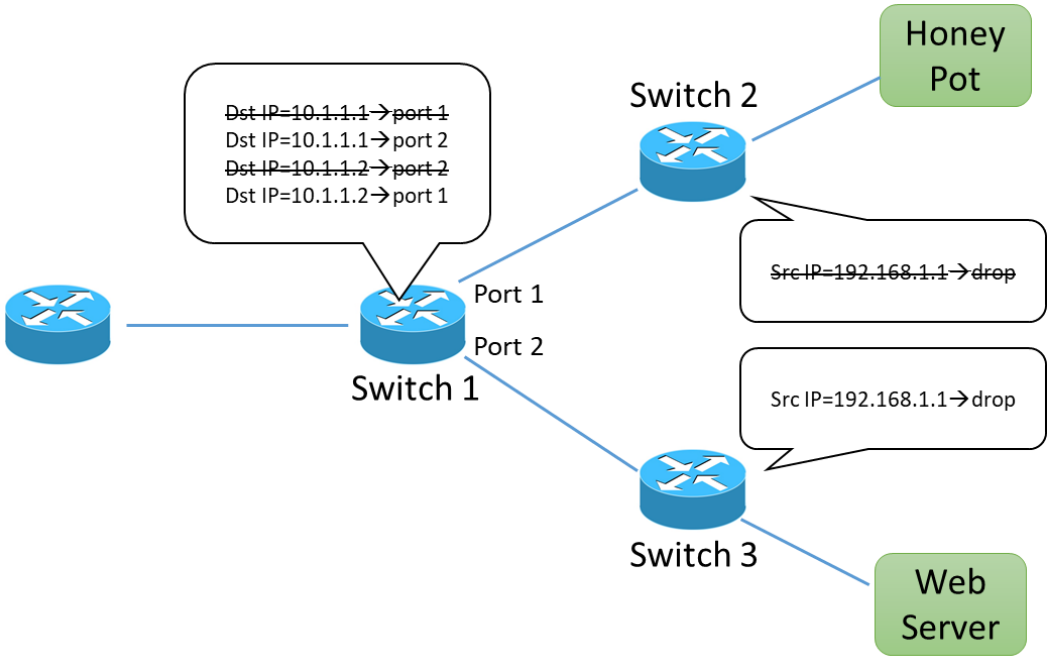
參、案例討論

在本章中，我們將討論規則更新的時間差在那些情況下可能會造成正確性上的問題。這些正確性上的問題會成為網路安全上很大的漏洞，在未來以軟體定義網路為主幹的 5G 網路中會成為一個嚴重的問題。

假設網路初期的設定為如圖一所示。網路中有兩台伺服器，每台伺服器中運行兩台虛擬機。其中一台虛擬機為網頁伺服器，它將接受外來的使用者要求網頁的內容。這個網頁伺服器是我們想要保護的對像，使它免於網路攻擊，如 DDoS。另一台虛擬機為蜜罐伺服器(honey pot server)，它主要的目的在收集網路的攻擊行為，進而可以做分析以求在未來可以辨視這些攻擊的網路流量，並加以阻擋防禦。因為需要盡可能地收集所有的攻擊行為和特徵，所有的攻擊封包都應該要被送往蜜罐。這兩台虛擬機的 IP 地址分別為 10.1.1.1(網頁伺服器)及 10.1.1.2(蜜罐伺服器)。因此，目的 IP 為 10.1.1.1 的封包會由網頁伺服器來處理，而目的 IP 為 10.1.1.2 的封包則由蜜罐伺服器來處理。在網路設定的初期，我們讓 Server 1 來處理網頁伺服器的功能，而讓 Server 2 去處理蜜罐伺服器的功能。

為了達到上述的政策，我們對網路中的 OpenFlow 交換機做了一些設定。目的地 IP 為 10.1.1.1 的封包應該要被送往 Server 1 中的網頁伺服器。就 Switch 1 而言，Server 1 在

port 1 的方向，因此在 Switch 1 中會被安裝相對應的規則。而 Server 2 在 port 2 的方向。假設已知來源 IP 為 192.168.1.1 的封包為惡意攻擊的封包，我們不希望這些封包進入受保護的網頁伺服器。因此在 Switch 2 中加入了一個規則將來源 IP=192.168.1.1 的封包丟掉。相反地，我們希望所有的惡意封包都進入蜜罐，使其行為可以被記錄，所以 Switch 3 會讓所有的封包都通過。



圖二: 當網頁伺服器和蜜罐伺服器交換位置後的網路設定

此時，假設網管人員要將網頁伺服器和蜜罐交換位置，則在 OpenFlow 交換機中相對應的規則也需要被更新，如圖二。在 Switch 1 中兩個規則的 Action 欄位要交換，也就是目的地 IP 為 10.1.1.2 的封包要送往 port 1，而目的地 IP 為 10.1.1.1 的封包要送往 port 2。此外，我們不希望惡意封包進入網頁伺服器，在 Switch 3 中我們必須加進一修規則將來源是 192.168.1.1 的惡意封包丟掉。相反地，這些惡意封包應該要進入蜜罐伺服器，因此在 Switch 2 中原本將封包丟掉封包的規則應該要被移除。

在交換了網頁和蜜罐伺服器，並做上述網路設定的更新時，OpenFlow 的中央控制器必須發出三個控制訊息來去更新三台交換機(Switch 1, Switch 2, Switch 3)中的規則(假設 Switch 1 中的兩條規則的更新可以匯集成一個控制訊息)。第一個訊息是更新 Switch 1 中的兩條規則，第二個訊息是將 Switch 2 中將惡意封包丟棄的規則移除，而第三個訊息是要在 Switch 3 中加入規則將惡意封包丟棄，以免惡意封包進入網頁伺服器。網路上的傳輸延遲時間是很不定的，中央控制器發出這些訊息之後，無法預測這些訊息到達交換機且更新完規則的先後順序。在不同的順序下，會對封包造成不同的影響。

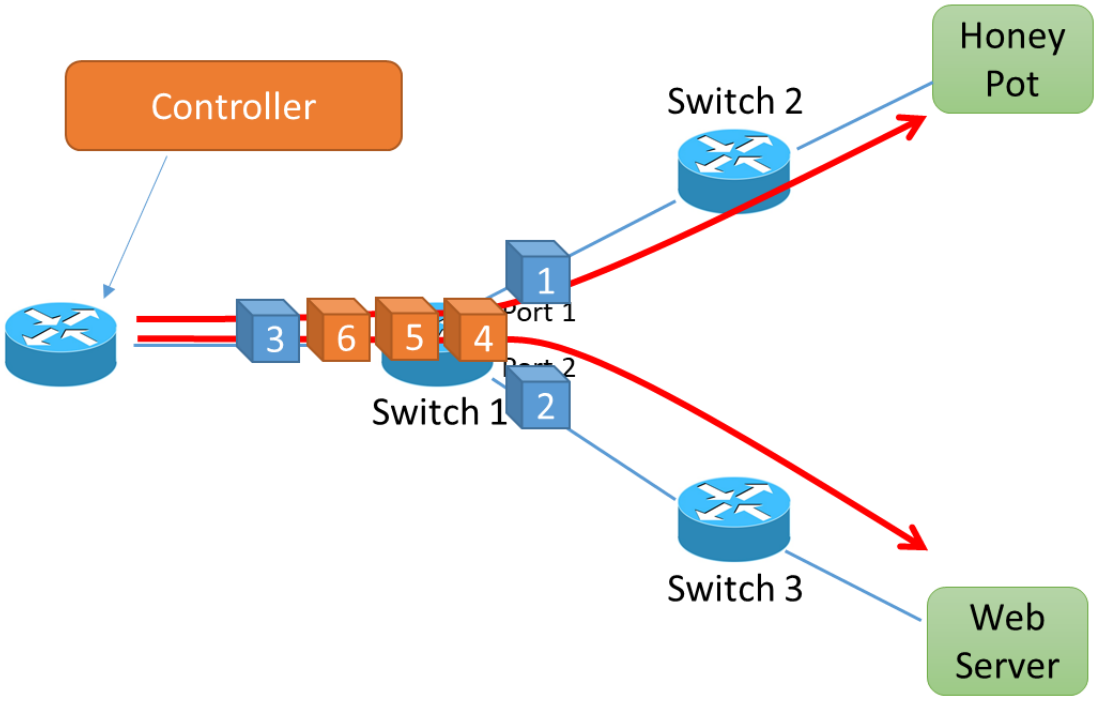
- 三個訊息同時在三個交換機中生效：因為網路中一直有封包在傳送，當三個訊息生效的那個時間點，會有些許封包在 Switch 1 和 Switch 2 之間。這些封包在經過 Switch 1 時是應用舊的規則，因此目的地 IP 為 10.1.1.1 的封包會經由 Port 1 到達 Switch 2。而此時 Switch 2 中的新規則已經生效了，造成所有要到網頁伺服器的封包都會通過 Switch 2，包含了來源為 192.168.1.1 的惡意封包，造成網頁伺服器受到攻擊。相對地，有些在 Switch 1 因舊規則而被送到 Switch 3 時，也因為新的規則生效而封包被丟棄。惡意封包將無法到達蜜罐被收集記錄。
- Switch 2 和 Switch 3 的規則先生效：在這個情況之下，有些封包在經過 Switch 1 時是使用舊的規則，而到達 Switch 2 或是 Switch 3 時使用的是新的規則。同樣的封包先後使用新舊規則，便會產生問題。
- Switch 1 的規則先生效：如 Switch 2 或是 Switch 3 的新規則生效時間晚於封包從 Switch 1 到 Switch 2 或是 Switch 3 的時間，則有些封包會在 Switch 1 時使用新規則，而在 Switch 2 或是 Switch 3 使用舊規則，也會造成問題。

在這個更新過程之中，只要有封包在路徑中使用了新舊封包，都會有可能產生問題。本篇論所提出的方法便是用來解決這個規則更新順序的問題。

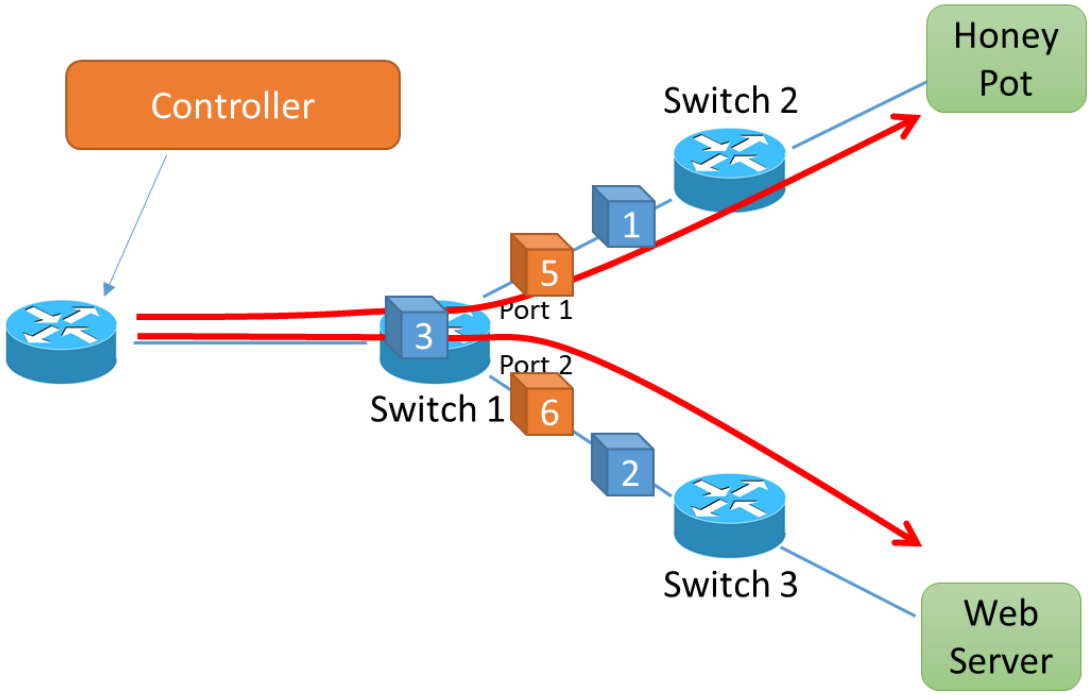
肆、系統總覽

為了解決交換機規則更新時的正確性問題，讓封包在路徑之中皆使用相同版本的規則，我們設計了個新的規則更新方式。基本的設計原則為，讓更新規則的控制封包順著資料封包的路徑傳送。如此可以確保在控制封包之前的封包會全程使用舊的規則，而控制封包之後的封包會全程使用新的規則，可以避免第二章所提到網路安全上的問題。

圖三為一個實例用來說明如何傳遞控制訊息以及如何更新交換機中的規則。更新交換機規則的控制訊號會由中央控制器發出，從資料流量路徑的第一個交換機進入網路。在我們的系統中控制訊息是以頻內(in-band)的方式到達交換機。交換機將控制訊息和資料流量做相同的處理，進入同樣的隊列(Queue)，並從同樣的界面傳送出去，因此，資料封包會和控制訊息封包以同樣的順序經過路徑上的交換機。如圖三所示，藍色的 Packet 1, Packet 2, Packet 3 為資料封包，而橘色的 Packet 4, Packet 5, Packet 6 為控制封包。Packet 4 排在最前面先更新 Switch 1，然後 Packet 5 延著 Switch 1 和 Switch 2 並對 Switch 2 做更新，而 Packet 6 則延著 Switch 1 和 Switch 3 並對 Switch 3 做更新，如圖四。Packet 1 和 Packet 2 會使用舊的交換機規則，因為它們比控制訊息早通每個交換機，而 Packet 3 會使用新的規則，因為控制訊息已經通過各個交換機。在我們設計的系統之中，和傳統的 OpenFlow 相同，每一個交換機更新會對應到一個控制訊息。因此，在圖三和圖四的例子之中，一共需要三個控制訊號。



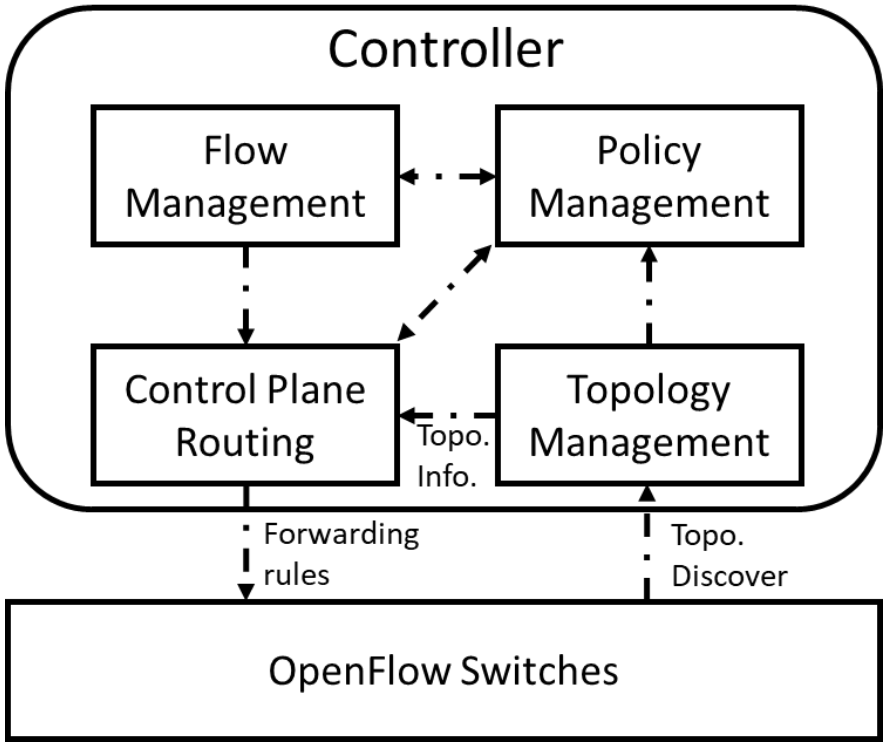
圖三: Packet 4 更新完 Switch 1 之後，Packet 5 往 Switch 2 前進，而 Packet 6 往 Switch 3 前進



圖四: 控制訊息順著資料流量依路徑順序更新交換機規則

4.1 中央控制器

在 OpenFlow 的系統架構中，所有網路上政策的更新都需要透過中央控制器。因此在我們的系統中，也在中央控制器中加入了一些元件。如圖五，在中央控制器中，Topology Management 會利用 LLDP 來找出交換機在拓樸中的連線情況。由 Topology Management 發出 packet out 訊息到達交換機。交換機收到後會發出 LLDP 訊息往所有連外介面做輸送。當鄰居交換機收到此 LLDP 後會對中央控制器發出 packet in 的控制訊息，藉由這些 packet in 的訊息，Topology Management 可以得到各個交換機的鄰居有那些其它交換機，因而組合成整個拓樸的連接情況。這個拓樸資訊會給 Policy Management 做為政策上的參考。網路政策主要是由網管人員來制定，也會參考到目前的網路拓樸情況。政策制定後，如有新的網路流量(flow)，則會通知 Flow Management 註冊一個新的網路流量。在我們的系統中，主要是在中央控制器中加入一個 Control Plane Routing 的元件。這個元件會決定交換機中的規則要如何來更新，這些更新訊息要如何來路由(routing)來達到正確性的要求。Control Plane Routing 會依據網路拓樸，資料流量的路徑，以及要更新的政策。當決定了控制訊號的路徑之後，Control Plane Routing 會將這些控制訊息傳送到路徑中的第一個交換機，並延路做交換機規則的更新。

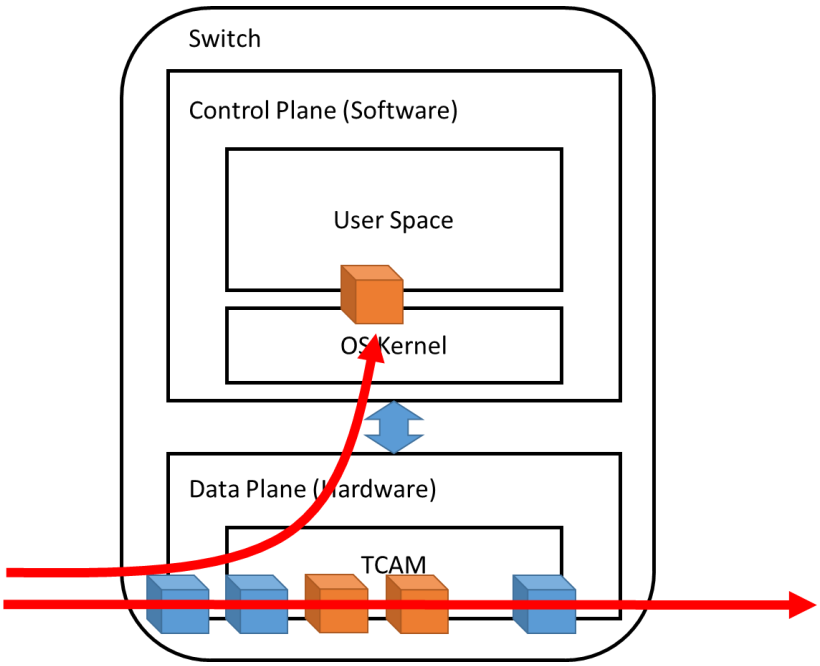


圖五: 中央控制器架構

4.2 交換機更新

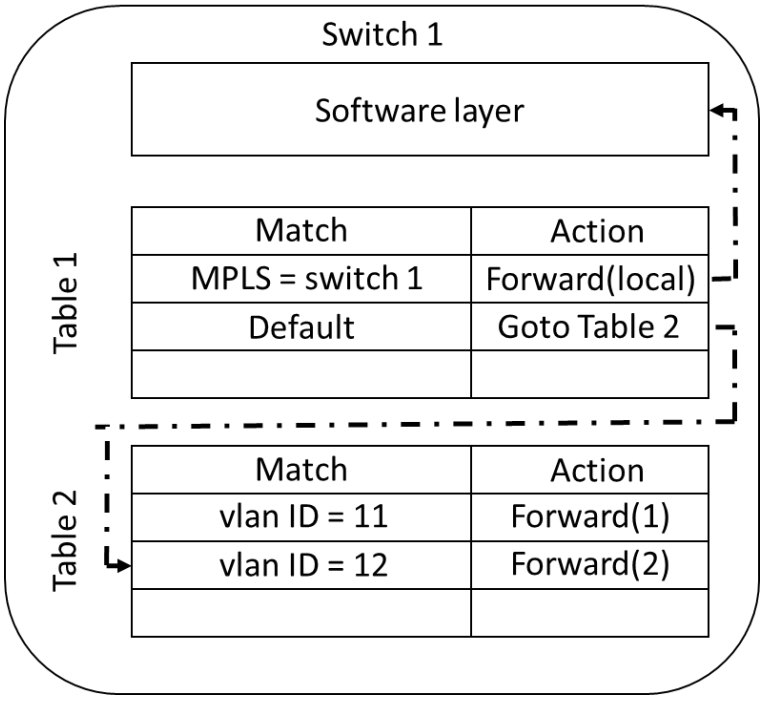
交換機需處理資料流量和控制訊息中的封包。控制訊息封包分別是用來更新不同的交換機的規則。當一個交換機收到一個用來更新其規則的控制訊息封包時，這個封包需要進入交換機的作業系統之中。作業系統會解析控制封包之中的控制訊息，並根據這些訊息去更新交換機中的規則。

圖六為 OpenFlow 交換機架構。交換機主要分為兩大部分：硬體和軟體部分。硬體包含了 TCAM，可以用來比對封包中的欄位。TCAM 可以快速地平行比對封包中的欄位和轉發表中的所有 entry。而在硬體之上為交換機軟體的部份，包含了交換機的作業系統，和一般伺服器中的作業系統類似，有核心的部分以及使用者空間(user space)的部分。軟體的部分主要是用來和中央控制器做溝通。當有交換機無法處理的封包時，會向中央控制器做訊問。同時，也接受中央控制器的命令並執行。我們的系統中，控制封包和資料封包以同樣的路徑在移動，並進入同樣的隊列以同樣的順序行進。在交換機中也跟資料封包相同，由 TCAM 來做比對和處理，如圖六。這些控制封包各自負責更新一個交機中的規則，因此當這些控制封包跟著資料封包到達所負責的交換機時，會被轉送到交換機的作業系統層。經過作業系統的核心(OS kernel)進入使用者空間(user space)。控制封包會在此被解析，作業系統接著會更新交換機中轉發表中的規則。其它不是負責此交換機的控制封包會隨著資料封包向外傳送。



圖六: 交換機中的控制訊息

在我們系統的設計之中，在交換機中我們會使用到兩張轉發表。第一張轉發表用來將負責此交換機(我們稱之為目標交換機)更新的控制封包導到交換機的作業系統層進行規則的更新。而第二張轉發表用來引導封包(包含資料封包和其它控制封包)延著路徑轉發。為了讓控制封包順著其所對應的資料封包前進，這些控制封包表頭的欄位要和相對應的資料封包是一致的。這樣的設定使控制封包在第二張轉發表會和資料封包以相同的方式被處理。然而為了能讓第一張轉發表得以將控制封包轉發到交換機的作業系統，控制封包需在封包內做記號。因為控制封包在到達目標交換機之前會和所對應的資料封包被相同的處理，我們做記號的位置要是資料封包做比對時不會用到的。例如，資料封包的規則中要比對 vlan ID，則我的便加入一個 MPLS，並把交換機 ID 加入 MPLS 之中。第一張表會去比對 MPLS，如果發現 MPLS 中的交換機 ID 是自己的(用來更新此交換機的控制封包)，便會將此控制封包轉發到交換機的作業系統做規則的更新。如 MPLS 不是自己的交換機 ID(更新其它交換機的控制封包)或是沒有 MPLS 表頭(資料封包)則會被導到第二張轉發表。因此，這些封包會順著資料封包的規則在路徑之中被轉發。



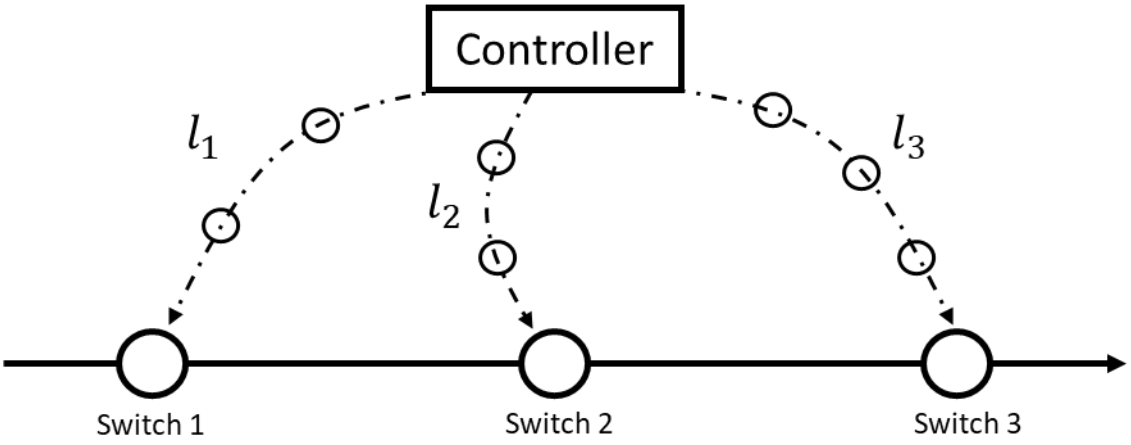
圖七: 交換機中, 轉發表的設定

圖七為一個轉發表設定的例子。假設這是 Switch 1 的轉發表。第一張轉發表的第一個規則將用來將更新 Switch 1 的控制封包轉送到交換機的軟體控制層。這個規則的 Match 欄位為 MPLS 等餘 switch 1 則其 Action 欄位為 Forward(local)也就是將封包轉送到交換機的作業系統由軟體來處理這個封包。而其它的封包則使用第二個規則，其 Match 欄位為 default，也就是所有封包內的欄位都設為 wildcard。其 Action 欄位為 Goto Table

2，也就是繼續去查詢第二張轉發表(Table 2)。在第一張轉發表中，第一個規則的優先權會高於第二個。而第二張轉發表便是 OpenFlow 使用者根據他們的政策對資料封包所設定的規則。在圖七的例子之中，如 vlan ID 為 11 則封包會被轉發到 Port 1。如 vlan ID 為 12，則封包會被轉發到 Port 2。藉由兩張轉發表的設計，我們可以正確地同時對資料封包和控制封包做正確的導向。

伍、效能分析

在本章中，我們將就規則更新的延遲來做分析。圖八為本章所使用的範例場景。假設我們要更新一條路徑，會經過數個交換機。在圖八顯示其中三台交換機。此時中央控制器要變更路徑，因此要更新三台交換機的規則。假設為了達到正確性的要求，我們必須要依照 Switch 1, Switch 2, Switch 3 的順序來更新規則。為了確保這個順序，傳統的更新方式要確定 Switch 1 已經做完更新了，才會對 Switch 2 進行更新，最後才是 Switch 3。然而，在 OpenFlow 協定之中，交換機並不會在更新完規則之後通知中央控制器，因此中央控制器沒辦法確定更新完成的時間。為了解決正確性的問題，在發出一台交換機的更新命令之後，需多等待一定的時間，才能去更新下一台交換機。假設從中央控制器到 Switch 1, Switch 2, Switch 3 的 hop 數分別為l_1, l_2, l_3。



圖八: 網路延遲比較範例

我們在此先定義規則更新延遲如下：

定義：L_o為使用傳統規則更新方式的規則更新延遲，而L_n為使用我們的方式所產生的延遲。在此我們將延遲定義為所經過的交換機數量，經過越多的交換機則排隊延遲(queue delay)會越長。

因此，傳統的規則更新的總延遲為

$$L_o = l_1 + \tau + l_2 + \tau + l_3 + \tau + \cdots = \sum_{i=1}^{|S|}(l_i + \tau) \quad (1)$$

在上面的式子之中，L_o為傳統方法的總延遲，$|S|$為路徑上的交換機數量。而τ為一個等待更新完的時間，來確保更新順序的正確性。而我們的方法是中央控制器先將控制封包送至 Switch 1，再延著路徑進傳送更新。此外，因為我們的控制封包是延著路徑前進，所以可以保證正確的更新順序。在我們的方法之中，我們的方法的總延遲如下。

$$L_n = l_1 + 1 + 1 + \cdots = l_1 + |S| - 1 \quad (2)$$

理論一：傳統方法的規則更新延遲會大於或是等於我們的方法($L_o \geq L_n$)。

證明：

在此，我們希望能夠比較兩種規則更新方式在規則更新延遲上的差異。我們將L_o整理如下：

$$L_o = \sum_{i=1}^{|S|}(l_i + \tau) \geq l_1 + \tau + (|S| - 1)(\min_{i=2\ to\ |S|} l_i + \tau) \geq l_1 + |S| - 1 \quad (3)$$

上面的式子之中，中央控制器到 Switch 1 的延遲可以先分開來看。到 switch 1 的延遲包含了從中央控制器到 Switch 1 所經過的交換機數量(l_1)以及中央控制器等待 Switch 1 更新完成的時間(τ)。因此，後面還剩下 Switch 2 到最後一台交換機的規則更新延遲。在這剩下部分的交換機中，先選定離中央控制器最近的交換機，其到中央控制器的延遲為$\min_{i=2\ to\ |S|} l_i$。因為此為交換機到中央控制器的最短延遲($l_i \geq \min_{i=2\ to\ |S|} l_i$)，

$(|S| - 1)(\min_{i=2\ to\ |S|} l_i + \tau)$必定小於$\sum_{i=2}^{|S|}(l_i + \tau)$。因此，可以得到

$\sum_{i=1}^{|S|}(l_i + \tau) \geq l_1 + \tau + (|S| - 1)(\min_{i=2\ to\ |S|} l_i + \tau)$。

因為中央控制器到任一個交換機必定會經過一個以上的交換機(也就是它自己本身)$\min_{i=2\ to\ |S|} l_i \geq 1$ 且 $\tau \geq 0$，可以得到$(|S| - 1)\left(\min_{i=2\ to\ |S|} l_i + \tau\right) \geq |S| - 1$。

最後，將傳統的方法和我們的方法所需的延遲相減，可以得到：

$$L_0 - L_n = l_1 + \tau + (|S| - 1)\left(\min_{i=2\ to\ |S|} l_i + \tau\right) - (l_1 + |S| - 1) \geq 0 \quad (4)$$

由上面的式子，可以得到結論為$L_0 \geq L_n$。理論一得到證明。

陸、模擬實驗

為了了解本系統的效能，我們以模擬的方式來做測試。在我們實驗的目標為比較傳統的更新方式和我們更新方式，在規則更新延遲上的差異。

6.1 實驗設定

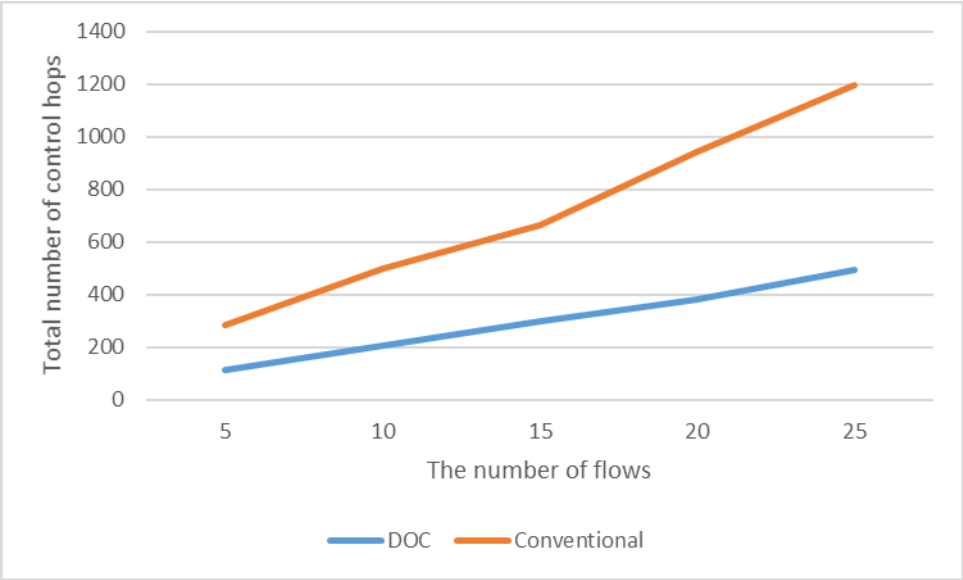
我們的實驗分為兩個部分，第一個部分我們使用 Topology zoo[14]裡面的真實拓樸。因真實的拓樸交換機數較少，因此在第二個部分中我們使用拓樸產生器產生交換機數較多的拓樸，來測試我們的方法在大型拓樸的表現。在真實拓樸實驗中，使用了兩個網路拓樸：Uninett2011 和 Pern。其中，Uninett2011 包含了 69 台交換機以及 98 條連線，而 Pern 包含了 127 台交換機以及 129 條連線。在這兩個真實拓樸中，我們改變網路流量的數量由 5 條網路流量增加到 25 條網路流量。而在大型網路拓樸的實驗之中，網路大小由 500 台交換機增加到 2000 台交換機，來測實在不同網路大小的情況下，我們方法在規則更新延遲上的減少量。同時，我們也改變網路流量的數量，由 50 條網路流量增加到 250 條網路流量。

在實驗之中，控制封包所經過的 hop 數將被統計。如經過的 hop 數多，代表了越高的規則更新延遲，也會影響到整個網路上資料傳輸的效能。在實驗中將比較我們的規則更新方法 DOC 和傳統的更新方法在控制封包所經過的 hop 數的差別。在傳統的更新方式中，為了確保更新的過程中，每個封包都會被正確的規則順序所處理，中央控器會延路去一個接著一個更新交換機的規則。而在 DOC 中，所有的規則更新控制封包先送到路徑中的第一個台交換機。接者，這些控制封包會延著路徑被傳送。實驗中，將比較傳統方式和 DOC 在路徑長短上的差異。

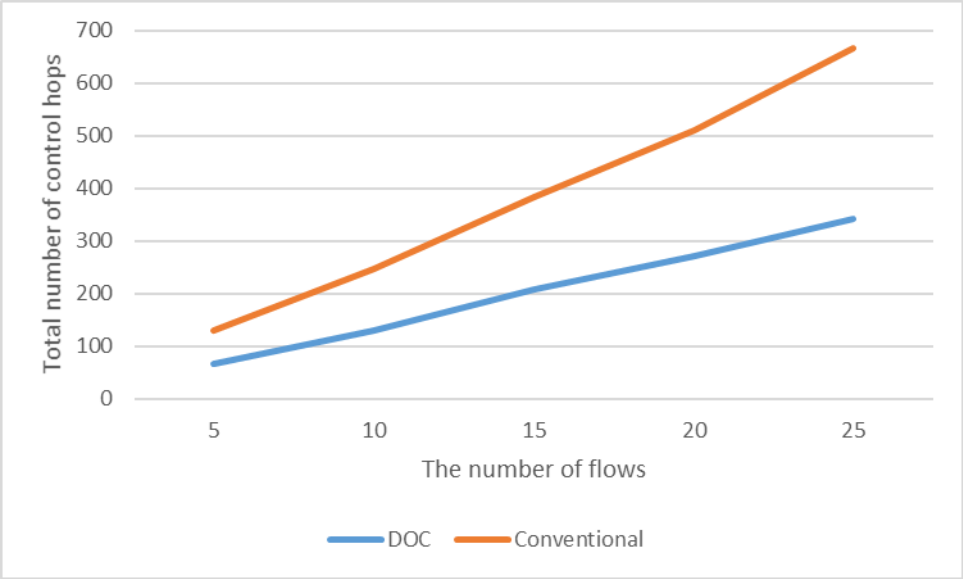
6.2 真實拓樸

圖九以及圖十為在真實拓樸中的實驗結果。其中圖九為 Uninett2011 的結果，而圖十為 Pern 的結果。在兩個網路的實驗結果中我們都可以看出來，DOC 可以減少一半的控制封包所經過的 hop 數。減少此 hop 數後，相對的交換機規則延遲也可以被減少。在網路管理上，也可以讓新的網路策略可以更快速地在網路上被實現。如新的策略是與網路安全有關，則可以使得網路可以更快進入較安全的狀態。此外，圖九和圖十也顯示了當網路中的網路流量數量增加時，因所有網路流量所經過的交換機數量增加，需要更新的交換機也會增加，也因此控制封包所要走的路徑長度也會增加。DOC 在網路流量增加的時候，其控制封包所經過的 hop 數也還是傳統更新方法的一半。新的更新方式可以有

效地減少規則更新的延遲。



圖九：Uninett2011 網路中，在不同網路流數量時的規則更新延遲

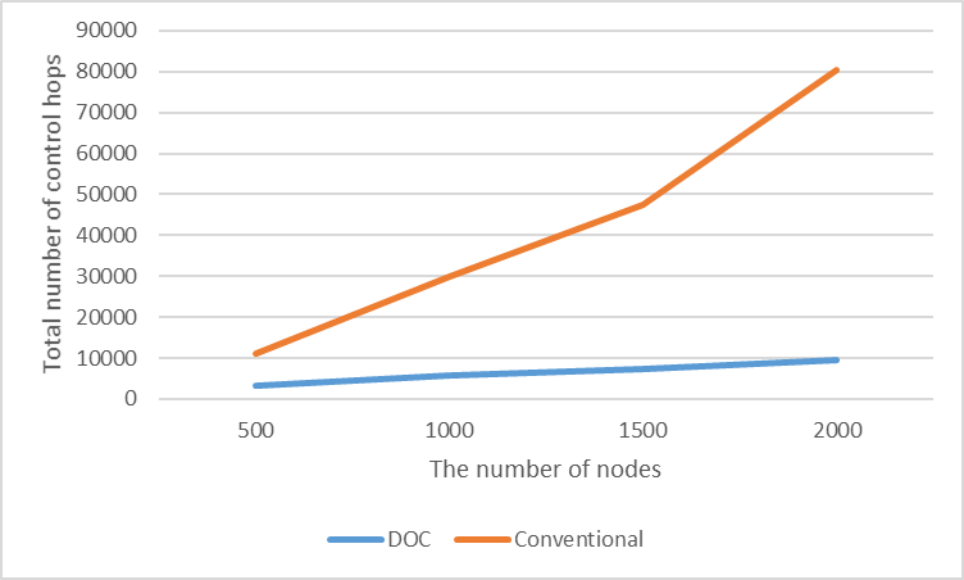


圖十：Pern 網路中，在不同網路流數量時的規則更新延遲

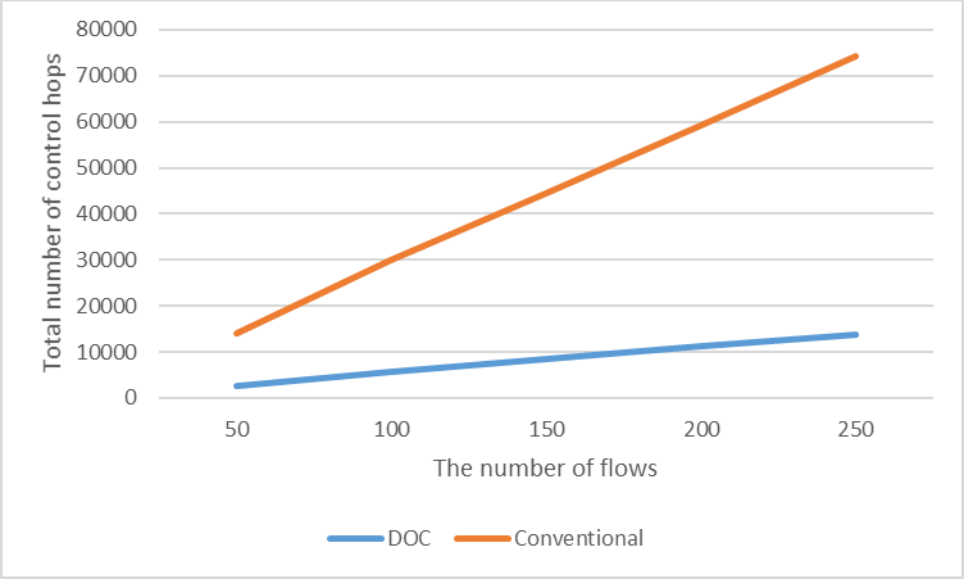
6.3 大型拓樸

圖十一和圖十二為 DOC 在人工產生的大型網路拓樸中的實驗結果。DOC 相較於傳統的更新方式比在小型拓樸中減少了更多的 hop。圖十一中顯示了在不同網路大小的情況下控制封包行經的 hop 數(網路流量的數量固定在 100 條)。當網路越大的情況下，每

條資料路徑所經過的交換機數會增加。也就是說要更新的交換機數量也會變多。此外，也可以觀察到，當網路越大的時候 DOC 可以減少更多 hop 數。當網路中有 2000 台交換機時，DOC 中控制封包所經過的 Hop 數是傳統更新方式的八分之一。因此，在越大的網路之中 DOC 可以取得更大的優勢。尤其在 ISP 網路或是大型的資料中心，以及目前正在快速發展的物聯網，網路交換機(節點)將會越來越多。圖十二則表示了在不同網路流量數量的情況之下(網路交換機數量固定在 1000 台)，控制封包所經過的 hop 數。當網路流量的數量越多時，和小型網路時的情況類似，因為所有網路流量經過的交換機總合較多，需要更多的控制封包去完成更新。DOC 所需的 hop 數在網路流量有 250 條時，是傳統更新方式的七分之一。



圖十一: 在不同網路流量流數量的規則更新延遲



圖十二: 在不同網路大小下的規則更新延遲

柒、結論

未來 5G 網路收以 SDN/NFV 為主軸，因此 SDN/NFV 的安全性問題應該要被嚴格的探討。軟體定義網路依賴中央控制器去更新交換機內的規則。在這些規則更新的過程之中，還是有網路流量在網路中進行著。因此，這些規則更新的順序會造成一些問題。例如，有些封包在上游的交換機可能是使用更新前的規則，但在下游的交換機卻使用了更新後的規則。這會造成安全上的問題，使得封包在網路中被處理的方式有別於我們的預期。相關文獻的解決方法是計算出正確的規則更新順序或是在封包內利用標籤來表明封包是要用新規則或是舊規則。前者可能會無解，且為了保持正確的更新順序，在不同交換機的更新之間需有等待時間，增加更新延遲。後者會增加封包的大小和所占用的頻寬，且因在封包加入了標籤，有影響到網路使用彈性的疑慮。為了解決規則更新的問題，本篇論文提出了隨資料流的 OpenFlow 控制訊息(Data flow aware OpenFlow control messages, DOC)。規則更新的控制封包隨著資料流量的路徑移動做更新，來確保封包被正確的規則處理。此外，相較於文獻，我們改善了更新時間，且不需在封包內加入額外的標籤。根據我們的實驗結果，我們的更新時間只需傳統更新方式的八分之一。

參考文獻

[1] G. Gheorghe, T. Avanesov, M. Palattella, T. Engel and C. Popoviciu, “SDN-RADAR: Network troubleshooting combining user experience and SDN capabilities,” *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*, 2015.

[2] X. Jin, H. Liu, R. Gandhi, S. Kandula, R. Mahajan, M. Zhang, J. Rexford and R. Wattenhofer, “Dynamic scheduling of network updates,” *Proceedings of the 2014 ACM conference on SIGCOMM*, 2014.

[3] N. Katta, J. Rexford and D. Walker, “Incremental consistent updates,” *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, 2013

[4] P. Kazemian, “Header Space Analysis: Static Checking for Networks,” *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, 2012.

[5] A. Ludwig, M. Rost, D. Foucard and S. Schmid, “Good Network Updates for Bad Packets: Waypoint Enforcement Beyond Destination-Based Routing Policies,” *Proceedings of the 13th ACM Workshop on Hot Topics in Networks*, 2014.

[6] R. May, A. El-Hassany, L. Vanbever and M. Vechev, “BigBug: Practical Concurrency Analysis for SDN,” *Proceedings of the Symposium on SDN Research*, 2017.

[7] J. McClurg, H. Hojjat, and N. Foster, “Efficient synthesis of network updates,”

Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation, 2015.

[8] J. Miserez, P. Bielik, A. El-Hassany, L. Vanbever and M. Vechev, "SDNRacer: detecting concurrency violations in software-defined networks," *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, 2015.

[9] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger and D. Walker, "Abstractions for network update," *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, 2012.

[10] X. Sun, A. Agarwal and T. Ng, "Controlling Race Conditions in OpenFlow to Accelerate Application Verification and Packet Forwarding," *IEEE Transactions on Network and Service Management*, 12, (2), pp. 263-277, 2015.

[11] S. Vissicchio and L. Cittadini, "FLIP the (Flow) table: Fast lightweight policy-preserving SDN updates," *The 35th Annual IEEE International Conference on Computer Communications*, 2016.

[12] A. Wundsam, D. Levin, S. Seetharaman and A. Feldmann, "OFRewind: Enabling Record and Replay Troubleshooting for Networks," *USENIX Annual Technical Conference*, 2011.

[13] Q. Zhi and W. Xu, "MED: The Monitor-Emulator-Debugger for Software-Defined Networks," *The 35th Annual IEEE International Conference on Computer Communications*, 2016.

[14] http://www.topology-zoo.org/