摘要

在現今的 SDN 網路架構中，將交換機的控制層與資料層分離，控制層由外部統一控管，以做到動態路由規劃，增加網路的彈性。然而當網路流量大，交換機上的路由表將因記憶體空間有限而不敷使用。若交換機上無法查到路由資訊，則需要向遠端控制器詢問路由規則，造成極大的延遲。本計畫提出依照資料流大小來決定交換機的路由規則安裝機制，將大流量資料的路由資訊盡可能存放在高效能的 TCAM，使大部分流量都能以最快速度查表得到路由規則，而小流量資料之路由資訊則存放在 RAM，最小流量則留在遠端控制台。因為路由的安裝須依照資料流大小動態搬遷存放位置，本計畫將利用 RDMA 快速存取的能力，有效率的完成動態路由規則遷移。透過初步模擬封包轉送表在交換機記憶體裡的搬遷演算法，實驗發現相較於過去演算法，本計畫提出方式能將傳輸總時間縮到最短。將來規劃將這些演算法利用 P4 語言實作，並結合真實的硬體設備和網路傳輸資訊，來找出最適合的演算法，藉此改善 SDN 網路的效能。

一、 研究動機與研究問題

近年來，由於雲端運算（cloud computing）、巨量資料（big data）等技術的快速發展與成長，SDN (Software Define Network、軟體定義網路) 成為當代最熱門的網路技術之一。傳統的網路架構中，交換機包含了控制層 (control plane) 和資料層 (data plane)。這樣的設計使網路管理者在需要改變交換機行為時不具靈活性。加上交換機設計不具大量演算能力，無法結合雲端運算、機器學習等巨量資料處理，在這個網路路由訊息量逐漸龐大的時代顯得不合時宜。SDN 將交換機的控制層與資料層分離，控制層統一交由一台大型的 server 管理，這台 server 就是整個網路的遠端控制器，交換機僅需依照遠端控制器給的路由訊息傳送資料。這會使網路管理者在管理整個網路狀態更為方便，僅需控制一台遠端控制器，就能操縱無數台交換機，而無須實際管理交換機的行為，實現所謂網路虛擬化。另外，遠端控制器具備大量計算能力，可以使用深度學習等方式計算最佳的傳輸路徑，並將路徑訊息傳送給交換機執行。

傳統的傳輸協定如 OSPF (Open Shortest Path First) 或 RIP (Routing Information Protocol) 都需要傳輸額外的封包或 header，讓交換機互相交換路徑訊息，更新自己的 routing table。然而，在 SDN 中，所有路徑相關訊息全部由遠端控制器集中管理，交換機僅需執行封包的傳送，無須進行計算，也省去多餘的封包或 header，減輕網路整體的負擔。路徑計算的靈活性也變得豐富多元，傳統的網路利用戴克斯特拉演算法(Dijkstra's algorithm) 或貝爾曼-福特演算法 (Bellman–Ford algorithm) 等計算最佳路徑，但現在路由表資訊量龐大、網路環境複雜多變，傳統演算法面對快速變化的網路狀態，逐漸顯得不夠靈活。而 SDN 能有彈性管理網路，有別於傳統的網路系統，路由器僅能找到最短路徑，可能會造成路線壅塞。SDN 能即時監控網路流量，由演算法來動態進行有效分流，避免壅塞，也優化網路資源使用率。

近期提出的的 P4 (Programming Protocol-Independent Packet Processors) 交換機程式語言，

不侷限使用的硬體與協定，僅需對應的編譯器即可完成網路的佈署。其靈活性使它成為近來最適合 SDN 的可編程交換機程式語言。在 SDN 架構中，遠端控制器控制了交換機的封包轉送表 (flow tables)，並透過 RPC (Remote Procedure Calls) 和該台交換機進行溝通。然而，要做到動態路由，需要將資料的路由機制安裝在交換機中，當網路流量過大，路由表將因交換機記憶體空間有限而不敷使用。若表格無法查到路由資訊，則需要向遠端控制器詢問路由規則。而與遠端控制器溝通需耗費的時間比直接在交換機中查表多許多。因此，本計畫要設計路由資訊的安裝演算法，讓出現較為頻繁的封包轉送表安裝在交換機的記憶體中，較為不頻繁的則保存在遠端。這樣便能減少遠端訊息交換次數，降低網路的負擔。另外，本計畫打算引進 RDMA (Remote Direct Memory Access、遠端直接記憶體存取) 技術，來取代一般的 RPC 溝通，更進一步增加路由管理效能。

二、 文獻回顧與探討

要進行 RDMA 傳輸，首先執行傳輸的兩端都必須在硬體上支援 RDMA。Infiniband (IB) 是一個用於高效能計算的電腦網路通信技術，RDMA 必須在擁有此技術的硬體設備上執行。目前這些硬體如交換機、伺服器適配晶片，主要由 Intel 及 Mellanox 等公司生產。RDMA 透過 CPU 繞道和零複製機制 (zero-copy design) [1]來實現傳輸加速。它利用作業系統建立一個專屬的傳輸通道，讓應用程式可以直接透過此通道傳收資料而不需要作業系統、CPU 的參與，也就實現了 CPU 繞道。傳統的 TCP/IP 傳輸在通道的兩端都需要進行資料的複製，而 RDMA 傳輸不需要複製資料。在本地端應用程式收到 RDMA Read/Write request 的時候，IB 硬體將資料打包成封包並直接送進目標端應用程式的 virtual buffer，在那裡將封包重組回原來的資料。從傳輸開始至結束都不需要複製資料，就是所謂零複製機制。

軟體上，RDMA 可以用 C 語言呼叫 API 進行編程。兩個重要的操作如下[2]: (1) RDMA Read---呼叫端 (caller) 指定遠端的虛擬記憶體位址 (virtual address) 進行讀取，在呼叫之前，遠端必須設定目標記憶體的權限為可讀取。操作完成後不會通知遠端讀取已完成。(2) RDMA Write / RDMA Write With Immediate---前者同 Read，只是將資料讀取改成資料寫入；而後者則是會在操作完成後通知遠端主機該操作已結束。

在 Beltman 等人的研究中[3]，他們想做出一台能透過 RDMA、有效率地蒐集網路資料的 server，希望在 P4 交換機傳收封包的同時，用 RDMA 將封包也寫入 server。他們創建了一個網路拓樸結構，讓 server 和每台交換機互相連結。一開始 server 透過 TCP 傳一些建立 RDMA 連線的初始參數給交換機後，他們就能以 RDMA 互相連線了。他們用 P4 實作了 RoCEv1 (RDMA over Converged Ethernet) 協定[4]，成功讓此協定的封包在 P4 交換機中傳收，同時也成功將封包負載 (payload) 透過 RDMA Write 操作寫入到 server 的檔案中。這證實了 P4 交換機與 RDMA 的可結合性是存在的。他們的研究是將交換機的資料寫入 server，而本計畫是打算將 server 的資料反向寫進交換機的記憶體中。

執行 P4 語言的硬體或軟體稱作 P4 target。在 P4 架構中，P4 target 上有一層中間層

(intermediate layer)。P4 完成編程後，會先經過一個前端編譯器 (front-end compiler)，將 P4 程式轉成一個 target-independent HLIR (High-Level Intermediate Representation)，再由中間層拆解並分配對應的部分程式給與之對應的硬體（P4 target)，最後硬體再由一個後端編譯器 (back-end compiler) 將程式轉成可執行的機械碼供硬體執行。P4 中控制封包轉送路徑的部分為 MATs (Match-Action Tables)。在 SDN 架構中，P4 程式只需要宣告 MATs，它的 entries 是由控制層透過 P4Runtime API 遠端控制的。遠端 server 和 P4 交換機透過 gRPC 溝通 [5] 。

在 Yuhan Xue 等人的研究中[6]，他們試圖用 RDMA，而不是用傳統的 gRPC 來讓交換機讀取資料。這裡的交換機用的是 PDP (Programmable Data Plane) 交換機，P4 交換機就是其中的一種。他們的研究指出，交換機的 SRAM (Static Random-Access Memory)只能夠存數千個封包轉送表，是不夠儲存現今龐大的網路路徑資訊的。他們造出了一個系統結構，讓交換機在查找路徑時，先搜尋自身的 SRAM 空間，若搜尋不到 (table miss)，就透過 RDMA 向 server 要求路徑資料，server 再回傳資料給交換機執行。而 server 中能儲存大量資料，對交換機來說，server 的記憶體就等於是交換機的外部記憶體。

本計畫是希望 server 不只是將路徑資料傳送給交換機執行，更將它透過 RDMA Write 操作寫入交換機的記憶體。這樣若是此封包轉送表被重複利用，就能直接從交換機的 SRAM 找到，進而省去多餘的 RDMA 存取時間。本計畫將透過傳送大量封包、大量不同的封包轉送表來測量此方法是否能有效增加網路效能，並和用 RDMA 溝通但不寫入記憶體、用傳統的 RPC 做效能的分析和比較。

在 Isaias Martinez-Yelmo 等人的研究中[7]，他們希望實作出一台使用 P4Runtime 控制封包轉送路徑的交換機，但若該路徑沒有被存在 controller 裡，交換機會自動用傳統的 ARP (Address Resolution Protocol) Path 來進行封包的轉送。他們用 P4 的 extern 語法配合 BMv2-based 交換機來實現此想法。然而，當傳送大量封包時，該交換機的 ARP 效能卻會比傳統只用 ARP 的交換機的效能還要差。原因是 BMv2-based 交換機使用 RR (Round Robin) 機制來控制封包的轉送順序，而不是他們所想的 FIFO (First In First Out)。因此，在本計畫中，交換機控制封包轉送順序的機制也會是我們應該注意到的細節。

三、 研究方法及步驟

(一) 研究方法

在可編程交換機上通常有兩種記憶體裝置，一種是 TCAM (Ternary Content Addressable Memory)，而另一種是 RAM (Random Access Memory) [8,9]。而 RAM 又分為存取較快速的 SRAM (Static Random Access Memory)及較慢的 DRAM (Dynamic Random Access Memory)，在此篇計畫中先將它統一稱為 RAM。和 RAM 相比，TCAM 透過平行化搜尋達到極高的存取效能，但儲存空間相當少，只有十多 MB (megabyte)。

Flow 進入交換機後，會先比對 TCAM 中儲存的 flow table。Flow table 的架構是一個 flow entry 對應到一個 flow content。要是找到 entry，就拿出對應的 content 中的路徑資料出來對封

包做 routing。若在 TCAM 中就找到 entry，就可以不用繼續往下搜尋，反之，就要在 RAM 中搜尋 entry。若在 RAM 中也搜尋不到，就代表該 flow table 並沒有被存在交換機的內部記憶體中，就必須從外部讀取 flow table 或是直接丟棄封包。在計劃中，我們想要做到的是大量 flow table 的處理，而 flow table 的數量是超過交換機記憶體能存的空間的。這表示交換機無法將所有的 flow table 都存進它自己的記憶體，有些必定要從外部存取，本計畫結合 RDMA，讓交換機在讀取外部資訊時能有更好的存取效能。既然交換機無法存取所有 flow table，那就必須在有限的空間中，做到最有效率儲存。理論上來說，將最常用到的資料存在 TCAM，不怎麼常的存在 RAM，不太會用到的就不需要存在交換機中，真的遇到再通知 server 將資料傳送過來。這樣就能達到最短的總存取時間，也就是本計劃的目標。

本計畫必須將常用到的資料存到 TCAM 中，因此資料搬移的演算法是是相當重要的。本計畫先設計各種演算法，並透過 python 進行模擬，確定各個演算法運作的初步效能，再將其結合 P4 語言及硬體設備。另外，如何透過 RDMA 讓交換機和 server 進行溝通也是本計畫需要解決的難題。

(二) 研究步驟

1. 先用 python 寫出一些簡單的 migration 演算法來進行效能測試。詳細作法見五。

   初始化: 令 TCAM 可以存 10 個 entries、RAM 可以存 100 個，總共 1000 種 flow。隨機挑選 10 個 entries 存進 TCAM、100 個存進 RAM。每個 entry 被挑選到的機率相同，並確保 TCAM 和 RAM 存的 entry 不重複。隨機產生 10000 個 flow，一個單位時間只讓一個 flow 進到交換機。換句話說，一個 flow 進入交換機後，演算法就會開始進行 entry migration。本計畫設計四種 migration:

   (1) No Action: 不做任何操作，完全不改動 TCAM 和 RAM 存的資料。以此方法作為對照組。

   (2) Aging (圖 1): 對每個在 TCAM 和 RAM 中的 entry 都給定一個不為 0 的 reference 值。當進到交換機的 flow entry 存在 TCAM 中(TCAM Hit)，就將該 entry 的 reference 值加 1；存在 RAM 中(RAM Hit)也如此。之後再將每個 reference 值都減 1，要是 TCAM 中的 reference 為 0，就和 RAM 中擁有最大 reference 值的 entry 進行交換，這個交換 entry 的動作本計畫將它稱為 swap。Swap 完畢後將 reference 值各自設定為初始值；若是 RAM 中的 reference 為 0，就隨機選一個在 server 中的 entry 透過 RDMA 傳給 RAM，取代掉原本 reference 為 0 的 entry，這個透過 RDMA 將資料從 server 寫入 RAM 或 TCAM 的動作，我將它稱為 move。Move 完畢後也要將新 entry 的 reference 設定為初始值。

   (3) Counter (圖 2): 對每個在 TCAM 和 RAM 中的 entry 都給定一個為 0 的 count 值。要是發生 TCAM Hit，就將該 entry 的 count 值加 1；RAM Hit 也一樣。之後檢查 TCAM 中最小的 count 值有沒有比 RAM 中最大的 count 值還小。若有，就執行 swap，連帶各自的 count 值也要進行交換，這樣就能確保 TCAM 中的所有 count 值一定都比

RAM 中的 count 值還大。若 flow 的 entry 沒有被存在 TCAM 或 RAM 中，稱之為 miss。則必須將 miss 的 entry 透過 RDMA 傳給 RAM，並取代掉 RAM 中擁有最小的 count 值的 entry，並將新的 count 值設為 1。因此 miss 的次數會等於 move 的次數。

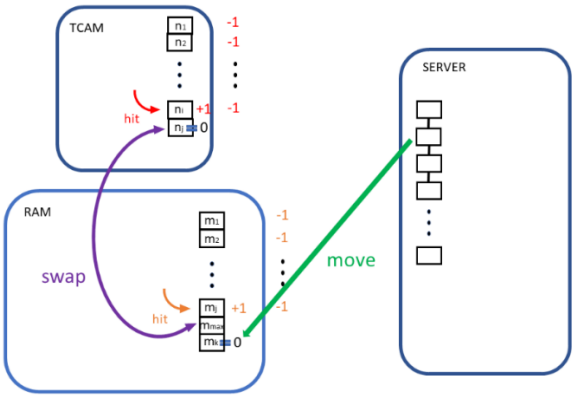

圖 1: Aging

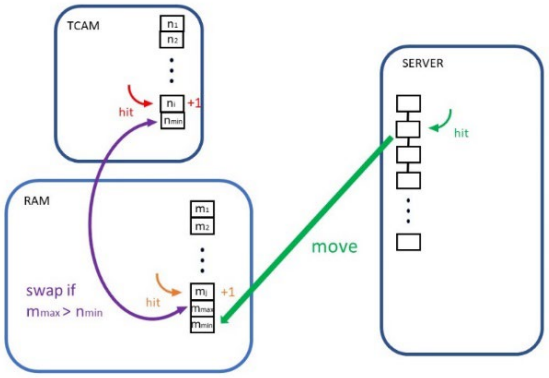

圖 2: Counter

(4) LRU(Least Recently Used) (圖 3): 不同於上述方法，LRU 將所有 entries 存成元素(link-list's elements)透過鏈結串列來維護。TCAM 中的元素串成一列，稱為 TCAM List，RAM 也是。若發生 TCAM Hit 或 RAM Hit，都將發生 Hit 的元素接到 TCAM List 的頭(第一個元素)之前，成為新的頭。若是 TCAM 中的元素數超過 10 個，就將 TCAM List 的尾(最後一個元素)去除，並接到 RAM List 的頭。發生 miss 就將 miss 的 entry 透過 RDMA 傳給 TCAM，並存成元素接到 TCAM List 的頭。若是 RAM List 中的元素超過 100 個，就直接將 RAM List 的尾去掉。因透過鏈結串列操作，swap 次數為 0，miss 次數等於 move 次數。

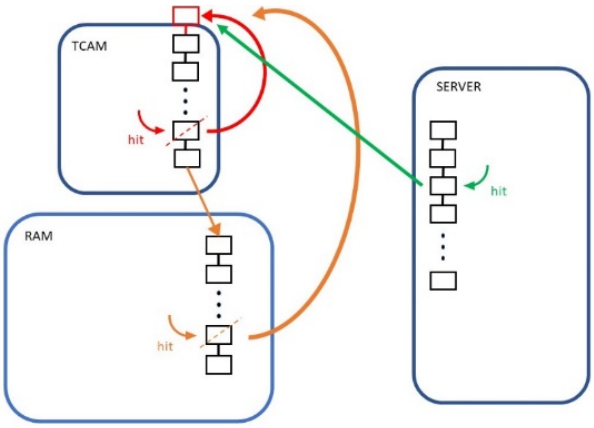

圖 3: LRU

2. 蒐集真實的網路封包資料，修改並強化演算法的準確性，實作出更符合真實情況、更為嚴謹的演算法。並用 python 模擬效能。
3. 利用 P4 語言實作出演算法，並架設真實的網路硬體設備，實際測試各演算法的效能。最後整理並分析結果。

四、 預期結果

(一) 初步研究結果及分析

本計畫利用 python 模擬了上述提到的四種方法 (No Action、Aging、Counter、LRU)。首先，假設一台交換機的 TCAM 可以存 10 個 entries，RAM 可以存 100 個 entries。總共有 1000 種不同的 flow，編號為 0 到 999。而共有 10000 個 flow 會進入交換機。為了展現快取概念的優勢，本計畫將 flow 分為 5 個等級 (表 1)。出現次數的總和是 10000 次，等級越高，代表該 flow 進入交換機的次數越多，代表它就越重要，就越應該將此 flow 存進交換機的記憶體中。

表 1: Flow 分析

Flow 編號	出現次數	等級(重要性)
0 ~ 300	0 ~ 3	1
301 ~ 600	3 ~ 7	2
601 ~ 800	7 ~ 20	3
801 ~ 900	20 ~ 40	4
901 ~ 999	40 ~ 80	5

至於產生 flow 的部分，先隨機產生每個 flow 出現的機率，再隨機選取一個數字決定 flow 出現的順序。比方說今有 flow f1、f2、f3 三種 flow，f1 出現的機率是 0.5、f2 是 0.3、f3 是 0.2。每次要讓一個 flow 進來，便從 0 到 1 隨機選一個數 x，若 $0 \le x < 0.5$，就是 f1 進來；若 $0.5 \le x < 0.8$，就是 f2 進來；若 $0.8 \le x < 1$，就是 f3 進來。假設隨機產生 5 個數: 0.35、0.92、0.21、0.56、0.47，這樣 flow 的順序便會是 f1、f3、f1、f2、f1。

在本計畫實驗中，四種方法都取 10 次測驗做平均。每次測驗的 flow 都是編號越大、機率越高，出現的次數也就越多、重要性也越高(表 1)。本實驗記錄的指標有 TCAM hit，代表 10000 個 flow 裡面中，總共有多少個 flow 的 entry 可以從交換機的 TCAM 中存取；RAM hit 則代表可以從 RAM 中存取；miss 代表 flow 不在 TCAM 或 RAM 中，必須從遠端 server 透過 RDMA 來存取。Swap 代表的是將 TCAM 的一個 entry 和 RAM 的一個 entry 對調的總次數；move 代表透過 RDMA 將 entry 寫入 TCAM 或 RAM 的總次數。TCAM content 代表在 10000 個 flow 都經過之後，TCAM 裡面存的 entry 有多少比例是 1 級的、2 級的、3 級的、4 級的、5 級的，ram content 同理。結果如表 2、表 3、表 4、表 5。

表 2: No Action

No Action	tcam hit	ram hit	miss	swap	move	tcam content	ram content	importance
	66.3	1064	8869.7	0	0	0.37	0.295	1
						0.33	0.29	2
						0.2	0.198	3

						0.03	0.118	4
						0.07	0.099	5

表 3：Aging

Aging	tcam hit	ram hit	miss	swap	move	tcam content	ram content	importance
	556.5	2554	6889.5	32.7	1179.2	0	0.104	1
						0	0.126	2
						0	0.09	3
						0.09	0.157	4
						0.91	0.524	5

表 4：Counter

Counter	tcam hit	ram hit	miss	swap	move	tcam content	ram content	importance
	710.4	3657.6	5632	90.2	5632	0	0	1
						0	0.012	2
						0	0.162	3
						0	0.319	4
						1	0.506	5

表 5：LRU

LRU	tcam hit	ram hit	miss	swap	move	tcam content	ram content	importance
	334.1	3619.7	6046.2	0	6046.2	0.01	0.002	1
						0.06	0.053	2
						0.26	0.177	3
						0.26	0.274	4
						0.41	0.494	5

根據上述結果可進行以下分析：

1. No Action: 以期望值來說，在 1000 種 flow entry 裡隨機選出 10 種存進 TCAM，TCAM hit 的機率就是 $10/1000$。總共有 10000 個 flow 進來，TCAM hit 次數的期望值就是 $10000 \times 10/1000$，也就是 100 次。RAM hit 次數的期望值以此類推，就是 1000 次。表中 tcam hit 的次數平均是 66.3 次，可能是因為 10 個 entry 太少，導致數值分布不夠穩定，偶有極端值產生；RAM hit 的次數就十分穩定，在 1000 次上下。至於 TCAMcontent 和 ram content 的部分，因越重要的 flow 編號範圍越小(表 1)，導致比例值隨著重要性主要呈現下降的趨勢。

2. Aging: 從表中可以得知，Aging 的 swap 和 move 的次數較少，miss 的次數稍較其他方法多，但仍比對照組 No Action 表現好上許多。Content 的部分大致上也都有成功存到重要性較高的 flow entry。
3. Counter: 表現最好的一個方法，TCAM hit、RAM hit 都是所有方法中最高，整體來說有將近一半的機率會發生 hit。Content 也都更為集中在重要的 flow entry 上。
4. LRU: 表現略輸 Counter。因為是維持鏈結串列的緣故，swap 的次數為 0。值得注意的是，RAM hit 的次數和 Counter 差不多，然而 TCAM hit 卻比 Counter 少了一半左右，TCAM content 也沒如此集中在重要的 entry 上。這是因為只要一發生 miss，server 就會透過 RDMA 將 entry 寫入 TCAM，就算這個 flow 是很稀少的也一樣。而這個稀少、不重要的 flow 會在 TCAM 待上至少 10 個 entry 進來的時間，才有可能被擠下去到 RAM。加上 TCAM 的空間非常稀少，才會導致此結果產生。

上述結果是用 python 模擬交換機中 flow table 搬遷的運作演算法，接下來本計畫將會結合硬體實作此系統。大致的硬體網路架構如下圖 4。由 Packet Generator 負責送封包給 P4 交換機，交換機在內部做記憶體的操作，並和遠端 server 用 RDMA 進行溝通。至於 Packet Generator 產生的封包要盡量模擬實際龐大的網路資料，這個部份我們可以參考 RouteViews [11]、CAIDA [12]等網站上的範例路由資訊。接著本計畫會監控、測量交換機中的數據，包括總傳輸時間、TCAM Hit 次數、RAM Hit 次數、Miss 次數、TCAM 資料搬遷次數、RAM 資料搬遷次數、RDMA 傳輸次數，這些數據來綜合評估各種演算法的優劣。若有演算法礙於複雜度太高無法實作在交換機中，本計畫會參考實際數據、運作方式並用 python 模擬，最後再和所有數據一起做比較、分析。

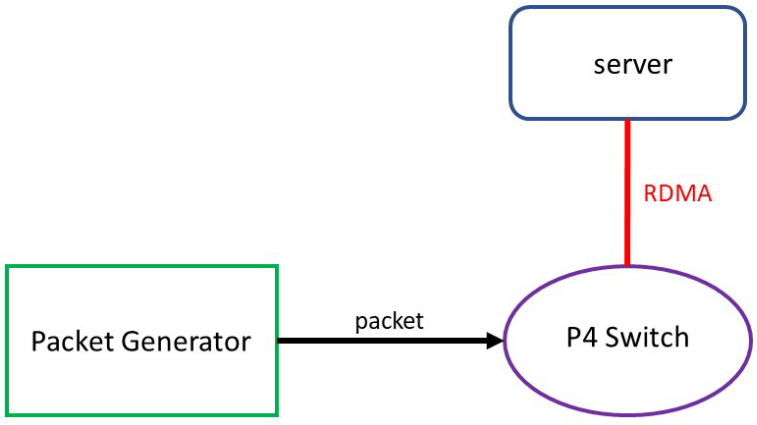

圖 4: 網路架構

(二) 預期之結果

理論上來說，使用 Least Recently Used policy 應該會使 cache hit 的機率最高[8]。然而，礙於 P4 語言的圖靈不完備性(圖靈不完備語言，無法執行條件跳轉 if、while、goto…語句)，使得許多演算法無法進行精準的實作。本計畫用 P4 實作以上演算法時，也必須特別留意到這

一點。在 Garegin Grigoryan 等人的研究中[8]，他們利用 hash table 配合 counter 的方法來實作，達到非常良好的 cache hit 比率。與之不同的是，他們直接將 cache miss 的封包捨棄掉，而本計畫會透過 RDMA 傳送路徑資料，再選擇要不要將它寫入交換機的記憶體中。在 Kaustubh Gadkari 的研究中[10]指出，在 ISP (Internet Service Provider) 網路中，1.93%的 flow entries 會有 99.5%的機率會被使用到，這比本計畫在模擬中預期的還要高出許多，代表在實際的網路環境中，所有演算法的效能都有望提升。另外，本計畫要測量的不只是傳輸時間，記憶體的消耗也是一個需要衡量的標準。雖然利用 RDMA 能將 CPU 的使用量減到最小，但交換機上的 CPU 仍然無法進行太過複雜的計算。而對於儲存的空間來說，像是 counter 之類的方法，都勢必要另外消耗空間儲存 count 值，這就有可能會壓縮到整體記憶體能存的 flow table 量。因此，本計畫設計的演算法必須要在時間、空間、硬體耗能、計算穩定度等方面取得平衡。總歸來說，本計畫將會完成:

1. 利用 RDMA 連結交換機與外部 server。
2. 以 P4 語言支援 RDMA，實現網路資料的溝通。
3. 在 P4 語言上再加上各種 entry 的搬移演算法。
4. 建構一個實際的網路結構。
5. 測量、分析，找出最有效率的演算法，並和普通的 SDN 網路一起比較。

五、 參考文獻

[1] U. Wickramasinghe, A. Lumsdaine, S. Ekanayake and M. Swany, "RDMA Managed Buffers: A Case for Accelerating Communication Bound Processes via Fine-Grained Events for Zero-Copy Message Passing," 2019 18th International Symposium on Parallel and Distributed Computing (ISPDC), Amsterdam, Netherlands, 2019, pp. 121-130, doi: 10.1109/ISPDC.2019.00025.

[2] RDMA Aware Programming user manual, Mellanox Technologies, 2015. [Online]. Available: https://www.mellanox.com/related-docs/prod_software/RDMA_Aware_Programming_user_manual.pdf

[3] R. Beltman, S. Knossen, J. Hill and P. Grosso, "Using P4 and RDMA to collect telemetry data," 2020 IEEE/ACM Innovating the Network for Data-Intensive Science (INDIS), GA, USA, 2020, pp. 1-9, doi: 10.1109/INDIS51933.2020.00006.

[4] J. Vienne, J. Chen, M. Wasi-Ur-Rahman, N. S. Islam, H. Subramoni and D. K. Panda, "Performance Analysis and Evaluation of InfiniBand FDR and 40GigE RoCE on HPC and Cloud Computing Systems," 2012 IEEE 20th Annual Symposium on High-Performance Interconnects, Santa Clara, CA, 2012, pp. 48-55, doi: 10.1109/HOTI.2012.19.

[5] F. Hauser, M. Häberle, M. Schmidt and M. Menth, "P4-IPsec: Site-to-Site and Host-to-Site VPN With IPsec in P4-Based SDN," in IEEE Access, vol. 8, pp. 139567-139586, 2020, doi: 10.1109/ACCESS.2020.3012738.

[6] Y. Xue and Z. Zhu, "Hybrid Flow Table Installation: Optimizing Remote Placements of Flow Tables on Servers to Enhance PDP Switches for In-Network Computing," in IEEE Transactions on Network and Service Management, doi: 10.1109/TNSM.2020.3045711.

[7] I. Martinez-Yelmo, J. Alvarez-Horcajo, M. Briso-Montiano, D. Lopez-Pajares and E. Rojas, "ARP-P4: A Hybrid ARP-Path/P4Runtime Switch," 2018 IEEE 26th International Conference on Network Protocols (ICNP), Cambridge, 2018, pp. 438-439, doi: 10.1109/ICNP.2018.00062.

[8] G. Grigoryan, Y. Liu and M. Kwon, "PFCA: A Programmable FIB Caching Architecture," in IEEE/ACM Transactions on Networking, vol. 28, no. 4, pp. 1872-1884, Aug. 2020, doi: 10.1109/TNET.2020.3001904.

[9] T. Nguyen-Viet and D. Le, "TCAM-based flow lookup design on FPGA and its applications," 2015 International Conference on Advanced Technologies for Communications (ATC), Ho Chi Minh City, 2015, pp. 378-382, doi: 10.1109/ATC.2015.7388355.

[10] K. Gadkari, M. L. Weikum, D. Massey and C. Papadopoulos, "Pragmatic router FIB caching," 2015 IFIP Networking Conference (IFIP Networking), Toulouse, 2015, pp. 1-9, doi: 10.1109/IFIPNetworking.2015.7145296.

[11] University of Oregon Route Views Project [Online]. Available: http://www.routeviews.org/routeviews/

[12] CAIDA Center for Applied Internet Data Analysis [Online]. Available: https://www.caida.org/home/

六、 需要指導教授指導內容

(一) 指導儀器的原理、操作及使用方法。

(二) 理論知識的傳授及指導。

(三) 討論實驗數據及結果分析。