

國立陽明交通大學

資訊學院 資訊工程學系

大學部專題論文

## **RDMA-based Adaptive Remote Lookup using P4**

組員一： 0716206 陳昱丞

組員二： 0716221 余忠旻

組員三： 0717035 王偉軒

指導教授：林靖茹

## 中文摘要

在一個複雜的 SDN 網路架構中，交換機上的路由表有可能因記憶體空間有限而不敷使用，此時需要向遠端控制器詢問路由規則，造成延遲。本研究提出一個路由規則安裝機制，將大流量資料的路由資訊盡可能存放在交換機記憶體中，而小流量資料路由資訊則留在遠端控制台。本研究成功設計出一套有效的路由規則遷移機制，使用 P4 語言、count- min sketch 演算法與 RDMA 機制，以達到即時且有效率的路由更新。此路由搬遷演算法能使交換機路由表使用率高達 88%，而 RDMA 根據不同的負載量，能使交換機和遠端控制器的溝通延遲縮短達最多 7 倍。本架構不僅能大幅降低交換機與遠端控制器的溝通次數及延遲，增進網路效能與路由資訊即時性，也能作為一種新型且高效的網路遙測手段。

## 英文摘要

In a complex SDN network architecture, a switch usually has a limited memory size, which cannot install the forwarding rules for all the flows. To overcome this limitation, an SDN integrates local table lookup with remote table lookup to ensure reliable forwarding. However, remote lookup typically incurs a much longer latency, which would lower the overall throughput of a network. In this project, we introduce an efficient rule migration algorithm that installs the rules of heavy flows in the data plane while migrating the rules of light flows to the remote controller. Since traffic patterns change dynamically, we designed a P4 pipeline that leverages count-min sketch and RDMA to enable adaptive rule migration. Using our migration algorithm, the usage rate of a switch's routing table reaches 88%. Furthermore, the signaling overhead reduces at most 7 times according to the load of RDMA. We have shown that the proposed migration algorithm not only enhances forwarding efficiency but also reduces signaling overhead. This can additionally become a new and efficient network telemetry method.

## 目錄

中文摘要.....	1
英文摘要.....	2
目錄.....	3
圖目錄.....	4
表目錄.....	5
第一章 緒論.....	6
1. 研究動機與研究問題.....	6
2. 文獻回顧與探討.....	7
第二章 實驗工具與理論.....	10
1. Python Scapy.....	10
2. Mininet.....	10
3. Bmv2.....	10
4. P4.....	11
5. P4 Runtime API.....	11
6. RDMA 傳輸模式.....	12
7. RDMA 基本元件和傳輸架構.....	14
8. Count-Min Sketch.....	17
9. 系統模組:網路環境.....	20
10. 系統模組: RDMA.....	21
11. 系統架構: P4 交換機.....	21
12. 系統架構: RDMA.....	22
第三章 研究方法及步驟.....	23
1. 簡單網路.....	23
2. 複雜網路.....	28
3. RDMA 實驗流程.....	28
4. RDMA 實驗步驟.....	29
第四章 結果分析與探討.....	33
1. 複雜網路.....	33
2. RDMA.....	36
第五章 結論與未來展望.....	39
參考文獻.....	40

## 圖目錄

圖一: RoCE header 架構 .....	9
圖二: bind_layers .....	10
圖三: P4 parser pipeline 中的 extract 和 select .....	10
圖四: P4 Runtime 指令集 .....	11
圖五: P4 交換機路由表 .....	12
圖六: 左圖是傳統網卡傳輸過程，右圖是 DMA 機制下傳輸過程 .....	13
圖七: 傳統網路透過 DMA 機制下傳輸模式 .....	13
圖八: RDMA 傳輸模式 .....	14
圖九: 上圖傳統傳輸模式(socket)架構，下圖是 RDMA 傳輸模式架構(kernel bypass 機制) ....	14
圖十: Work Queue 架構 .....	15
圖十一: Queue Pair 傳輸架構 .....	16
圖十二: Completion Queue 架構 .....	16
圖十三: RDMA 傳輸架構 .....	17
圖十四: count-min sketch 第一步 .....	18
圖十五: count-min sketch 第二步 .....	18
圖十六: count-min sketch 第三步 .....	19
圖十七: count of f1 = min (r1[2], r2[5]) = min (2, 3) = 2 .....	19
圖十八: P4 中的 hash function .....	20
圖十九: 網路系統模組統整圖 .....	21
圖二十: P4 Switch 演算流程圖 .....	22
圖二十一: h1 送封包給 h2 .....	23
圖二十二: h1 送封包給 h3，發生 table miss .....	24
圖二十三: s1 將封包轉傳給 controller .....	24
圖二十四: controller 修改封包欄位，將 port 資訊裝入 .....	25
圖二十五: routing table update .....	25
圖二十六: 實際操作畫面圖一 .....	26
圖二十七 實際操作畫面圖二 .....	27
圖二十八 實際操作畫面圖三 .....	27
圖二十九: 複雜網路架構 .....	28
圖三十: 查找路由步驟 .....	30
圖三十一: RDMA 指派工作架構 .....	30
圖三十二: 連續傳輸實作 .....	31
圖三十三: Memory Region 鎖定 .....	31
圖三十四: 相同低 CPU 負載下不同 local SDN switch 數目與傳輸時間比較圖 .....	36
圖三十五: 相同 local SDN switch 連線數目下 CPU 負載程度與傳輸時間比較圖 .....	37
圖三十六: 不同 local SDN switch 數目下兩傳輸方式時間之倍率比較圖 .....	38

## 表目錄

表一: RDMA、RoCE、軟體 RoCE、不使用 RDMA 之傳輸比較 RDMA、RoCE、軟體 RoCE、不使用 RDMA 之傳輸比較 .....	9
表二: 五次交換機 table hit 比率實驗結果.....	33
表三: 交換機 routing table entry 內容、count 值和 major flow 之封包數比較 .....	34
表四: 大規模系統之實驗結果.....	35

## 第一章 緒論

### 1. 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 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)或 socket 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，本研究引進 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。

另外，在執行路由安裝演算法的同時，我們需要統計每個 flow 流經一台交換機的流量。我們會用到 P4 的 register 功能，配合上 RDMA 的傳輸。這就是一種全新的網路遙測手段。過去許多網路遙測的方式，皆有一些無法避免的缺點。例如 IPMP (IP Measurement Protocol) 雖然能夠測量出丟包率等資訊，但卻會占用到許多的網路頻寬，影響運行。而若是採用帶內網路遙測(In-band Network Telemetry, 簡稱 INT)的模式，雖然能夠減少網路頻寬的佔用、也能得知較為全面的資訊，但其不斷增加的 INT Metadata，在測量中透過交換機轉發時容易造成問題。而我們這種遙測方式，直接將流量訊息存在每台 P4 交換機中，不需要任何的 Metadata、也不需要任何多餘的封包占用頻寬，且每台交換機透過 RDMA 連接 Controller，可將遙測數據透過 RDMA，以最即時的方式傳輸。

因此，我們設計的演算法，不但能使複雜的網路架構能有更好的傳輸效率，也能同時進行高效率的網路遙測，是一舉兩得的好方法。

## 2. 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撲結構，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 HIR (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。

本研究是希望 server 不只是將路徑資料傳送給交換機執行，更將它透過 RDMA 操作寫入交換機的記憶體。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]。存在記憶體中的好處是若是此封包轉送表被重複利用，就能直接從交換機的記憶體找到，進而省去多餘的 RDMA 存取時間。本研究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。

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遠端控制器裡，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)。

本研究中的演算法中，需要用 P4 中的 register 來實作 count-min sketch[13]來對流量進行估計。Count-min sketch 為利用 Bloom Filters 演算法的一種 sketch 方式。其儲存方式為一個二維陣列並透過多個獨立的雜湊函數，將其雜湊運算後得到的值作為索引，取出陣列中的值加一。Count-min sketch 利用雜湊函數計算的方式，減少空間的使用量。根據數據結果顯示[14]，它能夠擺脫傳統在處理資料流問題的空間複雜度下限  $\Omega(1/\epsilon^2)$  限制，將 sketch 之空間複雜度上限降為  $O(1/\epsilon)$ ，大幅降低 sketch 占用的記憶體空間。但減少記憶體空間使用的代價是需要利用 CPU 進行雜湊函數的計算。而在本研究中，對於每一筆網路流量都要經過多次的雜

湊函數運算，因此在執行時，對於交換機 CPU 存在著一定運算量的負擔。

另一方面，本研究中需要使 RDMA 能夠在乙太介面的 P4 交換機上進行傳輸，因此將會使用到 RoCE 這項技術，以達成在乙太介面上的 RDMA 傳輸之目的。RoCE 採用和 RDMA 相同的零複製資料傳輸模式，故和原生 RDMA 相比，仍然能夠保持低 CPU 占用率的特性[15]。然而由於其依然並非原生的 RDMA 傳輸，所以在傳輸的頻寬和延遲的表現上仍會有所不及。比較 RoCE、原生 RDMA 與未使用 RDMA 方式傳輸的差異(表一)[16]，其測量結果顯示，RoCE 相較於原生 RDMA，在效率方面雖然有下降的現象，但若與未採用 RDMA 方式傳輸進行比較，在降低延遲的提升上，仍然有 5.7 倍的進步，而在頻寬方面的提升也有 3.8 倍。P4 為不具圖靈完備性 (Turing-complete) 的語言[17]，意即無法執行條件跳轉 while、goto...等語句，無法直接使用迴圈的功能，也無法如 C 語言一般任意宣告分配記憶體。若要使 P4 交換機實現 RDMA 傳輸，必須透過實作出圖的 RoCE header (圖一)[3]，並針原本的傳輸方式進行修改，才能夠使 P4 交換機順利在乙太介面上進行 RDMA 傳輸。

表一: RDMA、RoCE、軟體 RoCE、不使用 RDMA 之傳輸比較 RDMA、RoCE、軟體 RoCE、不使用 RDMA 之傳輸比較

	RDMA	RoCE	軟體 RoCE	No RDMA
延遲(微秒)	1.96	3.7	11.6	21.09
頻寬(MB/s)	5481.9	2284.7	x	1136.1

Ethernet	Global Route Header	Base Transport Header	RDMA Extended Transport Header	Payload	Invariant CRC
----------	---------------------	-----------------------	--------------------------------	---------	---------------

圖一: RoCE header 架構

## 第二章 實驗工具與理論

### 1. Python Scapy

Scapy 是一個網路封包處理工具，以 Python 編寫。它可以讓使用者任意撰寫封包內容並發送。除了基本的 Ethernet、IP、TCP、UDP 等 header，還可以自訂義 header (圖二)。傳送自訂義的封包經過 P4 交換機時，可以透過 P4 語言來解析 (圖三)。

```
bind_layers(Ether, MyPort, type=TYPE_MYPORT)
bind_layers(MyPort, IP, pid=TYPE_IPV4)
```

圖二: bind\_layers

透過 scapy 的 bind\_layers[19] 方法，將自訂的 MyPort header 添加在 Ether 和 IP 層中間

```
state parse_myPort {
    packet.extract(hdr.myPort);
    transition select(hdr.myPort.proto_id) {
        TYPE_IPV4: parse_ipv4;
        default: accept;
    }
}
```

圖三: P4 parser pipeline 中的 extract 和 select

在 P4 的 parser pipeline 中，extract 和 select 方法同樣可以適用於自訂義的 header

### 2. Mininet

Mininet[20] 是一個軟體的網路模擬工具，運行在單一個 Linux kernel 中。代表可以在單一台電腦上模擬交換機、路由器、用戶端等網路設備的資料傳輸行為。我們可以利用它建立一個網路拓樸環境，並實際用 ssh 等遠端連線行為進入拓樸中的網路設備，並運行程式。本研究就是要用 Mininet 建立一個單一交換機的區域網路拓樸，並在交換機中運行 Python 和 P4 來控制交換機的行為。

### 3. Bmv2

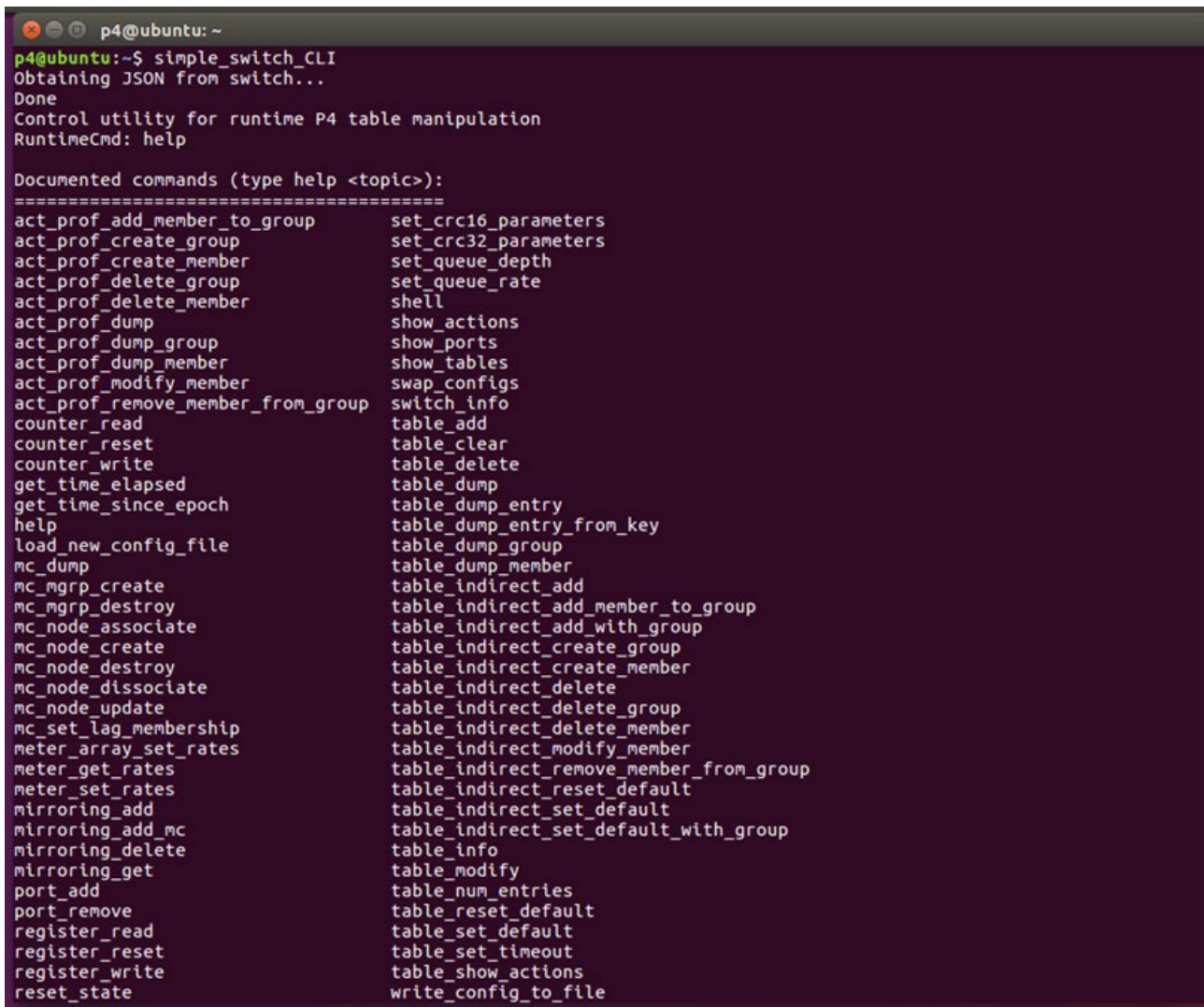
Bmv2[11] 的全名是 Behavior model version 2。如其名，它是一個用來模擬 P4 交換機的軟體。也可以說是一個運行 P4 語言的平台。P4 編譯器將一份 P4 原始碼編譯成一個 JSON 檔後，Bmv2 就可以透過那個 JSON 檔，模擬出 P4 交換機封包處理、轉送的行為。

## 4. P4

P4[12]全名為 Programming Protocol-Independent Packet Processors，是在 2013 年發表的新型交換機程式語言，以控制交換機的行為為目的。其特色是以 pipeline 的方式分區、分段對封包進行處理，包含封包的 분석、拆解、重組、轉送。P4 最大的優點就是 target independence 和 protocol independence。只要有對應的 P4 compiler，P4 程式就能運行在任何網路元件。且它不受 Ethernet、TCP、UDP 等傳統 protocol 限制，任何自訂的 protocol，P4 都能夠處理。本研究就是用上述的 Bmv2 模擬交換機，並撰寫 P4 程式控制其行為。

## 5. P4 Runtime API

至於動態控制交換機的路由表，可以用 P4 Runtime 這個使用者介面來達成。進入交換機介面並輸入 simple\_switch\_CLI 指令，便可以進入 P4 Runtime 控制台。在控制台輸入 help，可以看到所有控制指令，如下圖四。



```
p4@ubuntu: ~
p4@ubuntu:~$ simple_switch_CLI
Obtaining JSON from switch...
Done
Control utility for runtime P4 table manipulation
RuntimeCmd: help

Documented commands (type help <topic>):
=====
act_prof_add_member_to_group      set_crc16_parameters
act_prof_create_group             set_crc32_parameters
act_prof_create_member            set_queue_depth
act_prof_delete_group             set_queue_rate
act_prof_delete_member            shell
act_prof_dump                     show_actions
act_prof_dump_group               show_ports
act_prof_dump_member              show_tables
act_prof_modify_member             swap_configs
act_prof_remove_member_from_group switch_info
counter_read                       table_add
counter_reset                      table_clear
counter_write                      table_delete
get_time_elapsed                  table_dump
get_time_since_epoch              table_dump_entry
help                               table_dump_entry_from_key
load_new_config_file              table_dump_group
mc_dump                           table_dump_member
mc_mgrp_create                    table_indirect_add
mc_mgrp_destroy                   table_indirect_add_member_to_group
mc_node_associate                 table_indirect_add_with_group
mc_node_create                    table_indirect_create_group
mc_node_destroy                   table_indirect_create_member
mc_node_dissociate                table_indirect_delete
mc_node_update                    table_indirect_delete_group
mc_set_lag_membership             table_indirect_delete_member
meter_array_set_rates             table_indirect_modify_member
meter_get_rates                   table_indirect_remove_member_from_group
meter_set_rates                   table_indirect_reset_default
mirroring_add                     table_indirect_set_default
mirroring_add_mc                  table_indirect_set_default_with_group
mirroring_delete                  table_info
mirroring_get                     table_modify
port_add                          table_num_entries
port_remove                       table_reset_default
register_read                      table_set_default
register_reset                     table_set_timeout
register_write                     table_show_actions
reset_state                       write_config_to_file
```

圖四: P4 Runtime 指令集

使用 table\_add、table\_delete 可以增加、刪減路由表的 entry。table\_dump 可以看到現在交換機的路由表，如下圖五。

```
RuntimeCmd: table_dump ipv4_lpm
=====
TABLE ENTRIES
*****
Dumping entry 0x0
Match key:
* ipv4.dstAddr      : LPM      0a000101/32
Action entry: MyIngress.ipv4_forward - 080000000111, 01
*****
Dumping entry 0x1
Match key:
* ipv4.dstAddr      : LPM      0a000202/32
Action entry: MyIngress.ipv4_forward - 080000000222, 02
=====
Dumping default entry
Action entry: MyIngress.to_controller - 04
=====
RuntimeCmd:
```

圖五: P4 交換機路由表

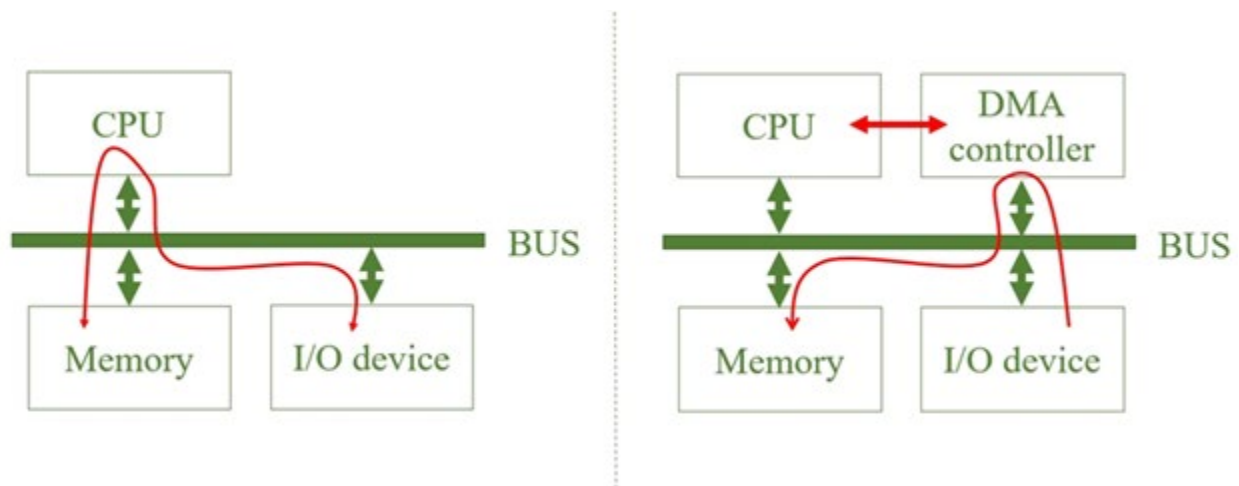
以上圖為例，在 entry 0 中，destination address IP 0a000101 拆解為 0a.00.01.01，換成十進位就是 10.0.1.1，其對應的 port 是 1。依此類推，10.0.2.2 對應的 port 是 2。而當前面的 entry 都不符合時，交換機就會選擇 default entry，在上圖中，也就是 port 4，封包將會從 port 4 被送出。在本研究中，將會使用腳本配合 P4 Runtime 來自動化動態增減路由表的過程，將使用較為頻繁的 entry 自動加入交換機的路由表，並將較不頻繁的剔除，以達成減少遠端要求路由訊息次數之目的。

## 6. RDMA 傳輸模式

傳統普通網卡，為了從內存拿到需要發送出去的數據，需要將數據重新分裝，再利用 physical layer 送出，但在這過程中，網卡需要告訴 CPU 傳輸的數據請求，然後 CPU 會將內存緩衝區的數據複製到自己內部的主 register 中，然後才複製到 I/O device 的儲存空間，在藉此發送出去，但如果數據量過大，CPU 會忙於搬移數據，而無法從事計算等工作，因此有了 DMA 機制。

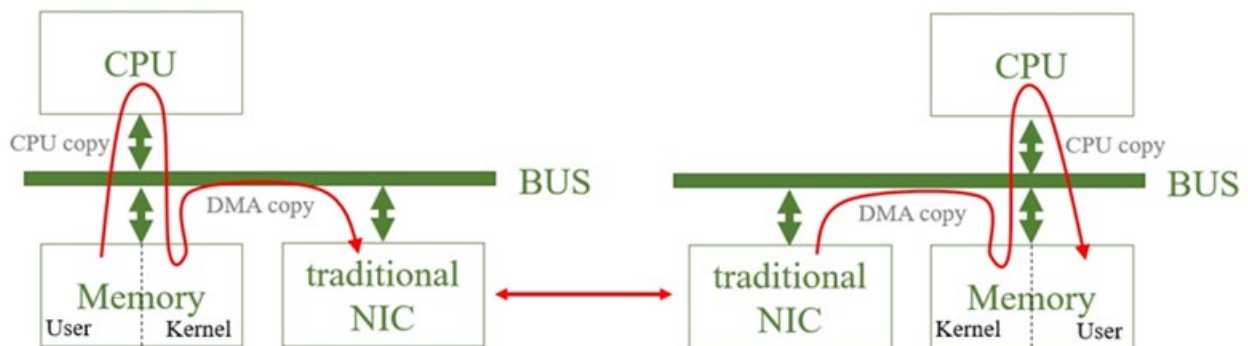
DMA(Direct Memory Access)是專門來讀寫內存的機制，如圖六。當網卡要複製數據時，大部分的過程會交由 DMA Controller 來完成，DMA Controller 會將內存的數據複製到 DMA Controller 的 register，再由 register 複製到 I/O device，藉此完成數據傳輸，在這全部的過程中，CPU 只需和 DMA Controller 溝通，不再需要參與數據複製，因此可以在此時間做其他工作，來提升整體效率。





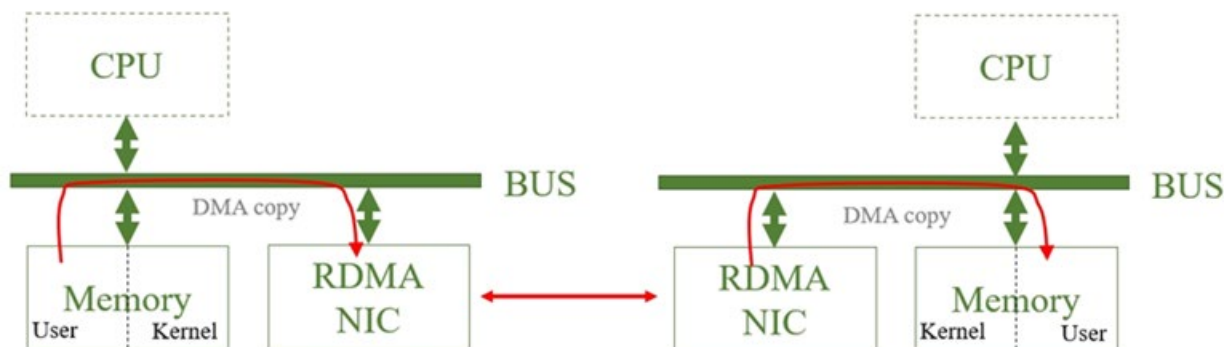
圖六：左圖是傳統網卡傳輸過程，右圖是 DMA 機制下傳輸過程

傳統網路中，要將節點 A 送數據給節點 B，無論是發送端 A 還是接收端 B 都需要 CPU 的控制，發送端需先透過 CPU 將數據從 user space 複製到 kernel space，網卡才得以訪問，接收端網卡收到後，會先透過 DMA 將數據複製到 kernel space，再透過 CPU 將數據從 kernel space 複製到 user space，CPU 需要完成數據在 kernel space 和 user space 的複製、網卡的控制、數據的封裝及解析等等，數據傳輸則是透過 TCP/IP 網路協定來完成讀寫遠端內存，如下圖七，可以看見即使有 DMA 機制，CPU 負荷還是相當大。



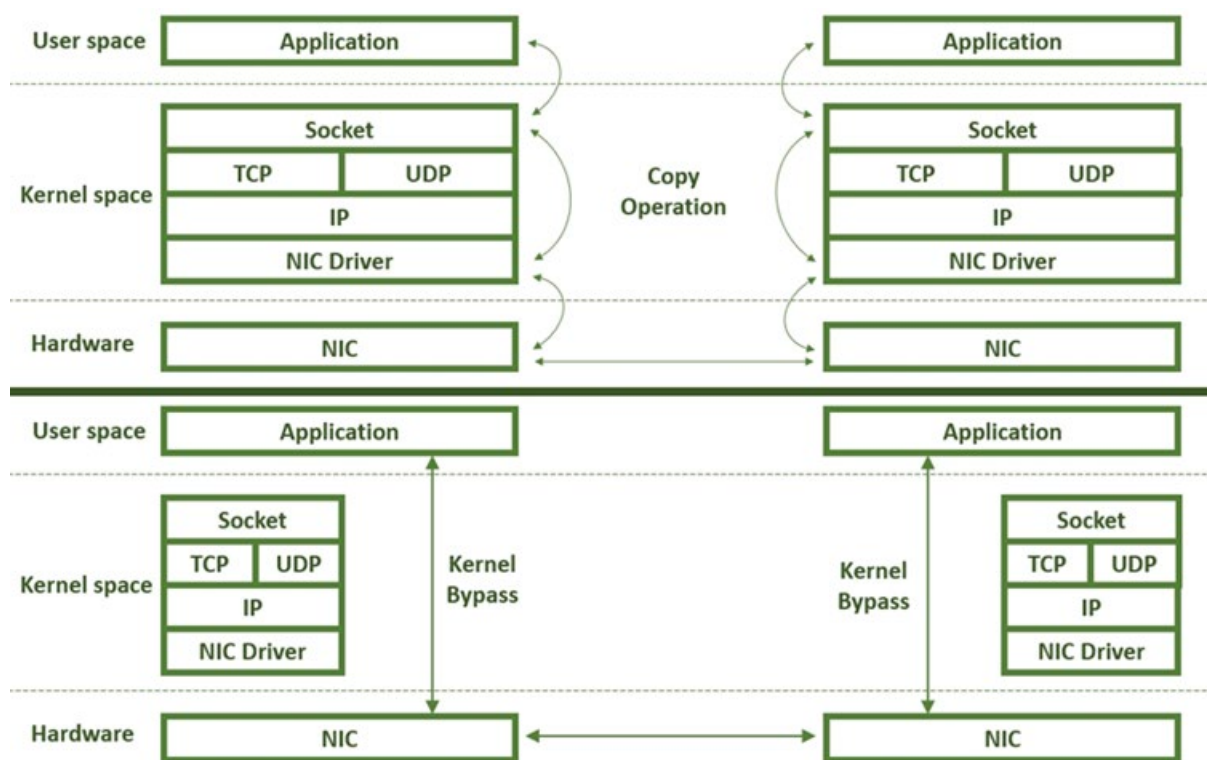
圖七：傳統網路透過 DMA 機制下傳輸模式

RDMA(Remote Direct Memory Access)是進一步提升遠程傳輸效率，讀寫過程大部分透過硬體來完成，達到直接訪問遠端節點的內存，透過 RDMA 能使兩端點 A,B 的 CPU 幾乎不參與數據傳輸，發送端網卡能直接從內存的 user space 將數據複製到內部儲存空間，在經由組裝、傳輸到接收端網卡，再將數據直接複製到內存的 user space。相比傳統網路，RDMA 不需要將數據在 user space 和 kernel space 來回複製，也就是 RDMA 有零複製的機制，能降低 CPU 在傳輸數據中的使用，如下圖八。



圖八: RDMA 傳輸模式

RDMA 除了有零複製的機制，還有 kernel bypass 的機制，在數據傳輸當中能繞過 kernel，application layer 能直接準備好數據並通知網卡發送以及接收，藉此避免系統調用和上下文切換的開銷，如下圖九。在 RDMA 對遠端進行續寫動作時，假如擁有遠端某區域內存的鑰匙之後，能不通知遠端 CPU 直接進行續寫動作，其原理是透過將 payload 的封裝和解析放到硬體來做，稱為 CPU 卸載機制，來減少傳統網路中 CPU 對各層 payload 的分析的開銷。



圖九: 上圖傳統傳輸模式(socket)架構，下圖是 RDMA 傳輸模式架構(kernel bypass 機制)

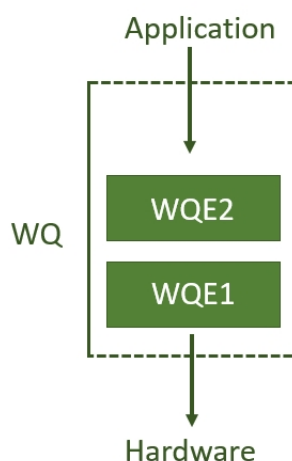
## 7. RDMA 基本元件和傳輸架構

RDMA 對內存進行數據傳輸時，RDMA 網卡需要先註冊 Memory Region，將需要操作的內存註冊到 RDMA Memory Region，讓這區域的內存交給 RDMA 保護，在這段期間，數據傳

輸中 application 不能修改數據所在的內存，也不能對數據所在的內存做 page out 的行為，這樣能避免非法的存取行為，也能夠避免當內存再次被 paging 時，數據會有所流失。

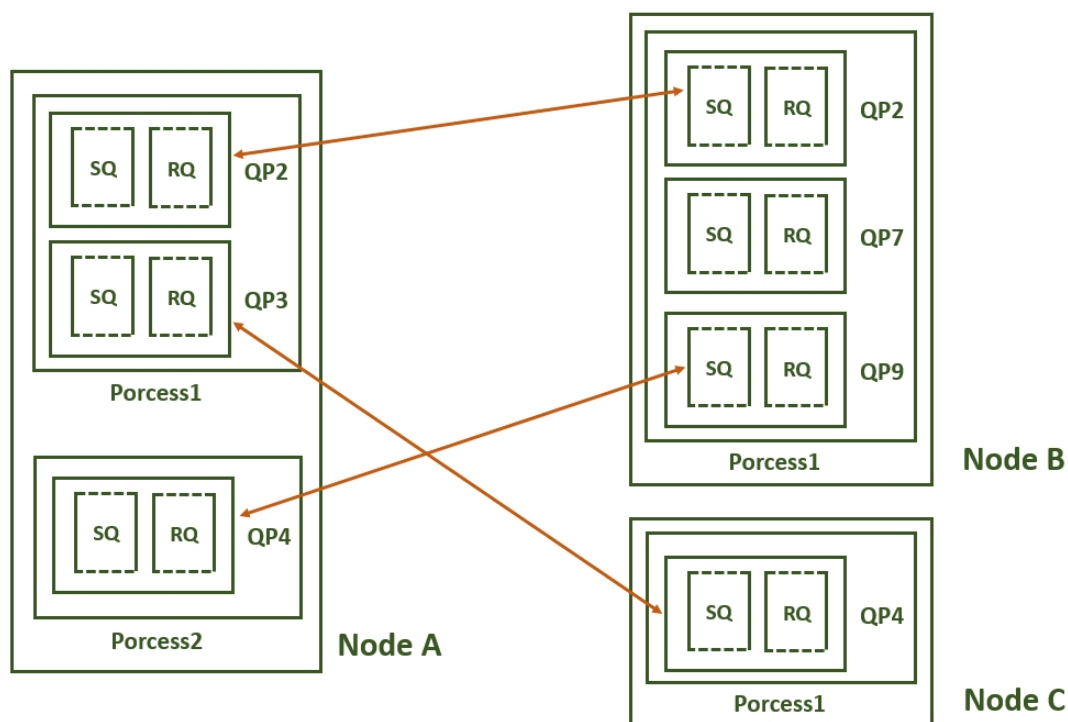
在 RDMA 註冊完 Memory Region 之後，RDMA 傳輸的基本單位是 QP(Queue Pair)，也就是我們前面提到的 WQ，每一個 QP 會有唯一一個 QPN(Queue Pair Number)，用來確定雙方是哪一個 QP 來進行傳輸數據的行為，而 RDMA 主要分為三種 QP，來支持 RDMA 傳輸，分別為 SQ(Send Queue)、RQ(Receive Queue)、CQ(Completion Queue)。

SQ 和 RQ 分別是用來存放發送傳輸任務和用來存放接收傳輸任務的 Queue，這兩個我們將之稱為 WQ(Work Queue)，WQ 是一個用來儲存任務請求的 Queue，WQE(Work Queue Element) 則是軟體發給硬體的任務請求，當中記錄著有關任務的詳細資訊，WQE 會交由軟體一一放進 WQ，再由硬體取出來下達任務，它的架構如下圖十。換句話說，前面提到的 SQ 和 RQ 其實就是 WQ 的實例，如下圖十一。每一個 QP 會和遠端的一個 QP 配對進行傳輸，在發送方，軟體將 WQE 放進 SQ，在交由硬體執行，稱之為 Post Send，反之，在接收方，軟體將 WQE 放進 RQ，在交由硬體執行，稱之為 Post Receive。



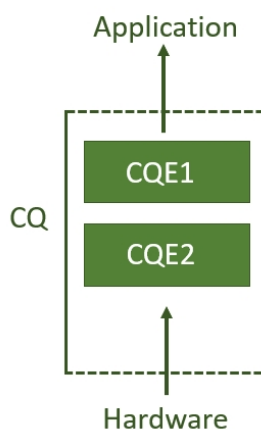
圖十: Work Queue 架構





圖十一: Queue Pair 傳輸架構

CQ 則是用來存放任務完成的 Queue，當硬體完成後會返回 CQE(Completion Queue Element)給軟體，其中 CQE 會描述 WQE 的完成資訊，包含此項任務是否有被正確的執行，如果有遭遇到錯誤，會回報是何種錯誤，CQ 的架構如下圖十二。

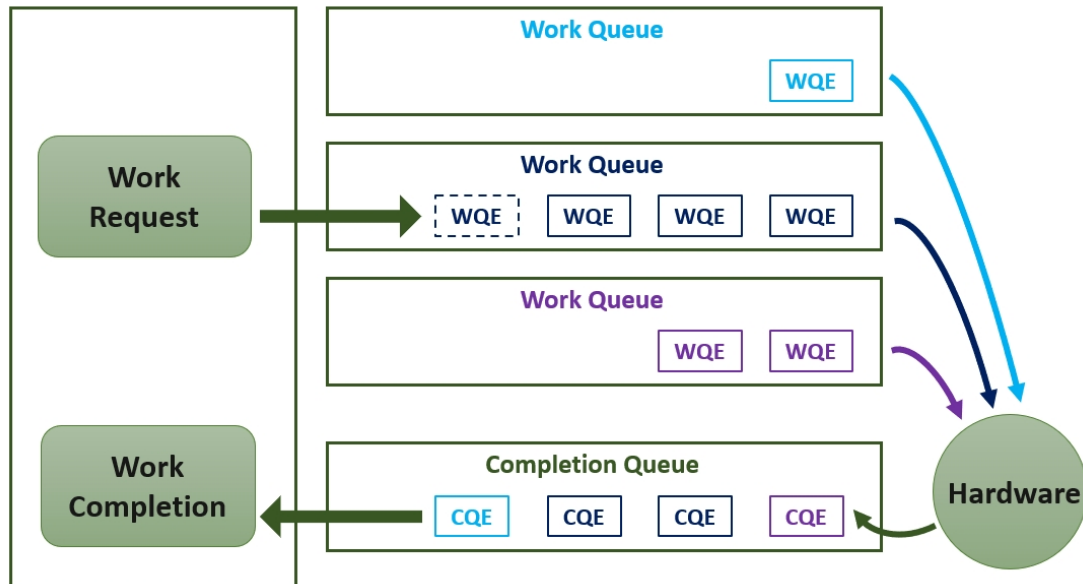


圖十二: Completion Queue 架構

總結來說，利用這些 QP，RDMA 傳輸操作主要可以分為五步，如下圖十三。

1. 用戶透過 API 提交 WR(Work Request)到 WQ，對於用戶來說 WQE 是不可見的，所以要在 application 提交 WR，讓驅動將 WQE 放入 WQ。
2. RDMA 硬體會將 WQ 中的 WQE 取出，執行任務。

3. 任務執行完成，RDMA 硬體會將 CQE 放入 CQ。
4. 因為 CQE 對於用戶也是不可見的，需透過驅動將 CQ 中的 CQE 取出，回報 WC(Work Completion)給 application，讓用戶知道任務完成。
5. 重複 1-4 步直到 WQ 為空。

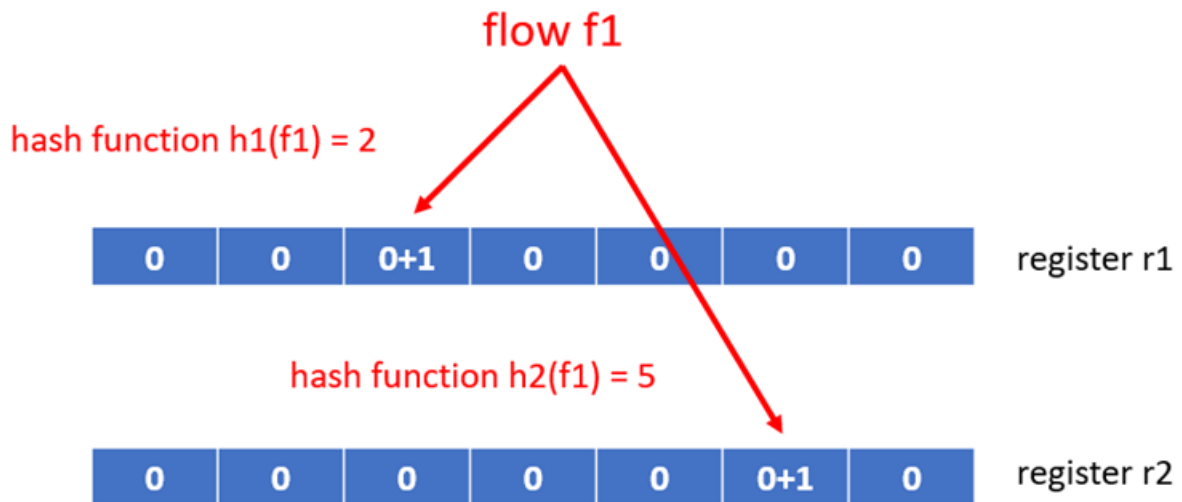


圖十三: RDMA 傳輸架構

## 8. Count-Min Sketch

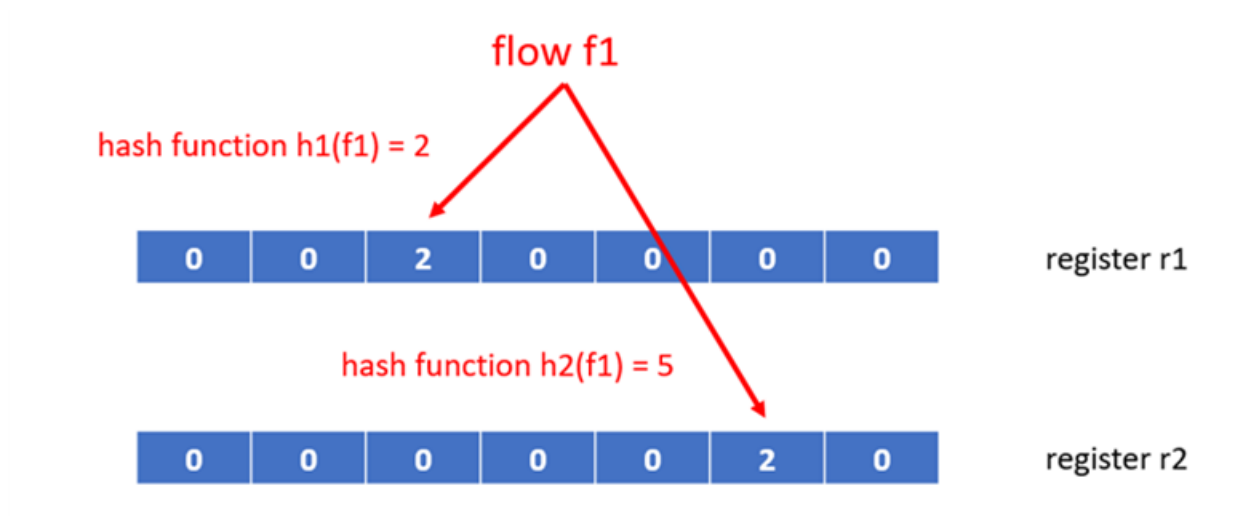
Count-Min sketch 在本研究中可以視為一種估計 flow 大小(封包個數)的方法，flow 愈大表示封包個數愈多。交換機記憶體空間有限，count-min sketch 雖犧牲些許準確率，但可減少大量的儲存空間。以接下來的圖(圖十四、十五、十六、十七)為例解釋交換機中的 count-min sketch 運作，一個 flow 中的一個封包以 fl 表示。

下圖十四中，兩個 register r1、r2 都先歸零。P4 中的 register 就如同其他語言中的 array，有儲存數據的功能。首先，將 fl 以 5 tuple 的形式代入第一個 hash function h1。在 flow 中所謂的 5 tuple 是指以 source ip, source port, destination ip, destination port, layer four protocol 這五個元素所組成的 tuple。代入得到的結果為 2，就將 r1 index 2 的值加一；代入 h2 得到 5，就將 r2 index 5 的值加一。



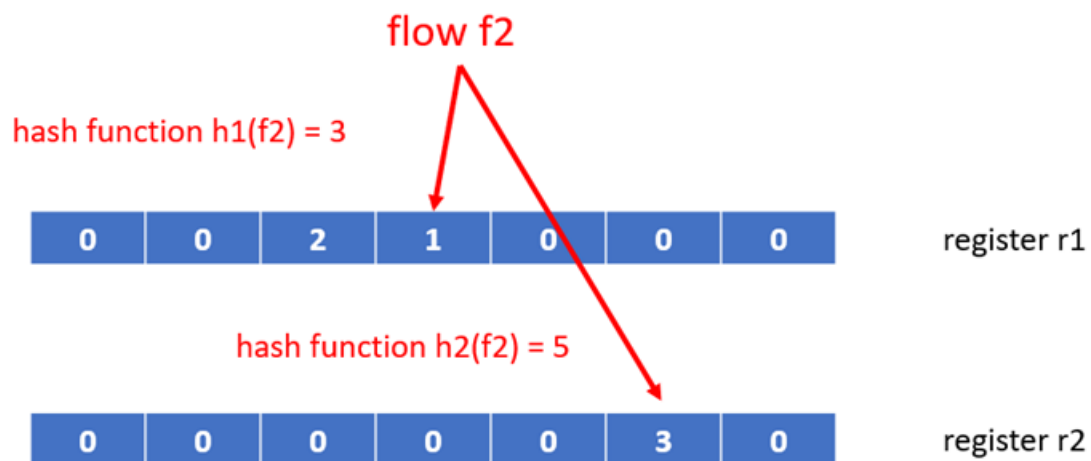
圖十四: count-min sketch 第一步

接著若 f1 又進來一個封包，則再將剛剛的步驟重複一次。



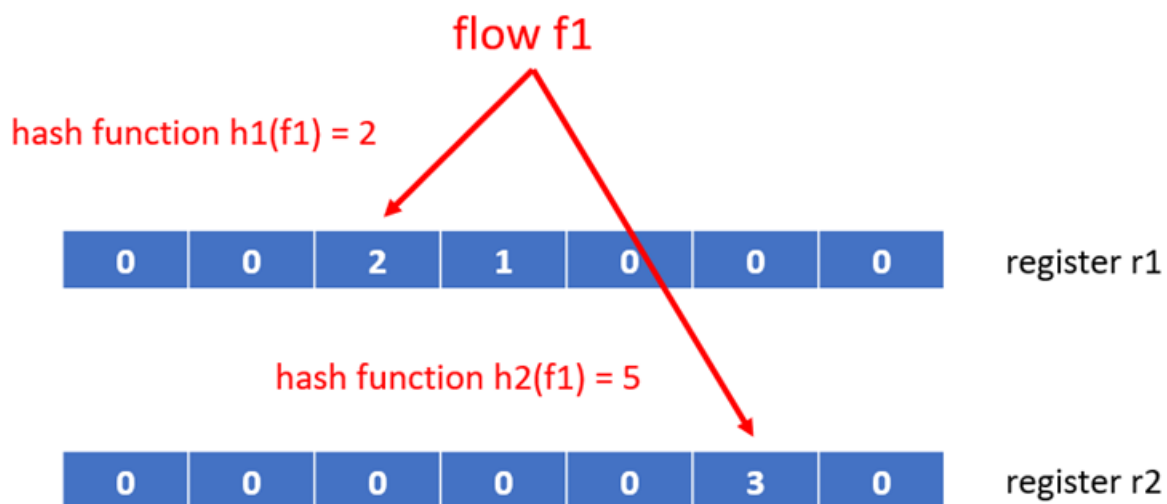
圖十五: count-min sketch 第二步

接下來進了另外一個 flow f2，f2 代入兩個 hash function 得到 3 和 5，就一樣將對應的 index 值內容加一。



圖十六: count-min sketch 第三步

最後，當我們要估計  $f1$  的大小時，一樣將  $f1$  代入  $h1$ 、 $h2$ ，得到 index 值 2、5。將  $r1$ 、 $r2$  的 index 2、5 之值取出來，並取最小值，就是 count-min sketch 之結果。此例得到 2、3，因此估計  $f1$  之大小為 2。



圖十七:  $\text{count of } f1 = \min(r1[2], r2[5]) = \min(2, 3) = 2$

```

hash(meta.hashIndex1,
    HashAlgorithm.crc16,
    IDX_BASE,
    {
        hdr.ipv4.srcAddr,
        hdr.ipv4.dstAddr,
        hdr.ipv4.protocol,
        hdr.tcp.src_port,
        hdr.tcp.dst_port
    },
    IDX_CNT);

hash(meta.hashIndex2,
    HashAlgorithm.crc32,
    IDX_BASE,
    {
        hdr.ipv4.srcAddr,
        hdr.ipv4.dstAddr,
        hdr.ipv4.protocol,
        hdr.tcp.src_port,
        hdr.tcp.dst_port
    },
    IDX_CNT);

```

圖十八: P4 中的 hash function

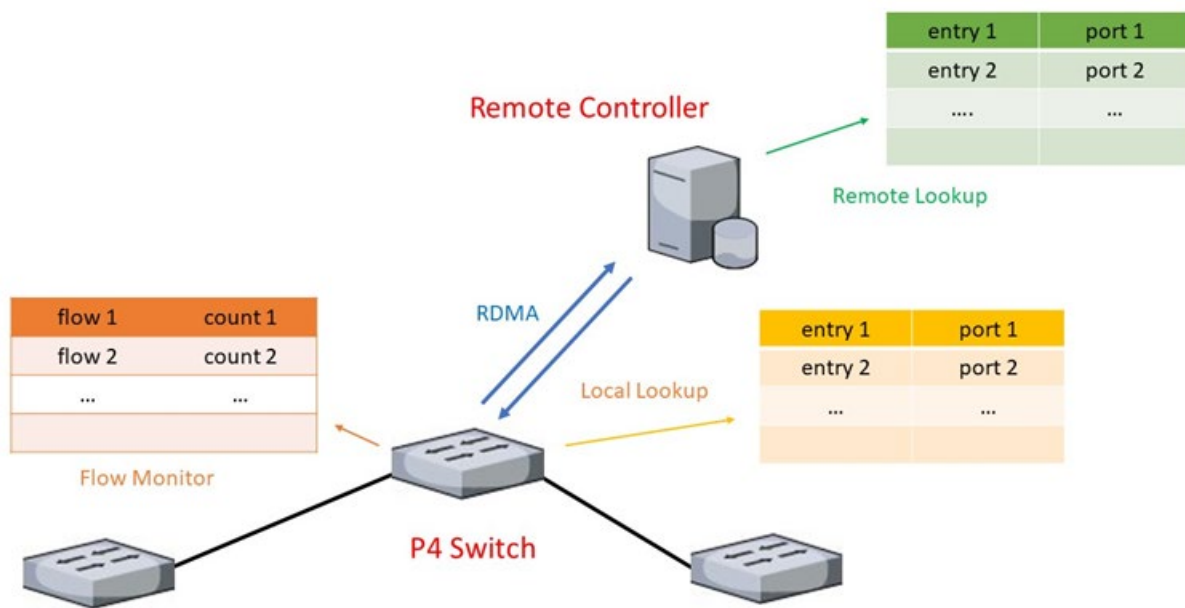
上圖中，其中第二個參數，h1 選 crc16；h2 選 crc32，只要 h1 和 h2 選不同種 hash function 即可。而第四個參數，則是傳了 flow 的 5 tuple 進去。

## 9. 系統模組:網路環境

本研究的網路系統主要有四大元素，分別為:

1. Remote Controller
2. Switch Flow Monitor
3. Switch Local Lookup
4. Remote Lookup

首先是第一點，遠端控制器傳送控制指令，控制 P4 交換機行為。最重要的一點是，它儲存了整個 LAN 的所有路由資訊。第二點是交換機中的 flow monitor，它會統計流經交換機的 flow 封包個數，並且會在固定條件下觸發 routing table update。詳細演算過程會在系統架構:交換機的部分做說明。第三點是 switch local lookup。當一個封包進到交換機時，會先進行 local lookup，檢查自身路由表有沒有相對應的路由資訊，若有，就直接進行轉送；若沒有，就要進行第四點的 remote lookup，向遠端控制器詢問路由資訊。下圖十九為統整圖。



圖十九：網路系統模組統整圖

## 10. 系統模組: RDMA

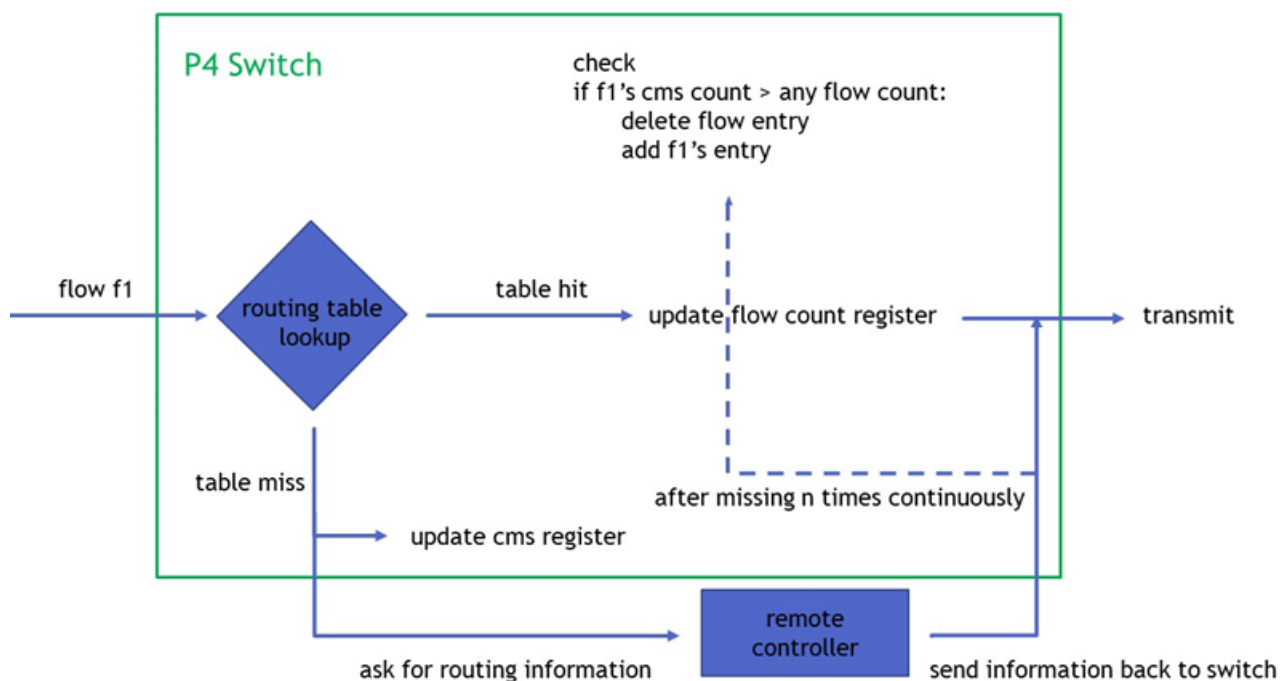
由於我們需要在模擬 RDMA 傳輸的行為，因此我們需要建立兩台虛擬機，並利用 rdma-core 運行 Soft-RoCE (Software RDMA over Converged Ethernet)，達成利用軟體的方式模擬 RDMA 在乙太介面當中的傳輸。為此，我們需要完成以下步驟：

1. 安裝兩台作業系統為 Ubuntu 20.04 LTS 的虛擬機
2. 在虛擬機中安裝 libibverbs 以及 librdmacm 這兩個 library
3. 下載 linux-rdma 當中的 rdma-core，並安裝相關套件
4. 輸入 rdma link add 讓虛擬 rdma 網卡開始運行
5. 輸入 rxe\_cfg status 確認虛擬 rdma 網卡是否正常運行

## 11. 系統架構: P4 交換機

接下來介紹 P4 交換機內部運作的詳細流程。當一個封包進到交換機內，交換機會先檢查自身的 routing table，也就是進行 local lookup。當路由資訊有在自身路由表內，即為 table hit，可以直接找到符合的 port 進行轉送；反之，即為 table miss。當發生 table hit，在進行轉送前，會先 update flow count register，即為紀錄這個 flow 的封包發生 hit 的次數，將對應到的 register 內容值加一。若發生 table miss，則會 update count-min sketch register，update 的流程即上述 count-min sketch 部分。接著交換機會將該封包轉傳給遠端控制器，遠端控制器進行 remote lookup 後，將對應 port 加進封包 header 後，回傳給交換機。交換機便能再將封包依 header 中的 port 傳出。如此一來對交換機來說，便能在不丟失任何封包的情況下，完成自身無路由資

訊的封包轉送。而當連續發生一定次數的 miss 時，便會進行 routing table update，以下簡稱 table update。交換機會檢查當下封包的 count-min sketch 結果值(圖十七)，和 flow count register 中的值以線性方式做比較。若該值大於 flow count register 中的任何一個值，便將 register 對應的 forwarding rule 從 routing table 刪除，並將該封包的 forwarding rule 加入 routing table。以此方式將較大的 major flow 之路由規則，存進交換機之路由表，並將較小、較不會用到的規則刪除。下圖二十為流程圖。在複雜網路實驗設定中，我們會令 flow 的數量遠超過 P4 交換機 routing table 能存的量，也超過 register 所能運用的記憶體容量。因此將所有 flow 的 flow count 全部存進 register 是不可行，也是沒有效率的。此時 count-min sketch 的空間優勢就能明顯發揮出來了。



圖二十: P4 Switch 演算流程圖

## 12. 系統架構: RDMA

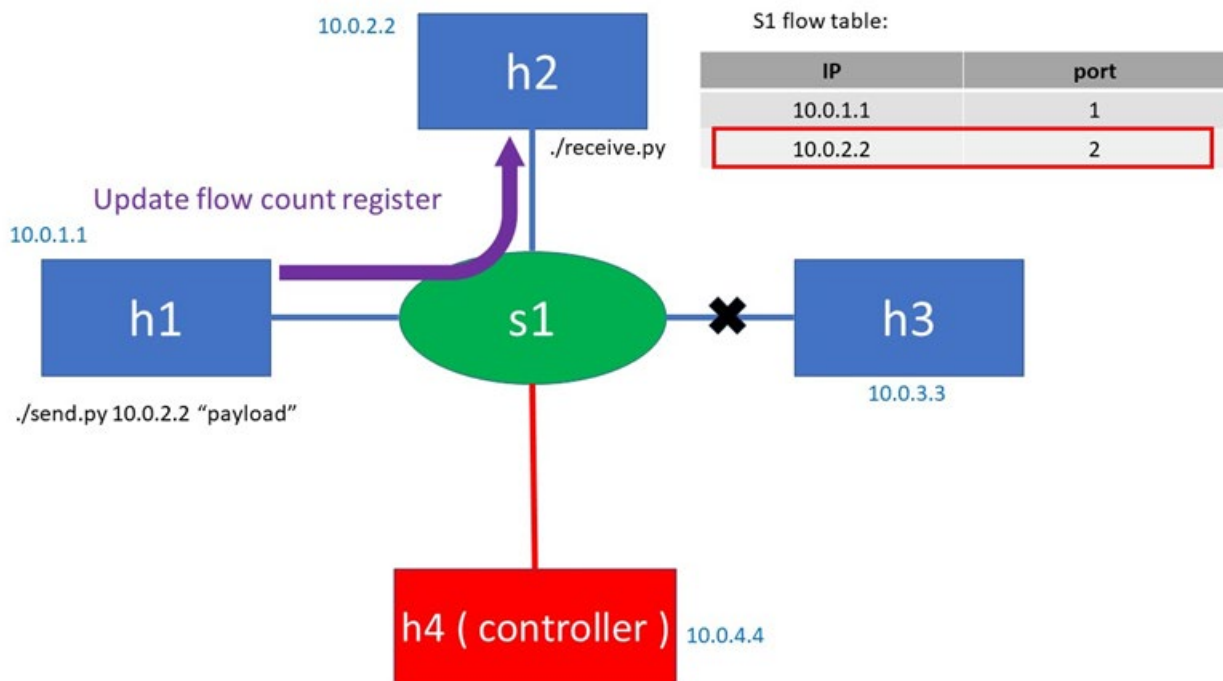
由於 RDMA 連線時需要 IP 位置以及 Port，因此兩台虛擬機的 Network Adapter 使用 Bridged，讓兩台虛擬機皆得以取得 IP 位置。實驗過程皆是一邊作為 Server，另一邊作為 Client，兩端利用 Post 進行傳輸過程的控制以及狀態的確認。

### 第三章 研究方法及步驟

#### 1. 簡單網路

首先，我們用 Bmv2 結合 mininet 簡單模擬上述的架構中交換機的行為。第一步先建立一個簡易拓撲網路(圖二十一)。這個架構由一個 P4 交換機 s1 連結三個 host 及一個遠端 controller 組成。host h1 的 IP 為 10.0.1.1、h2 為 10.0.2.2、h3 為 10.0.3.3，而 controller 為 10.0.4.4。s1 的路由表中目前有 h1、h2 的 forwarding rule，但沒有 h3 的。代表目前封包是無法傳到 h3 的。而 h4 是 controller，不屬於 host，故路由表中沒有其路徑。而 controller 存有所有 host 的 forwarding rule。

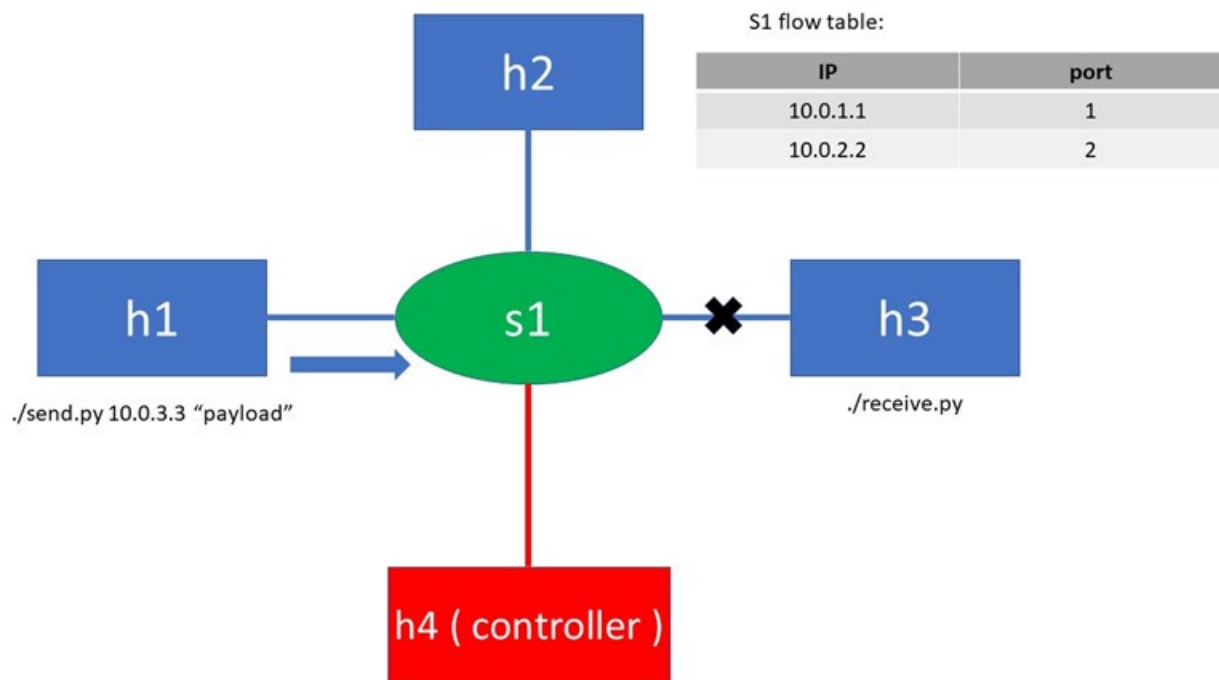
第一步，由 h1 發送一個封包給 h2。交換機 s1 收到這個封包後，檢查自身的路由表，發現封包目的地 h2 是有在路由表中的，其對應的 port 是 2，代表將封包由 port 2 送出，就能成功將封包送抵 h2。因此，交換機將其所對應的 flow count register 值加一，並將封包由 port 2 送出。路由表中有路由資訊的此種狀態，我們稱之為 table hit。



圖二十一: h1 送封包給 h2

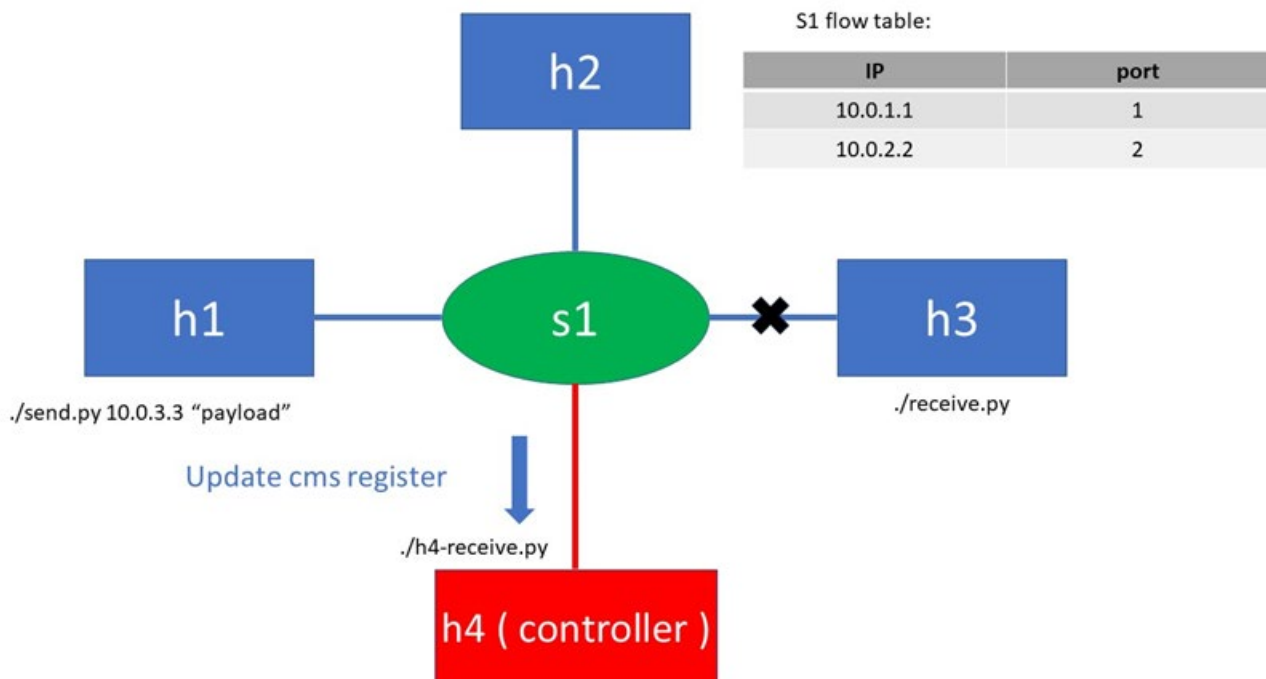
接著，h1 嘗試發送一個封包給 h3。s1 收到此封包後，一樣會先檢查自身的路由表。這次，它發現它沒有 h3 的 forwarding rule，代表它不知道封包要由哪個 port 送出(圖二十二)。路由表中沒有路由資訊的此種狀態，我們稱之為 table miss。





圖二十二: h1 送封包給 h3，發生 table miss

這時，s1 就會向 controller 進行詢問。它將封包發送給 h4，並且要更新 count-min sketch register (圖二十三)。



圖二十三: s1 將封包轉傳給 controller

Controller 收到由 s1 傳來的封包後，依照其 destination IP 來查詢路由資訊總表，得到 forwarding rule 是 port 3，代表此 IP 的封包必須從 s1 的 port 3 送出。controller 即修改此封包欄位，將這個訊息包進封包中（圖二十四），反向傳送回去給 s1。s1 收到封包後，便能依此訊息，將封包成功發送給 h3。



圖二十四: controller 修改封包欄位，將 port 資訊裝入

每當 s1 發生連續一定次數的 miss 時，會進行 routing table update。它會檢查當下的 flow，以下簡稱 f1，執行 count-min sketch 得到的值，以下簡稱 x1，是否有大於 flow count register 中的值。比較的方式為由前到後線性比較，要是一發現 flow count register 中有值是小於 x1 的，就將該 register index 對應的 routing table entry 刪除，並將 f1 的 forwarding rule 加進 s1 的 routing table。而 register 也要進行相同的操作以維持 table 和 register 之對應。以上述例子來說，令 s1 發生五次 miss 就要進行 update。若 h1 連續傳送五個封包給 h3，那就會連續發生五次 miss。並在第五次 miss 時觸發 table update。它會依序檢查 flow count register 中的值，假設此時為 7, 2，對應的 table entry 為 10.0.1.1, 10.0.2.2。再假設 count-min sketch 得到的值是 4，第一步比較  $7 > 4$ ，因此無動作；但接著比較  $2 < 4$ ，觸發了交換 (swap) 機制。s1 會將 10.0.2.2 這個 entry 刪除，並將 10.0.3.3 這個 entry 加入 table 中，flow count register 也會變成 7,0，如下圖二十五。

IP	port	count
10.0.1.1	1	7
10.0.2.2	2	2

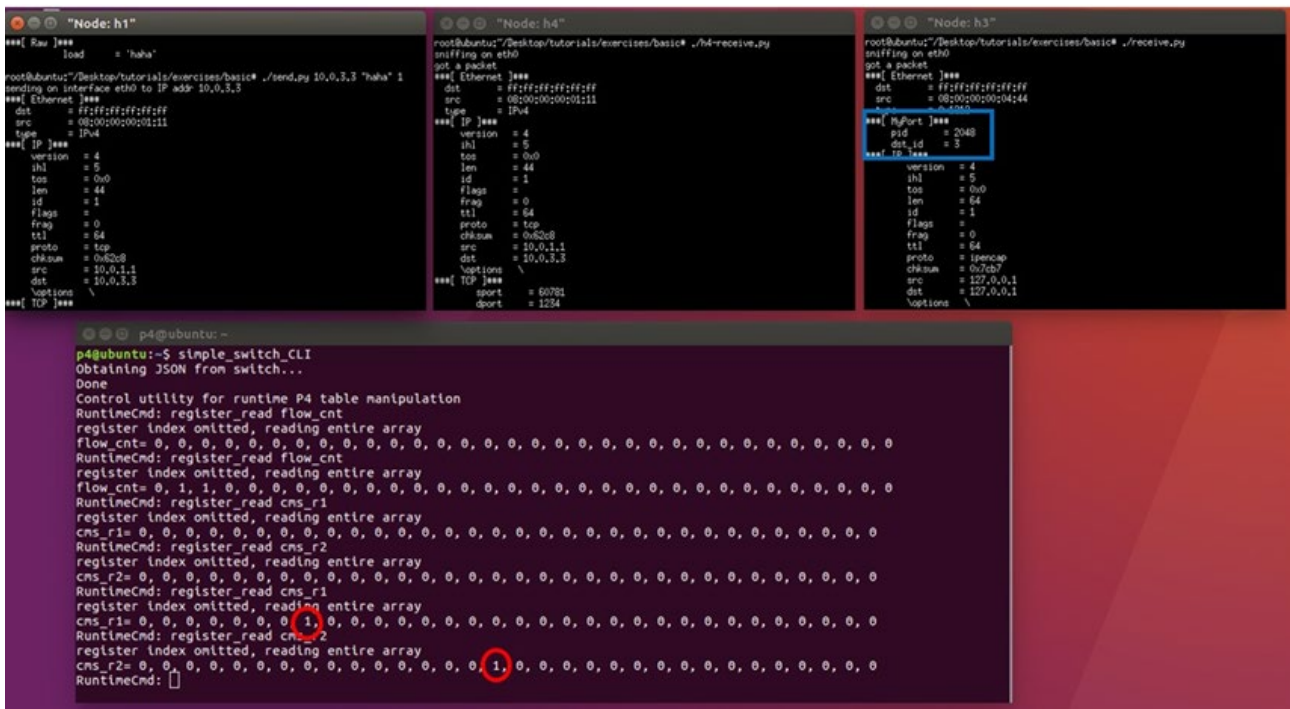
IP	port	count
10.0.1.1	1	7
10.0.3.3	3	0

圖二十五: routing table update

最後是在 Linux 系統上實際操作的畫面圖。首先是圖二十六，h1 送了一個 TCP 封包給 h2。這時交換機有到 h2 的路由資訊，可以直接轉送，並用 P4 runtime API 檢查 flow count register。

圖二十六：實際操作畫面圖一

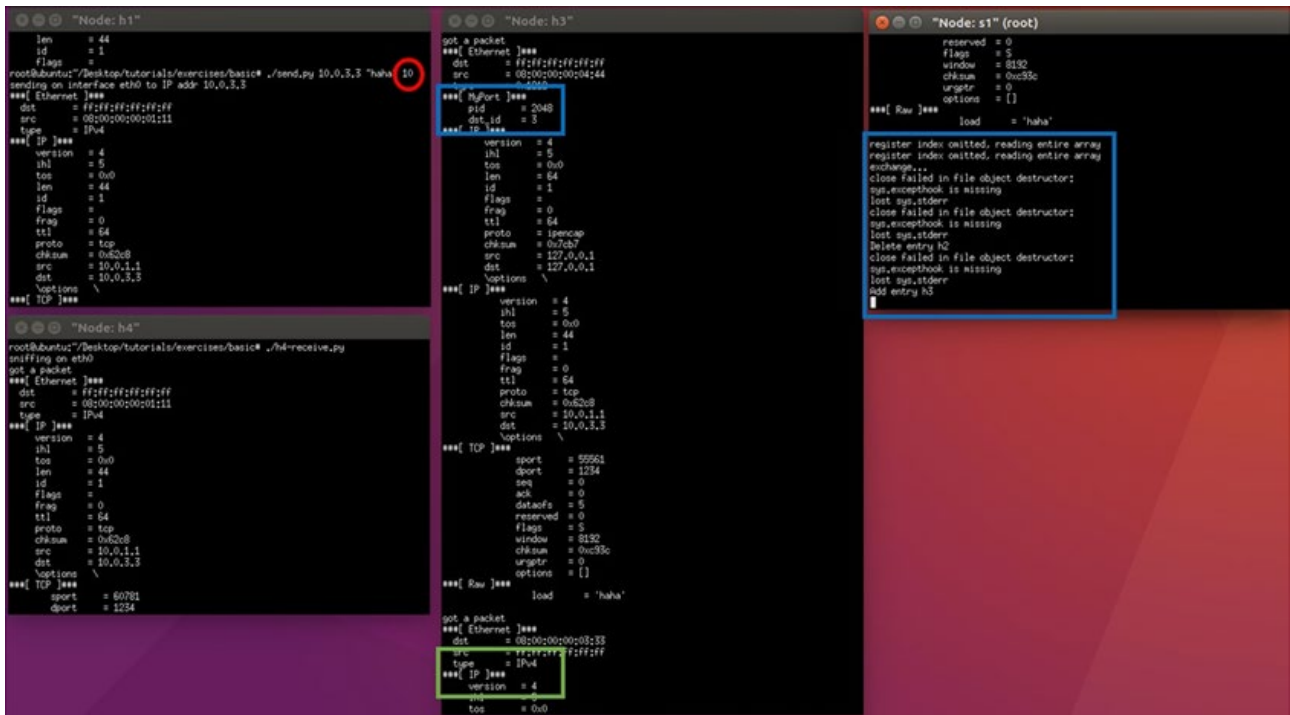
接著是圖二十七，h1 發送一個封包給 h3。此時交換機沒有到 h3 的路由資訊，需要轉傳給扮演 controller 的 h4。我們用一樣的方式檢查兩個 count-min sketch register r1 和 r2。



圖二十七 實際操作畫面圖二

上圖中，上半部左邊為 h1、中間為 h4、右邊為 h3。可以看到 h1 發送一個封包給 h3，並透過 h4 幫忙進行轉傳，而 h3 收到的封包也會多出一個 MyPort 欄位(藍色框框)。下方的 cms\_r1 和 cms\_r2 都有一個值從 0 變為 1(紅色圈圈)。

最後是圖二十八，h1 連續送 10 個封包給 h3。交換機仍然沒有到 h3 的路由資訊，因此前五個封包會連續發生 5 次 miss。這時就會執行 table update，將 h3 的路由資訊安裝進路由表。



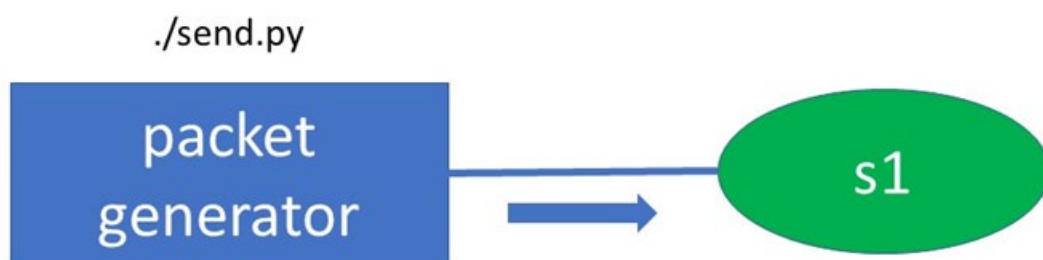
圖二十八 實際操作畫面圖三

上圖中，h1 連續傳送 10 個封包給 h3。中間是 h3，前面收到的封包有 MyPort 欄位(中間藍色框框)，而後面收到的沒有(青綠色框框)。右邊是交換機 s1，可以看到 s1 發生了 table update 訊息(右邊藍色框框)，h3 路由資訊被裝入而 h2 路由資訊被刪除。

## 2. 複雜網路

接著，一樣用 BMv2 和 mininet 來模擬一個複雜的網路架構。假設有一個 LAN，host IP 為 10.0.0.1 ~ 10.0.0.254、10.0.1.1 ~ 10.0.1.254、10.0.2.1 ~ 10.0.2.254、10.0.3.1 ~ 10.0.3.254，總共 1016 個 host。一台 P4 交換機總共要處理 1000 個 flow，每單位時間只處理一個 flow 中的一個封包。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。代表大約有 2%的 flow 為 major flow，因此隨機令 20 個 flow 為 major flow。Major flow 共有 500 ~ 1000 個封包，持續時間在 500 ~ 1000 單位時間；普通 flow 則是 1 ~ 50 個封包，持續時間在 1 ~ 50 單位時間。傳送的封包皆為 UDP，因此不會有 ACK。總體來說，這台交換機要處理的封包數量大約為  $20 \times 750 + 980 \times 25.5 = 39990$ ，大概 40000 個封包。假設單位時間為 1 秒，也就是一秒只傳送一個封包 (避免碰撞)，那全部封包傳送完就大約要花 40000 秒。

由於電腦效能限制問題，我們沒辦法用 mininet 實作出數千個 host 的 LAN。但我們在意的是交換機的 routing table hit 和 miss 的次數，因此可以假設封包不會因其他外界因素而丟失，只要交換機有選對 port 將封包發出，就能預設封包有傳達到 host。也就是說，我們只需要將眼光放在交換機的行為，而不用在乎封包在網路中的傳送。為此，我們用一個 packet generator 接上一台 P4 交換機。對於 packet generator，我們可以在上面運行 python，並用 scapy 組成封包並傳送。而用 scapy 的好處就是它可以幫助我們自由的生成任何形式、內容的封包，source IP、destination IP 都由我們自由決定。我們依此可以模擬各種不同的 flow、不同的封包傳入交換機，如下圖二十九所示。



圖二十九：複雜網路架構

## 3. RDMA 實驗流程

在實際網路中，SDN 交換機和遠端控制器溝通需要花費一定時間，使路由資訊和網路遙測的即時性下降。因此我們在上述複雜網路的基礎下，用 RDMA 改善 SDN 交換機和遠端控



制器溝通時間，確保資訊的即時性。本實驗首先需要將 RDMA 傳輸預先建立，如此才能夠及時送出查找要求，接下來需要將預查找的 IP 放入 RDMA 傳輸的 post 當中，讓遠端控制器得以得知本地交換機所欲查找的路由。而在這個整個查找架構運行當中，由於 RDMA 的驗證機制較為嚴謹，因此我們需要確保在能夠持續的進行查找不會斷線，最後則是要確保在控制器服務多個本地交換機時，能夠將路由資訊放在各個交換機特定的記憶體位置，避免本地交換機在讀取到路由時讀取到屬於其他交換機的路由資訊。

最後為了要驗證利用 RDMA 相較傳統傳輸方式，是否確實有降低本地交換機在查找路由時 CPU 的佔用以及查找的延遲，我們需要比較在本地交換機不同 CPU 負載下以及遠端控制器在服務不同數目的本地交換機時其傳輸的延遲，並比較其結果。

#### 4. RDMA 實驗步驟

為了要達成此架構之 RDMA 傳輸功能，需要以下五個步驟：

1. 須先建立連線，避免查找時建立連線的時間影響即時性。
2. 以本地 SDN 交換機向遠端控制器查找路由作為觸發事件，送出包含有待查找 IP 位置的 post，並將路由規則由 RDMA read 取回。
3. 持續的進行本地 SDN 交換機與遠端控制器的傳輸。
4. 各個本地 SDN 交換機之間，能夠存取的遠端控制器記憶體不可相同，如此遠端控制器才能夠決定其所控制的各個 SDN 交換機路由規則的存放位置。
5. 以交換機數量和交換機之 CPU 負載量為變因，傳輸並比較結果。

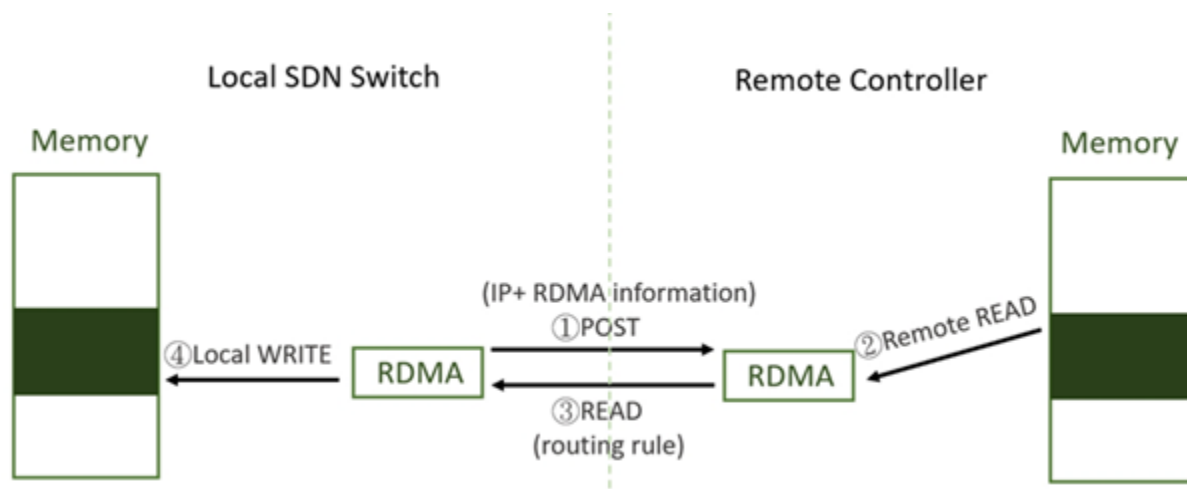
接下來我們詳細分析以上五步驟。首先是第一點的預先建立連線的部分。在運行此架構時需要先建立連線，避免在查找時才開始建立連線，影響查找路由的即時性。RDMA 傳輸基本上有分為：

- I. 建立 event channel，即開啟連線。
- II. 本地 SDN 交換機和遠端控制器皆先分配記憶體，並將其註冊至 Memory region。
- III. 在 Post 當中填入傳輸資訊、記憶體資訊以及其 Memory region 的金鑰。
- IV. 送出包含傳輸資訊的 post 並驗證內容之後，便會開始進行傳輸。
- V. 確認傳輸狀態並斷開連線。

為了達成查找時能夠直接進入傳輸的階段目標，遠端控制器在接收到本地 SDN 交換機連線要求後，兩端都會先完成上述 I 至 III 的部分，以便查找時能夠直接送出包含有讀取路由規則的要求的 Post，直接進行路由的讀取。

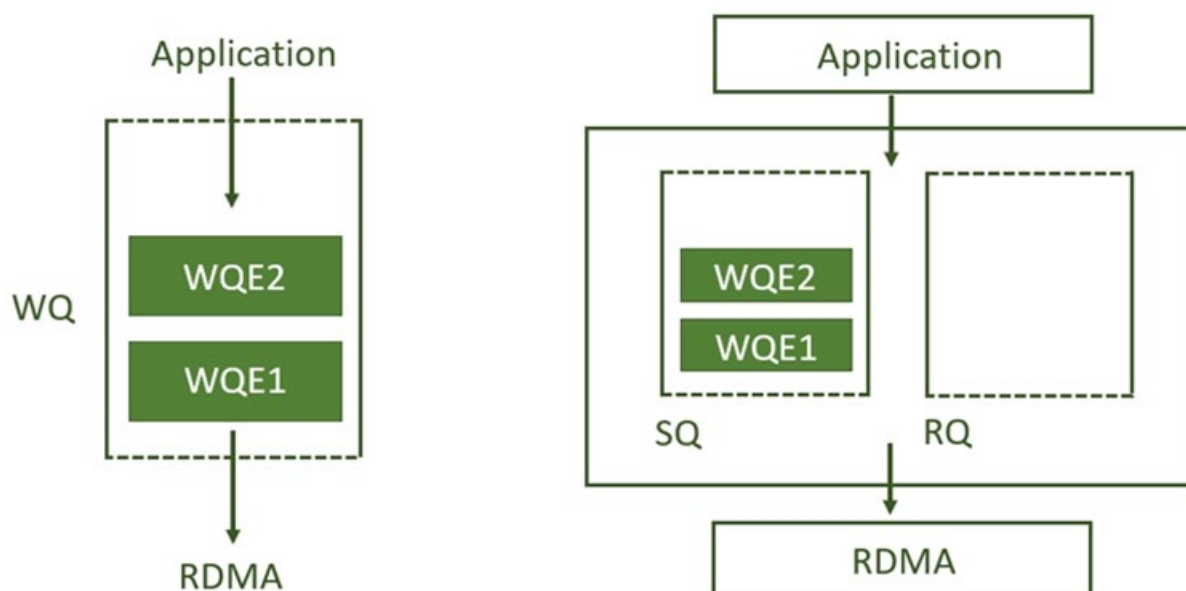
接著是第二點的查找路由機制。Post 在 RDMA 連線當中，扮演讓雙方能夠驗證彼此傳輸內容的資訊。由於連線以及交換資訊部分已經在先前完成，遠端控制器目前是處於等待本地 SDN 交換機送出之 post 的狀態。本地 SDN 交換機在需要向遠端控制器查找路由時，會欲查

找 IP 位置寫入 post，送出含有欲查找的 IP 的 RDMA read request 的 post，遠端控制器會在驗證 post 內容當中如金鑰，連線狀態等資訊無誤後，開放本地 SDN 交換機讀取其對應記憶體。



圖三十：查找路由步驟

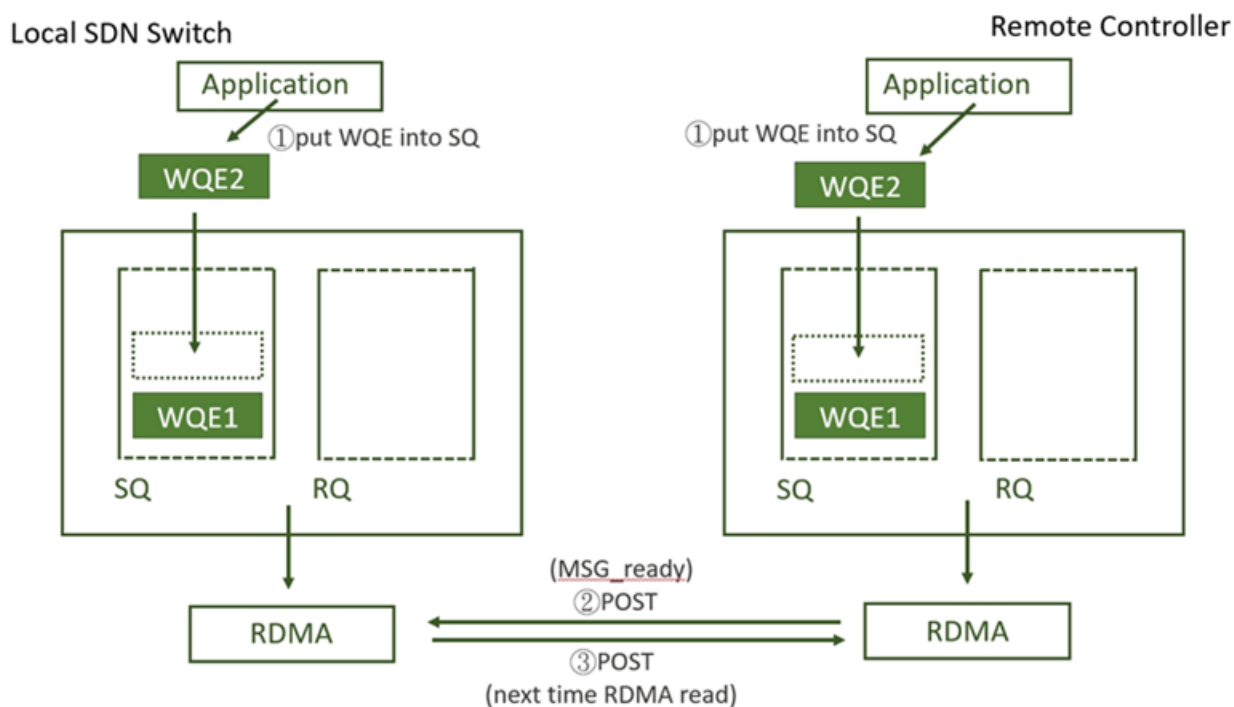
第三點是要持續進行本地 SDN 交換機與遠端控制器的傳輸。WQ (work queue) 為作為應用程式與 RDMA 媒介的一種 queue，儲存應用程式下達給予 RDMA 之工作，而工作的基本單位 WQE (work queue element)。其中 WQ 當中又分為 SQ (send queue) 和 RQ (receive queue)，用以儲存用於 send 與 receive 的 WQE。



圖三十一：RDMA 指派工作架構

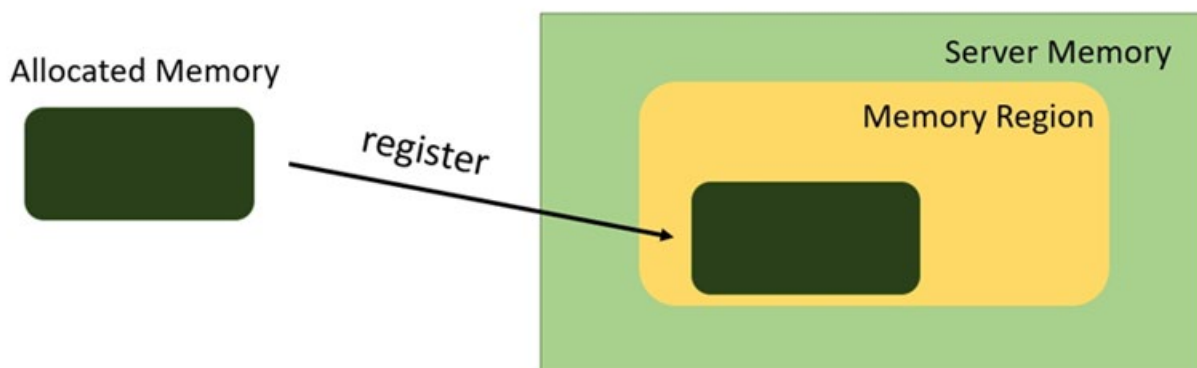
RDMA 傳輸過程中會以 work queue 確認本地狀態，以及以 post 作為傳輸兩端溝通彼此

傳輸狀態的媒介。為了達成本地 SDN 交換機能夠持續地進行 RDMA read，便須要兩方都在一次 RDMA 傳輸完成後，在各自的 send queue 放入 WQE，並且遠端控制器需要回傳送包含 MSG\_READY 的 post 使本地 SDN 交換機得知能夠繼續進行 RDMA read 傳輸，而需要注意的是雖然本地 SDN 交換機為接收資料，但在 RDMA read 當中，兩方都是將 WQE 中放入 send queue，只有在 RDMA receive 時，接收端才會在 SQ 放入 WQE。



圖三十二: 連續傳輸實作

然後是第四點，各個本地 SDN 交換機存取之遠端控制器記憶體須為唯一。RDMA 的 server 端(遠端控制器)收到來自 SDN 交換機的建立連線要求後，會在分配記憶體位置後，將記憶體位置註冊至 RDMA 的 memory region，此時 memory region 便會鎖定，避免其他 SDN 交換機在建立連線時占用到其他已經連線之 SDN 交換機存取的記憶體位置。



圖三十三: Memory Region 鎖定



最後是第五點。為了驗證 RDMA 與傳統傳輸方式的差異，我們以遠端控制器服務不同數量的本地 SDN 交換機以及本地 SDN 交換機在不同 CPU 負載下作為變因，分別測量數據並且比較結果。以遠端控制器服務不同數量的本地 SDN 交換機作為變因測量時，我們以 multithread 的方式在 client 端的虛擬機同時開啟多個模擬的本地 SDN 交換機，最後測量並記錄傳輸的延遲。以本地 SDN 交換機在不同 CPU 負載下作為變因測量時，我們則是以 python 檔撰寫 Dead loop，並在 client 端執行使本地 SDN 交換機能夠有不同的背景 CPU 負載，最後測量並記錄傳輸的延遲。

## 第四章 結果分析與探討

### 1. 複雜網路

上述實驗中有 1016 個 host，令 s1 的 routing table 空間只能存下 32 個 entries，也就是當 table entry 數量達到 32 時，我們便不會讓它再增加。如此一來，我們需要的 flow count register size 是 32。而我們令兩個 count-min sketch register r1, r2 的 size 皆為 32，連續發生 5 次 miss 時便進行 table update。下表為五次實驗的結果。

表二：五次交換機 table hit 比率實驗結果

	P4 Switch using cache			Normal P4 Switch	
	Total packet	Table entry hit	Hit rate	Table entry hit	Hit rate
1 <sup>st</sup>	38966	34501	88.54%	1646	4.22%
2 <sup>nd</sup>	39196	34703	88.54%	2201	5.62%
3 <sup>rd</sup>	39103	34627	88.55%	1501	3.84%
4 <sup>th</sup>	40288	35826	88.92%	1701	4.22%
5 <sup>th</sup>	39732	35272	88.77%	3363	8.46%

上表中，P4 Switch using cache 是有上述路由安裝演算法的交換機；而 Normal P4 Switch 是從 1016 個 flow table entry 中隨機挑選 32 個 entry 裝入交換機。從表中可以發現，有路由安裝演算法的交換機可以達到將近 90% 的 hit rate，而沒有的只有不到 10%。

接著，我們挑出其中一次，來觀察 routing table 裡的 entry 是否有存到所有 major flow。下表是實驗結果。

表三：交換機 routing table entry 內容、count 值和 major flow 之封包數比較

entry	major flow	packets	table entry	count
1	10.0.1.119	982	10.0.1.119	1008
2	10.0.2.80	975	10.0.2.4	43
3	10.0.0.118	946	10.0.1.16	36
4	10.0.3.227	941	10.0.3.227	936
5	10.0.2.171	940	10.0.2.171	935
6	10.0.1.154	934	10.0.1.154	932
7	10.0.1.157	927	10.0.1.157	970
8	10.0.0.48	895	10.0.0.48	898
9	10.0.0.121	867	10.0.0.121	862
10	10.0.2.103	864	10.0.2.103	859
11	10.0.0.218	821	10.0.1.141	34
12	10.0.3.46	810	10.0.3.46	805
13	10.0.1.246	754	10.0.1.6	33
14	10.0.0.217	681	10.0.0.217	676
15	10.0.0.104	606	10.0.0.104	601
16	10.0.2.156	602	10.0.2.156	597
17	10.0.2.112	601	10.0.2.112	596
18	10.0.3.24	543	10.0.2.56	31
19	10.0.2.196	506	10.0.2.196	501
20	10.0.0.31	500	10.0.0.31	545
21	x	x	10.0.2.148	30
22			10.0.2.249	28
23			10.0.0.243	26
24			10.0.0.12	23
25			10.0.1.148	20
26			10.0.3.37	20
27			10.0.0.167	16
28			10.0.1.169	12
29			10.0.3.41	7
30			10.0.0.214	7
31			10.0.1.235	4
32			10.0.1.213	3

從上表中可以看到，20 個 major flow 中，有 15 個被存進了 routing table 中。證明此方法是相當有效率的。

因此，我們可以得出結論：在 1016 個 hosts 的 LAN 中，我們只需要用到 P4 交換機中 size 32 的一個 flow count register、兩個 size 32 的 count-min sketch register，總共 96 的 register 空間，就能大幅上升 table hit 的機率。forwarding rules 數量和 register 消耗空間比是 1016 : 96，大約是 32: 3。為了印證此結論，我們再進行一個實驗。根據 mininet 官方網頁所述，mininet 可以在單一個核心中成功開啟 4096 個 hosts [18]。我們一樣不需要實際開啟 4096 個 hosts，只需要在交換機 table 中加入 4096 個 entries 即可。而這可以用 simple\_switch\_CLI 加上腳本來自動完成。依據比例，我們給定 flow count register 大小為 128，兩個 count-min sketch register 大小皆 128。為了模擬更大、更複雜的網路系統，我們將 flow 的數量放大 10 倍，從 1000 個變成 10000 個 flow，major flow 一樣依比例變 200 個，下表為實驗結果。

表四：大規模系統之實驗結果

	P4 Switch using cache			Normal P4 Switch	
	Total packet	Table entry hit	Hit rate	Table entry hit	Hit rate
1 <sup>st</sup>	397207	352365	88.71%	16301	4.10%
2 <sup>nd</sup>	405482	360594	88.93%	19436	4.79%
3 <sup>rd</sup>	402394	357695	88.89%	25074	6.23%
4 <sup>th</sup>	402164	356924	88.75%	23018	5.72%
5 <sup>th</sup>	401649	356784	88.83%	14528	3.62%

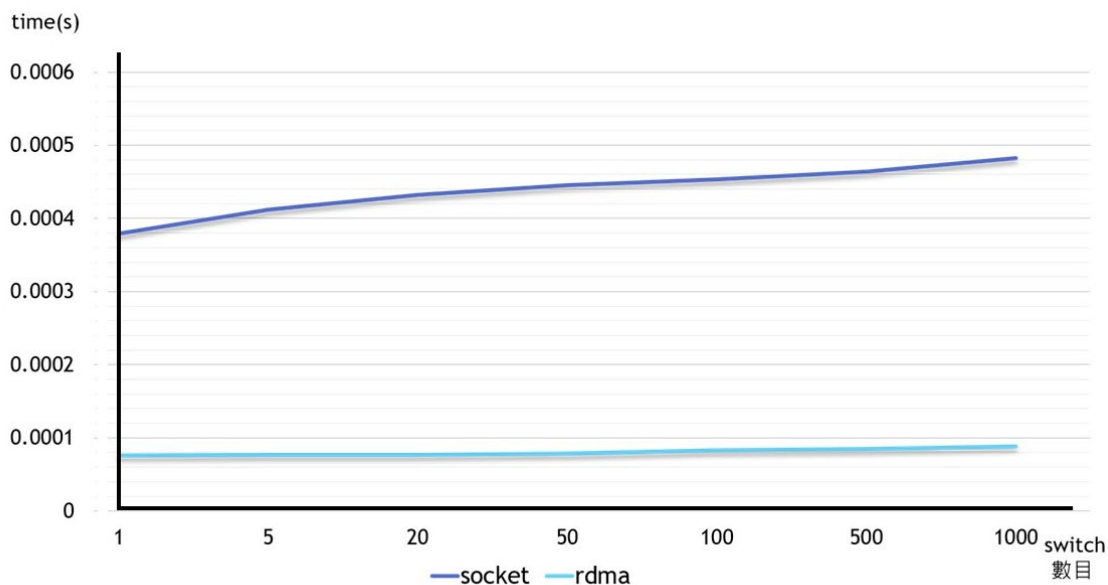
從上表中可以得到和表一差不多的結果。代表此方法在複雜網路結構中依然能保持穩定且良好的表現。

## 2. RDMA

對於 RDMA，本研究用下列三點方式進行實驗與比較。

### a. 低 CPU 負載下不同本地 SDN 交換機數目與傳輸時間比較:

本實驗是要模擬遠端控制器在服務處於低 CPU 負載之不同數量的本地 SDN 交換機，使用 RDMA 傳輸方式以及傳統傳輸方式的傳輸時間對比。由於本地 SDN 交換機在日常網路環境中皆會有一定程度的工作負載，因此在模擬過程中我們是將本地 SDN 交換機的 CPU 負載設為 30%，以此負載量來測試遠端控制器與本地交換機交換路由的傳輸時間。

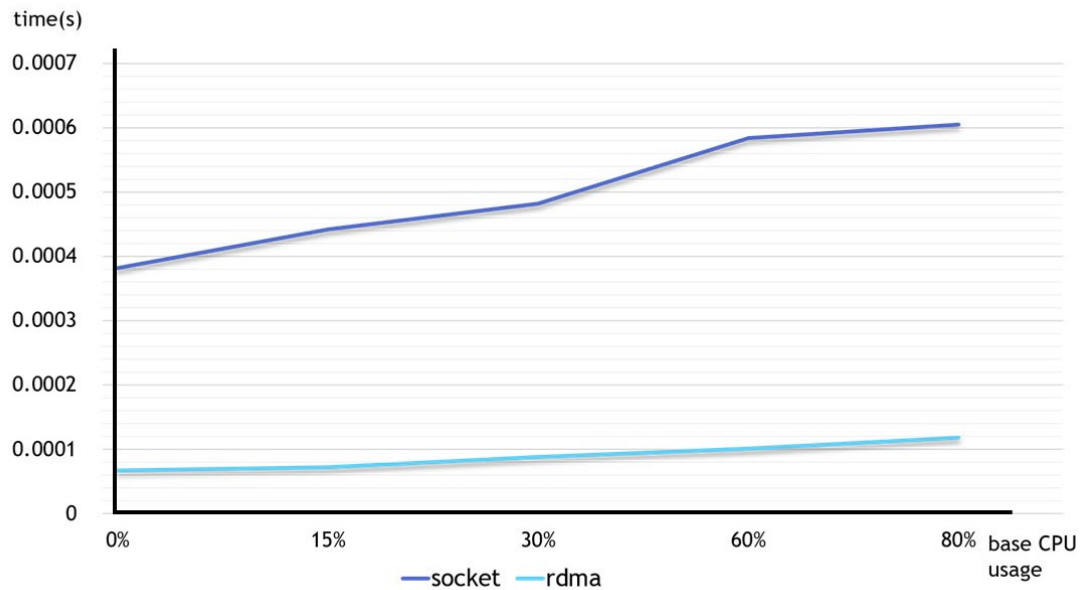


圖三十四: 相同低 CPU 負載下不同 local SDN switch 數目與傳輸時間比較圖

觀察上圖表可以看出，在遠端控制器服務相同數量的本地 SDN 交換器的情況下，RDMA 的傳輸方式相比傳統傳輸方式快了許多；而隨著遠端控制器服務的本地 SDN 交換器數量增長也可以看出，RDMA 傳輸方式時間的增長幅度也小於傳統傳輸方式時間的增長幅度。

### b. 相同本地 SDN 交換機連線數目下不同 CPU 負載程度與傳輸時間比較:

本實驗為模擬遠端控制器在服務不同 CPU 附載量之相同數量之本地 SDN 交換機，使用 RDMA 傳輸方式以及傳統傳輸方式的傳輸時間對比。由於 RDMA 傳輸方式占用之 CPU 較小，因此我們在本實驗模擬遠端控制器在服務 1000 個本地 SDN 交換機，並測量在不同 CPU 負載下遠端控制器與本地交換機交換路由的傳輸時間。

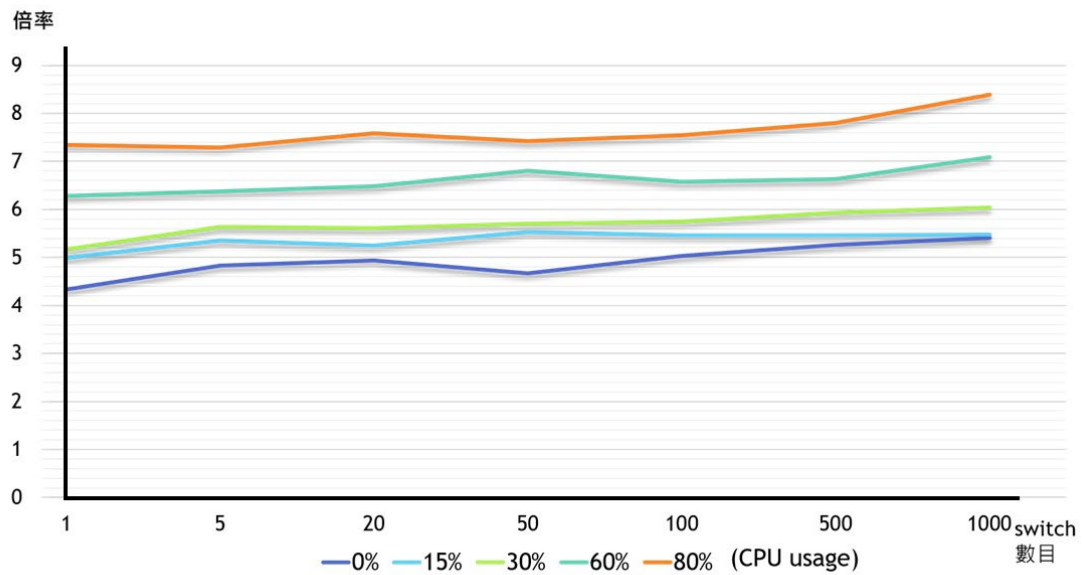


圖三十五：相同 local SDN switch 連線數目下 CPU 負載程度與傳輸時間比較圖

觀察上圖表可以看出，在遠端控制器服務相同數量的本地 SDN 交換機下，RDMA 傳輸方式的傳輸時間依然遠低於傳統傳輸方式。而隨著 CPU 附載量的上升，傳統傳輸方式的傳輸時間有很大程度的增長，RDMA 傳輸方式的傳輸時間之增長速度則是遠小於傳統傳輸方式之增長速度。

c. 不同本地 SDN 交換機數目下兩傳輸方式時間之倍率比較：

本實驗為比較遠端控制器在服務不同 CPU 以及不同數量之本地 SDN 交換機，使用 RDM A 傳輸方式以及傳統傳輸方式的傳輸時間對比。



圖三十六: 不同 local SDN switch 數目下兩傳輸方式時間之倍率比較圖

觀察上圖表可以看出，無論是隨著遠端控制器服務的本地 SDN 交換機增長或是隨著本地 SDN 交換機 CPU 負載增長的情況下，RDMA 傳輸方式都能夠維持著更高的傳輸效率，且在本地 SDN 交換機有 CPU 負載(也就是工作負載)時，RDMA 傳輸方式的優勢也更加明顯。

## 第五章 結論與未來展望

本研究使用 P4 語言成功實作出路由安裝演算法，在理論上不丟失任何封包的條件下，將較常用到的路由規則安裝進 P4 交換機。利用 P4 中的 register 儲存 flow count 和 count-min sketch 來協助安裝演算法進行估計與路由規則交換。然而，我們只關注在交換機路由表是否有需要的路由訊息，也就是文章上述的 table hit 次數，路由安裝演算法的目標在致力減少 table miss 次數、減少交換機和遠端控制器的溝通次數，而忽略了大規模 register 操作可能會造成的延遲。此外，在路由表中安裝、刪除路由規則也需要少量時間，若不是使用 Mininet，而是接上大量實際 hosts 端，並不能保證不會對封包的接收品質造成影響。關於路由安裝演算法內部運作機制，在搜尋 flow count 值時，我們使用索引值由小到大的線性搜尋，是因為 P4 Runtime API 中的 table\_add 方法是採用類似 push back 方式，將新的路由規則加在路由表最後方(索引值最大)。而剛加進來的 count 值是 0，為了避免剛剛加入的規則立刻又被刪除，我們將它放在後方就可以避免被檢索到的機率。這種方式的優點在簡單易實作、不耗費太多多餘的運算時間或空間。但缺點是有可能不夠精準，沒有找到最應該被刪除的路由規則。可能還有更適合的搜尋、比較方法，可以在效能和準確度中找到更好的平衡點。實驗中，路由規則和 register 空間比例用的是 32:3，也可能有更適合的比例能讓效能更好。

研究中也結合了 RDMA 的傳輸方式進行路由查找以及轉送路由，我們透過 Linux rdma-core 運行 Soft-RoCE (Software RDMA over Converged Ethernet)，達成利用軟體的方式模擬 RDMA 在乙太介面當中的傳輸，藉此來模擬整個複雜網路中本地 SDN 交換機與遠端控制器的溝通行為，解決傳統傳輸方式效率低落以及占用資源的問題。

實驗以 CPU 負載以及本地 SDN 交換機數目為控制變因，來模擬遠端控制器和本地 SDN 交換機的路由查找行為並測量傳輸時間，實驗結果顯示無論是較高的 CPU 負載或是較多的本地 SDN 交換機的情況下，RDMA 皆能以更快的速度完成查找及傳輸，大大提升傳輸的效率以及減少對於本地 SDN 交換機的資源占用。

我們期望運用 RDMA 傳輸方式，透過遠端控制器與本地 SDN 交換機的合作進行 count-min sketch 的網路遙測方式，來解決過去為求精準度，遠端控制器頻繁對本地 SDN 交換機採樣所造成的網路服務下降的問題，亦能解決僅在本地交換機進行 sketch 所造成準確性不佳之問題，為傳統的網路遙測方式提供全新的解決方案。



## 參考文獻

- [1] U. Wickramasinghe, A. Lumsdaine, S. Ekanayake and M. Swamy, "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](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] BEHAVIORAL MODEL (bmv2) [Online] Available: <https://github.com/p4lang/behavioral-model>
- [12] p4language [Online] Available: <https://github.com/p4lang>
- [13] Cormode, G., & Muthukrishnan, M. Count-Min Sketch. 2009. URL: <http://dimacs.rutgers.edu/~graham/pubs/papers/cmencyc.pdf>
- [14] Cormode, G., & Muthukrishnan, S. (2005). An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1), 58-75
- [15] Chelsio Communication. RoCE (RDMA over Converged Ethernet) : FAQ. 2012. URL: <https://www.chelsio.com/roce/>
- [16] Guo, C., Wu, H., Deng, Z., Soni, G., Ye, J., Padhye, J., & Lipshteyn, M. (2016, August). RDMA over commodity ethernet at scale. In *Proceedings of the 2016 ACM SIGCOMM Conference* (pp. 202-215).
- [17] Budi, M. (2019). Programming networks with P4. VMware Research Blog.
- [18] Mininet Overview [Online] Available: <http://mininet.org/overview/>
- [19] Scapy Adding new protocols [Online] Available: [https://scapy.readthedocs.io/en/latest/build\\_dissect.html](https://scapy.readthedocs.io/en/latest/build_dissect.html)
- [20] Introduction to Mininet [Online] Available: <https://github.com/mininet/mininet/wiki/Introduction-to-Mininet>