時戳式循環排程演算法在 Linux 核心之設計與實作[1]

The Design and Implementation of Timestamp Round-Robin Scheduling in the Linux Kerne

陳郁堂　陳俊榮

台灣科技大學電子系

TEL: (02)27376420

E-mail: ytchen@et.ntust.edu.tw　M8602038@mail.ntust.edu.tw

摘要

現有網路封包排程方式以 FCFS 方式無法提供使用頻寬保證，而 Weighted Fair Queue 在建構上複雜度過高，本論文主要目的探討網路上新封包排程方式，以減輕封包排程系統的額外負擔(overhead)，我們提出一個新的封包排程方法稱為 Timestamp Round-Robin (TRR)，將封包重新切割成固定大小封包，利用 Time-Shift 概念來計算時間戳記(timestamp)，配合雜湊函數(hash function)以循環的方式來安排封包輸出的先後順序。並在 Linux 的網路核心程式(kernel)建構封包排程系統的方法。經實驗測量結果顯示，我們所發展 TRR 演算法性能優越，在執行速度上優於 Time-Shift Scheduling 演算法，在延遲限制(delay bound)優於 Deficit Round-Roubin 演算法。

關鍵字：封包排程、核心程式

Abstract

The objective of this paper is to explore packet scheduling to guarantee the delay bound. To reduce overhead in existing scheduling algorithms, we propose the Timestamp Round-Robin(TRR) scheduling algorithm. The packet is fragmented into constant size and a hash function is used to avoid sorting overhea d. It not only includes the advantage of Deficit Round-Robin[2] but also referees that of Time-Shift scheduling[1]. The proposed schedulin algorithm is implemented in Linux kernel. The results of experiment demonstrate that the TRR is faster than Time-Shift Scheduling in terms of execution time, and it superior to Deficit Round-Robin in terms of delay bound.

Key Words: packet scheduling, Linux kernel

傳輸多媒體即時資料的網路環境必須要提供 Quality of Service(QoS)保證，傳統路由器頻寬的分配方式是 FCFS，在此的架構下無法保證 delay bound 與 fairness，然而 Rate-Based scheduling(封包排程)可以提供這兩項參數的保證，提供傳輸即時資料的能力。一個好的封包排程方法，應該具有下列幾項特點：

1. 公平性(Fairness): 對於每一要求服務的 flow，排程器依照其需求公平地分配頻寬。
2. Delay bound: 封包與封包之間的時間間隔(time interval)必須在固定範圍內。
3. 有效率(Efficiency):處理封包 enqueue、dequeue 所花費的時間與額外負擔越少越好。

一般而言封包的排程分為兩大類，第一類是 sort-based scheduling；WFQ[10]提出配置傳輸速率以交換方式公平分享輸出埠(output port)頻寬，保證網路傳輸及時性資料的 QoS，然而 WFQ 的關鍵問題是複雜度太高($O(\log N)$, $N \in$ *number of flo* , enqueue $O(N)$, dequeue $O(\log N)$)，它每送出一個封包就必須作一次 sort，因此不適合在高速的網路環境下實作。Time-Shift Scheduling(TSS)[1]每一 flow 都會被指定一個遞增的時間戳記，並且挑選最小的時間戳記傳送，TSS 使用一個稱為 time shifting 的新技術，可以調整 flow 的時間戳記比 real-clock 更快，保證排程的公平性。

另一類是 frame-based scheduling，以循環的方式依

[1] 本研究由國科會 NSC 88-2213-E-011-047 補助支持

序服務每一個 flow，沒有排序的額外負擔。Deficit round-robin[2]的基本概念為路由器以循環方式輪流服務每一 flow，DRR 為每一 flow 維護一個計數器稱為 DC，而記數器的數值稱為配額決定每一 round 可以輸出多少數量的封包。DRR 是最有效率的封包排程演算法，其時間複雜度為 *O(1)*。DRR 的缺點，隨著動態加入路由器要求服務的 flow 增多，其 delay bound 也會隨著增加。

為了解決 WFQ 複雜度太高以及避免 DRR delay bound 遞增的問題，研究如何在路由器中加入封包排程的機制，使每一 flow 能夠公平的分享頻寬，我們提出一個新的排程演算法稱為 Timestamp Round-Robin(簡稱 TRR)，參考 Shift-Clock[1]與 Round-Robin[2]的概念，採取切割封包方式，每一封包都具有相同大小，使得每一 round 服務封包的時間相同，TRR 的 enqueue 方式採用 hash functio 取代 sort，而 dequeue 以循序方式檢查輸出陣列，複雜度為(enqueue 為 O(1), dequeu 為 O(P), $P \in$ *prime number*)。我們在 Linux Kernel 2.0.3 實作封包排程演算法，並且證實 TRR 演算法的執行時間優於 WFQ，而 delay bound 不會隨著 flow 的數目增加而增長。

這篇論文架構敘述如下，第 2 節提出 Timestamp Round-Robin 演算法，第 3 節說明如何在 Linux 核心製作 TRR 排程演算法，第 4 節說明實驗測量的結果，最後第 5 節作結論，appendix 詳細說明推導 TRR fairness 與 latency 的過程。

2 Timestamp Round Robi 演算法

我們將接收加以切割或重組成固定大小的封包並以 source 與 destination IP 位址分類稱為 flow。Enqueue 的方法是將每一新加入 scheduler 的 flow 必須參考 Shift-Clock 計算時間戳記，避免 WFQ enqueue 需要 *O(N)* 的負擔，而時間戳記計算方式是將封包長度除以 flow 傳送速率。

Dequeue 的方法是以時間戳記的大小來決定封包輸出的順序，利用 hash 方式取代 sort 避免 WFQ 的負擔 *O(log N)*，而 hash functio 是以時間戳記除以質數取其餘數得到索引值(Index = Timestamp mod Prime)，將封包指標以此索引值掛在輸出陣列，以循序方式搜尋整個陣列。

2.1 Shift-Clock

Timestamp Round-Robin 引進 TSS 中 Shift-Clock 的概念，它負責調整新加入 flow 的時間戳記；如圖 1 所示，在路由器中已經有三個 flow(F_1 ~ F_3)存在，而且他們的實際的傳送速率都超出其預約的速率；如果有第四個 flow(F_4)要進入路由器，則其時間戳記(T_4 = Real-Clock + L_4/R_4)會遠小於 flow(F_1 ~ F_3)，使得路由器會有一段時間一直在服務 F_4，如此一來 F_1 ~ F_3 的延遲時間就會拉長。

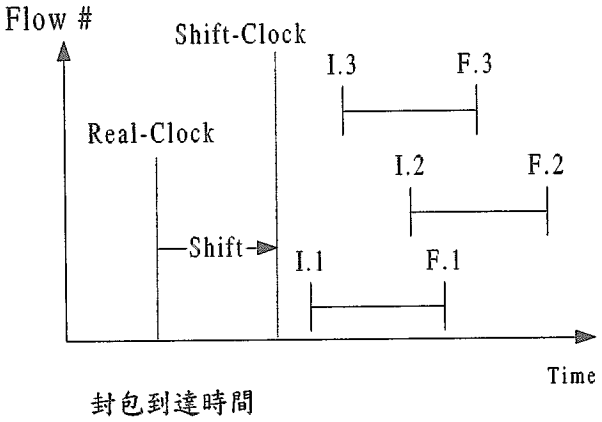


圖 1 Shift -Clock 調整新 flow 的時間戳記

Time Shift Scheduling 認為偷用了多餘的頻寬不應該受懲罰，而 Shift-Clock 就是要解決這種不公平的現象，因此必須調整 F_4 的時間戳記，使它的數值盡量接近 T_0~T_3。方法有兩種：第一種方法是調整實際的 clock 並更動所有 flow 的時間戳記；第二種方法是將實際的 clock 加上一個差異值(稱為 Shift-Clock)，顯然第二種方法比較有效率。

$$ShiftClock = \max(ShiftClock, I_{min}) \qquad (3\text{-}1)$$
$$I_{min} = \min(I_{.0}, I_{.1}, \ldots, I_{n-1}),$$

I_{min} (代表 flow 的 ideal arrival time)

TRR 引進 Shift-Clock 方法主要是在 round-robin 的服務方式下能保證公平性，使用多餘的頻寬不受處罰。Dequeue 的作法是用 hashing functio 的方式取代 WFQ 的 sorting 作法，加速 scheduler 處理封包的效率，並且能夠解決 Deficit Round-Robin 的 delay bound 隨 flow 數目增加的問題。TRR 引進 round-robin 的概念，每一 round 服務一個封包，將所有進入 router 的封包切割或重組成相同大小，保證服務的時間都相同，而且輸出能達到 constant bit rate。以下介紹 TRR 演算法，前置作業部分

說明封包分類方式，而演算法 部分包含了 enqueue 與 dequeue。

Timestamp Round-Robin 前置作業

Step A. 控制訊息(Control Signal)

對於每一個到達路由器的封包，先檢查是否為 start 封包，如果是則到第 B 步驟，否則歸類為 best-effort service。

Step B. 封包分類(Packet classification)

讀取 start 封包資料，依據 IP 位址與埠號(Port number)分類，以雜湊函數[17]提高查表速度，在取得 flow ID 之後到 Step 1。

Timestamp Round-Robin 演算法

Enqueue:

Step 1. 切割封包與配置佇列(Packet fragmentation and per-flow per-queue)

為每一 flow 配置專屬的佇列暫存封包，而每一個封包都會被切割成相同大小，一旦有封包在佇列中時就通知排程器。

Step 2. 計算時間戳記(Calculate timestamp)

排程器就以循環方式服務每一flow，每一新加入的flow都必須等到round結束，以Shift-Clock為參考為 flow 貼上時間戳記，其計算方式如下

$$T_{\cdot f} = T_{\cdot f-1} + \frac{L}{R_{\cdot f}} \qquad (3\text{-}2)$$

($T_{\cdot f}$: flow f 的時間戳記，L : 封包長度，$R_{\cdot f}$: flow f 的預約速率)

Step 3 分配封包到輸出陣列(Dispatch packet)

每一 flow 取得時間戳記後使用雜湊函數將封包指標送到輸出陣列，雜湊函數是將時間戳記除以一個質數得到輸出陣列的索引值並且標上記號，有碰撞以鍊結串列方式解決。

Index = T_f mod P , where P is prime number

Output_Queue[Index] = Data pointer

Step 4. 更新 Shift-Clock(Update)

當路由器有新的 flow 加入就必須更新 Shift-Clock 內容。

Shift-Clock = min(T_1, …, T_n)

Dequeue: 送出封包(Send packet)

送出封包模組，從頭開始掃瞄整個輸出佇列，有封包存在就送出去，如果還有鍊結則一起送出並清除記號。

2.2 TRR 演算法範例

舉一個例子說明 TRR 演算法，假設封包長度為 300 長度單位，服務 4 個 flow 分別預約頻寬為 100、50、75、75 速率單位，每一 flow 在不同時間點所出現的時間戳記計算方式如圖 2(a)所示，Shift-Clock 的初值設為 0，以下列出四個 flow 在時間(t0,t10)的時間戳記，參考圖 2(b)。

if (new flow ID)

$$A_{\min} = \min(A_{\cdot 0}, A_{\cdot 1}, \ldots, A_{n-1});$$

$$ShiftClock = \max(ShiftClock, A_{\min});$$

$$T_{\cdot f} = ShiftClock + \frac{L}{R_{\cdot f}};$$

else

$$T_{\cdot f} = T_{\cdot f-1} + \frac{L}{R_{\cdot f}};$$

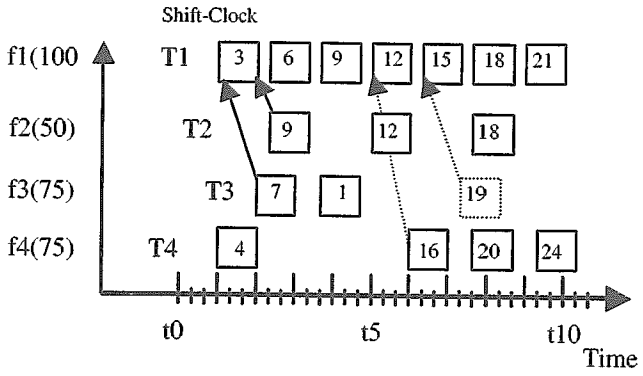
$T_{\cdot f}$:guarantee service flow f，L: 封包長度，$R_{\cdot f}$: scheduler 對 flow f 的服務速率，A_{min}:flow 的到達時間

圖 2(a) TRR timesta m 計算公式

T1 = 0 + 300/100 : 3, 6, 9, 12, 15, 18, 21

T2 = 3 + 300/50 : 9, 12, 18

T3 = 3 + 300/75 : 7, 11, x, x, 19 ; x 表示封包未到



T4 = 0 + 300/75 : 4, x, x, x, 16, 20, 24

圖 2(b) TRR 範例之計算時間戳記圖

更新 Shift-Clock 的內容，取該 round 最小 flow 的時間戳記，計算時間戳記必須參考 Shift-Clock，除了新加入的 flow 之外如果有 flow 在某時間停止送封包，過

了一段時間又開始送，為了公平的考量必須參考 Shift-Clock 調整新的時間戳記，如圖 2 中 flow 3, 4。

2.3 分析與比較排程演算法特性

Fairness [12]，其傳送資料時間的定義為封包資料長度除以平均傳送速率，任意兩個 flow 之間的資料傳送時間差異定義為 fairness，而 TRR 的 fairness 表示式如(1)所示。

Latency [12]是指在忙碌的傳送週期內，指當封包從進入一個空的輸入佇列到離開所花費最久的時間，TRR 的 latency 表示式如(2)所示。

$$F^{TRR} \le \frac{2\Phi}{\rho_i} \quad \# \qquad (1)$$

$$L^{TRR} = \frac{\Phi}{r} + \frac{\Phi}{\rho_i} \qquad (2)$$

(r: router service rate, ρ_i : flow i rate, Φ :packet size)

以下在附錄 B 的表格 1 中，我們將比較各種封包排程演算法的 fairness、latency、enqueue complexity 與 dequeue complexity。

3 TRR 演算法在 Linux Kernel 實作

封包排程系統要改變路由器轉送封包的順序取代傳統式先來先服務的作法，以 Linux 網路核心架構為主加入封包排程模組實作封包排程系統，在附錄 B 的圖 3 說明 Linux 網路模組架構，灰色方塊是本論文實作修改的部分。

為了在實際的網路環境下實作封包排程系統，選擇修改 Linux 的網路核心程式，擴充 Linux 的 software router 功能，使其具有封包排程的能力。在硬體裝置抽象層實作封包排程系統，保持原有的網路程式相容性。在圖 3 中，最底層是網路的 driver，負責控制硬體接收以及傳送封包的工作，第二層是硬體裝置(device)抽象層，提供一致化的網路軟體介面(system call)。

3.1 實作方法

介紹在 Linux IP 層實作封包分類模組，並解釋與 Linux 核心模組之間的關係，下圖 4、5 中有陰影方塊為 Linux 原有的模組。ip_rcv()接收封包後加入過濾封包模組，以濾除多餘的封包，接下來做選徑(route)。我們修改 ip_queue_xmit()的程式加入封包分類模組，當封包都切割成相同大小後，為每一 flow 配置佇列，最後將封包送到硬體裝置抽象層。

Linux 在硬體裝置抽象層送出封包的函式為 dev_queue_xmit()如圖 6 所示，保證在處理封包的過程當中，在還沒有處理完畢之前不會再接受其它中斷，它會

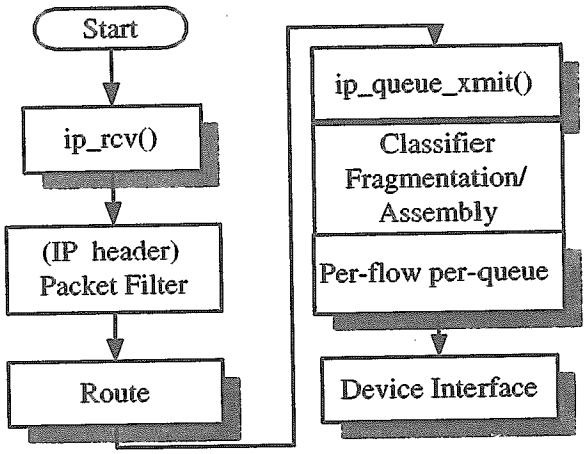


圖 4 在IP層分類器與封包切割模組之關係

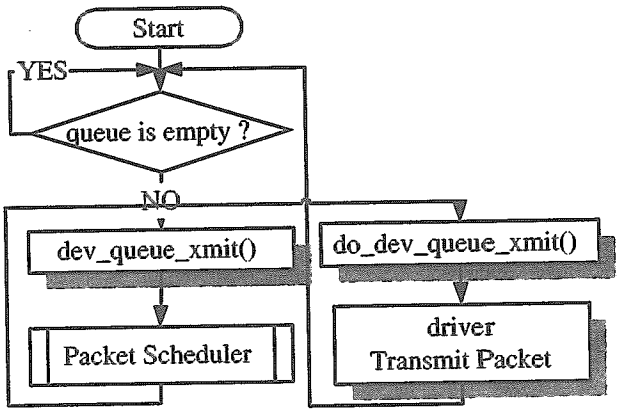


圖 5 封包排程器在Linux device層的處理流程

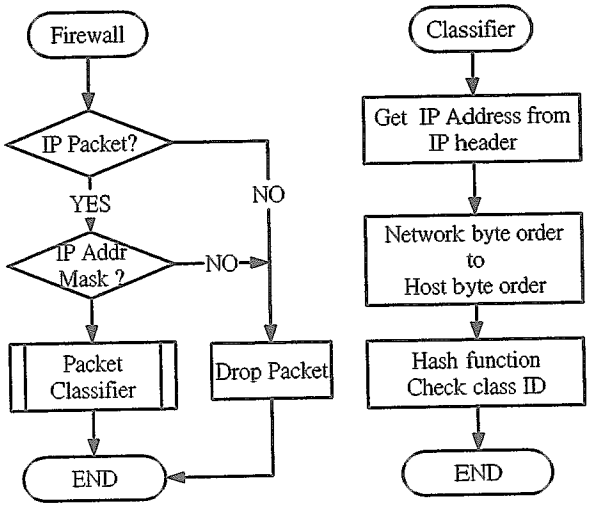


圖 6 Packet filter與Classifier的流程圖

呼叫 do_dev_queue_xmit()負責維護中斷處理、產生網路封包 CR 碼並且呼叫網路卡驅動程式，最後由網路卡實際送出封包。我們修改 Linux 硬體裝置抽象層核心程式加入封包排程模組，作法是在封包要送出之前做排程，決定封包服務的順序，當封包送到 dev_queue_xmit()時，將封包攔截下來並由排程器決定哪一個 flow 的先後順序然後再由 do_dev_queue_xmit()送出封包，一直重複此流程直到輸入佇列中所有封包都送完為止。

定義控制訊息封包

定義兩個特殊封包，分別是 start 與 end 封包的格式，一般 IP 封包其 protocol ID 為 0x0800(十六進位)，依順將開始與結束封包的 protocol ID 分別訂為 0x0801 與 0x0802。排程系統的封包過濾模組會參考此 protocol 欄位，判別 flow 是屬於 guarantee servic 或是 best-effort service 的封包，若兩者都不是就將該封包丟棄。

start 封包的格式

Protocol ID = 0801						
Source IP Addr	Destination IP Addr	Source port number	Destination port number	Reserved bandwidth	Starting time	Finished time

end 封包的格式

Protocol ID = 0802						
Source IP Addr	Destination IP Addr	Source port number	Destination port number	Reserved bandwidth	Always 0	Always 0

3.2 封包的分類方式

參考[17]的方法以來源位址、目的位址以及應用程式通訊埠作為分類的依據。參照圖 6 所示說明如何製作封包過濾器與 flow 分類器，在 IP 層接收封包函式 ip_rcv()時設定過濾封包的條件，形成一個防火牆，然後依據 <source , destination> IP,port 作為分類的方式，接收到一個封包並取出 header 的資料，將之從網路位元組順序轉換為主機位元組順序，取出來源位址、埠號與目的位址、埠號，四個 32 位元整數做 XO 運算之後會得的一個整數，再將這個數值除以質數可以得到一個索引值，依據此索引查詢表格。

封包分類雜湊函數 **Classification hash function**

CAM32 = [SIP⊕Sport⊕DIP⊕Dport]

Index = CAM32 mod Prime

(SIP: Source IP Address, Sport: source port; DIP: Destination IP Address, Dport: Destination port)

乙太網路是廣播型網路，在同一分段上連線的電腦，隨時都會收到任何一部電腦所發出的封包。路由器對於收到的封包首先判別是否屬於 IP 封包，如果不是就丟棄，第二步再繼續檢查來源端與目的端位址如果不符合過濾條件就丟棄該封包，所以建立封包過濾機制的優點是(1)過濾多餘或不合法封包，減輕路由器的負擔提高處理封包速度。

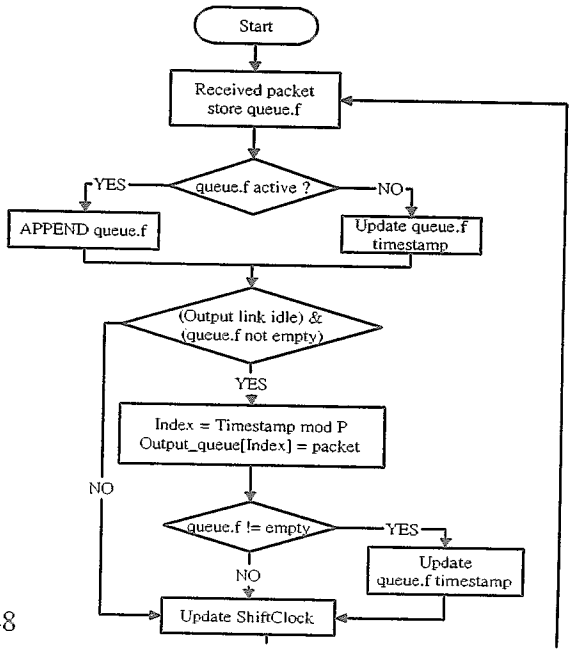
如果封包通過封包過濾器的檢查，就進入封包分類器，而分類器會依據封包的來源位址、埠號與目的位址、埠號做分類，並且產生相對應的佇列儲存封包，為了加快分類器處理速度，採用了雜湊函數。

3.3 封包切割與排程實作

在 Linux 上設定 MTU 的大小，使每一個輸入的封包都能切割或組合成相同的大小，方法是修改網路組態設定檔，修改/etc/rc.d/rc.inet1 這個網路組態設定檔記錄 Host name, Host IP, Gateway, DNS, Route table 以及 MTU，也就是當 Linux 作業系統啟動時會參考這一個設定檔，完成乙太網路卡的設定，下面的範例將 eth0 網路卡 MTU 設定為 500 bytes。

ifconfig eth0 140.118.2.48 netmask 255.255.0.0 broadcast 140.118.255.255 mtu 500

封包排程器再根據分類資訊來作封包輸出先後次序的安排，配置封包輸入佇列為每一個 flow 計算時間戳記並且使用雜湊函數分配到輸出佇列，最後更新 Shift-Clock。實作時使用靜態配置記憶體方式，不會動用到 Linux 核心的動態配置記憶體，因為 Linux 核心最



大只能配置到 128K bytes，實作流程如圖 7 所示。

4 實驗測量的結果

實驗環境是利用兩部 K6-II 350Mhz 的 PC，其中一部電腦執行 router 功能，另一部當作 host 產生大量封包測試各項參數。

4.1 切割封包 overhead

比較切割封包的額外負擔對不同演算法的影響，MTU(Maximum Transmit Unit)是決定封包切割的因素，觀察圖 8 對 TRR 而言 MTU 等於 800 bytes 時執行時間最短。

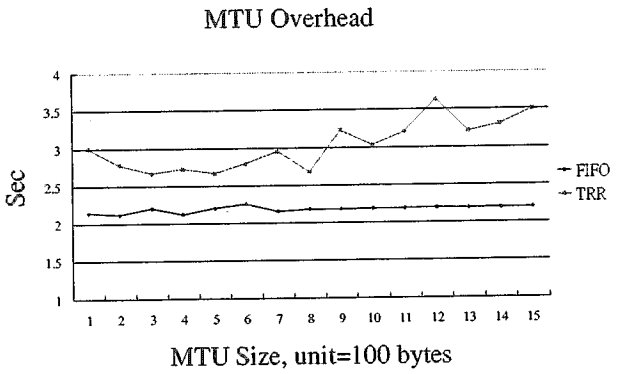


圖 8 MTU 大小對封包排程系統的影響

Linux 的記憶體配置是以分頁的方式，每一分頁有 4096 bytes，MTU 愈大則所造成的記憶體碎片愈大。以 FIFO 的運作方式並不需要大量記憶體來暫時存放封包，所以記憶體碎片很少，因此其執行時間不受 MTU 的影響，但是封包排程系統就需要為每一flow 配置一個佇列，因此會佔用較多的記憶體暫時存放封包，所以記憶體碎片會隨著 MTU 大小與封包數量的增加而增加。在表 1 中探討 MTU 大小對記憶體分頁的利用效率，可以看出當 MTU 分別為 500、800、1000 bytes 時剩餘記憶體最少。

表1 Linux記憶體分頁在不同MTU下所造成的記憶體碎片大小

MTU	100	300	500	700	800	900	1000	1100	1200	1400	1500
#Packet	40	13	8	5	5	4	4	3	3	2	2
Mem	96	196	96	596	96	496	96	796	496	1296	1096

4.2 執行時間(Execution time)

測量演算法在計算封包輸出順序所花費的時間，

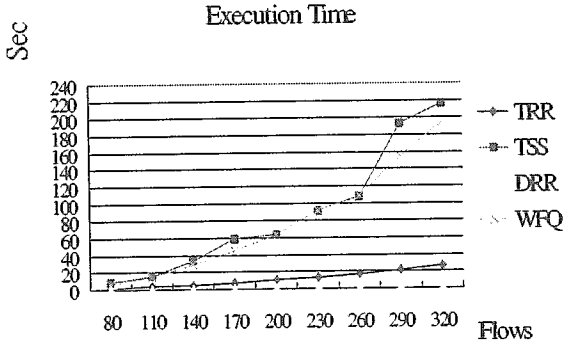


圖 9 測量系統處理大量 flow 所需時間

實驗結果發現，TRR 演算法在執行時間的花費比 WFQ 快，而 flow 數目愈多則 sort-based 封包排程演算法所花費在 sort 的時間就愈長，而 DRR 演算法是線性增加，參考圖 9。

4.3 輸出量(Throughput)

實際測量線路最高的傳輸量約 7.5Mbps，比較 TRR 與 DRR 的輸出量如圖 10 所示。以 fairness 而言，TRR 對於多餘頻寬是採取平均分配的策略，然而 DRR 就是採取記數器(deficit counter)補償策略，偷用頻寬者受限於 input queue 的大小，而 WFQ、TSS 演算法因為 sortin overhead 實際測量線路最高的傳輸量約 7.5Mbps，比較 TRR 與 DRR 的輸出量如圖 10 所示。以 fairness 而言，

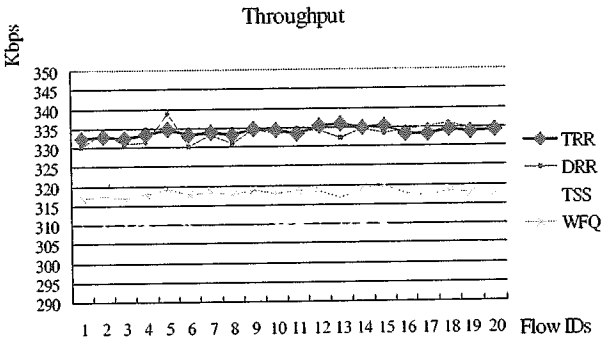


圖 10 封包排程演算法輸出量比較圖

TRR 對於多餘頻寬是採取平均分配的策略，然而 DRR 就是採取記數器(deficit counter)補償策略，偷用頻寬者受限於 input queu 的大小，而 WFQ、TSS 演算法因為 sorting over h e a的影響，使得它們的 throughput 較低。

4.4 延遲限制(Delay bound)

比較四個演算法的平均 delay bound 所得結果如圖

11 所示，結論是 TRR 以 timestam 方式服務近似 sort-based 的方法，因此 delay bound 接近 WFQ。Sort-based 演算法對每次輸出封包後都重新排序，最大優點就是能保證 delay bound 能隨時控制，所以 WFQ 平均 delay bound 最短；TRR delay bound 比 WFQ 長；DRR 平均 delay bound 最長。

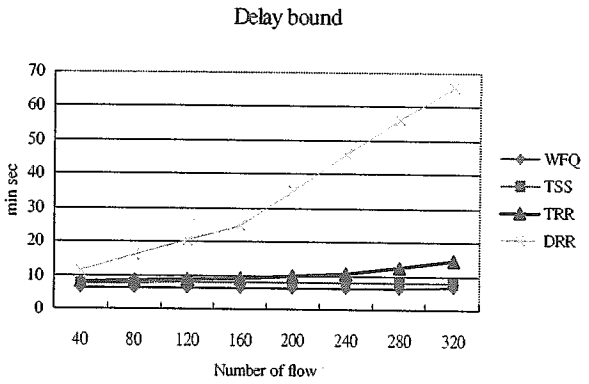


圖 11 排程演算法平均 delay bound 的比較

4.5 排程演算法 flow 數的上限

Linux 系統核心的可用記憶體為 128K bytes，因此能負荷最大的 flow 數與封包大小有關，Linux sock buffer 資料結構大小不包含資料部分為 148 bytes，以最小的封包 64 byte 來計算，理論上 flow 的上限為 618。DRR 的記憶體負擔最少因此可容納較多的 flow 數而 TRR 不需要排序但是記憶體負擔比 DRR 多，因此可容納的 flow 數比 DRR 略少，實際測量結果列在表 2。

表 2 排程演算法flow數的上限比較表

演算法	TRR	DRR	WFQ	TSS
Flow數上限	445	450	396	412
資料結構大小(bytes)	42	40	62	56

5 結論

TRR 採用了時間戳記的方法以保證封包 delay bound，並且使用固定大小(800 bytes)的封包達成利用循環的方法，快速處理輸入與輸出封包的目的，enqueue 時間複雜度為 *O(1)*,而 dequeue 為 *O(P)*，P 愈大則 TRR 演算法的特性就愈類似 sort-based 演算法，能夠保證 delay bound，相反 P 愈小則類似 frame-based，delay bound 控制較差，但處理封包速度變快。比較重新切割與重組封包所花費的時間比不作這些動作要多出 7.3%。

最後提出一套可以在 LINUX kernel 上實作封包排程的方式，驗證系統的可行性並且與 WFQ、DRR、TSS 等演算法做了效能上的比較，實驗結果發現在 delay bound 的控制，TRR 與 TSS 相近但比 DRR 好，而執行時間 TRR 僅次於 DRR 演算法比 TSS 快。

References

[1] Jorge A. Cobb, Mohamed G.Gouda, "Time-Shift Scheduling Fair Scheduling of Flows in High -Spee Networks" in IEEE/ACM Transactions O Networking, VOL. 6, NO. 3, JUNE 1998 pp. 274-284

[2] M. Shreedhar and G. Vrghese , "Efficient Fair Queuing Using Deficit Round-Robin" in IEEE/ACM Transactions On Networking, VOL. 4, NO. 3, JUNE 1996 pp. 375-385

[3] P.Goyal, H. M. Vin, and H.Cheng. "Start-time Fair Queuing: A Scheduling Algorithm for Integrated Services Packet Switching Networks" In Proceedings of ACM SIGCOMM'96, August

[4] L. Zhang, "Virtual clock: A new traffic control algorithm for packet-switched networks", ACM Trans. Comput. Syst., vol 9, no 2, pp. 101-124 May 1991

[5] R. BREYER & S. Riley, "SWITCHED & FAST ETHERNET" from table 5-1 in Chapter 5, ISBN 1562763385

[6] S. Floyd, V. Jacobson "Link-Sharing and Resource Management Models for Packet Networks". in IEEE/ACM Transactions On Networking, VOL. 3 NO. 4, AUGUST 1995 pp. 365-386

[7] S. Keshav "On the Efficient Implementation of Fair Queueing" In Internetworking: Research an Experience, Vol. 2, pp. 157-173

[8] H. Zhang, S. Keshav "Comparison of Rate-Based Service Disciplines" In University of California at Berkeley 1991

[9] W. R. Stevens "TCP/IP Illustrated volume 1" ISBN 957-8682-77-1

[10] J. Nagel, December RFC 970, "On Packet Switche with Infinite Storage"

[11] P. Ferguson, G. Huston "Quality of Service: Delivering QoS on the Internet and in Corporat Networks" ISBN 0-471-24358-2

[12] D. Stiliadis, A. Varma "Latency-Rate servers: A General Model for Analysis of Traffic Scheduling Algorithms" in University of california, Santa Cruz

[13] D. Stiliadis, A, Varma "Frame-based Queueing: A New Traffic Scheduling Algorithm for Packet-Switched Networks" in University of California, Santa Cruz

[14] S. Golestani, "Self-Clocked Fair Queueing Scheme for Broadband Applications" in Proc. IEEE INFOCOM 1994

[15] A. Rubini "LINUX DEVICE DRIVERS" ISBN 1-56592-292-1

[16] M. Beck, H. Böhme, M. Dziadzka "LINUX KERNE INTERNALS" second edition ISBN 0-201-33143-8

[17] R. Edell "Billing Users and Pricing for TCP" in IEEE Journal on selected areas in communications VOL. 1 NO. 7 September 1995

[18] M. Waldvogel, G. Varghese, J. Turner, B. Plattne "Scalable High Speed IP Routing Lookups" SIGCOMM 1997 Cannes, France

[19] J. C.R. Bennett, H. Zhang "WF^{2}Q: Worst-case fair

weighted fair queuing" in Proc. INFOCOM'96, Mar 1996, pp.120-127
[20] K. Cho, H. Yoon "Design and Analysis of a Fai Scheduling Algorithm for QoS Guarantees i Hig -Speed Packet-Switched Networks" in Proc. IEEE NFOCOM'98 pp.1520-1525

Appendix A

Table 1. 各種封包排程演算法之比較

演算法	Latency	Fairness	Enqueue	Dequeue
WFQ	$\frac{L_i}{\rho_i}+\frac{L_{\max}}{r}$	$\max(C_j+\frac{L_{\max}}{\rho_i}+\frac{L_j}{\rho_j}, C_i+\frac{L_{\max}}{\rho_j}+\frac{L_i}{\rho_i})$ $C_i=\min((N-1)\frac{L_{\max}}{\rho_i}, \max_{1\le n\le N}\frac{L_n}{\rho_n})$	O(N)	O(log N)
DRR	$\frac{(3F-2\Phi_i)}{r}$	$\frac{3F}{r}$	O(1)	O(1)
SCFQ	$\frac{L_i}{\rho_i}+\frac{L_{\max}}{r}(N-1)$	$\frac{L_i}{\rho_i}+\frac{L_j}{\rho_j}$	O(log N)	O(log N)
VTRR	$\frac{L_{\max}}{\rho_{\min}C}+\frac{L_{\max}}{r}+\frac{L_i}{\rho_i}$	$\frac{2L_{\max}}{\rho_{\min}}(\frac{1}{C}+1)$	O(log log C)	O(log log C)
TRR	$\frac{\Phi}{\rho_i}+\frac{\Phi}{r}$	$\frac{2\Phi}{\rho_i}$	O(1)	O(P)

(符號說明： ρ_i :平均速率，L_{max}:最大封包長度，N: flo 數目，r: server rate，Φ: 封包平均長度，F: $frame = \sum_{i=1}^{N}\Phi_i$， $C = \frac{L_{\max}}{\rho_{\min}T}$, T: virtual time interval between rounds, P: prime number

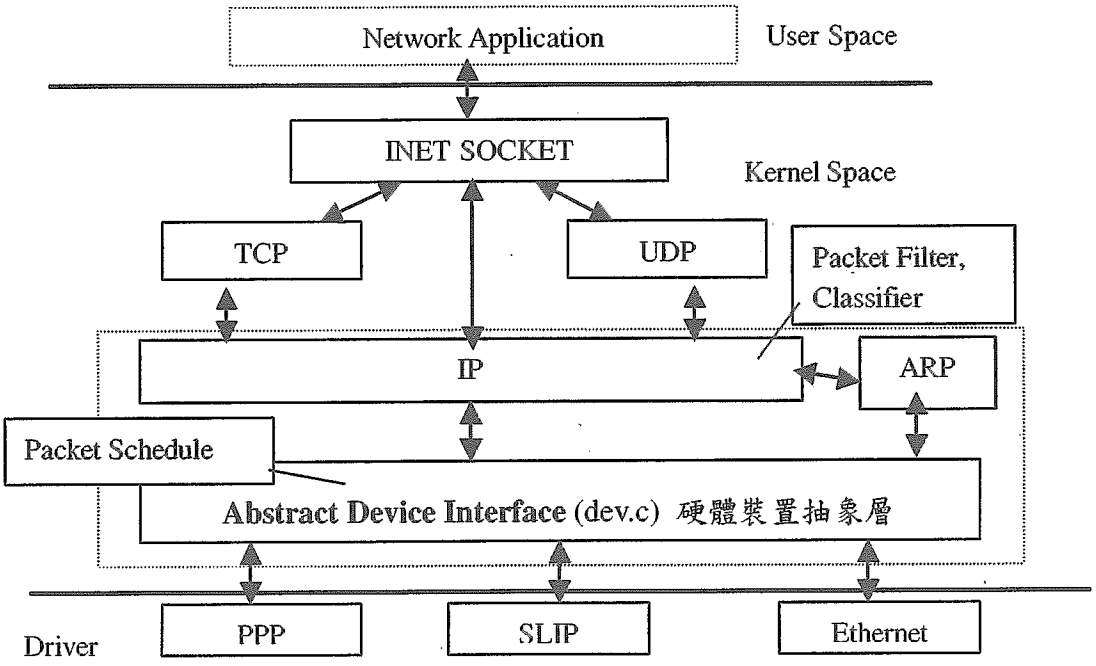


圖 3 封包排程模組在 Linux Kernel 之架構圖