中華民國八十四年全國計算機會議論文集
Proceedings of National Computer Symposium 1995

一個新佇列排程法:先窺式最短服務優先法
A NEW QUEUEING DISCIPLINE: LOOK AHEAD SHORTEST-SERVICE-FIRST

朱延平　　黃一泓　　紀裕華
Y.P. Chu, E.H. Hwang and Y.H. Jee

國立中興大學應用數學所
台中市國光路250號
E-mail:ypchu@flower.amath.nchu.edu.tw

摘要

本篇論文提出一種路由器佇列排程演算法，以有效控制及改進在分封交換網路上之路由器及網路線等分散式資源。經由文獻探討，最簡單的先到先服務佇列排程法，它不能顧慮到每條起終點連線（發送端至接收端）的封包長度，更不能保証公正性。而另一種著名的「公平排程法」，它雖確保傳輸網路線間彼此的公平性，但在計算時卻要花相當多的時間來計算。在這篇論文裡，我們提出一種叫作「先窺式最短服務優先法」的排程方法，這種新的佇列排程法不僅計算簡單，而且能像公平排程法一樣，滿足公平原則。爲了証實本文所提排程法的優越性，我們設計一個與眞實網際網路架構及通訊協定一致的模擬實驗環境，利用實驗的結果，將本文所提的排程法與公平排程法，在公平性、執行複雜度、以及整體平均延遲上，作一比較。

關鍵字：佇列排程法、公平服務法、先到先服務法、公平性、網路擁塞

ABSTRACT

In this paper, we proposed a queueing discipline working on switching node. It can control and improve efficiently the distributed resources of switching node, like buffer space and channel bandwidth. As we know the simplest FCFS queueing discipline can't consider about packet's length and fairness of each conversation. Another famous discipline, Fair Queueing, which promises the fairness among coonversations but needs much time to compute. Our method called "Look Ahead Shortest-Service-First" that is not only ease to implement but also satisfying the fairness as familiar as Fair Queueing. In order to verify the adventage of our algorithm, we design a simulation environment as same as the real Internet architecture and protocols. And we will use the results of simulation and focus on fairness, queueing delay and implement complexity to make comparisons with Fair Queueing.

Keywords: Queueing Discipline, Fair Queueing, FCFS Queueing, Fairness, Network Congestion.

一、引言概述

由於資訊的普及化，電腦網路的充分利用使得網路上流通的資料量變得更多且更頻繁，因此網路上的資料擁塞控制就顯得相當重要[1、2、4、18]。欲完成網路上的擁塞控制大致上有兩個方向[12]。第一是在起點端 (source host) 運用改變傳送封包 (packet) 的速率所設計出的流量控制演算法 (flow control algorithm)，雖然流量控制的主要目的在確保接收端 (destination host) 的緩衝區 (buffers) 能夠有效使用，但我們更關心的是它在整個網路裡所扮演的限制封包送出的角色。另一種則是在匣通道 (gateway) 上透過路徑 (routing) 和排程 (queueing scheduling) 演算法來控制擁塞，比如說利用可調適路徑法 (adaptive routing) 讓封包遠離網路中的瓶頸路段 (network bottlenecks)，在排程演算法中它只管封包被傳出的順序與緩衝區分配情形，並不改變匣通道上整個對外傳輸的交通 (traffic)，感覺上好像跟擁塞控制無關，可是我們發現在排程的過程中，所有的起始端點的交通皆一起參與並且相互影響，若是封包遲遲未被送出，導至發送端 (sender) 無法收到封包回饋 (packet ACK)，推斷認爲在傳送的路徑上，可能有擁塞發生，而限制封包傳出的流量速度。因此匣通道可說是擁塞狀況的前哨站，經由它的控制可間接的解決擁塞的問題。一個好的佇列排程法，必須能避免網路中各個使用者，遭受不當使用者 (ill-behaved user)[15]及網路負載變動 (network load fluctuations) 的干擾[14]。不當使用者經常以較高的速率傳出資料，造成頻寬 (bandwidth) 大量的被佔據；網路負載的變動則造成網路交通突增 (bursty) 的現象，形成擁塞。近年的研究指出，採用速率基礎服務排程法 (rate-based service discipline)[5、6、7、9、10、11、13、14]可以解決此種問題。速率基礎服務排程法無關於 (independent) 網路中交通的特性，可以保障網路中的使用者

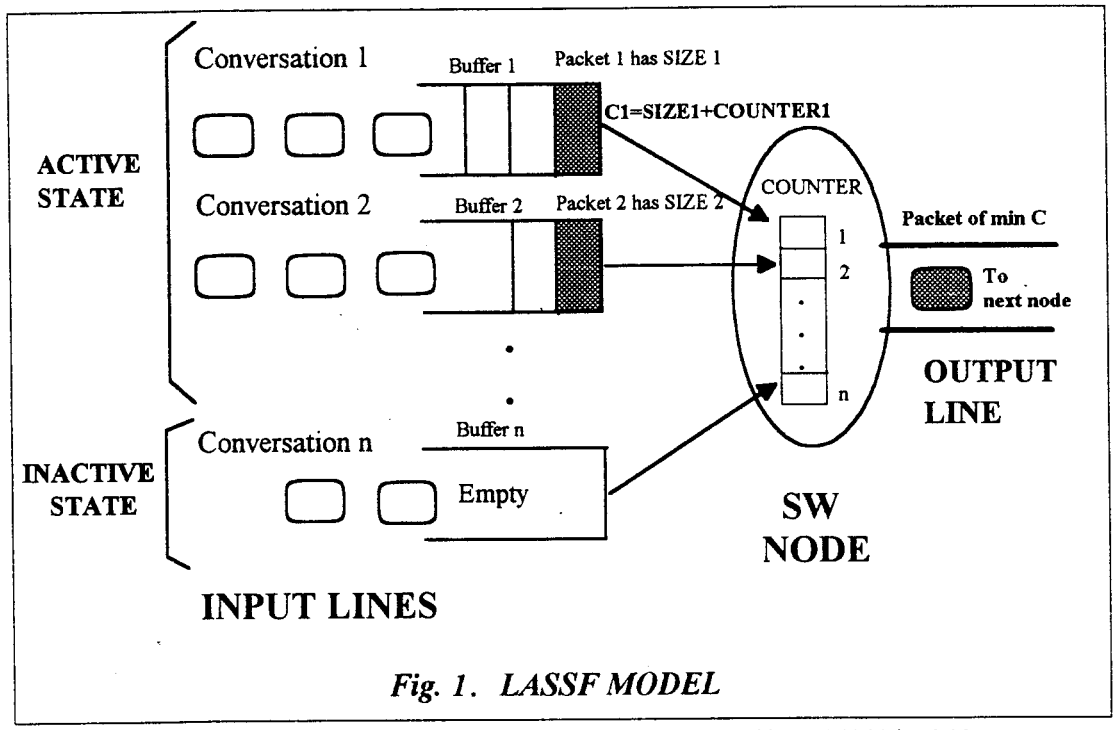

Fig. 1. LASSF MODEL

最低限度的服務率，此演算法，基本上可分配三種獨立的資源[12]：頻寬 (bandwidth)、提示 (promptness) 以及緩衝區空間 (buffer space)。經由流量控制演算法的合作，此演算法可以使網路中使用者，公平的分享網路的資源。

在我們最熟知的先到先服務法 (First-Come-First-Service) 裡，幾乎是把擁塞問題留給發送端來處理，而頻寬、提示與緩衝區等資源分配完全基於封包到達時間來決定。FCFS 雖簡單，然而公平性卻不易達成，因此在今天的環境中漸漸地不被採用。另一種由循環式 (Round-Robin) 服務的觀念所提出的公平排程演算法 (Fair Queueing Algorithm)[8、12、17]，不論是封包接封包 (packet-by-packet) 或位元接位元 (bit-by-bit) 的方式都是對每一個起終點連線 (source-destination conversation)，給予一個屬於自己的佇列 (queue)，並依序循環的傳送封包，讓個別起終點連線的封包數可被設限，而達到公平分配資源的原則。

我們都知道在執行時間長短已知的前題下，最短服務優先 (Shortest-Job-First) 的排程演算法，可讓平均等待時間最短。基於這個原理，爲了降低封包傳送時，所產生的延遲問題，以及頻寬公平分配原則，我們提出了一個新的佇列排程演算法：「先窺式最短服務優先法」(Look Ahead Shortest-Service-First Algorithm)，簡稱爲 LASSF。經由此排程法的運作，不管起點端的流量控制採用何種方式，它均可使得使用者公平的分享網路的資源。

在這篇論文的第二節部份，將詳細描述我們所提出的 LASSF 演算法，並藉著模擬實驗來檢測我們的結果。接著在第三節裡我們把實驗的結果和 FQ，在演算法複雜度、公平性與平均延遲作一比較。最後一節提出我們的結論。

二、LASSF 演算法

我們的演算法是基於無連結 (connectionless) 的網路環境(在非連結導向的網路中，頻寬等資源不會事先被分配給每個使用者)，而且在起終點連線一旦建立連結 (setup connection) 後，路徑不會改變。與公平排程法相同的，採用持續工作的佇列排程法 (work-conserving queueing discipline)[3]，當交換節點 (switching nodes 如 routers, gateways) 內有封包時，會立即傳送封包到下一個節點；當交換節點無法傳送封包時，封包便被儲存在佇列(或緩衝區)中 (queues or buffers)，並且每條起終點連線，都有一個指標可以立即找到第一個在佇列的封包資料。當佇列空間容納不下過多的封包時，則可依所定的傳輸協定 (protocol) 拿掉最先、最後或隨機一個封包，以減少佇列中的封包個數。除此之外，我們允許封包長度是任意且變動的，以及對發送端所採用的傳送模型亦不設限制。

由圖 1我們來說明 LASSF 演算法，在圖中有 n條起終點連線，每條的連線 i 在交換節點裡都有一個 $\boldsymbol{COUNTER}_i$ 來記錄連線 i 上已送了多少資料(單位是 Byte)。封包在送到交換節點前，均先放在自己所屬的 $\boldsymbol{BUFFER}_i$。對於 $\boldsymbol{BUFFER}_i$ 的第一個封包我們用 $\boldsymbol{PACKET}_i$ 表示，其大小爲 $\boldsymbol{SIZE}_i$。另外我們定義，當佇列中有封包被儲存下來，則這條起終點連線爲可動的 (ACTIVE) 連線(如連線 1、2)，反之爲休止的 (INACTIVE)。最後爲了方便計算，我們定義 $C_i=\boldsymbol{COUNTER}_i+\boldsymbol{SIZE}_i$。

首先當對外傳輸線 (OUTPUT LINE) 是可傳送時 (free)，我們由佇列中找出一條最小的 C_i值，並傳回 $\boldsymbol{PACKET}_i$ 爲要被傳送的封包，這個 C_i取法的意義，即依照最短工作優先 (Shortest-Job-First) 的原則而先窺 $\boldsymbol{SIZE}_i$的大小。爲了要讓頻寬公平地分配給不同的連線，我們藉著調整 $\boldsymbol{COUNTER}_i$的值來滿足這項要求。因此我們對 $\boldsymbol{COUNTER}_i$的更新有下列三種考量方式：

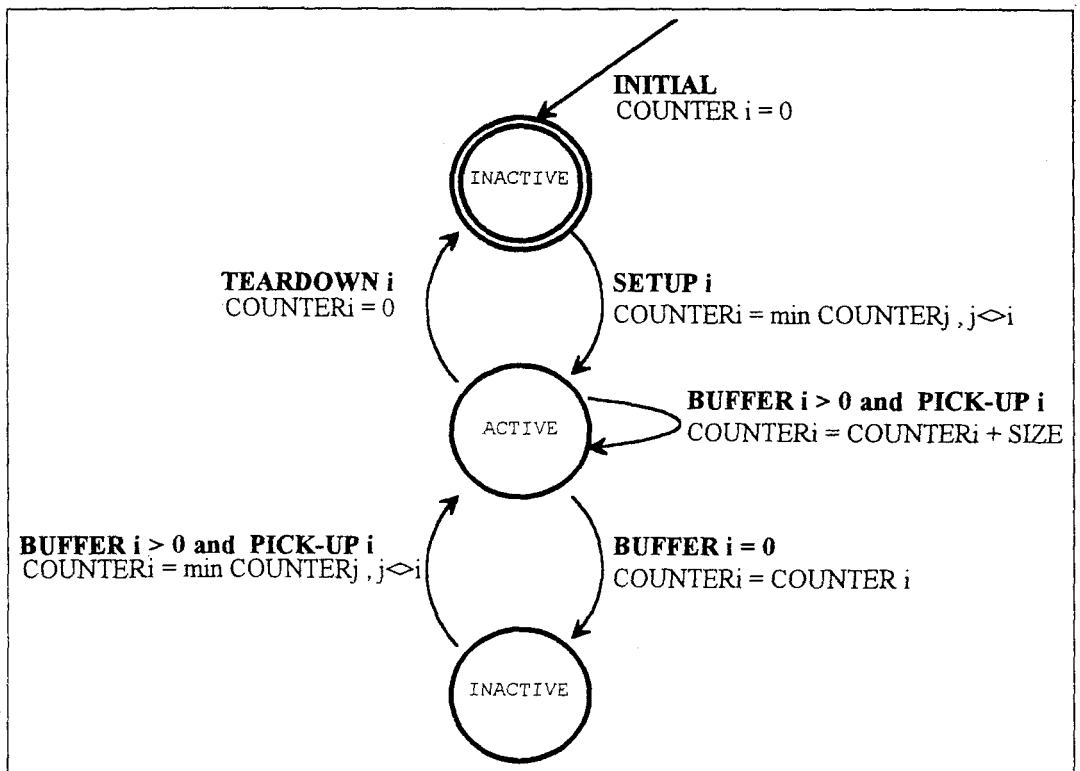

Fig. 2 Diagram of STATE changing and EVENT / COUNTER updating

(1) 當連線 *i*的第一個封包到達時，則令 $COUNTER_i$ 等於其他連線 *j*的最小值。(最小值不能爲零)

(2)當連線 *i*欲結束連結，它的最後一個封包到達時我們則令其值爲零。(重設，使其不影響其他的 *j*)

(3)當 $STATE_i$由休止的轉入可動狀態時，我們再令 $COUNTER_i$等於其他連線 *j*的最小值。(最小值必須大於 $COUNTER_i$)

在演算法中，***STATE*** 轉換與 ***COUNTER*** 更新方式的關連性，可由圖 2 來表示。利用這些更新方式，便能在瞬間讓頻寬公平地分享給每個起終點連線，在(1)(3)點中清楚指出，頻寬不會被新的連線(或休止很久又再傳送資料的連線)給強制壟斷。此演算法最後一個步驟，則是偵測連線 *i*是否轉入可動的狀態。

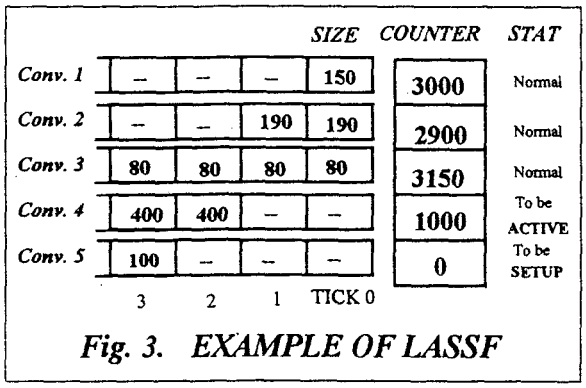

Fig. 3. EXAMPLE OF LASSF

我們利用圖 3的例子來說明 LASSF 演算法，其結果在表 1中。從圖 3中我們清楚地看出連線 1、2 和 3在時間 0 (TICK 0) 時佇列都有資料，而連線 4 會在時間 2時，由休止轉入可動狀態。連線 5將在時間 3時被建立連結。所以依照 LASSF 法，在時間 0，交換節點會選擇連線 2的封包來傳送(因爲 2900+190 < 3000+150 且 2900+190 < 3150+80)。時間 1時，將選擇連線 1的封包(同理)。我們留意，時間 3在選完連線 4之後， $COUNTER_4$ 變成 3490；以及時間 4後，$COUNTER_5$變成 3190。這兩個值分別因爲符合更新的條件 (3) 與 (1) 之故。

TICK	*CNT 1 (SIZE)*	*CNT 2 (SIZE)*	*CNT 3 (SIZE)*	*CNT 4 (SIZE)*	*CNT 5 (SIZE)*	NEXT
0	3000(150)	2900(190)	3150(80)	1000(-)	0(-)	Conv. 2
1	3000(150)	3090(190)	3150(80)	1000(-)	0(-)	Conv. 1
2	3150(-)	3090(190)	3150(80)	1000(400)	0(-)	Conv. 4
3	3150(-)	3090(190)	3150(80)	3490(400)*	0(100)	Conv. 5
4	3150(-)	3090(190)	3150(80)	3490(400)	3190(-)*	Conv. 3
5	3150(-)	3090(190)	3230(80)	3490(400)	3190(-)	Conv. 2
.	.	.	.	.	.	.

CNT means *COUNTER*

*3490=3090(Conv 2)+400 *3190=3090(Conv 2)+100

Tab. 1. RESULTS OF LASSF

整個詳細的演算法如圖 4 所示：

VARIABES DEFINITON:

$COUNTER_i$:counter of the conversation *i* (packet sent)

$STATE_i$:state of the conversation *i* (ACTIVE or not)

$BUFFER_i$:buffer in switching node of conversation *i*

$PACKET_i$:first (head) packet of $BUFFER_i$
$SIZE_i$:size of $PACKET_i$
C_i=$COUTER_i$+$SIZE_i$
Main Function:
let all $COUNTER_i$=0 and
let all $STATE_i$=INACTIVE;
forever
    if output line is free
        get packet from queue
        [call get_shortest_packet()] ;
        update the $COUNTER_i$
        [call get_least_service()] ;
        $COUNTER_i = COUNTER_i + SIZE_i$;
        remove packet from Q, send it to next;
        update the $STATE_i$
        [call change_state()] ;
    endif
end.
Function get_shortest_packet()
    for all conversation i compute the values of C_i ;
    we choice the $PACKET_i$ of $min\ C_i$; (for all $SIZE_i > 0$)
    return $PACKET_i$;
end.
Function get_least_service()
    if first packet comes
        $COUNTER_i = min\ COUNTER_j$; (for all j<>i and $COUNTER_j > 0$)
    else if last packet comes
        $COUNTER_i = 0$;
    else (normal situation)
        if $STATE_i$ = INACTIVE (INACTIVE => ACTIVE)
            $COUNTER_i = min\ COUNTER_j$; (for all j<>i and $COUNTER_j > COUNTER_i$)
end.
Function change_state()
    if $BUFFER_i > 0$
        $STATE_i$ = ACTIVE;
    else
        $STATE_i$ = INACTIVE;
end.

Fig. 4. LASSF ALGORITHM

三、模擬實驗與比較

現在我們就把 LASSF 與 FQ 作一比較。比較的部份有兩大項，第一是兩者間演算法的複雜度，其次是利用模擬實驗來評估兩者間的效能。

3.1 LASSF 與 FQ 的執行複雜度

在作比較之前我們先看一下 FQ 的演算法[12]，首先，我們對 FQ 的變數定義作一描述：

P^i_k 表連線 i上第 k個封包的封包長度 (Packet size)

F^i_k 表連線 i上第 k個封包的完成傳送數 (Finish number)

$R(t^i_k)$ 表連線 i上第 k個封包在時間 t到達時的循環數 (Round number)

d^i 表連線 i上誤差值 (delta)。它影響著，當連線爲休止時 F^i_k 與 F^i_{k-1} 的關聯程度(correlation)。

B^i_k 表連線 i上第 k個封包的投標數 (Bidding number)

並且定義，當 $R(t) \le F^i_k\ \forall k = MAX(j|t^i_j \le t)$時，這條連線爲可動的。接著，我們來看 FQ 的演算法[17]：

if conversation i is ACTIVE
$$F^i_k = F^i_{k-1} + P^i_k$$
else
$$F^i_k = R(t^i_k) + P^i_k$$

$$B^i_k = P^i_k + MAX(F^i_{k-1}, R(t^i_k) - d^i)$$
find the packet whose B^i_k is minimal and send

簡單的說，FQ 每次都是選擇在佇列封包中最小 B^i_k 的封包來傳送，而 B^i_k 值的決定由封包長度 P^i_k、已完成總數 F^i_k 以及循環數 $R(t^i_k)$ 三者所控制。雖然 FQ 與我們的 LASSF 的時間複雜度，均爲 $O(log(active\ N))$[17]。但是，我們知道在 FQ 演算法中，若有一個封包 k 進入連線 i 時，就必須給此封包一個 B^i_k。因此，在時間 t 時，更新 B^i_k 所執行的次數爲：

$$Cost\_Bid\_Num(t) = Pkt\_in\_Q(t) + Pkt\_sent(t) + Pkt\_dropped(t) \quad \text{..... (1)}$$

(我們甚至忽略 $R(t^i_k)$ 的計算中，需包含：發送端平均速率、傳送出的頻寬與瞬間可動連線數的乘除運算)。

在 LASSF 中只有當封包要離開交換節點時，才作 $COUNTER_i$的更新，因此，更新的次數爲：

$$Cost\_Counter(t) = Pkt\_sent(t) \quad \text{..... (2)}$$

很明顯地，當佇列中有封包被儲存下來，或是有封包被丟掉時，(1)要比 (2) 花時間。而 LASSF 的演算法裡，只用加法。遠比 FQ 所使用的乘除法容易而且簡單。

3.2 LASSF 與 FQ 的效能評估

我們在 U.C. Bekerley 所發展出的模擬實驗軟體 REAL 4.0[19] 上作測試，實驗的模型 (scenarios) 有三個(見圖 5、6)，主要的拓樸圖形 (topology) 是四個起點1、2、3與 4，透過交換節點 SW 送封包到終點 D，其中交換節點的佇列大小爲 8KBytes，封包長度爲 128Bytes，所有起點至交換節點的頻寬均爲 19.2 Kbits/sec，而遲延(latency)爲 0.5ms。模型 1、2中交換節點至終點的頻寬爲 38.4 Kbits/sec 遲延是 1ms；在模型 3 中用的是22.4 Kbits/sec 與 1ms。

模型 1所架構四條起點均是採用 Poisson distribution 作爲封包間的間隔(packet interarrival time)控制，平均速度爲 16.8 Kbits/sec，同時向終點端傳送

資料。

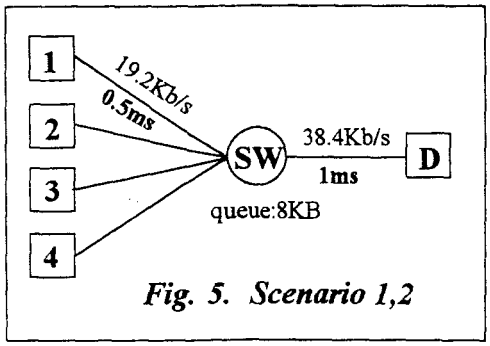

Fig. 5. Scenario 1,2

模型 2與模型 1不同的是，第四個起點採用 ON-OFF發送模型 (即在 ON 的區間用尖峰速率產生封包，而 OFF 的區間不產生任何封包)，ON 與 OFF 的間隔分別爲 15 秒及 20 秒，且尖峰速率我們定爲 18.4 Kbits/sec，其餘部分與模型 1相同。這個實驗主要是考量頻寬對突發資料 (bursty data) 的影響。

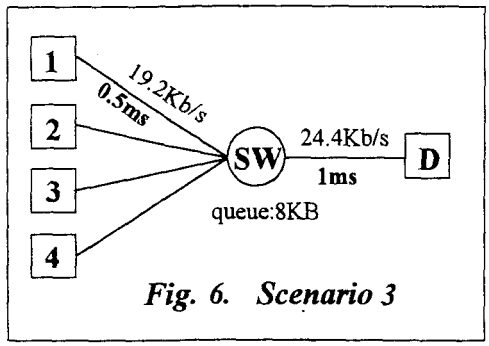

Fig. 6. Scenario 3

第三個實驗我們把重點放在新連結建立，以及連結結束後頻寬的分配情形；四個起點依舊使用 Poisson distribution，平均速率爲 19.2 Kbits/sec，而四點連結建立的時間分別是 0、15、25以及 40秒，其中第三個發送端只產生 1000個封包(如此一來可讓此連線提早結束)，其餘的均爲 5000個(封包個數不考慮重送處理)。

3.2.1 LASSF 與 FQ 的公平性

我們利用公平性方程式(Fairness equation)[20]來作計量, 方程式如下:

$$Fairness = \frac{(\sum Pkt\_sent\_on\_i)^2}{n\sum(Pkt\_sent\_on\_i^2)} \quad \text{.....(3}$$

其中*Pkt_sent_on_i* 是指個別的連線 *i* ，在每個單位測量時間內，共傳了多少封包。依據此式，我們得到圖 7、8和 9的結果。

在圖 7中，LASSF 與 FQ 的公平曲線 (fairness-line) 似乎是相同的，這表示在一般正常情況，兩種演算法的效果幾乎是一樣的。

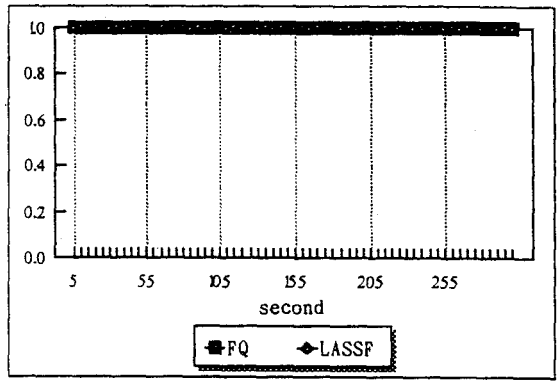

Fig. 7. Fairness Line in Scenario 1

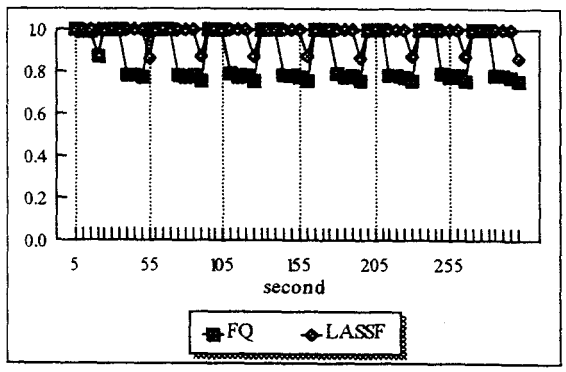

Fig. 8. Fairness Line in Scenario 2

在圖 8中，面臨 ON-OFF 發送模型時，FQ 對頻寬的分配就顯得不公平。而 LASSF 不僅公平曲線的震盪 (oscillation)比 FQ 小，而且不公平的周期也比較短。這說明 LASSF 對於頻寬的瞬間分配能力比 FQ 來的好，異常行爲的連線亦能立即被解決。

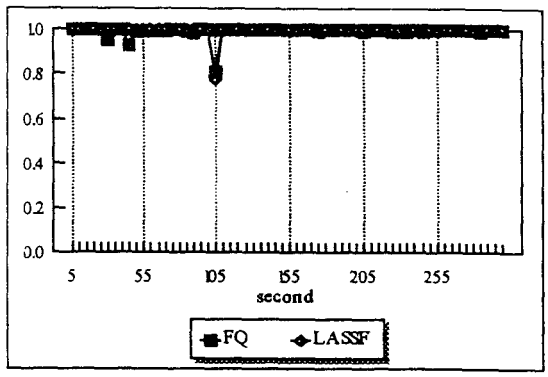

Fig. 9. Fairness Line in Scenario 3

在圖 9中，我們先說明在第 105 秒發生的震盪現象，是由於第三條連線提早結束，在它最後一次傳送時，封包個數遠比其他連線來的小而形成這種現象，但在其它時刻 LASSF 仍保持直線。反觀 FQ，遇到新連線建立時(如 25及 40秒)均有小幅震盪發生，所以可以發現 LASSF 對新連線加入時，較能立即平分頻寬給每一條連線。

3.2.2 LASSF 與 FQ 的整體平均延遲

在圖 10中，兩個演算法對於不同連線的整體平均佇列延遲幾乎是一樣的，再一次說明了在一般情況下 LASSF 的優點。

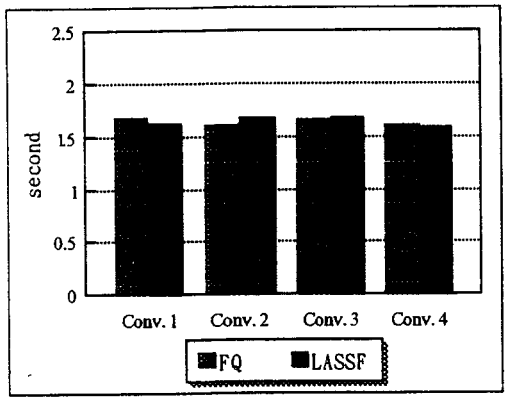

Fig. 10. Total Ave-Qing Delay in Scenario 1

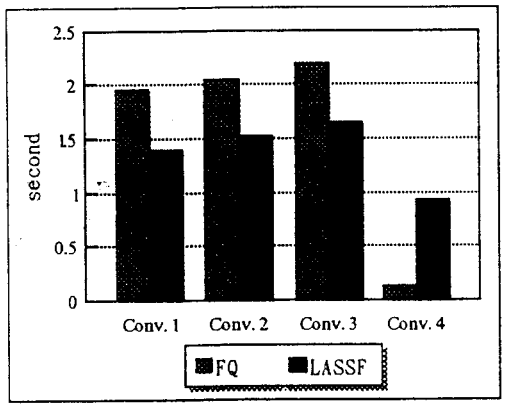

Fig. 11. Total Ave-Qing Delay in Scenario 2

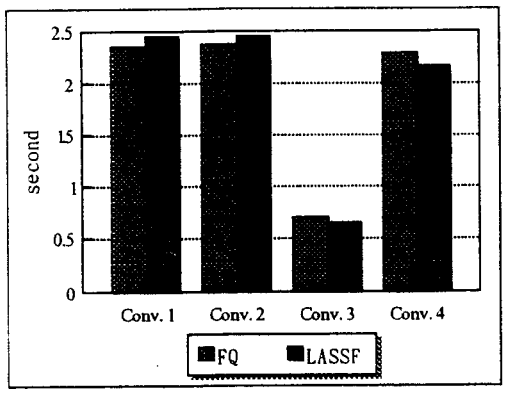

Fig. 12. Total Ave-Qing Delay in Scenario 3

在圖 11 中，我們留意第四條連線，LASSF 所產生的值似乎是比 FQ 來的大，主要的原因是，LASSF 爲了達成瞬間的公平性，對於在 ON/OFF 發送模型的 ON 時段裡，平均地將頻寬分給所有連線。而 FQ 則是在 ON 時段裡，彌補少傳的封包數，所以 LASSF 傳的封包個數比 FQ 來的少[12]，而增加了儲存在佇列的時間所致；反之，其他三條連線就比 FQ 來的低了。

在圖 12 中，值得注意的是第三條連線，因爲它提早結束，所以傳的封包就少，而使得整體平均佇列延遲相對的比其他連線低。而兩演算法的效果亦幾乎相同。

四、結論

在本文中，我們提出了一個新的佇列排程法：先窺式最短服務優先法。不像 FCFS 排程法，它可公平的分配網路上的資源，使得網路中的使用者，能避免不當行爲使用者及網路負載變動的干擾，達到最低限度服務的需求。

經由模擬比較的結果，顯示它在公平性及平均延遲時間的效能都不亞於公平排程法。除此之外，由於它實施上 (implementation) 比公平排程法簡單，在高速網路上更適合爲交換節點，採用作爲排程的方式。傳統的資料網路只提供單一型態的資料傳輸，未來的 BISDN 廣域網路，必須傳輸不同型態的資料，如語音 (voice)、影像 (image)和視訊 (video) 等。因此，這些網路必須提供服務品質的保証 (quality of service quaranteed)給不同型態的使用者，對於即時交通 (real-time traffic) 的保証尤其重要[1、9、10、11、13、14、16、18]。最近，Parekh[21]結合了 Fair Queueing 及 Leaky-bucket，能提供給即時交通效能上的保証 (performance guarantee)，然而，由於 Fair Queueing 實施的困難，尚未能符合實際的需求。未來，我們將朝此方向研究，提出一種方式來滿足即時交通使用者的需求。

參考文獻：

[1] C. Partridge, *Gigabit Networking* , Addison Wesley, 1993.

[2] V. Jacoboson, *Congestion Avoidance and Control*, In Proceedings of SIGCOMM'88, Aug. 1988.

[3] L. Kleinrock, *Queueing Systems Vol. II*, Wiley-Interscience, 1976.

[4] A. S. Tanenbaum, *Computer Networks*, 2nd ed. Englewood Cliffs, Prentice-Hall, 1988.

[5] S. P. Morgan, *Queueing disciplines and passive congestion control in byte-stream networks*, IEEE Trans. Comm. 1989.

[6] C. R. Kalmanek and H. Kanakia, *Rate Controlled Servers for Very High-Speed Networks*, IEEE Globecomm'90 ,pp.300.3.1-300.3.9., Dec. 1990.

[7] E. L. Hahne, *Round robin scheduling for fair flow control in data communication networks*, Lab. for Information and Decision Systems, MIT, Rep. LIDS-TH-1631, 1986.

[8] A. Greenberg and N. Madras, *How fair is fair queueing*? Proc. Performance'90, 1990.

[9] D. Ferrari, *Real-Time Communication in Packet Switching Wide-Area Networks*, Tech. Rep.. TR-89-022, International Computer Science Institute, Berkeley, May 1989.

[10] D. Ferrari and D. Verma, *A Scheme for Real-Time Channel Establishment in Wide Area Networks*, IEEE J. Selected Areas Comm., Nol. SAC-8 n.3, pp. 368-379, Apr. 1990.

[11] D. Verma, H. Zhang and D. Ferrari, *Delay Jitter Control for Real-Time Communication in a Packet Switching Network*, International Computer Science Institute, Berkeley, 1991.

[12] A. Demers, S. Keshav, and S. Shenker, *Analysis and simulation of a fair queueing algorithm*, In Proc. ACM SIGCOMM'89, pp 3-12. 1989.

[13] D. Vermas, H. Zhang, and D. Ferrari, *Guaranteeing delay jitter bounds in packet switching networks*, In Proc. of Tricomm'91, Apr. 1991.

[14] L. Zhang and S. Keshav, *Comparison of rate-based service disciplines*, In Proceedings of ACM SIGCOMM'91, pp 113-122, Zurich, Sep. 1991.

[15] L. Zhang, *Virtual Clock: A new traffic control algorithm for packet switching networks*, In Proceedings of ACM SIGCOMM'90, pp 19-29, Sep. 1990.

[16] L. Zhang. *A New Architecture for Packet Switched Network Protocols*, Ph.D. dissertation, MIT, Jul. 1989.

[17] S. Keshav, *On the efficient implementation of fair queueing*, 1991. To appear in Jour. of Internetworking Research and Experience.

[18] J. Kurose, *Open issues and challenges in providing quality of service guarantees in high-speed networks*, ACM Computer Comm. Review, 23(1):6-15 Jan. 1993.

[19] S. Keshav. *REAL: A network Simulator, Technical Report 8814723*, Department of Computer Science, UC Berkeley, 1988.

[20] R. Jain, D.M. Chiu, and W. Hawe, *A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Systems*, Digital Equipment Corporation, Technical Report TR-301, Septemper 1984.

[21] A.K. Parekh, R.G. Gallager, *A Generalized Processor Sharing Approach to Flow Control in Integrated Service Networks: The Single Node Case*, IEEE/ACM Transaction on Networking, Vol.1, No. 3, pp.344-357,Jun. 1993.