國立中山大學電機工程研究所

碩士論文

指導教授：李錫智　　博士

ATM 網路細胞排程

的技術研究

A Simple Cell Scheduling Mechanism

for ATM Network

研究生　李明志　撰

中華民國 八十九年 七月

摘要

在這一篇論文中，我們針對 Carry-Over Round Robin(CORR)排程法的一些缺陷進行改進。和 CORR 比較起來，修正後的方法降低了實作的複雜度，在分配頻寬上也更加的公平，在網路的延遲方面則是互有高低。整體而言，改進後的方法可說是精簡了 CORR 的演算流程，CORR 使用了某些設計來維持訊框的最大值，我們則證明了訊框最大值其實是可以不存在的，移除這些維持最大值的多餘步驟反而可讓排程法分配頻寬更加的公平。除此之外，我們也注意到排程法和 Shaper 各自獨立運作時，網路頻寬使用效率較低的問題，我們利用 CORR 原來將訊框分為 Major Cycle 及 Minor Cycle 的構想，設計將排程法內的一些資訊適時的回饋給 Shaper，然後在 Minor Cycle 中處理由 Shaper 傳輸過來的細胞，如此一來可望能在不影響排程法與 Shaper 的基本效能下，成功的提高網路頻寬使用率。

Abstract

In this thesis, we propose a cell scheduling mechanism to overcome some drawback of Carry-Over Round Robin (CORR) algorithm. Compare with CORR, the modified scheme reduces complexity of implementation and allocates bandwidth more fairly. In general, it simplifies CORR algorithm, which applies some design to maintain the maximum frame size. We prove that the maximum frame size is not necessary for deriving end-to-end delay. We also show that it results in fair distribution of bandwidth.

As long as the schedulers and traffic shapers work independently, significant underutilization is expected. In order to solve this problem, we borrow the concept of CORR which divides each allocation cycle into two subcycles—a major cycle and a minor cycle. By designing some information feedback to shapers, schedulers can transmit more cells in minor cycles. Hence we can improve bandwidth utilization successfully.

目錄

圖目錄

第一章 簡介

為了讓高速網路能支援多媒體應用程式，網路必須提供一些必要的服務品質保證(QoS)，排程法(Scheduling)就是在網路上負責這樣的一項工作，它會根據網路上各個連線不同的要求進行網路資源的分配，然後保證每個連線都能得到它所應得的服務品質。這幾年來，已有大量關於排程法的研究論文出現，整體而言，早期的研究強調的多半是效能，目標是在理論上能達到最佳化，但近來大家逐漸考慮到實作複雜度的問題，希望能在犧牲一小部分效能的情況下讓排程法能簡單的被實作出來，在本篇論文中我們就是試著去提出一個簡單的排程演算法。我們首先參考了許多的排程法的論文，接著我們選定將 CORR 作為我們主要參考的目標，其主要原因就是在於 CORR 的複雜度相對於其它排程法是比較低的，特別是 CORR 是被設計使用在 ATM 網路上，ATM 網路固定長度的細胞讓整個排程法結構更加的簡化，因此我們希望藉由改善 CORR 的過程中能得到一個非常實用的排程法。

本篇論文的組織架構如下：第二章概述排程法的研究背景，也一併將 CORR 做了詳盡的介紹及分析。第三章描述新的排程方法以及它如何修正 CORR 的一些缺失，內附排程法的詳細演算流程。第四章則是數學的分析，我們參照 CORR 所使用的數學模組推導排程法

內最重要的兩項指標：Fair Index 及 End to End Delay，並在這一章的最後一節與 CORR 做了詳細的分析比較。第五章是程式模擬的結果，模擬與 Shaper 結合後的排程法其效率會有何種程度的提升。最後第六章總結這篇論文。

第二章　排程法概論

2.1 排程法簡介和研究動機

多媒體時代的來臨，網路使用者對網路有了更多的需求，例如視訊及音訊都需要即時的傳送以便接收端能在時效內順利撥放，而讀取圖片則需要使用大量的頻寬，高速的區域網路於是在此需求下因應而生。而隨著高速網路來的問題是：我們需要建立一些有效的制度及方法來管理整個網路，並且保證服務品質能滿足所有使用者的要求。

排程(Scheduling)是網路管理制度中的一部分，它針對每個連線的不同特性及需求來安排網路封包的傳送法則，如果再配合上其他的網路管理技術，網路的整體運作就會井然有序且保持最佳效能。

衡量一個排程法的優劣主要有三項參考指標：Fair、End to End Delay 以及 Complexity。在此敘述它們的特性及在排程法中扮演的角色：

- **Fair**

傳統網路是採用所謂的 FCFS(First in First Out)的方法

來處理封包，這種方法在要求服務品質的網路上就顯得不太適用，特別是網路呈現擁塞情況時，有些連線可能會因為網路的擁塞而取得較多的頻寬，有些連線則取得較少的頻寬。因此排程法的第一目標就是希望無論網路負載是在何種情況下，每個連線都能公平的得到它們應有的頻寬。一個理想中的排程法會滿足下列式子

$$\frac{send_i(t_1,t_2)}{d_i} = \frac{send_j(t_1,t_2)}{d_j} \qquad \text{for any two connections } i,j \text{ during any interval } (t_1,t_2)$$

其中 $send_i(t_1,t_2)$表示連線 i 在 t_1 到 t_2 這段時間內實際傳輸的資料量；$send_j(t_1,t_2)$表示連線 j 在 t_1 到 t_2 這段時間內實際傳輸的資料量。 δ_i及 δ_j 分別為連線 i 及連線 j 所預設保留的頻寬。

然而封包的傳遞並非每個連線平行處理, 在僅有一個輸出通道的情況下封包傳遞勢必有個先後順序, 因此任何一個排程法只能儘量使封包的傳遞趨近於公平, 一般研究會使用下列式子來評斷一個排程法的公平性：

$$\textbf{Fair Index} = \max_{0 \le i < j \le n-1} \left\{ \left| \frac{send_i(t_1,t_2)}{d_i} - \frac{send_j(t_1,t_2)}{d_j} \right| \right\} \qquad \text{during any interval } (t_1,t_2)$$

Fair index 的值越小，即表示此排程法越公平。理想的排程法則會使 Fair index 等於 0。

● End to End Delay

延遲(delay)是網路分析裡最重要的一項數據，在排程法中延遲越大會造成網路中爆發式資料量(burst traffic)越大，使得網路 Buffer 的需求量增加，進而間接促使成本上升。同時網路的延遲越大也會增加網路的不可預測性，使得我們難以有效管理。然而如何正確的預估網路延遲是一項相當艱鉅的工作，在排程法的論文中一般較常見的是計算其可能的最大延遲(Worst case delay)，經過數學的假設限制，最大延遲將比較容易推導並可相互比較。

● Complexity

複雜度也是排程法中一項最重要的指標，許多研究在理論上都能夠達到相當良好的效果，但複雜度高的方法往往就會造成硬體實作上的困難，甚至會成為整個系統的瓶頸。即使現在的硬體科技突飛猛進，但考量到硬體成本的因素時，較簡單的排程法就能有效降低成本並具有實際效益。

若僅考量公平性及網路延遲，Packet-by-Packet Generalized Processor Sharing(PGPS)[2]是個最理想的演算法。PGPS 是對每一個連線在 buffer 裡待傳的第一個封包給予一個 Timestamp 的值，Timestamp 的值代表著這個連線在 buffer 裡的第一個封包應該被傳遞出去的時間。系統會比較每個連線的 Timestamp，依照由小到大的順序將其封包送出，每送出一個封包系統就重新計算該連線下個封包的 Timestamp。PGPS 不僅接近完美的分配頻寬同時也擁有最小的網路延遲，但它在計算 Timestamp 時牽涉到的複雜度卻是

$O(N)$，其中 N 表示當時 buffer 裡有封包待傳的連線的數目。當這種方法使用在高速網路時，成千上萬的連線 N 會使得 Timestamp 的計算量非常大，也就是說計算一次 Timestamp 的時間可能會相對的比較久，而系統必需等待 Timestamp 計算完畢才能繼續執行下一步驟，如此一來就會可能造成效能明顯的下降而形成系統的瓶頸！！

為了解決 PGPS 在計算上的缺陷，許多改良的演算法也相繼被提出[1]、[4]，它們成功的將 Timestamp 的計算複雜度降低至 $O(1)$ 或 $O(\log N)$，但是包含 PGPS 在內，這類有 Timestamp 計算的演算法其封包仍需按照 Timestamp 的大小依序排列才能提供服務，這使得它們在處理一個封包的的複雜度為 $O(\log N)$，如同 PGPS 在 Timestamp 計算上的困擾一般，當 N 值趨向極大時，這類的演算法在硬體實作上仍是相當困難的。

所以，我們將研究目標定為是一個簡單且複雜度為 $O(1)$的排程法。接下來的兩節我們簡單的介紹一下 ATM 網路以及應用在 ATM 網路上複雜度為 $O(1)$的 CORR 排程法。希望藉由改良 CORR 能得到一個有效率且易於實作的新排程法。

2.2 ATM 網路簡介

非同步傳輸模式(Asynchronous Transfer Mode)網路是目前最受重視的高速網路，近來其相關的技術已日漸成熟，許多骨幹網路的角色現在都是由 ATM 扮演重要的角色，以提供高速的傳輸量。我們簡介

ATM 網路的特性如下：

1. 多種傳輸速率
2. 傳送的封包為固定長度之細胞(cell)
3. 多種傳輸媒介
4. 累加型頻寬
5. 連線導向模式
6. 提供傳輸服務品質保證(即 QoS)
7. 提供多元化傳輸服務

2.3 CORR 排程法

CORR 是使用在 ATM 網路上的排程法，ATM 網路強調的正是服務品質保證，與排程法的目標是一致的。ATM 網路使用固定長度的細胞作為傳送資料的單位，目的就是為了網路在運作時能迅速的處理細胞，這對排程法的研究也是一大利多，因此排程法與 ATM 網路的結合將會是不可避免的趨勢。

CORR 如同一般複雜度為 $O(1)$的 Frame-based 排程法，將傳輸時間分成一個一個的單位—訊框(frame)，再依據各個連線的需要及訊框的大小，計算及分配每個連線在一個訊框內所能傳送的細胞數，和其它 Frame-based 的排程法不同的在於 CORR 予許每個連線在一個訊框中能傳送的值為一實數而非正整數，同時訊框的長度有一最大值的上限。有些排程法的訊框並沒有最大值，這樣會造成最大延遲隨著連線

數 N 不停的增長，以致於它的最大延遲難以求得或者不甚理想；有的排程法則使用固定長度的訊框，但這樣會造成網路頻寬使用效率降低；使用正整數來分配細胞數則會使得分配方法不夠彈性，有時也同樣會造成頻寬的浪費，特別是當訊框很小的時候，而 CORR 都克服了上述的這些問題，是個相當實用的排程法。圖 2.1 為 CORR 完整的演算法流程。

Carry-Over Round-Robin Scheduling

Constants

T : Cycle length.
R_i : Slots allocated to C_i

Variables

$\{C\}$: Set of all connections
t : Slots left in current cycle
n_i : Number of cells in C_i
r_i : Current slot allocation of C_i

Events

Initialize (C_i) /* Invoked at connection setup time */
add C_i to $\{C\}$;
$n_i \leftarrow 0$; $r_i \leftarrow 0$;

Enqueue ( ) /*Invoked at cell arrival time */
$n_i = n_i + 1$
add cell to connection queue ;

Dispatch ( ) /* Invoked at the beginning of a busy period */

$\forall C_i :: r_i \leftarrow 0$;
while not end of busy period **do**
t $\leftarrow$ T ;
1. Major Cycle :
For all $C_i \in \{C\}$ **do** /* From head to tail */
$r_i \leftarrow \min(n_i, r_i + R_i)$; $x_i \leftarrow \min(t, \lfloor r_i \rfloor)$;
$t \leftarrow t - x_i$; $r_i \leftarrow r_i - x_i$; $n_i \leftarrow n_i - x_i$;
dispatch x_i cells from connection queue C_i ;
end for
2. Minor Cycle :
For all $C_i \in \{C\}$ **do** /* From head to tail */
$x_i \leftarrow \min(t, \lceil r_i \rceil)$;
$t \leftarrow t - x_i$; $r_i \leftarrow r_i - x_i$; $n_i \leftarrow n_i - x_i$;
dispatch x_i cells from connection queue C_i ;
end for

end while

圖 2.1 CORR**演算法**

CORR 排程法分為三個事件：Initialize、Enqueue 與 Dispatch。Initialize 事件發生於有新連線加入時，CORR 將此連線加入連線的清單內。這個清單是按照每個連線所分配的 R_i 的小數部分由大到小排列，這樣一來就能保證小數部分較大的連線會較早被傳送封包，這個設計已在[9]中被證明能有效控制 CORR 的最大 Frame 長度為一定值，同時這樣的一個過程並不影響 CORR 在處理單一封包時的複雜度為 $O(1)$。Enqueue 事件發生於 ATM 細胞到達時，系統將細胞放入佇列(queue)中等待傳遞並將 n_i 值加 1，n_i 表示在佇列中等待傳送的細胞的個數。Dispatch 事件則是整個排程法的運作核心，只要任何連線的佇列中有細胞等待傳送，Dispatch 就會不停的將它們傳送出去。

CORR 將每個 Frame 分為 Major Cycle 及 Minor Cycle，每個連線在單一 Frame 中所應被傳遞的細胞其整數部分會在 Major Cycle 中獲得服務，剩下的小數部分則留至 Minor Cycle 中處理。很顯然的，系統無法將單一細胞拆成好幾塊來滿足各個連線的小數部分，但又不能讓每個連線都得到一個完整的細胞，否則傳送的細胞總數將可能超過一個訊框所應有的細胞總數，所以某些連線勢必得先獲得完整的一個細胞。CORR 於是設計在 Minor Cycle 中有些連線可以先“借”細胞，而在下個訊框中償還，沒有借到細胞的連線則在之後的訊框獲得補償。接下來我們用個簡單的例子來觀看整個 CORR 的運作流程。

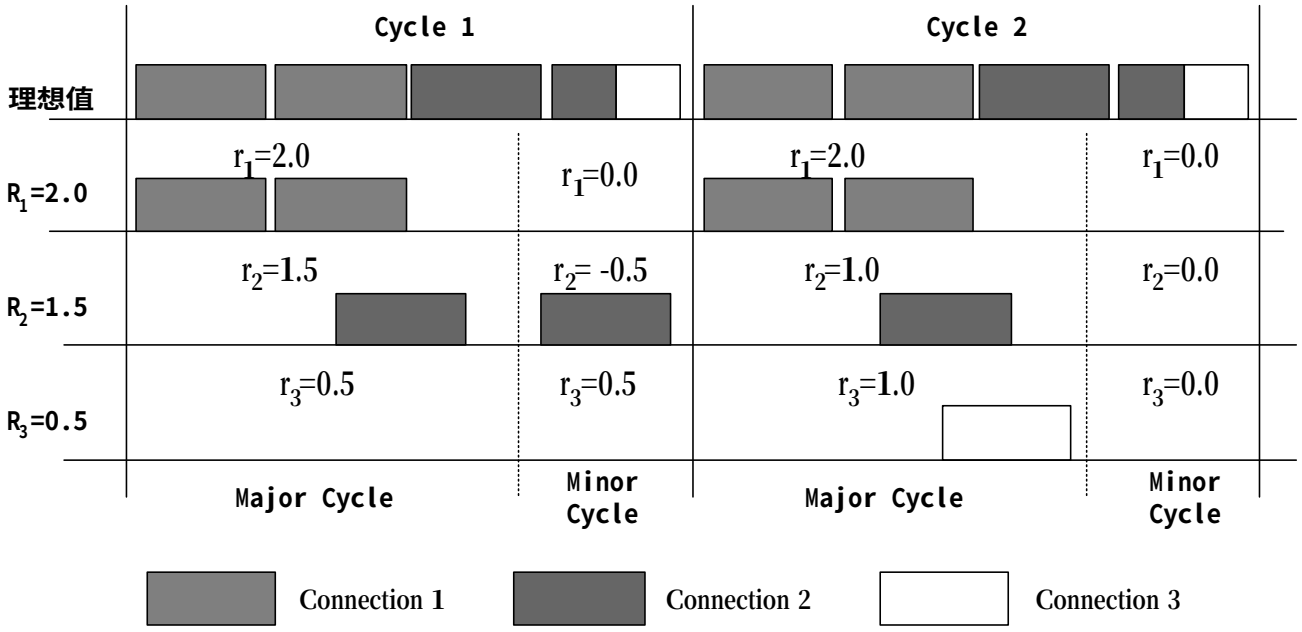

圖 2.2 CORR**範例**

圖 2.2 **為一** CORR **的範例。我們假設系統剛開始運作，**Frame **預設的長度** T **為四個細胞的時間，正好和各連線所預設保留的細胞數** R_i **總合相等，這樣我們可以期待網路的使用效率為** 100%**，且** CORR **的** Frame size **會達到最大值** 4**。**R_1**、**R_2**、**R_3 **分別是連線** 1**、**2**、**3 **所預設保留的細胞數。**r_1**、**r_2**、**r_3 **則是連線** 1**、**2**、**3 **計數器的值，在每個訊框開始後代表著在這個訊框中各個連線分別該送出多少細胞。**

在第一個訊框的 Major Cycle **中，各連線計數器** r_i **的整數部分都獲得了服務，連線** 1 **得到了** 2 **個細胞，連線** 2 **得到** 1 **個細胞，連線** 3 **則沒有得到任何細胞。接著在** Minor Cycle **中對小數部分進行"借細胞"的工作，連線** 2 **借得了半個細胞所以能夠送出一個完整的細胞，所以第一個訊框結束後，各連線計數器的值分別為** $r_1 = 0.0$**、**$r_2 = -0.5$**、**$r_3 = 0.5$**。進入第二個訊框後則變成**

$$r_1 = r_1 + R_1 = 0.0 + 2.0 = 2.0$$

$r_2 = r_2 + R_2 = -0.5 + 1.5 = 1.0$

$r_3 = r_3 + R_3 = 0.5 + 0.5 = 1.0$

由於全部計數器的值均成為正整數，第二個訊框中各連線都會在 Major Cycle **中得到應有的服務，**r_1**、**r_2 **及** r_3 **成為** 0**，所以** Minor Cycle **無需再運作。所有的連線在第二個訊框結束時都得到理想中的服務。**

2.4 CORR 的優缺點

CORR **的最大優點在於它的易於實作，不僅僅是在處理單一細胞上的複雜度為** $O(1)$**，同時整個運作過程也相當簡單明瞭。在效能方面，**CORR **能有效的使用並公平分配頻寬；加上** Leaky Bucket **後藉由其網路特性，**CORR **也能在數學上導出最大延遲，雖然因為數學模形和定義的關係，**CORR **無法和使用** Timestamp **的排程法直接作延遲的比較，但相較於複雜度為** $O(1)$**的其它排程法，**CORR **是其中表現最傑出的一個。**

然而 CORR **雖然號稱複雜度為** $O(1)$**，為了保證其訊框有一最大值，某些設計減低了它的效能並增加了複雜度，最明顯的就是它的連線必需依照每個連線** R_i **值的小數部分作排列。比起那些** Timestamp **的排列，這種將新連線作排列的過程是比較不會成為系統的瓶頸而且可以在硬體上與細胞傳送的工作進行平行處理，但當高速網路上連線數目** N **極大時，任何對** N **的搜尋、比較、或者排序都可能使得系統**

效能嚴重降低或者付出高昂的成本。此外，同樣是為了使訊框有最大值，CORR 設計讓計數器成為一個可大於-1 的值，也就是之前提過的借細胞或借頻寬，因為有些連線能借而有些不能借，這使得 CORR 在公平性方面仍有改進的空間。

於是我們有了這樣的一個構想：也許我們能解決訊框最大值的問題，或者在無訊框最大值的情況下找出點對點的最大延遲，就可移除新連線加入時必需排序的步驟，並改善”借頻寬”所帶來的問題，只要修正後的方法其最大延遲與 CORR 相去不多，這樣的修正就算是成功的。而 CORR 的 Minor Cycle 在修正後就無存在的必要，移除後整個流程可把程序簡化一半左右，這讓排程法在實作時，硬體計算上的延遲也可預期較 CORR 減少一半。

第三章　我們提出的排程方法

3.1 修正 CORR

我們在第二章最後解釋了 CORR 的兩個缺陷，在第三章開始對 CORR 進行改良，修正後的演算法如圖 3.1 所示。和 CORR 相比主要有三點不同：

1. 新連線加入時無需進行排列
2. 沒有了 Minor cycle，所以計數器值大於或等於 0，而不是如 CORR 大於或等於 –1
3. 訊框大小無上限值

第 3 點是由於第 1、2 點的改變而來。CORR 因為訊框有最大值，就可以用這個值來計算最大延遲 (需假設 ATM 在 Call Admission Control 的控制下，R_i 的總合不超過 T)，方法是先求出一個連線可能產生的最大延遲的時間相當於幾個訊框數，再將其乘上訊框最大值便可得到答案。直覺上，修正後的方法其訊框若無最大值，訊框大小會

隨著連線的數目 N 增加而不停澎脹, 所以最大延遲也因此無限增加導致無法求出。但實際上新的方法在運作上並未比 CORR 浪費頻寬，在網路使用效率和 CORR 相等的條件下，我們認為最大延遲不應該有過大的差別，甚至該預期有個相近的結果，所以我們先用個例子來觀察：

Modified Algorithms

Constants

R_i : Slots allocated to C_i

Variables

$\{C\}$: Set of all connections

n_i : Number of cells in C_i

r_i : Current slot allocation of C_i

Events

Initialize(C_i) /* Invoked at connection setup time */

add C_i to $\{C\}$;

$n_i \leftarrow 0$; $r_i \leftarrow 0$;

Enqueue ( ) /*Invoked at cell arrival time */

$n_i = n_i+1$

add cell to connection queue ;

Dispatch ( ) /* Invoked at the beginning of a system busy period */

$\forall C_i :: r_i \leftarrow 0$;

while not end-of-busy period **do**

for all $C_i \in \{C\}$ **do** /* From head to tail */

$r_i \leftarrow \min(n_i, r_i+R_i)$;

$r_i \leftarrow r_i - \lfloor r_i \rfloor$; $n_i \leftarrow n_i - \lfloor r_i \rfloor$;

dispatch $\lfloor r_i \rfloor$ cells from connection queue C_i ;

end for

end while

圖 3.1 修正後的演算法

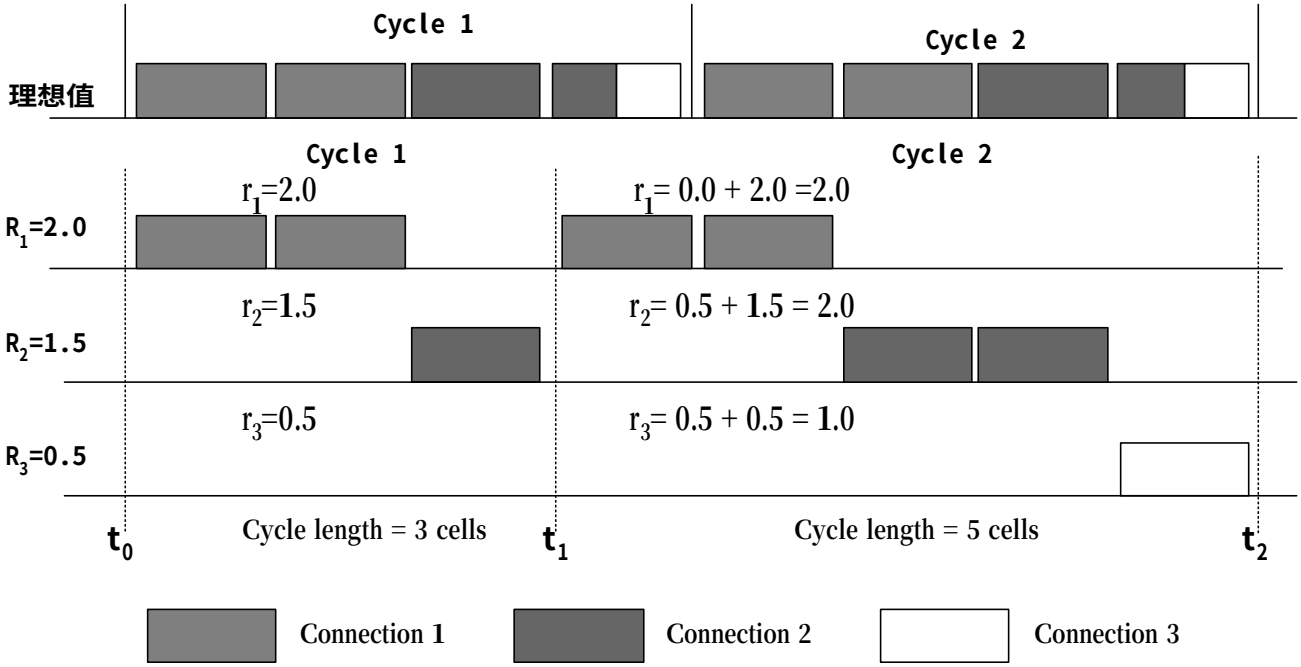

圖3.2 **修正後的演算法範例**

我們使用跟 CORR **相同的例子。如圖** 3.2 **所示，**T **是** 4 **個細胞，**R_1**、**R_2**、**R_3 **分別是** 2.0**、** 1.5 **及** 0.5**。**t_0 **到** t_2 **這段時間的結果和** CORR **是相同的，**CORR **的兩個訊框都是** 4 **個細胞的長度，而新的方法則是** 3 **和** 5 **個細胞，但總長度兩者同為** 8 **個細胞。我們接著觀察各連線** r_i **值的變化情形；時間為** t_0 **時每個連線的** r_i **值均為** 0**，到圖中的** t_2 **時，兩個訊框應該傳送的總細胞數為兩倍的** R_i**，其值為** 8 **個細胞，實際經過兩個訊框後得到的服務亦為** 8 **個細胞；時間為** t_1 **時連線** 1**、**2**、**3 **的** r_i **值分別為** 0.0**、** 0.5**、** 0.5**，到** t_2 **時應傳送** 4 **個細胞，實際傳送為** 5 **個細胞。**

我們將觀察得到的情形作出如下的結論

1. **若不考慮每個連線的** r_i **的初始值(或開始時** r_i **值均為** 0)**時，新排程**

法的任一個訊框結束的時間必小於理想中的時間(如圖 3.3 所示)，即

$$\sum_{k=1}^{n} C_k \le \sum_{k=1}^{n} T = nT \quad \text{----------(1)}$$

C_k 為新排程法中訊框 k 的長度，T 為系統預設的訊框長度。以圖 3.2 的例子而言，t_0 為觀察點時 r_i 值都為 0，到 t_1 時傳了 3 個細胞，小於理想中的 1 ×4 = 4 個細胞；到 t_2 時共傳遞了 3 + 5 = 8 個細胞，等於理想值的 2 ×4 = 8 個細胞。

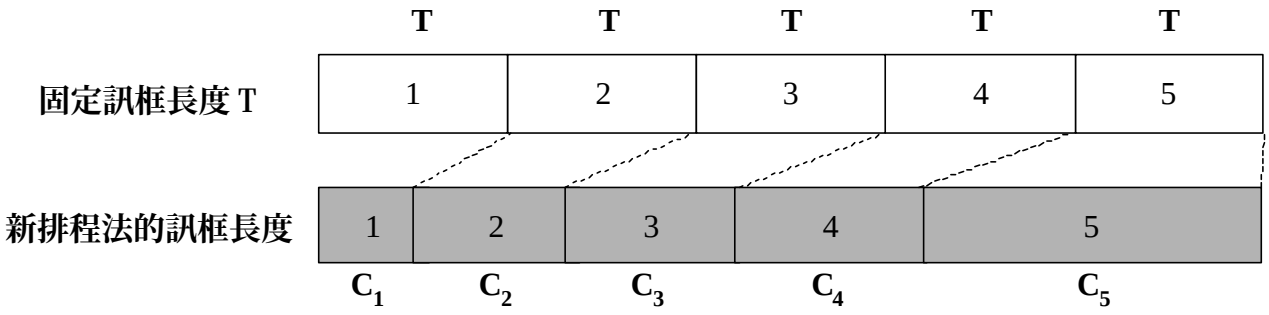

圖3.3 訊框長度圖

2. 考慮初始的 r_i 時，訊框總合的長度小於等於理想中的長度再加上各 r_i 的總合，即

$$\sum_{k=1}^{n} C_k \le nT + \sum_{i=1}^{N} r_i \quad \text{----------(2)}$$

以圖 3.2 而言，t_1 為觀察點時 r_1、r_2、r_3 分別為 0.0、0.5、0.5，新排程法的第二個訊框使用了 5 個細胞正好等於 4 + (0.0 + 0.5 + 0.5) = 5。

這樣的兩個式子是相當有幫助的，如此一來我們就能比照 CORR 將各個訊框視為有一最大值 T 來求得一個連線的最大延遲，僅需多加

一項 r_i 的初始值總合，式子(2)的證明請參閱附錄 1。

至止我們提供了解決訊框最大值問題的一個方案，移除了 CORR 在新連線加入時需進行排列的缺陷，此外還一併省略掉 Minor Cycle 及它的借頻寬步驟，這使得新的排程法在操作的步驟上較 CORR 更為簡單而容易實作，同時我們在第四章還會證明少了借頻寬的過程，新排程法在公平性方面也較 CORR 來的優異。

3.2 排程法結合 Shaper

許多的排程法在推導最大延遲時, 都假設細胞在進入網路前需先經過 Shaper 的調整，CORR 也是如此。我們在此簡介 CORR 所使用的 Shaper ― Leaky Bucket

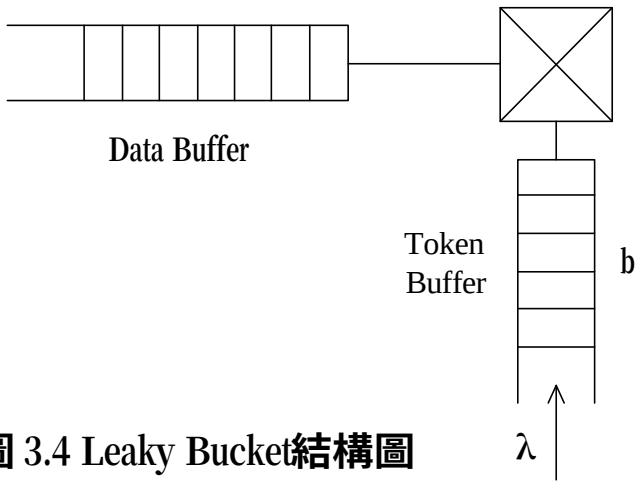

圖 3.4 Leaky Bucket結構圖

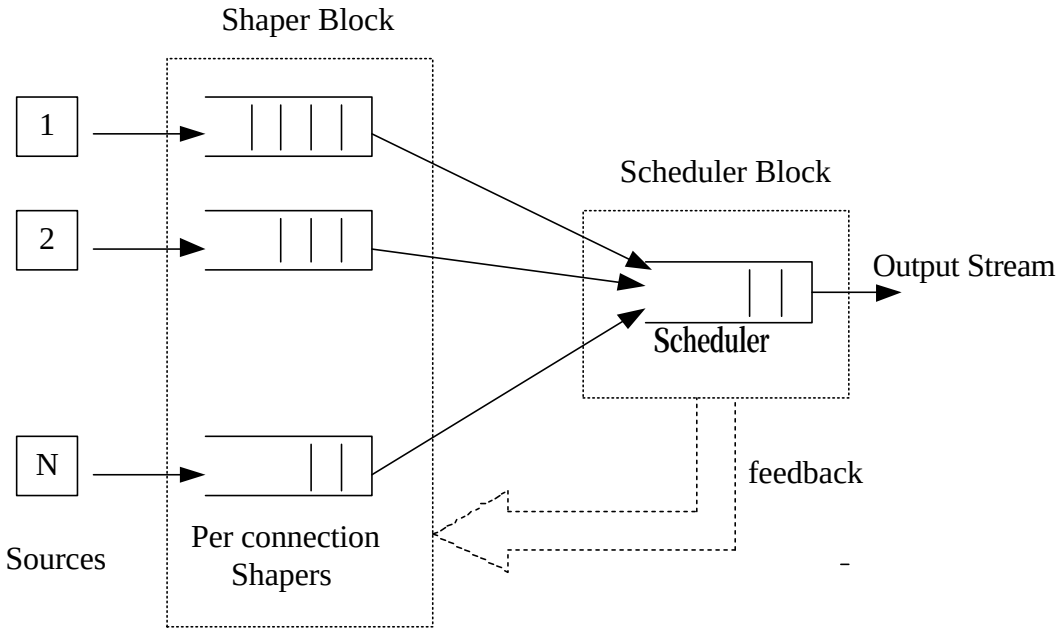

圖 3.5 Scheduler**結合**Shaper**架構圖**

Leaky Bucket **的用途是用來重整網路細胞的排列狀況。如圖** 3.4 **所示，細胞到達** Leaky Bucket **時先儲存於** Data Buffer **中，再視** Token Buffer **中的訊標**(token)**數來決定細胞能否傳送。** λ **是訊標產生的週期時間，每一個訊標只能予許一個細胞使用，使用後訊標自動消失，若細胞在** Data Buffer **中時發現** Token Buffer **裡沒有訊標，細胞就必需等待新的訊標產生後才能傳送出去。** b **是訊標的上限值，或者說是** Token Buffer **的容量，當** Token buffer **中的訊標數到達** b **時，就不會有新的訊標產生。**

如上述，Leaky Bucket **是用來重整網路細胞的排列狀況，通常使用在細胞正式進入網路前，一方面避免短期內大量的細胞進入網路，進而造成網路的擁塞；一方面可使爆發式資料的細胞進入網路時變得比較和緩。**

一般而言，排程法藉由 Leaky Bucket 調整及限制的特性來推導最大延遲，但兩者各自獨立運作，這樣的一個結構其實是有點瑕疵的。如果我們能把它們如圖 3.5 般的結合，效率應該會更好才是。

舉例而言，CORR 是一個網路頻寬能充分利用的排程法，只要有細胞堆積在它的 buffer 裡(圖 3.5 的 scheduler 處)且有足夠網路頻寬時，它就能適時的將其送出。但今天可能發生的情形是，有細胞堆積在 Leaky Bucket 的 buffer 裡(圖 3.5 的 Shaper 處)等待訊標的到來，卻沒有細胞堆積在 CORR 的 buffer 裡，因為兩者是獨立運作的，所以 Leaky Bucket 並不知道 CORR 那裡有充分的頻寬可送細胞，結果這樣就造成了網路頻寬的浪費！！

因此我們使用圖 3.5 的架構，嘗試將排程法的一些資訊適當的 feedback 給 Leaky Bucket，利用 CORR 裡 Major Cycle 與 Minor Cycle 的構想，在 Major Cycle 中傳送一般的細胞，在 Minor Cycle 中傳送由 Leaky Bucket 額外傳來的細胞，以提高網路的使用率，其演算法如圖 3.6 所示。

許多排程法雖然是和 Leaky Bucket 共同使用，但卻未對 Leaky Bucket 做 feedback 的動作，所以網路使用效率上較低是可以預期的，我們利用原來 CORR 的 Minor Cycle 作 feedback，所增加的操作步驟並不會超出 CORR，但卻可有效的結合 Leaky Bucket，而且由於這是利用多餘的頻寬來作 feedback，其過程並不會對最大延遲或公平性產生影響。在第五章我們會用模擬來檢驗這樣的一項設計能確實提高網路頻寬的使用效率。

Modified Algorithms With Leaky Bucket

Constants

R_i : Slots allocated to C_i

Variables

$\{C\}$: Set of all connections

n_i : Number of cells in scheduler of C_i

b_i : Number of cells in Leaky Bucket of C_i

r_i : Current slot allocation of C_i

Events

Initialize (C_i) /* Invoked at connection setup time */

add C_i to $\{C\}$;

$n_i \leftarrow 0$; $r_i \leftarrow 0$;

Enqueue ( ) /*Invoked at cell arrival time */

$n_i = n_i + 1$

add cell to connection queue ;

Dispatch ( ) /* Invoked at the beginning of a system busy period */

$\forall C_i :: r_i \leftarrow 0$;

while not end-of-busy period **do**

1. Major cycle

for all $C_i \in \{C\}$ **do** /* From head to tail */

$r_i \leftarrow r_i + R_i$; $x_i \leftarrow \min(n_i, \lfloor r_i \rfloor)$;

$r_i \leftarrow r_i - x_i$; $n_i \leftarrow n_i - x_i$;

Feedback ($\lfloor r_i \rfloor$) ;

dispatch x_i cells from connection queue C_i ;

end for

2. Minor Cycle :

for all $C_i \in \{C\}$ **do** /* From head to tail */

$r_i \leftarrow \min(n_i, r_i)$;

$r_i \leftarrow r_i - \lfloor r_i \rfloor$; $n_i \leftarrow n_i - \lfloor r_i \rfloor$;

dispatch $\lfloor r_i \rfloor$ cells from connection queue C_i ;

end while

Feedback (r_i) /* on the Leaky Bucket */

$x_i \leftarrow \min(b_i, r_i)$;

$b_i \leftarrow b_i - x_i$;

dispatch x_i cells from Shaper queue ;

圖 3.6 演算法結合 Leaky Bucket

第四章　數學分析

4.1 Fairness Property

首先我們對新排程法的公平性進行分析，依據第二章的定義，我們必需求出在任何一段時間內(t_1 至 t_2)任意兩個連線之間的最大差值，為求計算方便，我們利用任兩個訊框(c_1 至 c_2)來表示任意的一段時間。假設連線都進入了忙碌週期，所謂的忙碌週期表示此連線的 buffer 內一直有細胞在等待傳送，一直到 buffer 空了這段忙碌週期才算結束，所以任意兩個訊框 c_1 到 c_2 之間所傳送的細胞數應當為

$$send_i(c_1,c_2)=\lfloor (c_2-c_1)R_i+r_i \rfloor \quad \text{---------- (3)}$$

r_i 即是進入訊框 c_1 時，計數器的初始值，注意它必需是個小於 1 的正實數。我們將上式如第二章定義般的除以連線 i 的預設頻寬，即

$$W_i(c_i,c_2)=\frac{send(c_1,c_2)}{R_i}=\frac{\lfloor (c_2-c_1)R_i+r_i \rfloor}{R_i} \quad \text{---------- (4)}$$

接著求取任意兩個連線 i , j 之間絕對值的差，

$$\left|W_i(c_1,c_2)-W_j(c_1,c_2)\right|=\left|\frac{\lfloor(c_2-c_1)R_i+r_i\rfloor}{R_i}-\frac{\lfloor(c_2-c_1)R_j+r_j\rfloor}{R_j}\right| \text{ ---------- (5)}$$

拆除 ⌊ ⌋ 符號，我們可加入一個介於 0、1 之間變數 x

$$\left|W_i(c_2-c_1)-W_j(c_2-c_1)\right|=\left|\frac{(c_2-c_1)R_i+r_i-x_i}{R_i}-\frac{(c_2-c_1)R_j+r_j-x_j}{R_j}\right| \text{ ----- (6)}$$

接著消除 c_2 及 c_1

$$\left|W_i(c_2-c_1)-W_j(c_2-c_1)\right|=\left|\frac{r_i-x_i}{R_i}-\frac{r_j-x_j}{R_j}\right| \text{ ---------- (7)}$$

r_i、r_j、x_i、x_j 都是 0 至 1 之間的實數，所以就最壞的情況而言

$$\left|W_i(c_2-c_1)-W_j(c_2-c_1)\right|\le\frac{1}{R_i}+\frac{1}{R_j} \text{ ---------- (8)}$$

新的排程法的 Fair Index **即為上述式子，如同我們預期的，這個值和** CORR **比較起來大約只有** CORR **的一半左右，因為** Fair Index **是越小越公平，所以在數學分析上我們驗證了修正後的排程法在公平性方面是超越了** CORR**。**

4.2 Delay in Single node

一般求取延遲的方法是，將 Shaper 置於連線細胞進入網路之前，經過 Shaper 後的網路細胞具有某些規律性，利用這樣子的規律性，在數學上我們就能得到最大延遲。因為每種排程法的結構及假設不同，不同的排程法要在延遲的數學式上相互比較有其困難之處，我們在此是採用和 CORR 相同的方法來推導最大延遲，這樣我們就能在數學分析之後與 CORR 作一番比較，4.2 與 4.3 節中所使用到相關的定理及證明請參閱附錄[6]、[9]及[13]。

首先我們要先介紹 CORR 所使用的混合式 Leaky Bucket，它將 *n* 個 Leaky Bucket 串連在一起，如圖 4.1 所示。

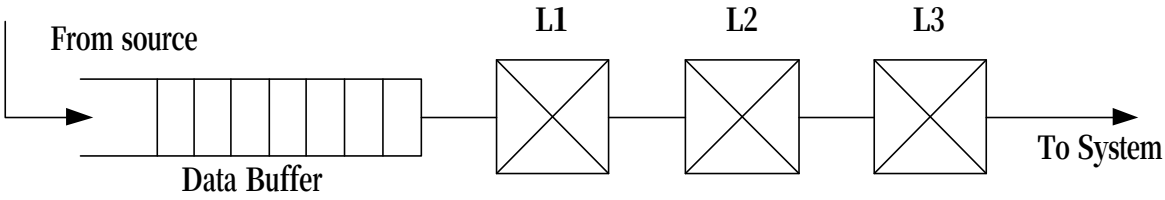

圖 4.1 Composite Leaky Bucke結構圖

混合式的 Leaky Bucket 的 date buffer 仍然只有一個，但每個細胞需要每個 Leaky Bucket 都至少有一個訊標才能順利傳遞出去。混合式的 Leaky Bucket 有著一些特性，如圖 4.1 所示我們由前至後將 Leaky Bucket 編號為 L1、L2、L3 等等，那麼一個混合式 Leaky Bucket 會滿足 $b_1 > b_2 > b_3 > \cdots > b_n$ 和 $\lambda_1 > \lambda_2 > \lambda_3 > \cdots > \lambda_n$。這樣的設計能夠使得一連串爆發式的細胞在通過混合式的 Leaky Bucket 後細

胞的排列情況更加的和緩。有關組合式 Leaky Bucket 的其他資料在[13]中有詳盡的描述。

我們要推導的最大延遲即是細胞離開排程法所在節點的時間和細胞進入節點的時間的差，若假設傳輸時間很小可忽略不計，那麼進入節點 buffer 的時刻也就可視為是離開 Leaky Bucket 的時刻。現在我們考慮第 i 個細胞離開混合式 Leaky Bucket 的時間，我們將它表示成 $a(i)$，$a(i)$在[13]已經被證明可表示成下列式子

$$a(i)=\sum_{k=1}^{n+1}(i-b_k+1)\,l_k[U(i-B_k)-U(i-B_{k-1})],\quad i=0,1,...,\infty \quad \text{------ (9)}$$

其中

$$B_k=\begin{cases}\infty & k=0,\\ \left\lfloor\dfrac{b_k l_k-b_{k+1}l_{k+1}}{l_k-l_{k+1}}\right\rfloor & k=1,2,...,n,\\ 0 & k=n+1,\end{cases}$$

$$U(x)=\begin{cases}0 & x<0\\ 1 & x\geq 0\end{cases}$$

另外注意 B_k 具有這樣的特性：$B_1>B_2>B_3>\cdots>B_n$。

接下來的工作就是推導第 i 個細胞被排程法傳送出去的時間，首先假設某個連線在時間 $t=0$ 時進入了忙碌週期，因為最大延遲必然出現在網路高負載時，此時我們推導時都假設連線長期處於忙錄週期中。

我們用 $d(i)$來表示第 i 個細胞在這段忙碌週期內離開系統的時間；要想確實的得知此細胞離開系統的時間是相當困難的，幸好我們要的是最大延遲，也就是最壞的情況，所以我們可以假設細胞是在某

一個訊框的最後一刻，也就是這個訊框結束時離開系統。

如果第 i 個細胞是在第 L 個訊框結束時離開系統的，那麼在這段忙碌週期的時間內，系統所被分配到的細胞數應為 $L \times R + r$，r 是第一個訊框開始時計數器的初始值. 我們知道 r 是介於 0 到 1 之間的數, 那麼就算壞的情況而言，r 應該要趨近於 0。舉個例子來說，若某連線某個忙碌週期內的第一個細胞進入 buffer 時 $r = 0$、$R = 0.5$，進入後 $r = 0 + 0.5 = 0.5$，此細胞要在第二個訊框才被送出；若進入 buffer 時發現的是 $r = 0.9$、$R = 0.5$，此細胞在第一個訊框中就會被送出， r 值越小表示細胞可能會被延遲較久才被送出, 所以我們為因應最壞情況假設 $r = 0$。接下來考慮在第 L 個訊框第 i 個細胞被送出的條件為 $L \times R - i \geq 1$ (細胞從 0 開始算起)，利用此不等式可得到

$$L \geq \frac{1+i}{R} \quad \text{----------} \quad (10)$$

根據第三章所提到的, 我們可把延遲視為連線所需的訊框數 L 乘以訊框理想中的大小 T 再加上網路的初始值 r 總合(假設 R_i 的總合小於等於 T 的情況下)，另外考慮 L 是個正整數這項條件，我們可得到

$$d(i) = \frac{l}{C}\left\lceil \frac{1+i}{R} \right\rceil T + \frac{l}{C}\sum_{k=1}^{N} r_k \quad \text{----------} \quad (11)$$

C 是細胞輸出通道的頻寬(單位為 bits/s); l 則是細胞的長度，在 ATM 中即為 48 bytes。

有了進入系統的時間 $a(i)$及離開系統的時間 $d(i)$, 第 i 個細胞延遲的時間就可由 $d(i) - a(i)$來獲得, 但究竟那一個細胞會造成最大延遲則

要仰賴組合式 Leaky Bucket **的數學模組，在**[6]**中已被證明組合式** Leaky Bucket **置於系統前時，當下列情況成立時，最大延遲會出現在第** B_j **個細胞**

$$\frac{1}{l_j} < \frac{R}{T} < \frac{1}{l_{j+1}}$$

於是 $a(B_j)$**可計算如下**

$$\begin{aligned} a(B_j) &= \sum_{k=1}^{n+1} (B_j - b_k + 1)\, l_k \left[U(B_j - B_k) - U(B_j - B_{k-1})\right] \\ &= \sum_{k=1}^{j-1} (B_j - b_k + 1)\, l_k \left[U(B_j - B_k) - U(B_j - B_{k-1})\right] \\ &\quad + (B_j - b_j + 1)\, l_j \left[U(B_j - B_j) - U(B_j - B_{j-1})\right] \\ &\quad + \sum_{k=j+1}^{n+1} (B_j - b_k + 1)\, l_k \left[U(B_j - B_k) - U(B_j - B_{k-1})\right] \end{aligned}$$

根據之前提到的

$$B_1 > B_2 > B_3 > \cdots > B_n \qquad\qquad U(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases}$$

我們求得

$$a(B_j) = (B_j - b_j + 1)\, l_j \text{ ---------- (12)}$$

和

$$\max\, delay \leq d(B_j) - a(B_j) = \frac{l}{C}\left\lceil \frac{1 + B_j}{R} \right\rceil T - (B_j + b_j + 1)\, l_j + \frac{l}{C}\sum_{k=1}^{N} r_k \text{ --------- (13)}$$

when

$$\frac{1}{l_j} < \frac{R}{T} < \frac{1}{l_{j+1}}$$

4.3 Delay in Multinode

上一節我們推導的是單一網路節點上的最大延遲，這一節我們介紹一種數學模組可用來表示一個連線經過多個網路節點的最大延遲。這種表示方法並不只限於某些特定的排程法，並且可以描述整個多節點網路上的延遲情況。首先我們定義一些會使用到的表示符號：

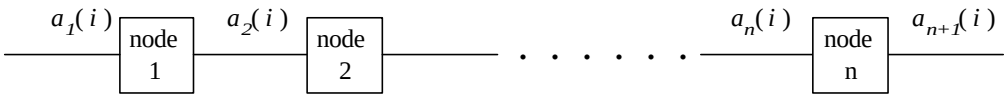

圖4.2 多節點網路圖

考慮一個連線在傳送和接收的兩個端點之間經過了 n 個網路節點，我們用 $a_k(i)$來表示此連線的第 i 個細胞抵達第 k 個節點的時間(參考圖 4.2)，用 $s_k(i)$來表示第 i 個細胞在第 k 個節點所遭遇到的延遲時間，$S_k(p,q)$表示細胞 p 到細胞 q 之間的延遲總合，即 $s_k(p) + s_k(p+1) + \cdots + s_k(q-1) + s_k(q)$ 。假設節點之間的傳輸延遲(propagation delay)很小，則第 i 個細胞離開第 k 個節點的時間即為到達第 $k+1$ 個節點的時間 $a_{k+1}(i)$，如此一來 $a_1(i)$就是細胞 i 到達節點 1 的時間，也是此細胞進入網路的時間、$a_{n+1}(i)$就是此細胞離開節點 n 的時間，同時也是此細胞離開系統的時間。

為了充分了解這個數學模式的結構，我們先以一個 2 節點的網路來作個說明。如果我們現在想要計算 $a_3(2)$，必需考慮下列幾種情況

情況 1： 細胞 2 到達節點 1 及節點 2 時, 細胞 1 都已經不在 buffer 中、也不在被傳遞的過程中。所以細胞 2 從進入網路到離開網路，中

途沒有任何其它細胞所造成的延遲，因此細胞二在節點 1 及節點二所花的時間分別是 $s_1(2)$及 $s_2(2)$。這樣一來，$a_3(2) = a_1(2) + s_1(2) + s_2(2) = a_1(2) + S_1(2,2) + S_2(2,2)$。

情況 2： 細胞 2 到達節點 1 時細胞 1 已經離開，但細胞 2 到達節點 2 時細胞 1 卻正在傳遞或在 buffer 中等待傳遞。這種情況下，細胞 2 開始得到服務的時間是細胞 1 離開節點 2 的那一刻，細胞 2 離開系統的正確時間受到了細胞 1 的影響，我們必須將細胞 1 的延遲考慮進去，變成由細胞 1 進入的時間來計算細胞 2 離開的時間，也就是

$$\begin{aligned} a_3(2) &= a_1(1) + s_1(1) + s_2(1) + s_2(2) \\ &= a_1(1) + S_1(1,1) + S_2(1,2) \end{aligned}$$

情況 3： 細胞 2 到達節點 1 時細胞 1 正在節點 1 中，此時要考慮兩種子情況

情況(a)： 細胞 2 等細胞 1 離開節點 1 後開始接受服務，當細胞 2 離開節點 1 到達節點 2 時，細胞 1 已經離開節點 2。這種情況下細胞 2 在節點 1 受到細胞 1 的延遲影響，但在節點 2 並不受影響，所以

$$\begin{aligned} a_3(2) &= a_1(1) + s_1(1) + s_1(2) + s_2(2) \\ &= a_1(1) + S_1(1,2) + S_2(2,2) \end{aligned}$$

情況(b)： 細胞 2 到達節點 2 時細胞 1 仍在節點 2 內，此時的條件與情況 2 相同，我們仍需計算細胞 1 離開節點 2 的時間後才能得到細胞 2 離開節點 2 的時間，在節點 1 發生的延遲變得無足輕重，數學的表示仍為

$$a_3(2) = a_1(1) + s_1(1) + s_2(1) + s_2(2)$$
$$= a_1(1) + S_1(1,1) + S_2(1,2)$$

把這三種情況可整理成下列表示法

$$a_3(2) = \max\{a_1(2) + S_1(1,1) + S_2(1,2), a_1(1) + S_1(1,2) + S_2(2,2), a_1(2) + S_1(2,2) + S_2(2,2)$$
$$= \max\{a_1(1) + \max\{S_1(1,1) + S_2(1,2), S_1(1,2) + S_2(2,2)\}, a_1(2) + S_1(2,2) + S_2(2,2)\}$$

如果我們有興趣的是第 i **個細胞，式子則變成**

$$a_3(i) = \max_{1 \le j \le i}\{a_1(j) + \max_{j \le l_1 \le l_2 \le l_3 \le i}[S_1(l_1, l_2) + S_2(l_1, l_2)]$$ ---------- (14)

我們解釋一下這個式子的物理意義，它表示著細胞 i **在整個系統中的延遲往往要考慮與某它細胞的互動關係。任何編號** j **的細胞(** $j < i$**)都可能使得細胞** i **在某個節點遭遇到延遲；單一節點上，細胞** i **必需等到所有之前的細胞都傳遞出去後才獲得服務，因此其離開系統時間的計算必須仰賴之前的細胞才能進行。舉個例子來說** $a_3(12)$**的最大延遲可能會是下列的情況：**$a_3(12) = a_1(5) + S_1(5,8) + S_2(8,12)$**。這表示細胞** 12 **在進入節點** 2 **時有細胞** 8、9、10、11 **正在等待服務中，因此細胞** 12 **離開節點** 2 **的時間需要追朔到細胞** 8 **進入節點** 2 **的時間，也就是** $a_3(12) = a_2(8)+S_2(8,12)$**，而細胞** 8 **進入節點** 1 **時有細胞** 5、6、7 **正在等待服務，所以細胞** 8 **離開節點** 1 **的時間** $a_2(8)$**必須追朔到細胞** 5 **進入節點** 1 **的時間。於是** $a_3(12) = a_2(8)+S_2(8,12) = a_1(5) + S_1(5,8) + S_2(8,12)$**。當然這只是情況的一種，其它的可能譬如** $a_3(12) = a_1(3) + S_1(3,7) + S_2(7,12)$ **或** $a_3(12) = a_1(9) + S_1(9,9) + S_2(9,12)$**等可能發生。**

了解上面的式子意義後，我們可以把細胞 j 推廣為 1 至 i 之間

$$a_3(i) = \max_{1\le j\le i}\left\{\max_{1\le j\le i}\{a_1(j) + \max_{j\le l_1\le l_2\le l_3\le i}[S_1(l_1,l_2) + S_2(l_1,l_2)]\right\} \quad \text{---------- (15)}$$

根據[9]中的定理 4.2，細胞 i 進入第 k 個節點的時間亦可由上式再擴展成

$$a_k(i) = \max_{1\le j\le i}\left\{a_1(j) + \max_{j=l_1\le l_2\le \ldots\le l_k=i}(\sum_{h=1}^{k-1} S_h(l_h,l_{h+1}))\right\} \quad \text{---------- (16)}$$

此式子可被廣泛的運用在任何的排程法上來表示延遲的時間，但要得知確實的延遲時間仍然十分不容易，式子中的 $a_1(j)$可以在使用 shaper 後得到答案，但不同細胞在不同節點實際的延遲時間卻是相當困難，所幸我們想得到的是最大延遲而不是實際值。觀察 $S_h(l_h,l_{h+1})$表示的是在單一節點 h 上，細胞 l_h 到 l_{h+1} 之間所需的實際延遲時間，我們在 4.2 節中所求出的即是單一節點上細胞 i 的最大延遲，如今將 l_h 視為細胞 0；l_{h+1} 視為細胞 i 就能得到 $S_h(l_h,l_{h+1})$的最大值，所以我們可用單一節點上的最大延遲時間 $S_w(l_h,l_{h+1})$式子取代實際的延遲時間 $S_h(l_h,l_{h+1})$，式子因此改變成

$$a_k(i) \le \max_{1\le j\le i}\left\{a_1(j) + \max_{j=l_1\le l_2\le \ldots\le l_k=i}(\sum_{h=1}^{n} S_w(l_h,l_{h+1}))\right\} \quad \text{---------- (17)}$$

根據 4.2 節的結果

$$S_w(l_h,l_{h+1}) = \frac{l}{C}\left\lceil\frac{(l_{h+1}-l_h)+1}{R_w}\right\rceil T + \frac{l}{C}\sum_{k=1}^{N} r_k \quad \text{---------- (18)}$$

我們求得

$$\sum_{h=1}^{n} S_w(l_h, l_{h+1}) = \frac{l}{C}\sum_{h=1}^{n}\left\lceil \frac{(l_{h+1}-l_h)+1}{R_w} \right\rceil T + \frac{l}{C}\sum_{h=1}^{n}\sum_{k=1}^{N_h} r_k^h \quad \text{---------- (19)}$$

$$\begin{aligned}
\sum_{h=1}^{n}\left\lceil \frac{(l_{h+1}-l_h)+1}{R_w} \right\rceil T + \sum_{h=1}^{n}\sum_{k=1}^{N_h} r_k^h &\le \sum_{h=1}^{n}\left[\frac{(l_{h+1}-l_h)+1}{R_w} + 1 \right] T + \sum_{h=1}^{n}\sum_{k=1}^{N_h} r_k^h \\
&\le \left[n + \frac{n-1}{R_w} \right] T + \left[\frac{l_{n+1}-l_1+1}{R_w} \right] T + \sum_{k=1}^{N} r_k + \sum_{h=1}^{n-1}\sum_{k=1}^{N_h} r_k^h \\
&\le \left[n + \frac{n-1}{R_w} \right] T + S_w(l_1, l_{n+1}) + \sum_{h=1}^{n-1}\sum_{k=1}^{N_h} r_k^h \\
&\le \left[n + \frac{n-1}{R_w} \right] T + S_w(j,i) + \sum_{h=1}^{n-1}\sum_{k=1}^{N_h} r_k^h
\end{aligned}$$

所以我們可推導出 $a_k(i)$的最大值

$$\begin{aligned}
a_k(i) &\le \max_{1\le j\le i}\left\{ a_1(j) + \max_{j=l_1\le l_2\le \ldots \le l_k=i} \left(\sum_{h=1}^{k-1} S_w(l_h, l_{h+1}) \right) \right\} \\
&\le \frac{l}{C}\left[n + \frac{n-1}{R_w} \right] T + \max_{1\le j\le i}\left\{ a_1(j) + S_w(j,i) \right\} + \frac{l}{C}\sum_{h=1}^{n-1}\sum_{k=1}^{N_h} r_k^h \quad \text{---------- (20)}
\end{aligned}$$

細胞 i 的最大延遲等於 $a_k(i)$-$a_1(i)$

$$Delay(i) \le \frac{l}{C}\left[n + \frac{n-1}{R_w} \right] T + \max_{1\le j\le i}\left\{ a_1(j) + S_w(j,i) \right\} + \frac{l}{C}\sum_{h=1}^{n-1}\sum_{k=1}^{N_h} r_k^h - a_1(i) \quad \text{-------- (21)}$$

4.4 Comparison with CORR

	CORR	Our approach
Fair	$\frac{2}{R_i}+\frac{2}{R_j}$	$\frac{1}{R_i}+\frac{1}{R_j}$
End To End Delay	$\frac{l}{C}\left[n+2\frac{n-1}{R_w}\right]T+\max_{1\le j\le i}\{a_1(j)+S_w(j,i)\}-a_1(i)$	$\frac{l}{C}\left[n+\frac{n-1}{R_w}\right]T+\max_{1\le j\le i}\{a_1(j)+S_w(j,i)-a_1(i)+\frac{l}{C}\sum_{h=1}^{n-1}\sum_{k=1}^{N_h}r_k^h$
Complexity	$O(1)$	$O(1)$

表格內為 CORR **和新的排程法在** Fair、Delay **及** Complexity **方面的綜合比較。如之前所提到的，在公平性方面新的排程法** Fair Index **只有** CORR **的一半，複雜度方面雖然同為** $O(1)$**，但新的排程法不用對新連線作任何的排序或搜尋，並且在不結合** Leaky Bucket **的情況下，傳送一個細胞所需的步驟比** CORR **更簡化了將近一半，所以這兩方面新排程法表現較佳應該是無庸致疑的，較具爭議性的是在最大延遲。**

比較最大延遲，兩種排程法是互有高低，但新的排程法似乎會隨著連線 N **的數目增加而變大，同時當** N **及** n **值很大時，有人可能會認為計算** r_k^h **的總合是件相當費時的工作，下面我們分別對這兩點做出解釋。**

首先是 r_k^h **的計算問題，若真有計算的必要，且不是拿來做比較，我們認為這一項求取它近似值即可，由於** r_k^h **必然為一介於** 0 **至** 1 **之間的數，我們可用一個平均值來替代，例如** r_k **的小數部分是用** 2 **個** bit **來表示的話，它只能表示出** 0、0.25、0.5 **及** 0.75**，這時我們就取**(0

+ 0.25 + 0.5 + 0.75) / 4 = 0.375 來當平均值，再假設各 N_h 值均為同一數 N，所以

$$\sum_{h=1}^{n-1}\sum_{k=1}^{N_h} r_k^h = 0.375 \times N \times (n-1) \quad \text{---------- (22)}$$

接著我們討論連線 N 的數目對最大延遲的影響。首先 N 值雖然很大，但在 ATM 的 call admission control(CAC)的控制下，N 不可能趨近於∞，否則若 $N \to \infty$，T 也會趨近於∞，這樣的條件下無論是那一個排程法，它的最大延遲都是∞。同時我們試著將 CORR 和新排程法的最大延遲相減來觀察兩種方法的差異性

$$D^{CORR}(i) - D^{OURS}(i) = \frac{n-1}{R_w}T - \sum_{h=1}^{n-1}\sum_{k=0}^{N-1} r_k^h \quad \text{---------- (23)}$$

就最壞的情況而言，r_k^h 都視為 1。為求比較方便，也是假設各節點的連線數 N_h 均為 N，所以

$$D^{CORR}(i) - D^{OURS}(i) = \frac{n-1}{R_w}T - \sum_{h=1}^{n-1}\sum_{k=0}^{N-1} r_k^h = \frac{n-1}{R_w}T - N(n-1) = (n-1)(\frac{T}{R_w} - N) \quad (24)$$

由上式可看出當 R_w 大於 T/N 時, CORR 的連線有較好延遲表現；當 R_w 小於 T/N，新排程法有較好的延遲表現。而 T/N 正好是所有連線的平均速率(在負載為 100%時)，因此我們可大略估計，約有一半的連線其 R_w 值大於 T/N，一半的連線 R_w 小於 T/N。同時 N 若趨近∞，R_w 的平均值將趨近於 0，使得兩種排程法的延遲都將接近∞。由上面的分析我們相信，兩種排程法在延遲方面的表現應該是齊鼓相當的。

4.5 Comparison with Flow-Timestamps scheduler

前面有提到過，計算 Timestamps 然後再與以排序的排程法較一般複雜度為 $O(1)$的排程法在 fair 及 delay 方面表現都較好，在這一節我們便試著用改良過的 CORR 來和它們做個比較，其實在參考文獻[6]及[9]當中，我們也都能找到一些 CORR 與這些俗稱 Flow-Timestamps 排程法的比較，但這些比較在數學模組的定義上都或多或少的有些問題存在，也因此影響了比較的可信度。譬如在 Fairness 方面，CORR 使用每個 cycle 開始的時間來代替真實的時間，這樣的方法只能表現出 CORR 長時間的公平性，卻看不出其短期間內的公平性。也就是說，CORR 可能在某一個 cycle 中的一段時間內產生大量擁塞的連續細胞，但在此 cycle 結束時每個連線都獲得了應有的頻寬，這就是所謂短時間的不公平而長時間卻是公平。這樣和原始的 Fairness 定義是不同的，在第二章裡我們有提到，Fair Index 是在任何兩段時間的任意兩個連線中求取最壞的情況，因此即使是發生了如上述的情況，我們還是能發現它會表現在數學式裡，很遺憾的，在參考文獻[6]及[9]中，CORR 的 Fair Index 是看不到這個情形的。而若將這個數學式拿來直接跟 Flow-Timestamps 的排程法做比較，顯然更是不盡公平。

所以我們必須將 4.1 節中的取樣點 c_1、c_2 改成實際的時間 t_1、t_2。如此才能和 Flow-Timestamp 做公平的比較。方法其實也並不複雜，假設 t_1、t_2 分別屬於系統中的 c_1、c_2，在 4.1 節中 t_1、t_2 只能取樣於 c_1、c_2 的開頭，如今我們把它擴展成 t_1、t_2 能取樣在 c_1、c_2 的任意一點，於是 4.1 節中的 $send_i(t_1,t_2)$產生了一上限值和一下限值

$$\lfloor (c_2 - c_1 - 1)R_i + r_i \rfloor \le send_i(t_1, t_2) \le \lfloor (c_2 - c_1 + 1)R_i + r_i \rfloor \text{ -------- (25)}$$

接下來如同 4.1 **節所進行的步驟, 但這次** $send_i(t_1,t_2)$**及預設頻寬所使用的單位為** bits/s**，我們用** δ_i **及** δ_j **分為代表連線** i、j **的預設頻寬**

$$\begin{aligned}
&\left| \frac{l\lfloor (c_2 - c_1 + 1)R_i + r_i \rfloor}{\delta_i} - \frac{l\lfloor (c_2 - c_1 - 1)R_j + r_j \rfloor}{\delta_j} \right| \\
&= \left| \frac{l(c_2 - c_1 + 1)R_i + lr_i - lx_i}{\delta_i} - \frac{l(c_2 - c_1 - 1)R_j + lr_j - lx_j}{\delta_j} \right| \\
&= \left| \frac{l(c_2 - c_1 + 1)\frac{\delta_i T}{C} + lr_i - lx_i}{\delta_i} - \frac{l(c_2 - c_1 - 1)\frac{\delta_j T}{C} + lr_j - lx_j}{\delta_j} \right| \\
&= \left| \frac{2lT}{C} + \frac{lr_i - lx_i}{\delta_i} - \frac{lr_j - lx_j}{\delta_j} \right| \\
&= l\left| \frac{2T}{C} + \frac{r_i - x_i}{\delta_i} - \frac{r_j - x_j}{\delta_j} \right|
\end{aligned}$$

和 4.1 **節一樣，就最壞的情況而言**

$$\left| W_i(t_2 - t_1) - W_j(t_2 - t_1) \right| \le l\left(\frac{2T}{C} + \frac{1}{\delta_i} + \frac{1}{\delta_j} \right) \text{ ---------- (26)}$$

接著我們繼續推導 delay**，首先我們得注意到，**Flow-Timestamps **的排程法在推導** delay **時是假設沒有排列延遲**(queuing delay)**的，這是因為它們的數學式不像我們之前有使用** Leaky Bucket **作輔助，以** Leaky Bucket **來調整細胞進入網路時的數學模組**(arrival function)**，藉此得到細胞進入系統時的時間，然後再將細胞離開系統的時間求出，兩者相減就得到** delay**。**Flow-timestamp **一般而言都是利用** timestamps **來反推所謂"細胞期待中的到達時間"，所謂期待中到達時間就是不會有排列延遲的情況發生。所以，如果我們想與這些排程法比較時就得考慮**

到這些數學模組不同的問題，將雙方的差異消除後才能比較。基本上若要把延遲的條件加入 Flow-Timestamps 排程法的數學式是有其困難處，畢竟它們所使用的方法也已行之多年，但要假設我們的方法沒有延遲倒是容易的多，實際上只要把 4.2 節中的 *d(i)*改成 *d(0)*即可，因為 *d(0)*在數學上是到達 buffer 的第一個細胞，它沒遭遇任何在它之前抵達的細胞的延遲，於是

$$d(0) = \frac{l}{C}\left\lceil \frac{1}{R} \right\rceil T + \frac{l}{C}\sum_{k=1}^{N} r_k \quad \text{---------- (27)}$$

這裡的連線數目 N 原本是包括此一細胞的連線本身, 但實際上第一個細胞到達時 r_k 的值是為 0，也就是說這個式子可以使用 N-1 代替 N，在 4.1 節中我們的表示法是用 N，在此為了配合 Flow-Timestamps 的表示法，我們於是修改式子成為

$$d(0) = \frac{l}{C}\left\lceil \frac{1}{R} \right\rceil T + \frac{l}{C}\sum_{k=1}^{N-1} r_k \le \frac{l}{C}\left(\frac{1}{R}+1\right)T + \frac{l}{C}(N-1) \quad \text{--------- (28)}$$

我們將所有的比較整理如下表

	Flow-Timestamps	**Our approach**
Fair	$l\left(\frac{1}{f_i}+\frac{1}{f_j}\right)$	$l\left(\frac{2T}{C}+\frac{1}{f_i}+\frac{1}{f_j}\right)$
End To End Delay	$\frac{1}{R_i}\frac{l}{C}+(N-1)\frac{l}{C}$ *SCFQ* $\frac{1}{R_i}\frac{l}{C}+\frac{l}{C}$ *PGPS*	$\left(\frac{1}{R_i}+1\right)\frac{lT}{C}+(N-1)\frac{l}{C}$
Complexity	*O* (*N*) or *O*(log *N*)	*O* (1)

Flow-Timestamp 的 delay 方面我們列出了兩種排程式的 delay，PGPS 是所有 Flow-timestamps 中 delay 表現最好的，SPCQ 則是最差的。很顯然，我們的排程法即使是僅和 SPCQ 比，在效能上還是有著一段不小的差距，最大的因素來自於 frame 的長度 T 是個不小的數字，但我們有沒有可能把 T 減到最小值呢??其實在 ATM 網路中，我們是有很大機會的，因為 ATM 採用了固定長度的資料傳送單位，如果是在一個傳統的 packet-switched 網路中, 隨時變動的 packet 長度會使這個問題相當棘手，而 ATM 網路中，最小值就是讓 T=1。

	Flow-Timestamps	**Our approach**
Fair	$l\left(\frac{1}{f_i}+\frac{1}{f_j}\right)$	$l\left(\frac{2}{C}+\frac{1}{f_i}+\frac{1}{f_j}\right)$
End To End Delay	$\frac{1}{R_i}\frac{l}{C}+(N-1)\frac{l}{C}$ *SCFQ* $\frac{1}{R_i}\frac{l}{C}+\frac{l}{C}$ *PGPS*	$\left(\frac{1}{R_i}+1\right)\frac{l}{C}+(N-1)\frac{l}{C}$
Complexity	O (N) or O(log N)	O (1)

上面的這一個表格就是我們讓 T=1 後的比較情形，先觀察 Fair Index，和輸出通道的頻寬 C 比較起來，δ_i 及 δ_j 都是極小的數目，因此 $2/C$ 這一項幾乎是可以省略，這樣在 Fair Index 上新排程法的表現是相當接近 Flow-Timestamps 的，這表示著即使是在一段很小的時間內, 新排程法也不會產生一大串擁塞的連續細胞。同樣在 delay 方面，當 T=1 時, 每個 R_i 都應該遠小於 1, 所以 $1/R_i$ 遠大於 1, 這樣一來 delay 方面我們所使用的排程法也相當接近 Flow-Timestamps, 幾乎和 SCFQ 不相上下。於是我們可是這樣宣稱:

在 ATM 網路下，若把 frame 縮小為一個細胞的長度，一個複雜度為 *O*(1)的排程法在 fair 及 delay 上都能與 Flow-Timestamps 的排程法媲美。

然而這個方法並不是完美的，因為現在 frame **的長度僅為一個細胞的大小，所以我們可以想像每一個** cycle **在經過許許多多的硬體計算後僅能送出約一個細胞，這是相當浪費頻寬的，特別是當連線數目** N **極大時更是明顯。因此這個方法變得跟** Flow-Timestamps **排程法有著相同的問題，在高速網路中可能因過多的硬體計算而形成系統的瓶頸。但至少我們發現了在** ATM **網路中，一個** $O(1)$**的排程法也能擁有與** $O(N)$**或** $O(\log N)$**相近的效能。同時我們也想到一種可行的方法，就是在網路負載較小時，系統可以將** frame **長度** T **設為** 1 **或比** 1 **稍大，而在網路負載較大時，將** T **設為較大的值。這種動態的調整就能避免網路頻寬的浪費，且整個系統具有相當大的彈性。在負載小時擁有** Flow-Timestamps **排程法的公平性及較小延遲，同時在網路負載大時擁有** frame-based **排程法的高網路頻寬使用率。**

第五章　模擬結果

在這一章節中我們將對和 Leaky Bucket 結合後排程法作程式模擬。之前也提到過，結合 Leaky Bucket 後是利用 Minor Cycle 來增加網路頻寬的使用率，由於公平性及最大延遲都是在忙碌週期下計算，所以有沒有結合 Leaky Bucket 在數學上來分析上都是相同的，也因此結合前和結合後的成效我們只得用程式模擬來觀看結果。相反的，公平性及最大延遲都是推導其最壞的狀況，在模擬時往往最壞的情況不容易出現，因此較適合用數學加以分析。

設計上我們在 feedback 後才進行 Minor Cycle 處理細胞主要是考慮到，細胞由 Leaky Bucket 接到訊息後釋放至系統的 buffer 中會有時間上的延遲，如果讓系統在 Major Cycle 中等待勢必會降低網路頻寬的使用率，所以我們讓系統先處理其它連線的細胞，等到 Major Cycle 結束後再處理 Leaky Bucket 釋放過來的細胞。假設系統的處理細胞的速度夠快且傳輸延遲很小的話，理想上是不會浪費任何頻寬的，若系統處理細胞的延遲過大，則 Minor Cycle 中可設計更精密的計算，如此一來系統仍可以有效的利用頻寬並且不會造成延遲。在模擬中我們仍是假設系統處理系胞的時間為 0，因此網路頻寬的效率理想上可達 100%。

圖 5.1 是結合 Leaky Bucket 的排程法和未結合 Leaky Bucket 的排程法在不同負載下的比較，其中 Leaky Bucket 的訊標產生速率 λ_i 我

們將其設為跟連線預設的頻寬 R_i 相同，訊標的上限值 b_i 設為 30。很顯然的結合 Leaky Bucket 後，無論網路負載的大小，網路頻寬都能有效且明顯的提升。

圖 5.2 我們試著讓網路在短期內大量增加負載，但各連線預設保留的頻寬及 Leaky Bucket 的訊標產生速率並沒改變，b_i 仍為 30，網路預設負載為 30%。這種情況在現實中是很有可能發生的，特別是在傳送即時的影像或視訊的檔案時。當然網路的整體負載要同時倍數增長的機會是微乎其微，而我們的程式正是模擬這樣的情況，但這也可看作各連線在各別遭遇這樣情況後所求出的平均曲線圖。很顯然的，結合 Leaky Bucket 後的頻寬使用率幾乎是隨著實際的負載成線性增長，而未結合 Leaky Bucket 始終保持相同的頻寬使用率。

圖 5.3 則是我們觀察訊標上限值對頻寬使用率的影響，雖然網路實際流量是不可預測的，短時間內可能遠大於預設的頻寬，也可能遠小於預設的頻寬，但長時間來看是會與其預設頻寬大致相同，若訊標上限值越大雖然耗費成本，但可望能在短期內傳遞大量細胞來有效提供頻寬使用率，因此我們做此模擬觀察究竟 b_i 要有多大，排程法就可以不用與 Shaper 結合而有相同的效率。

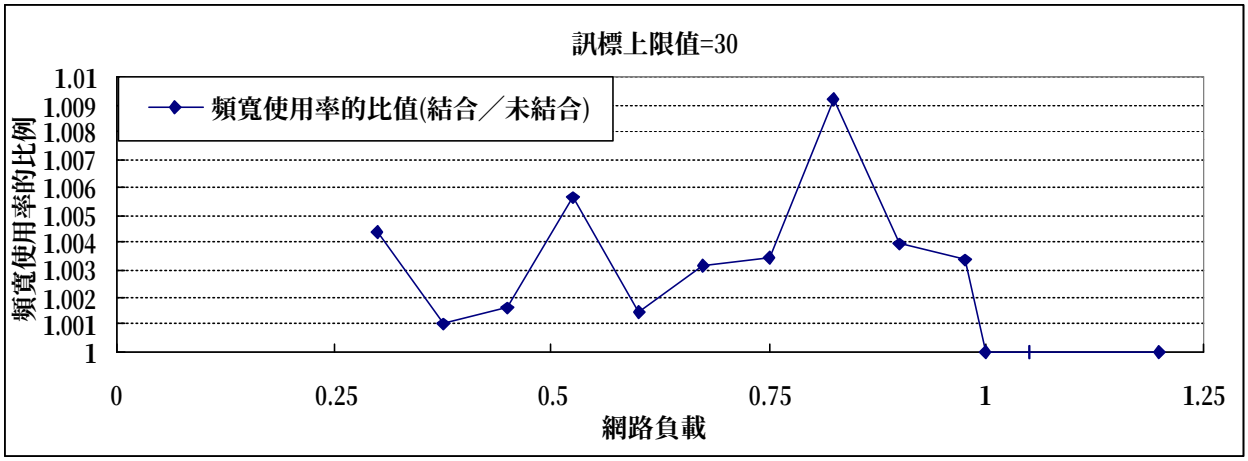

圖 5.1 網路負載對應頻寬使用率圖

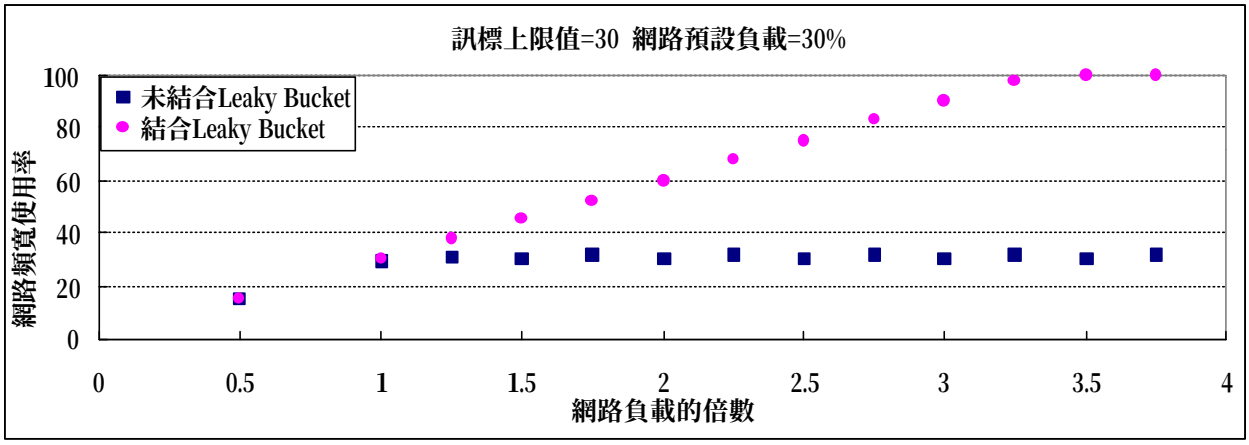

圖 5.2 網路負載倍數對應頻寬使用率圖

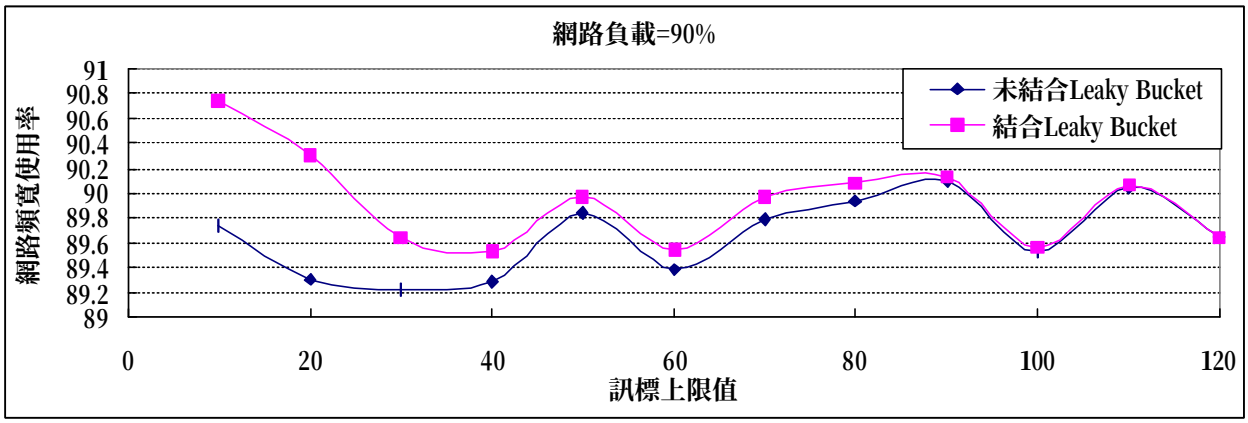

圖 5.3 訊框上限值對應頻寬使用率圖

第六章 總結

CORR 是個簡單卻有實際效益的排程法，雖然純就公平性及網路延遲的表現來看，計算 Timestamp 的排程法表現是無懈可擊，但若要考慮成本問題，簡單的排程法仍有其發展的必要性，而 CORR 排程法正是此類排程法中的最佳選擇。

我們在此篇論文中證實了 CORR 的演算法其實是可以更簡單而表現更好的，或許當初為了推導網路的最大延遲時，訊框的上限值可讓數學分析過程簡化許多，但並不表示著訊框沒有上限值時排程法就一定表現比較差，於是我們秉持著這樣的想法將 CORR 作改良，數學分析證明了修改過後的方法能比 CORR 更加公平的分配頻寬，而最大延遲則是互有高低，在複雜度上雖然處理單一細胞時同為 $O(1)$，但我們的方法移除了新連線加入時需排序的步驟，解決了這個步驟可能帶來的問題，除此之外，這個排程法的運算流程也較 CORR 更簡單，因此我們認為這個改良過的排程法會比 CORR 更易於實作，我們有理由相信這次改良 CORR 的嘗試是成功的。同時我們也提出將 frame 的長度動態調整的觀念，我們已在 4.5 節中推導出 frame 長度 T=1 時新排程法的效能，並與 SCFQ 和 PGPS 等相比較，數學分析中可看出當 T=1 時整個系統的延遲及公平性是最理想的，但在負載大時卻會浪費網路頻寬。另外一方面，較大的 frame 長度產生較大的延遲與較多壅塞的細胞，但卻有較好的網路頻寬使用率，因此我們認為，

一個動態調整 frame 長度的方法將能結合兩者的優點，有效增進網路的效能。

CORR 和 Leaky Bucket 在運作時是相互獨立的, 在此篇論文中我們嘗試用最簡單的方法將兩者結合起來，模擬結果顯示這樣的結合對網路頻寬的使用是有相當幫助的，雖然這只是個簡單的嘗試，但相信能給排程法的研究帶來一些啟發，如何能讓排程法和 Shaper 的組合效益更好，應該是未來研究的一個好課題。

附錄 1

Theorem：***在新的排程法中，任一個連線由加入排程後開始算起，*** *cycle* ***長度的總和符合下列式子***

$$\sum_{k=1}^{n} C_k \le nT + \sum_{i=1}^{N} r_i$$

證明：

首先考慮任一個連線的任一個 cycle 均符合下列情況

$$C_k \le \sum_{i=1}^{N_k} r_i' = \sum_{i=1}^{N_k} r_i + \sum_{i=1}^{N_k} R_i$$

C_k 是第 k 個 cycle 的實際長度，r_i' 是每個 cycle 在開始傳送細胞前其計數器的值，r_i 則是各計數器在各 cycle 剛開始時計數器的值，N_k 是此 cycle 中連線的數目，每個 cycle 開始傳送細胞前都會執行

$r_i' \leftarrow r_i + R_i$，很顯然的，$C_k$ 的最大值即為 $\sum_{i=1}^{N_k} r_i'$。因此我們得知 C_1 為

$$C_1 \le \sum_{i=1}^{N_1} r_i^1 + \sum_{i=1}^{N_1} R_i^1$$

為方便解釋，我們對 r 及 R 標上一個數字表示其為第幾個 cycle 的 r_i 或 R_i 值，如上式中 r_i^1 即為 cycle 1 的 r_i 值。而 C_2 為

$$C_2 \le \sum_{i=1}^{N_2} r_i^2 + \sum_{i=1}^{N_2} R_i^2$$

其中 $\sum_{i=1}^{N_2} r_i^2$ 值其實就是 cycle 1 中各 r_i^1 留下的小數部分的總合，所以

$$\sum_{i=1}^{N_2} r_i^2 \le \sum_{i=1}^{N_1} r_i^1 + \sum_{i=1}^{N_1} R_i^1 - C_1$$

所以

$$C_2 \le \sum_{i=1}^{N_2} r_i^2 + \sum_{i=1}^{N_2} R_i^2 \le (\sum_{i=1}^{N_1} r_i^1 + \sum_{i=1}^{N_1} R_i^1 - C_1) + \sum_{i=1}^{N_2} R_i^2 = \sum_{i=1}^{N_1} r_i^1 + \sum_{i=1}^{N_1} R_i^1 + \sum_{i=1}^{N_2} R_i^2 - C_1$$

$$C_3 \le \sum_{i=1}^{N_3} r_i^3 + \sum_{i=1}^{N_3} R_i^3 \le (\sum_{i=1}^{N_2} r_i^2 + \sum_{i=1}^{N_2} R_i^2 - C_2) + \sum_{i=1}^{N_3} R_i^3 = \sum_{i=1}^{N_1} r_i^1 + \sum_{i=1}^{N_1} R_i^1 + \sum_{i=1}^{N_2} R_i^2 + \sum_{i=1}^{N_3} R_i^3 - C_1 - C_2$$

$$\vdots$$

$$C_n \le \sum_{i=1}^{N_1} r_i^1 + \sum_{k=1}^{n} \sum_{i=1}^{N_k} R_i - \sum_{k=1}^{n-1} C_k$$

無論每個 cycle 中連線數 N_k 為多大，在 ATM 的 CAC 控制下 $\sum_{i=1}^{N_k} R_i$ 都會小於等於 T，再將 $\sum_{k=1}^{n-1} C_k$ 移至不等式的右邊，即可得到

$$\sum_{k=1}^{n} C_k \le nT + \sum_{i=1}^{N_1} r_i^1$$

參考文獻

[1]. P. Goyal, H. M. Vin, and H. Cheng, “Start-time fair queuing: A scheduling algorithm for integrated services network,” *IEEE/ACM Trans. Networking*, vol. 5, pp. 690-704, 1997.

[2]. K. Parekh and R. G. Gallager, “A generalized processor sharing approach to flow control in integrated services network: The single node case,” in *Proc. IEEE INFOCOM’92* vol. 2, May 1992, pp.915-924.

[3]. M. Shreedhar and G. Varghese, “Efficient fair queuing using deficit round robin, “ *IEEE/ACM Trans. Networking*, vol. 4, pp. 375-385, June 1996.

[4]. J. Cobb, M. Gouda, and A. El-Nahas, “Time-Shift scheduling: Fair scheduling of flows in high-speed networks,” *IEEE/ACM Trans. Networking*, vol. 6, pp. 274-285.

[5]. S. J. Golestani, “Congestion free communication in high-speed packet networks,” *IEEE Trans. Commun.*, vol. 32, pp. 1802-1812, Dec. 1991.

[6]. D. Saha, “Supporting distributed multimedia applications on ATM networks, “ PH.D. dissertation, Dep. Comput. Sci., Univ. Maryland, College Park, 1995.

[7]. S. J. Golestani, “A framing strategy for connection management,” in *Proc. SIGCOMM’90*, 1990.

[8]. D. Stiliadis and A. Verma, “Efficient fair queuing algorithms for packet-switched networks,” *IEEE/ACM Trans. Networking*, vol. 6, pp. 175-185, Apr 1998.

[9]. D. Saha, S. Mukherjee, and S. Tripathi, "Carry-over round robin: A simple cell scheduling mechanism for ATM networks," *IEEE/ACM Trans. Networking*, vol. 6,pp. 779-796, Dec 1998.

[10]. D. Stiliadis and A. Verma, "Rate-proportional Servers: A design methodology for fair queuing algorithms," *IEEE/ACM Trans. Networking*, vol. 6, Apr 1998.

[11]. S. Iatrou and I. Stavrakakis, "A dynamic regulation and scheduling scheme for real-time traffic management," *IEEE/ACM Trans. Networking*, Vol. 8,Feb 2000.

[12]. D. Stephens and H. Zhang, "Implementing scheduling algorithms in high-speed networks," *IEEE J. Select. Areas Commun.*, Vol 17, Jun 1999.

[13]. S. Raghavan and S. Tripathi, Networked multimedia systems. Prentice Hall, 1998.

[14]. L. Zhang, "VirtualClock: A new traffic control algorithm for packet switching networks," ACM trans. Comput. Syst., vol.9, pp. 101-124, May 1991.