GREEN:網際網路中具服務品質之緩衝區管理機制

黃蓮池	許俊萍	羅煥英	李欣紘
義守大學電機系	義守大學資工系	義守大學資工系	義守大學電機系
高雄縣大樹鄉學城	高雄縣大樹鄉學城	高雄縣大樹鄉學城	高雄縣大樹鄉學城
路 1 段 1 號	路 1 段 1 號	路 1 段 1 號	路 1 段 1 號
lain	steenhsu	m883301m	m893218m
@isu.edu.tw	@isu.edu.tw	@isu.edu.tw	@isu.edu.tw

摘要

服務品質控制是為了能成功部署多媒體網路。而緩衝區管理又扮演了非常重要的角色。因此，本篇論文提出了一個新的緩衝區管理演算法，稱為 GREEN。GREEN 是藉由 ERD 和 RED 所延伸出來的概念。此外，GREEN 更進一步地利用整體性地考量隨機機率，而將「隨機」擴展為「全域隨機」。另一方面，利用了先期預估網路狀態，而將原本的「先期」擴展為「先期預估」。GREEN 演算法中的「全域隨機」與「先期預估」主要是藉由通盤性的考量網路狀況和服務品質需求,並且預先估計部分的決策因素。將 GREEN 的演算法加入 DiffServ 架構中,藉由區分不同種類的服務能提供 scalabillty，而藉由考慮和預估多媒體網路狀況使網路更順暢。在效能評估方面，我們以 RED 作為比較對象,並藉由模擬各種不同的情況來偵測 GREEN 的特性。從模擬的結果我們可以看到，GREEN 的 goodput 比 RED 還要來得高。另外很明顯可以看出，GREEN 作決策的速度會比 RED 要快，因為部分的計算在之前就已經先被完成了。

第一節、簡介

在多媒體通訊時期的來臨。要解決符合多媒體通訊的傳輸特性,不僅要有高速的網路而且要能提供不同種類的服務。然而，網路的頻寬不可能無限制的增加來滿足不同種類的資料流量。為了能滿足不同服務品質的需求，發展了許多流量控制機制，例如：多佇列排程和緩衝區管理。因此，本論文將研究一個緩衝器管理(buffer management)的方法。

在過去的二十幾年裡，已經有許多相關的研究,都是利用緩衝區管理的技術來提供網路傳輸服務品質的。例如：Drop tail [1,2,a1]，Random Drop [1,a2]， Early Random Drop (ERD) [1]，Random Early Detection (RED) [3,4,5]，等等，都是相關的研究文獻。在下文裡，我們將簡介一些現有的緩衝區管理演算法,其中最簡單的管理方法,就是 Drop Tail 這個方法。在該方法裡，只要緩衝區滿了，封包就會直接被丟棄。這些封包通常都是來自各個不同的連線，例如 TCP 連線。在同時，這些連線的來源端通常會因此而降低他們的 windows size。這也導致了 TCP 連線的 global synchronization。而在 Random Drop 裡，當封包到達緩衝區，而且緩衝區已經滿了的時候,緩衝區裡面的某一個封包將被隨機地取出,並且予以丟棄。而在 Early Random Drop 裡，如果佇列長度超過了某一個預設的丟棄門檻值，這時,每一個收到的封包將依據某一個預設的機率被選擇丟棄。

另一個被提出的 Random Early Detection (RED)方法，並不直接參考佇列的大小來作緩衝區管理，而使用佇列大小的指數權重移動平均(exponential weighted moving average)作為管理參考的依據。因此，在這個方法裡，會需要兩個門檻值，分別是最小門檻值與最大門檻值。如果平均佇列低於最小門檻值時，則接受到達的封包。如果平均佇列大於最大門檻值時，則丟棄到達的封包。而如果平均佇列是介於兩個門檻值之間,則到達的封包就會依照某個標記機率而打上標記。事實上，ERD 和 RED 是相當類似的。其兩者的差異點在[3]裡有相當詳盡地描述。平均佇列大小的使用，對於猝發性的訊務(traffic)或是暫時性壅塞會有比較好的調和效果[3]，但是卻會造成網路狀態反應時間的增加。想在暫態調和效果和網路狀態反應時間之間尋找到一個適合的平衡點則不是一件容易的事情。此外 RED 可以不將封包丟棄，而只是將封包

標記，這是一個相當不錯的觀念。這個觀念很有可能是沿襲 leaky bucket 方法[6,7,8]或是非同步傳輸模式(ATM)之訊務控制(traffic control)方法而來。在 ATM 的訊務控制裡，超量的訊務就是會被打上標記以作為日後壅塞時訊務丟棄的重要準則。不過在 RED 裡的標記卻是和 ATM 中所使用的標記技巧在意義上，是有一些不太一樣的。在 RED 裡，所謂的標記有可能是指丟棄或是貼上標籤兩者其中的任何一種，而在 leaky bucket 裡的標記則一般是專指貼上標籤的意思。我們認為，RED 這個方法最重要的創意在於利用變動的機率，取代原有的固定式機率來選擇所要丟棄的封包。而這個觀念和 ERD 是不一樣的。在 ERD 裡所使用的是固定的機率來選擇所要丟棄的封包。Pipas[9]考慮延遲的控制，所以將原 RED 中的佇列平均長度取代為平均延遲。Hasegawa[10]使用一個增強的 RED(事實上，它是 ERD 而不是 RED)來改善多個 TCP 連結的系統公平性，經由不同的 Throughput 得到不同丟棄機率的方法來獲得增強的 RED(ERD)。

網路上大部分的緩衝區管理的演算法主要考慮的是 TCP 連結的 global synchronization。然而，對於未來的多媒體網際網路而言，很多應用將會是強調即時性，此時使用 UDP 是比較適合的。此篇論文不特別強調 TCP 或是 UDP。再者，ATM 網路也是設計用來支援不同的服務品質，然而原本的網際網路只提供盡力而為傳輸的方式。因此，ATM 便成為未來多媒體網路最熱門的候選者。但由於 ATM 的複雜性再加上網際網路的普及，固定大小的 cell 將不會取代非固定大小的封包。這就是為什麼這篇論文強調封包的緩衝區管理，而不是如同許多文獻中所探討 cell 的管理[4]。雖然有些論文例如[5]是在討論非固定大小封包的緩衝區管理，在這些方法中，特別是數值的例子裡，主要是著重在固定大小的實例。

1.1 動機與目的

這篇論文提出了一個新的緩衝區管理機制，稱為全域式隨機先期預估封包夾取法(Global Random Early Estimation for Nipping packets, GREEN)。和 RED 相較之下，我們所提出的方法有兩項重要的新設計：第一是隨機機率的全域性考量。在其他的緩衝區管理方法裡，大多只是利用網路的現況當作隨機機率的參數，而 GREEN 則更進一步地考慮了服務品質的需求，在這個情形之下，系統的 goodput 可以更為提升。第二是我們使用了先期預估(early estimation)來取代 RED 裡的先期偵測(early detection)。這個先期預估，可以在緩衝區壅塞發生之前，就預估將來是不是會發生壅塞。如果預估結果是肯定的，那麼下一個到達的封包會被夾取(nip, 亦即 RED 裡的 mark)的機率就會相對提高。在我們所提出來的方法之下，封包夾取與否的決定時間會非常地短暫，因為決定的部分運算早就已經做好了。因此，GREEN 不但可以改善系統的 goodput，更可以減少封包標記決策所需耗費的時間。

1.2 分析緩衝區管理演算法

這些方法大致上考量了兩個因素[5]：第一個因素是決定封包是否丟棄的時機。一般來說，封包丟棄的決策，大多是在壅塞即將發生之時才作的。而緩衝區裡的某些封包則有可能被丟棄，以便於讓新的或是更高優先權的封包可以進入緩衝區。另外一項考量因素則是需參考哪些資訊來作封包丟棄的決策。此外，我們認為緩衝區管理演算法可以詳細地分為以下五個因素：

1. 執行緩衝區管理的觸發事件，或是觸發準則。例如以固定長度時間週期性地檢查緩衝區的使用狀況，然後再依照檢查結果做出必要的處理。
2. 壅塞發生與否的判斷準則；例如 RED 是使用平均佇列長度和兩個門檻值作比較作為判斷依據的。而 ERD 則是使用佇列長度和一個門檻值的比較來當作依據的。
3. 懲處之封包或是連線(connection)的選取。例如在緩衝區裡隨機地選取封包，或是從新來的封包裡面利用某個機率去隨機地選取。
4. 對所選取出之封包的處理方法。例如丟棄，或是標記。
5. 端點站台(end stations)的反應。例如傳送速率的降低。TCP 壅塞控制演算法為了避免快速增加傳輸速率而使得封包被丟棄，所以在偵測到封包被丟棄或逾時發生，便使用壅塞避免演算法去減少它的傳輸速率[14]。

在一般的情況之下，例如之前我們所回顧的幾個相關研究裡，上述的考慮有可能是個別的，或是部份整合起來考量的，且不見得會全部同時用到。例如 ERD 和 RED 就是整合了因素一和因素二。他們分別使用了佇列長度或是平均佇列長度是否超過門檻值，來當作他們的壅塞判斷準則與緩衝區管理的觸發機制。事實上，因素一和因素二通常是被整合在一起設計的。另外，ERD 與 RED 則都沒有考慮到因素五。這也是一般情況，因為緩衝區管理演算法很少會考慮到訊務來源端的流量控制演算法。不過，整合該項因素卻是一個相當值得研究的課題。在本篇論文裡頭，我們主要只考慮緩衝區管理演算法而不管相關的回饋控制與訊務源的反應。根據過去幾個受歡迎的演算法，例如 Drop Tail, Random Drop, ERD 和 RED 等，以及綜合以上我們所提及之觀點與[5]所述之相關準則，緩衝區管理的設計可以大致分為兩個主要的類型：第一類是先期或非先期的考量，第二類則是隨機或非隨機的考量。所謂先期或非先期的考量是指選擇懲處封包的時機是在緩衝區滿載之前，或是之後。這個類型主要整合了前述之因素一和因素二。而另一個隨機或非隨機的考量則是指當開始選取懲處封包時，應該

利用隨機的機率或是非隨機的機率來選取。這個類型通常是整合了前述之因素三與因素四。假使是使用「先期」時，則不可避免地，會需要配合使用「隨機機率」，因為在緩衝區資源尚未耗盡之前，就貿然地固定選取某些封包是不太合理的。不過，這種情況在 RED 裡卻是有可能發生的。因為當緩衝區尚未滿載時，RED 所使用的平均佇列長度卻可能會大於最大門檻值。雖然這個問題可以藉由選取更大的門檻值來避免，不過卻會使得最大門檻值變得沒有作用而使得丟棄機率相對地變得很小，而造成「先期」與「隨機」變得沒有其實質效果。

「先期」與「隨機」的結合是緩衝區管理演算法的主流。它是從 ERD 開始被提出來的，並且在 RED 裡更進一步地被加強。

1.3 論文的組織結構

本篇論文其他部分的組織架構如下：在第二節裡，我們解釋 GREEN 的意思和介紹 GREEN 演算法。並給予數值例子，證實 GREEN 的方法有好的效能與特性。事實上，在網際網路中，每個節點可能藉由不同的資料流得到不同種類的服務品質。所以在第三節中，為了要符合網際網路實際的情形，我們加強 GREEN 演算法。主要是修改 GREEN 演算法中的一些參數，使得每個節點不只在單一種類的資料流，在多個種類的資料流也能使用 GREEN 演算法做緩衝區管理。在數值例子中，證實在多個種類的資料流使用 GREEN 的方法有好的效能與特性。最後我們在第四章裡作總結並指出未來工作的方向。

第二節、Global Random Early Estimation for Nipping

2.1 GREEN 演算法

2.1.1 GREEN 的意思

GREEN 仍然使用「先期」與「隨機」的結合。然而，GREEN 將該觀念發揮淋漓盡緻。首先，GREEN 將封包選取的隨機機率予以全域化。在這個動作裡，明白地指出了隨機機率不只是網路狀態的函數，更可以利用服務品質的需求來當作該函數的參數。在 GREEN 裡，懲處封包選取的全域隨機(General Random)機率稱為夾取機率(nipping probability)。在全域隨機的考量之下，那些不符合服務品質需求的封包，將會被夾取出來，以便將其所佔據之緩衝區資源讓予符合服務品質需求之封包，以改善整體之 goodput。 其次，不僅是「先期」，GREEN 更進一步地使用了先期預估(Early Estimation)的「雙先期(dual early)」技巧。更具體地來說，GREEN 不但參考了網路狀況(例如佇列長度)來決定是否要開始進行管理行為，並且還考慮了相關的網路狀態變化，因此它可以依照實際的情況來作預估。由於預估會比偵測更早獲取狀態資料，因此我們才會說這是一種雙先期的技巧。使用預估技巧時，其作決策所需的時間會比偵測技巧來的短，因為一些相關決策所需之計算已經在先前預估時就完成了。在 GREEN 裡的最後一個縮寫字母 N，其所代表的含意就是夾取(nipping)。這個「夾取」的意義和 RED 裡面的標記(marking)是一樣的意思。三個主要使用「夾取」來取代原有「標記」一詞的的理由如下：

1. 「標記」一詞容易與 leaky bucket 和 ATM 所使用的「標記」混淆。一般標記一詞並不含有「丟棄」的意義。
2. 取自俗諺：防患未然(牛津英漢字典)。夾取一詞所代表的含意是挑出來，而挑出來之後的動作也許是在封包上打上一個標記，也許是將該挑選出來之封包直接予以丟棄，不論如何，在語意的認知上，都是很適當的，而且也不容易造成混淆。
3. 讓整個演算法的縮寫名稱更為優美，其動機就類似於 RED 方法的最後一個縮寫字母 D。事實上，RED 所使用的縮寫 D 很容易被誤認為丟棄(discard) [5]，而不會是偵測(detection)。

2.1.2 GREEN 的演算法

首先，表 2.1 提供了在 GREEN 演算法中所利用的符號。此外，在演算法中每一個參數的初始值也列在表 2.1 中。為了決定是否要啟動懲處封包選取功能，首先我們先訂立一個門檻值，稱為夾取門檻(nipping threshold)，並且以 T_n 來表示。佇列的長度通常被用來當做是判斷的標準，像是 ERD。但不與 RED 同，因為使用平均佇列長度當做是判斷準則的優點尚值得懷疑。如果一個封包到達了，而且剛好網路的狀況也超過了該門檻值，則稱為夾取機率(nipping probability)的機率值(P_n)會被計算出來，以決定是否夾取封包。如果封包到達時，緩衝區已經滿載了，那麼該封包就會被丟棄。而 GREEN 的核心即在於夾取機率的計算，同樣地該部分也是 RED 演算法的主軸。GREEN 保留 RED 均勻(uniform)的特性，所以使用一個變數 c_p 計算進入緩衝區的封包數量直到某個封包被夾取為止。和 RED 一樣，夾取機率是其他機率的函數。那些機率函數在 GREEN 裡是稱為亂度(entropy)機率，並且用 P_{e_pre} 來表示。

使用全域隨機時，夾取機率不但是網路狀況(例如佇列長度和連結的使用率)的函數，也是服務品質需求(例如延遲需求和漏失需求等)的函數。很自然地，我們會夾取一個不符合服務品質需求的封包。舉個例子來說，一個已經逾時的即時性封包就應該被夾取，因為即使它到達了目的地，也會因為逾時的關係而一點用處都沒有。在這樣子的機制之下，緩衝區的資源，甚至是頻寬資源就不會因為儲存或傳送這一類沒有用的封包而造成資源的浪

費，進而可以提高系統的 goodput。因此，夾取機率可以用這樣的式子來表示：

$$P_n = \max\left\{U(q/B_a - d), [(P_{e\_pre}/(1 - c_p P_{e\_pre})]U(l - \hat{P}_l)\right\} \quad (2.1)$$

$U(x)$ 是單位函數(unit function)即

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

在式子(2.1)中包含了兩個單位函數，分別為 $U(q/B_a - d)$ 和 $U(l - \hat{P}_l)$，是對於GREEN全域性隨機的性質來考慮服務品質的需求。如果不能滿足延遲需求，$U(q/B_a - d)$ 會表示到達的封包將會被夾取。如果到達封包所考慮的佇列長度的服務時間大於或者等於延遲需求，它會得到1，導致 P_n 等於 1。也就是說，封包無法及時得到服務，到達的封包會被夾取。相反的，$U(q/B_a - d)$ 等於0，夾取機率將只依賴在函數中其他項目計算出來。同樣地，如果漏失比率的需求大於或等於預估的漏失比率，則 $U(l - \hat{P}_l)$ 得到1。也就是說，仍然滿足漏失比率的需求，如果有必要的話，到達的封包可以被夾取。相反的，當無法滿足漏失比率的需求，到達的封包盡可能不要被夾取。在 $U(l - \hat{P}_l)$ 中所得到預估的漏失比率 $\hat{P}_l$，是藉由一個權重移動的平均值所計算出來。$\hat{P}_l$ 的式子表示式為：

$$\hat{P}_l = w_l(L/(c_b + L)) + (1 - w_l)P_l .$$

這個預估值是假設到達封包已經被夾取時所計算出來。它用來預估漏失比率，如果它的夾取使漏失比率需求未滿足，到達的封包將盡可能不被夾取。再來我們提到有關在式子(2.1)中的 $P_{e\_pre}/(1 - c_p P_{e\_pre})$，對於一些大的 c_p 而言，它可能產生大於 1 或負數的情況。在 RED 中同樣也會發生這種情況，但是在[3]中並未被提及。事實上，應該更進一步研究在[3]中有關推倒均勻特性的議題：(1) 關於標記機率變化的影響，RED 在說明均勻的特性將可變的標記機率固定，(2) 關於封包長度的影響，RED 在說明均勻的特性時，不去考慮封包的長度，(3)假設兩個標記封包(包含第二個標記封包)之間到達的封包個數大於 $1/p_b$（p_b 是封包標記機率）之機率為 0。這些問題並不容易，也是 GREEN 尚未解決的問題。為了避免 $P_{e\_pre}/(1 - c_p P_{e\_pre})$ 計算出來的值大於 1 或為負數，所以限制計算出來的值保持在[0,1]之間，也就是，

$$0 \leq P_{e\_pre}/(1 - c_p P_{e\_pre}) \leq 1 ,$$

或

$$c_p \leq 1/P_{e\_pre} \quad \text{and} \quad c_p \leq (1 - P_{e\_pre})/P_{e\_pre} \text{。}$$

最後簡化成

$$c_p \leq \min(1/P_{e\_pre}, P_{e\_pre} - 1) = (1 - P_{e\_pre})/P_{e\_pre} \text{。}$$

假如 $c_p > (1 - P_{e\_pre})/P_{e\_pre}$，則 $P_{e\_pre}/(1 - c_p P_{e\_pre})$ 之值等於 1。

在上一段，陳述 GREEN 全域隨機的性質。這段將陳述 GREEN 先期預估的性質，它是透過將夾取機率函數的部分先行算出來獲得先期性。事實上，這些部分產生了亂度機率，並且用 P_e 來表示。亂度機率是用來代表正規化的亂度，其值不會大於 1，且用以決定夾取封包的強度。而亂度是按照緩衝器的佔有率及連結頻寬的佔有率來計算。

為了先期評估的目的，亂度是由估測而得。在緩衝器佔有率的方面，一個佇列的長度將被估測。 如果佇列長度變化表成 Δq，利用權重為 w_q 的移動平均值，Δq 可以用這樣的式子來表示：

$$\Delta q \leftarrow w_q(q - q_{pre}) + (1 - w_q)\Delta q.$$

箭號在新的 Δq 和舊的 Δq 之間表示遞迴的關係，透過使用 $q + \Delta q$ 去預估下次到達時間的佇列長度。$\max(q + \Delta q, 0)/k$ 是 P_e 的構成因素之一。0 在 $\max(q + \Delta q, 0)$ 中使運算值最小為0，但不會小於0。聰明的讀者可以發現 $\max(q + \Delta q, 0)/k$ 其值未被限制小於1，但是所得到的機率值理應小於1。此限制會呈現在最後的亂度機率中。$\max(q + \Delta q, 0)/k$ 這個因素表示假如(預估的)下一個封包到達時間的佇列長度較大的話， 下個到達封包的夾取機率會比較大。在連結使用率方面，假如可用的頻寬較小的話，其亂度的機率會較大，因素 B_u/B_a 是連結的繁忙機率（繁忙率），而 B_u 是此連結在這次的封包和前個的封包期間，的平均使用的頻寬。它表示假如可用的頻寬較小，則有較大的機率去夾取到達封包。另外我們也仿效RED給予一參數 $\max_p$。根據上述提的這些因素，這個亂度的機率表示如下：

$$P_e = \max_p \min\{[\max(q + \Delta q, 0)/k](B_u/B_a), 1\} .$$

此 $\min\{[\max(q + \Delta q, 0)/k](B_u/B_a), 1\}$ 的運算用來限制亂度機率小於 1。 它不需要限制運算結果不為負數，因為 $\max(q + \Delta q, 0)/k$ 和 B_u/B_a 這兩個構成因素不會為負數。要注意的是這個機率值並不是用來計算當次到達封包的夾取機率，而是下一次到達封包的夾取機率。這也正是我們所謂先期預估的本質。在圖 2.1 中描述這個 GREEN 演算法。在圖 2.1 中，P_l

不會被 $\hat{P}_l$ 替代直到一個封包真正被夾取。$\hat{P}_l$ 只是到達封包可能被夾取的想像預估值，漏失比率不會真的被統計，除非到達封包真正被夾取或丟掉。式子(2.1)是一個同時參考了「全域隨機」和「先期預估」的夾取機率表示式，在實作上，不一定要利用式子(2.1)來計算，而可以利用「if-then-else」的邏輯語法來描述之，如圖 2.2 所示。

可以想像GREEN比RED來的複雜，這是非常合理的，透過較多的開銷(overhead)而換來更多額外的特性。然而，這不意味著在決策時需要更多的計算。因為GREEN考慮許多其他的條件限制(例如：緩衝區是全滿時、計數器超過某些限制時等)，在GREEN中有較多其他的步驟。在RED中也應該考慮這些條件，但是在[3]並沒有。當RED和GREEN相互比較時，他們的核心分別是 p_a (RED) 和 P_n (GREEN)的計算。在RED，首先應該找到平均佇列長度，然後求出 p_b，最後求 p_a 及作出決策。在GREEN，首先應該找到 P_l，然後求出 P_n。對於 P_{e_pre} 不需要在決策時計算，它在上次的封包到達時就計算過了。按照數量的比較，RED需要2個比較，2個加法，4或5個減法，4或3個乘法，和2個除法。此外，有一個線性函數和對於非空佇列的一個次冪函數。另一方面，GREEN需要2個比較，2個加法，3個減法，4個乘法，和3個除法。所以明顯的GREEN會比RED更快速的作出一個決策。

2.2、數值例子(Numerical Examples)

為了說明 GREEN 有好的效能和特性，此節給予一個簡單的系統模型，及給一這些系統參數。在 GREEN 演算法中，系統的符號顯示在表 2.1 中。由於在網際網路中，有大量多媒體流量，如 I-phone。吾人使用聲音的模型做為 I-phone 的到達程序。它的模型類似分散式 ON-OFF 的隨機過程。為簡化到達程序的參數，以 α 代表從 ON 到 OFF 的轉換機率，而 $1-\alpha$ 則代表從 OFF 到 ON 的 轉換機率。透過這個方法，ON 和 OFF 的逗留時間是參數 α 和 $1-\alpha$ 的幾何分佈。為使 ON/OFF 的逗留時間相較於封包的到達間隔時間有合理大小且不為 0，吾人將幾何隨機變數所得的值加 1 然後再乘 10。在下面的例子中，選擇 α 為 0.2。ON 的到達率是一個常數，OFF 為 0。在每一個模擬實驗，產生了 100000 個封包。圖 2.3-2.7 所顯示的 goodput ratio 為有效已接受封包對所有產生封包的封包長度的比率。當 GREEN 和 RED 做比較，有高的 goodput ratio 即表示有高的 goodput。這篇論文使用 goodput ratio 替代 goodput，因為從 goodput ratio 的值能夠容易得到有效已接受封包的數目之粗略值。它比 goodput 更具意義，goodput ratio 可看成在單位時間內有效已接受封包的大小。

2.2.1 案例 1: 負載(Loading)

緩衝區的大小和夾取門檻值各為 100 單位和 50 單位。最大封包長度值為 20 單位。延遲和漏失比率的需求值各為 75 單位和 1。測量佇列長度變化和預估漏失比率的權重的值皆等於 0.6。$\max_p$ 設定為 0.02，和 RED 的 $\max_p$ 設定值相同。負載從 0.4 至 1.6。RED 使用的參數相同，除了最大門檻值，最小門檻值和測量平均佇列長度的權重為[3]中所建議的設定值外各為 50，25，0.002 。此模擬的結果分別顯示於圖 2.3a，2.3b 和 2.3c。從圖 2.3a 可得知，當負載增加，Goodput ratio 減少。這是因為當負載增加，所接受的封包數目減少，注意，在模擬中是以固定封包數為模擬終止條件，且當負載較高，瞬間輸入太大易丟失封包，相對的無效的封包數目增加(可參考圖 2.3b 和 2.3c)，導致 goodput ratio 降低。對於 GREEN 和 RED，此特性是可確定的。另外在圖 2.3a 中也可得知，GREEN 優於 RED，因為 RED 和 GREEN 的已接受封包數幾乎相等，但 RED 有許多接受的封包是無效的，而 GREEN 則沒有，可參考圖 2.3b 和 2.3c。

2.2.2 案例 2: 漏失比率需求(Loss ratio requirement)

緩衝區的大小和夾取門檻值各為 100 單位和 50 單位。最大封包長度為 20。延遲需求值為 75 單位。測量佇列長度變化和預估漏失比率的權重的值皆等於 0.6。$\max_p$ 設定為 0.02，和 RED 的 $\max_p$ 設定值相同。負載分別為 0.8 及 1。漏失比率需求值依到達間隔時間而改變。此模擬的結果分別顯示於圖 2.4a 和 2.4b。當漏失比率需求值增加時，goodput ratio 減少。這原因可由式子(2.1)中得知，當漏失比率需求較小時，這表示 GREEN 將盡可能接受大多數的封包。在式子(2.1)中的因素 $U(l-\hat{P}_l)$ 幾乎等於 0，換言之，在式子(2.1)中的第二部分幾乎是 0。因素 $U(q/B_a - d)$ 變成了重要的條件。在此狀態下，佇列長度容易增加，使得封包在超過延遲需求或超過緩衝區大小時被夾取或丟棄。所以許多封包會漏失而導致較小的 goodput ratio。

2.2.3 案例 3: 延遲需求(Delay requirement)

緩衝區的大小和夾取門檻值各為 100 單位和 50 單位。最大封包長度為 20。漏失比率需求為 1。測量佇列長度變化和預估漏失比率的權重的值皆等於 0.6。$\max_p$ 設定為 0.02，和 RED 的 $\max_p$ 設定值相同。負載分別為 0.8 及 1。延遲需求從 50 至 95。RED 的參數相同，除了最大門檻值，最小門檻值和估測平均佇列長度的權重各為 50，25，0.002 外。此模擬的結果顯示於圖 2.5 所示。當 RED 之延遲需求較大時得到較大 goodput ratio。然而，GREEN 延遲需求並未明顯的影響 goodput ratio。這是因為 GREEN 可以保持高 goodput ratio，這也

GREEN 好的特性。原因如下：當延遲需求小時，由於封包超過延遲需求而被夾取，這也形成式子(2.1)中的第二部分的值較低，因為 c_p 通常會是 0。另一方面，當延遲需求大時，夾取機率在此種情況互相靠近。當延遲需求增加時 RED 的 goodput ratio 也增加，而且接近 GREEN。因為超過延遲的封包減少。當在較大負載時，RED 和 GREEN 二者差異變的更明顯，因為 goodput ratio 變大，在較大的負載使差異更大。事實上，小的負載和較大的負載相對的差異性是相似的。

2.2.4 案例 4: 夾取門檻值(Nipping threshold)

緩衝器大小及最大封包長度分別為 100 單位和 20 單位。延遲和漏失比率的需求值各為 100 單位和 1。延遲需求設定成 100，使得在式子(2.1)中 $U(q/B_a - d)$ 的因素不影響觀察特性。測量佇列長度變化和預估漏失比率的權重的值皆等於 0.6。$\max_p$ 設定為 0.02，和 RED 的 $\max_p$ 設定值相同。負載為 1。夾取門檻值從 50 至 100。RED 的參數相同，除了估測平均佇列長度權重為 0.002，最大門檻值等於 GREEN 的夾取門檻值，最小門檻值為最大門檻值的一半外。此模擬實驗的結果分別顯示在圖 2.6a 及 2.6b 中。如圖 2.6a 所示，對於 GREEN，當夾取門檻值增加，goodput ratio 幾乎沒有變化。這可以表示 GREEN 有穩定的 goodput ratio，能提供品質的保證，而對於 RED 而言，當夾取門檻值增加，goodput ratio 也隨之緩慢的增加，超過某一夾取門檻值時，RED 的 goodput ratio 就不再變化。另外見圖 2.6b 的平均封包長度，當夾取門檻值接近緩衝區大小時，較少的緩衝區空間去容納較大的封包，即 goodput ratio 減少。也就是說當夾取門檻值接近緩衝區大小，系統將選擇較小的封包。RED 選擇較小的封包的現象更明顯。

2.2.5 案例 5: 緩衝區大小(Buffer size)

緩衝器大小及最大封包長度分別為 50 單位和 20 單位。延遲和漏失比率的需求值各為 100 單位和 1。測量佇列長度變化和預估漏失比率的權重的值皆等於 0.6。$\max_p$ 設定為 0.02，和 RED 的 $\max_p$ 設定值相同。負載分別為 0.8 及 1。緩衝區大小從 50 至 100。RED 的參數相同，除了最大門檻值，最小門檻值和估測平均佇列長度的權重各為 50，25，0.002 外。此模擬的結果顯示於圖 2.7。如圖 2.7，buffer size 增加，goodput ratio 也隨之緩慢的增加。緩衝區大小增加使阻隔機率減少。然而，我們為了要使緩衝區夠大讓封包不會被阻隔，而增加緩衝區大小是幫助不大的。因本案例延遲及漏失要求的值皆設定成不影響觀察，所以 GREEN 與 RED 之結果相當類似。

在所有數值的例子中，固定 QoS 需求的假設對於 DiffServ [11] (core router)是適合的，在一個佇列中，相同類別的封包能提供相同的 QoS 。在這情況下，需求資料對於每個佇列是固定的。如果多個佇列共用一個共同的緩衝區，per-class 或 per-flow 資料應該也可以使用 GREEN 演算法。透過這方法，我們相信 per-class 資料在 DeffServ 結構下會夠少。

第三節、The GREEN algorithm for multiple QoS classes

3.1 GREEN 演算法的修正

在第二節中，我們介紹對於單一資料流型態的 GREEN 演算法，只有一種類別的服務品質需求。事實上，在網際網路中，每個節點可能藉由不同的資料流得到不同種類的服務品質。所以在本節中，為了要符合網際網路實際的情形，我們加強 GREEN 演算法。主要是修改 GREEN 演算法中的一些參數，使得每個節點不只有單一種類的資料流，有多個種類的資料流下也能使用 GREEN 演算法做緩衝區管理。表 3.1 提供了在多個種類資料流的 GREEN 演算法中所利用的符號。此外，在演算法中每一個參數的初始值也列在表 3.1 中。不管單一種類或多個種類資料流的 GREEN 演算法，GREEN 使用共同的緩衝區做緩衝區管理並以佇列的長度來當做是判斷的標準。如果一個封包到達了，而且剛好網路的狀況也超過了該門檻值，則稱為夾取機率(nipping probability)的機率值(P_n)會被計算出來，以決定是否夾取封包。如果封包到達時，緩衝區已經滿載了，那麼該封包就會被丟棄。

當考慮多種 QoS 需求時，夾取機率可以用這樣的式子來表示：

$$P_n = \max\left\{U(q/B_a - d[i]), [(P_{e\_pre}/(1-c_p[i]P_{e\_pre})]U(l[i] - \hat{P}_l[i])\right\} \quad (3.1)$$

當第 i 類資料流發現其延遲需求不能滿足，$U(q/B_a - d[i])$ 會表示到達的封包將會被夾取。同樣地，當第 i 類資料流的漏失比率的需求大於或等於預估的漏失比率，$U(l - \hat{P}_l[i])$ 得到 1。也就是說，仍然滿足漏失比率的需求，如果有必要的話，到達的封包可以被夾取。相反的，當無法滿足漏失比率的需求，到達的封包盡可能不要被夾取。在 $U(l - \hat{P}_l[i])$ 中預估的漏失比率 $\hat{P}_l[i]$，仍是藉由一個權重移動的平均值所計算出來。$\hat{P}_l[i]$ 的式子表示式為：

$$\hat{P}_l[i] = w_l[i](L[i]/(c_b[i] + L[i])) + (1 - w_l[i])P_l[i].$$

如同單一資料流，多資料流之

$P_{e\_pre}/(1-\sum_i c_p[i]P_{e\_pre})$，可能大於 1 或小於 0。為了避免 $P_{e\_pre}/(1-\sum_i c_p[i]P_{e\_pre})$ 計算出來的值大於 1 或為負數，所以限制計算出來的值保持在[0,1]之間，同樣可得當 $\sum_i c_p[i] > (1-P_{e\_pre})/P_{e\_pre}$ 時，$P_{e\_pre}/(1-\sum_i c_p[i]P_{e\_pre})$ 之值就給予 1。

在亂度函數方面，因其主要因素為佇列長度及頻寬使用率，所以與是否為多資料流類型無關，即仍為

$$P_e = \max_p \min\{[\max(q+\Delta q,0)/k](B_u/B_a),1\}.$$

在圖 3.1 中描述這個 GREEN 演算法。在圖 3.1，$P_l[i]$ 不會被 $\hat{P}_l[i]$ 替代直到一個封包真正被夾取。$\hat{P}_l[i]$ 只是到達封包可能被夾取的想像預估值，漏失比率不會真的被統計，除非到達封包真正被夾取或丟掉。式子(3.1)是一個同時參考了「全域隨機」和「先期預估」的夾取機率表示式，在實作上，不一定要利用式子(3.1)來計算，而可以利用「if-then-else」的邏輯語法來描述之，如圖 3.2 所示。

3.2 數值例子(Numerical Examples)

為了說明在多個種類的資料流使用 GREEN 的方法有好的效能和特性，此節給予一個簡單的系統模型，及給一些系統參數。在 GREEN 演算法中，系統的符號顯示在表 3.1 中。假設這些到達的封包有兩種類型的資料型態進入緩衝區。一種的模型類似分散式 ON-OFF 的隨機過程，而另一種模型類似 Bernulli 的隨機過程。各以 2:8、8:2、5:5 的比率來摸擬 100000 個封包。圖 3.3-3.4 所顯示的 goodput ratio 為有效已接受封包對所有產生封包的封包長度的比率，以 GREEN 和 RED 做比較。由於多類型輸入較為複雜，所以尚有相當多案例仍待研究，這些研究也正在進行。

3.2.1 案例 1: 延遲需求(Delay requirement)

緩衝區的大小和夾取門檻值各為 100 單位和 50 單位。最大封包長度為 20。漏失比率需求為 1。測量佇列長度變化和預估漏失比率的權重的值皆等於 0.6。$\max_p$ 設定為 0.02，和 RED 的 $\max_p$ 設定值相同。延遲需求從 50 至 100。RED 的參數相同，除了最大門檻值，最小門檻值和估測平均佇列長度的權重各為 50，25，0.002 外。此模擬的結果顯示於圖 3.3a-3.3c 所示。當 RED 之延遲需求較大時得到較大 goodput ratio。然而，GREEN 延遲需求並未明顯的影響 goodput ratio。這是因為 GREEN 可以保持高 goodput ratio，這也是 GREEN 好的特性。原因如下：當延遲需求小時，由於封包超過延遲需求而被夾取，這也形成式子(3.1)中的第二部分的值較低，因為 $c_p[i]$ 通常會清成 0。另一方面，當延遲需求大時，夾取機率在此種情況互相靠近。當延遲需求增加時 RED 的 goodput ratio 也增加，而且接近 GREEN。我們也可以看出愈 burstiness goodputratio 就愈低。

3.2.2 案例 2: 最大的封包長度(Maximum packet length)

緩衝區的大小和夾取門檻值各為 100 單位和 50 單位。延遲和漏失率的需求值各為 50 單位和 1。測量佇列長度變化的權重和預估漏失率的值皆等於 0.6。最大封包長度從 5 至 40。$\max_p$ 設定為 0.02，和 RED 的 $\max_p$ 設定值相同。與 RED 的參數相同，除了最大門檻值，最小門檻值和測量平均佇列長度的權重各為 50，25，0.002。在實驗模擬結果顯示在圖 3.4a-3.4c。在圖中 GREEN 的 goodput ratio 隨著最大封包長度增加而減少，因為較大的封包長度是較難進入緩衝區。而在圖中 RED 的 goodput ratio 隨著最大封包長度增加而增加，因為最大封包長度增加會導致封包到達率變小，這也形成記錄自上次封包夾取時到現在，進入緩衝區的封包數量值變小。使得封包不易被丟棄，所以 goodput ratio 緩慢的增加。我們也可以觀察到不管是 GREEN 或 RED 變化的幅度都不大。我們也可以看出愈 burstiness goodput ratio 就愈低。

第 四節、總結

在多媒體網路裡，緩衝區管理將會是服務品質控制的一個重要元件。因此，本篇論文發展出了 GREEN 演算法。GREEN 承襲了現今各個緩衝區管理演算法的主要核心，即「隨機」與「先期」的並用，而發展出一套完整的方法。此外，GREEN 更進一步地利用整體性地考量隨機機率，而將「隨機」擴展為「全域隨機」。另一方面，還利用了先期預估網路狀態，而將原本的「先期」擴展為「先期預估」。我們從不同觀點設計 GREEN 演算法可以得到不同的結果。在數值例子中，由模擬結果可以得知，GREEN 比 RED 的效能好。如同預期的，GREEN 的 goodputs 比 RED 更好。另一方面，由廣泛模擬各種情況去探討 GREEN 的特性。在 GREEN 的參數中仍有許多複雜的關係，需要更深一層的研究。

這篇論文可以被視為是研究緩衝區管理演算法系列的開始，並且依 QoS 需求的資訊和先前預估的一些決策因素來做緩衝區管理。而在某些觀點，把一個違規封包加標籤再把它放入網路中進一步的觀察而不直接丟棄它，將得到更好的效能。因此，考慮標記替代丟棄或標記丟棄綜合使用，又是另一個吸引人的問題。

References

[1] E. Hashem, “Analysis of random drop for gateway congestion control”, *Rep. LCS TR-465*, *Lab. for Comput. Sci.*, *M.I.T.*, p.103, 1989.

[2] L. Zhang, “A new architecture for packet switching network protocols”, *MIT/LCS*, *Lab. For Comput. Sci.*, *M.I.T.*, 1998.

[3] S. Floyd and V. Jacobson, ”Random Early Detection Gateways for Congestion Avoidance,” *IEEE/ACM trans. networking*, vol.1, no.4, pp.397-413, 1993.

[4] O. Elloumi and H. Afifi, “Improving RED algorithm performance in ATM networks,” *IEEE GLOBECOM '97*, vol.2, pp.1062 -1066, 1997.

[5] R. Guerin and V. Peris, ”Quality-of-service in packet network: basic mechanisms and direction”, *Computer Networks*, vol. 31, 1999, pp.169-189.

[6] K.V. Waal, M. Dirksen and D. Brandt, “Implementation of a police criterion calculator based on the leaky bucket algorithm,” *IEEE Globecom'93*, pp. 713-718, 1993.

[7] J. M. Pitts and J. A. Schormans, ”Introduction to ATM design and performance,” Wiely, New York, 1997.

[8] D. Gan, and S. McKenzie, ”Traffic policing in ATM networks with multimedia traffic: the super leaky bucket,” *Computer Communication*, vol. 22, pp.439-450, 1999.

[9] J. B. Pippas and I. S. Venieris, “A RED Variation for Delay Control,” *IEEE ICC 2000*, vol.1, pp.475 –479, 2000.

[10] G. Hasegawa, T. Matsuo, M. Murata, and H. Miyahara, “Comparisons of packet scheduling algorithms for fair service among connections on the Internet,” *IEEE INFOCOM 2000*, vol.3, pp.1253 –1262, 2000.

[11] S. Blake, D. Blake, M. Carlson, E. Davies, Z. Wang and W. Weiss, "An architecture for differentiated services," *Internet RFC 2475*, Dec. 1998.

[12] W. H Amy and S. Mischa, ”Achieving bounded fairness for multicast and TCP traffic in the Internet,” *Proceedings of the ACM SIGCOMM '98 conference on Applications, technologies, architectures, and protocols for computer communication, 1998.*

[13] A. Mankin, “Random drop congestion control,” *Proceedings of the ACM symposium on Communications architectures & protocols, 1990, Pages 1 – 7.*

[14] RFC2001.

符號	意義	初始值
P_n	夾取機率	Null
T_n	夾取臨界值	Given
Δq	佇列長度變化	0
q	目前封包到達的佇列長度	Null
q_{pre}	先前封包到達的佇列長度	0
w_q	計算佇列長度變化的權重	Given
P_{e_pre}	先前封包到達所計算的亂度機率	0
P_e	目前封包到達所計算的亂度機率	Null
B_u	在兩個連續的封包到達期間使用的連結頻寬平均值	0
B_a	所有連結的頻寬	Given
$\max_p$	最大亂度機率	Given
P_l	一個封包夾取/丟棄後的漏失率	0
$\hat{P}_l$	預估的漏失率,假定夾取到達封包	Null
w_l	計算預估漏失率的權重	Given
l	漏失率的需求	Given
c_p	紀錄自上次封包夾取時到現在，進入緩衝區的封包數量	0
c_b	總計 c_p 封包的大小	0
L	目前到達的封包大小	Null
$L_{\max}$	最大的封包大小	Given
d	延遲需求	Given
k	緩衝區大小	Given

表 2.1、符號說明

Initialization: Set the initial values (see Table 2.1)

For each packet arrival

$\Delta q \leftarrow w_q(q - q_{pre}) + (1 - w_q)\Delta q$

$P_e = \max_p \min\{[\max(q + \Delta q, 0)/k](B_u / B_a), 1\}$

$q_{pre} = q$

$\hat{P}_l = w_l(L/(c_b + L)) + (1 - w_l)P_l$

if (Buffer full)

Drop the packet

$P_l = \hat{P}_l$

$c_p = 0$

$c_b = 0$

else if $(q > T_n)$

if $(c_p > (1 - P_{e\_pre})/P_{e\_pre})$

$P_n = (U(q/B_a - d), U(l - \hat{P}_l))$

else

$P_n = \max\left\{U(q/B_a - d), [(P_{e\_pre}/(1 - c_p P_{e\_pre})]U(l - \hat{P}_l)\right\}$

$P_{e\_pre} = P_e$

if (a random value < P_n)

Nip the packet

$P_l = \hat{P}_l$

$c_p = 0$

$c_b = 0$

else

$c_b += L$

$c_p ++$

else

Accept the packet

$c_p = 0$

$c_b = 0$

圖 2.1、GREEN 演算法

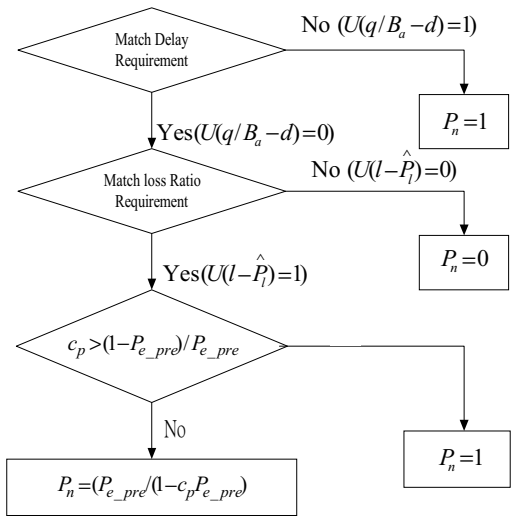


圖 2.2、 The if-then-else logic for nipping probability

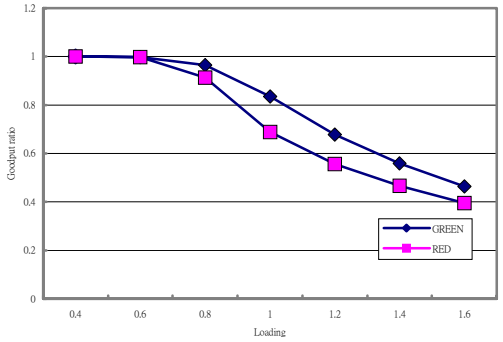


圖 2.3a、改變負載之 Goodput ratios

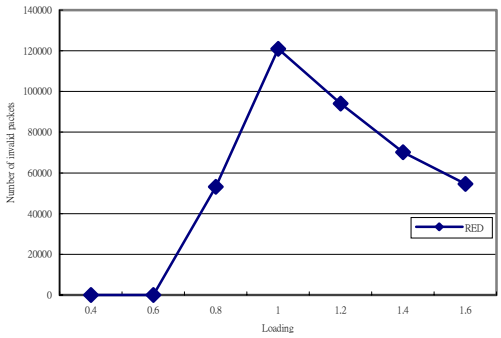


圖 2.3b、負載對無效的封包數目

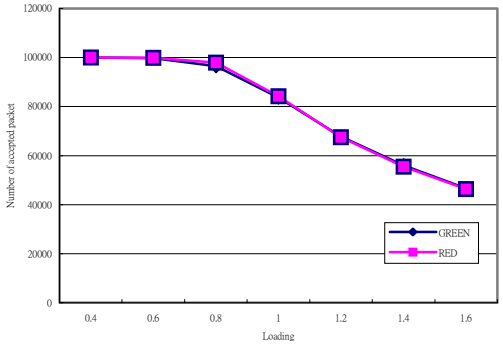


圖 2.3c、負載對所接受的封包數目

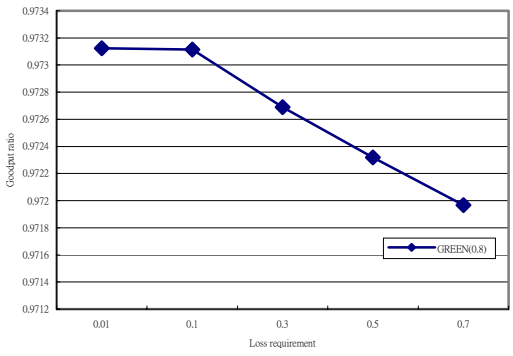


圖 2.4a、改變漏失比率需求對 Goodput ratios

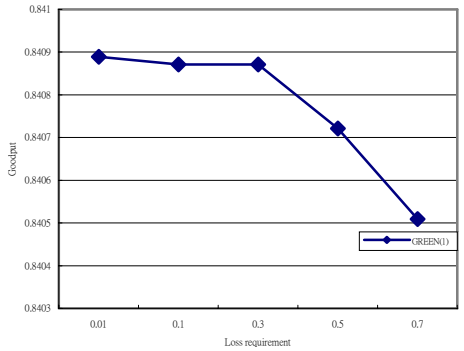


圖 2.4b、改變漏失比率需求對 Goodput ratios

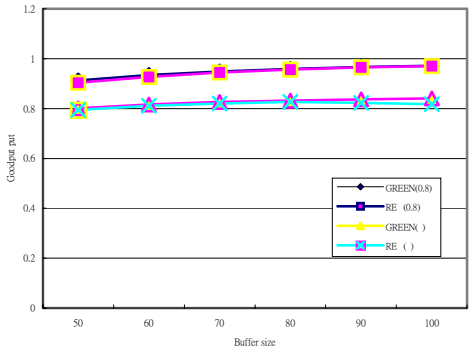


圖 2.7a、改變緩衝區大小對 Goodput ratios

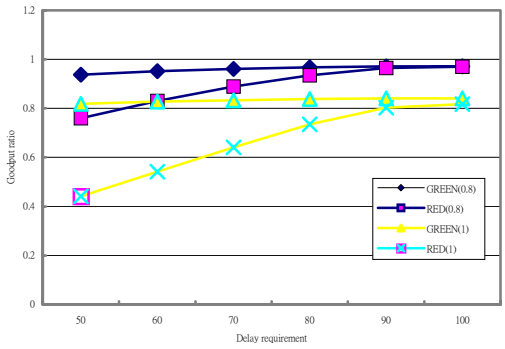
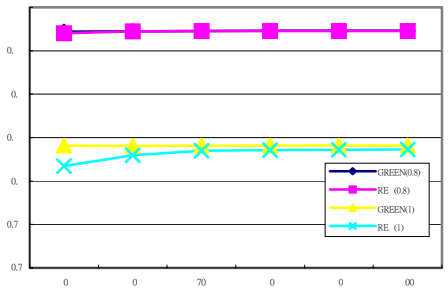


圖 2. 5、改變延遲需求對 Goodput ratios



2.6a、改變夾取臨界值對 Goodput ratios

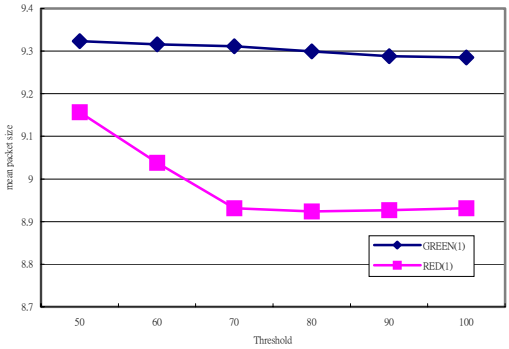


圖 2. 6b、改變夾取臨界值對平均封包長度

符號	意義	初始值
P_n	夾取機率	Null
T_n	夾取臨界值	Given
Δq	佇列長度變化	0
q	目前封包到達的佇列長度	Null
q_{pre}	先前封包到達的佇列長度	0
w_q	計算佇列長度變化的權重	Given
P_{e_pre}	先前封包到達所計算的亂度機率	0
P_e	目前封包到達所計算的亂度機率	Null
B_u	在兩個連續的封包到達期間使用的連結頻寬平均值	0
B_a	所有連結的頻寬	Given
$\max_p$	最大亂度機率	Given
$P_l[i]$	對於類別 i 的一個封包夾取/丟棄後的漏失率	0
$\hat{P}_l[i]$	對於類別 i 去預估的漏失率,假定夾取到達封包	Null
$w_l[i]$	對於類別 i 去計算預估漏失率的權重	Given
$l\ i]$	對於類別 i 的漏失率的需求	Given
$c_p[i]$	對於類別 i，紀錄自上次封包夾取時到現在，進入緩衝區的封包數量	0
$c_b[i]$	對於類別 i 去總計 c_p 封包的大小	0
$L\ i]$	對於類別 i 的目前到達的封包大小	Null
$L_{\max}$	最大的封包大小	Given
$d\ i]$	對於類別 i 的延遲需求	Given
k	緩衝區大小	Given

表 3.1、對於類別 i 的符號說明

Initialization: Set the initial values (see Table 3.1)

For each packet arrival

$\Delta q \leftarrow w_q(q - q_{pre}) + (1 - w_q)\Delta q$

$P_e = \max_p \min\{[\max(q + \Delta q, 0)/k](B_u / B_a), 1\}$

$q_{pre} = q$

$\hat{P}_l[i] = w_l[i](L[i]/(c_b[i] + L[i])) + (1 - w_l[i])P_l[i]$

For each i

if (Buffer full)

Drop the packet

$P[i] = \hat{P}[i]$

$c[i] = 0$

$c_b[i] = 0$

else if $(q > T_n)$

if $(\sum c_p[i] > (1 - P_{e\_pre})/P_{e\_pre})$

$P_n = (U(q/B_a - d[i]), U(l[i] - \hat{P}_l[i]))$

else

$P = \max\{U(q/B - d[i]), [(P_{\_}/(1 - \sum c[i]P_{\_}))]U[l[i] - \hat{P}_l[i]]\}$

$P_{e\_pre} = P_e$

if (a random value < P_n)

Nip the packet

$P_l[i] = \hat{P}_l[i]$

$c[i] = 0$

$c_b[i] = 0$

else

$c_b[i] += L[i]$

$c_p[i]++$

else

Accept the packet

$c_p[i] = 0$

$c_L[i] = 0$

圖 3.1、對於類別 i 的 GREEN 演算法

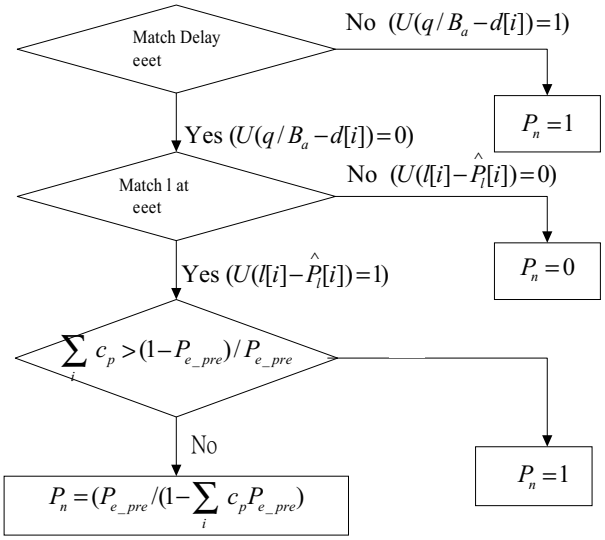


圖 3.2 The if-then-else logic for nipping probability

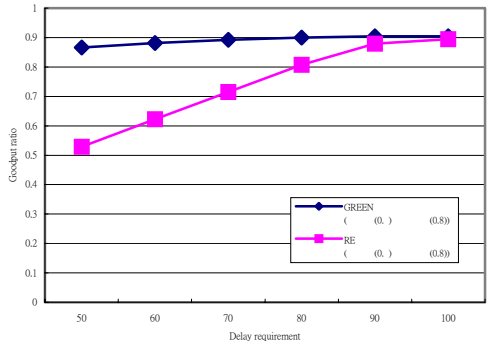


圖 3.3a、改變延遲需求對 Goodput ratios

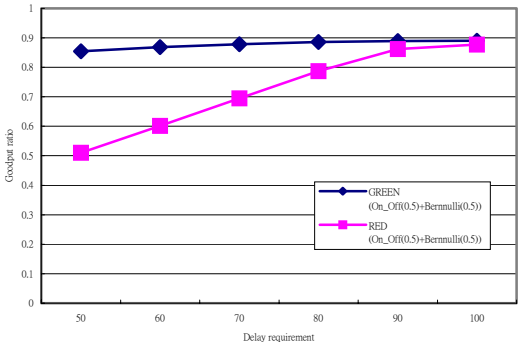


圖 3.3b、改變延遲需求對 Goodput ratios

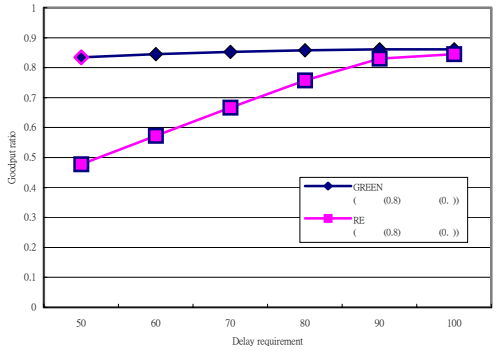


圖 3.3c、改變延遲需求對 Goodput ratios

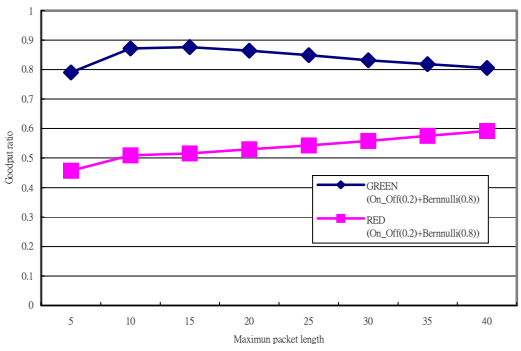


圖 3.4a、改變最大封包長度對 Goodput ratios

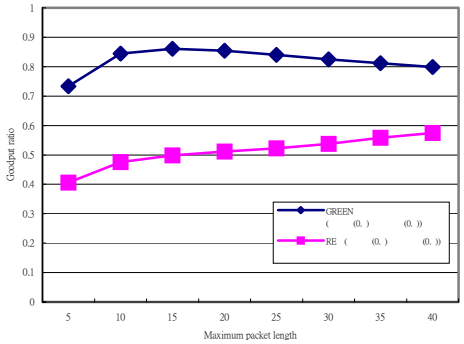


圖 3.4b、改變最大封包長度對 Goodput ratios

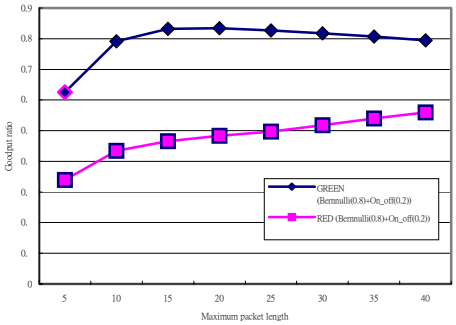


圖 3.4c、改變最大封包長度對 Goodput ratios