一種依據連線緩衝區使用率的流量控制演算法

朱延平　　黃一泓　　曾晞凌

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

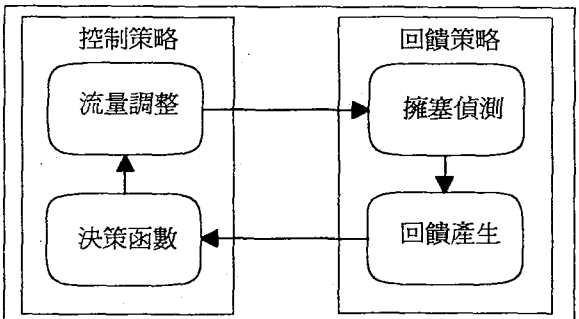
本篇文章提出一種新的流量控制演算法,經由在封包中加入兩個位元,做為回饋訊息。發送端送出封包後,交換節點依連線緩衝區使用率來設定位元值,封包到達目的端後,此二位元被複製到回應中,隨回應傳回發送端,發送端根據所得回饋訊息調整流量。模擬實驗結果顯示,本文方法不僅有效控制網路中的交通流量,並且不需借助交換節點的排程演算法便能維持公平性。

1. 簡介

在這個科技進步的時代,資訊的流通日趨頻繁,資料的傳遞一日千里,愈來愈多的資訊透過網路相互傳遞,導致Internet的交通流量急遽增加,網路擁塞情形日趨嚴重,所以解決Internet擁塞情形的流量控制演算法(flow control algorithm)便格外重要。

流量控制演算法包含兩部分:回饋策略(feedback strategy)與控制策略(Control strategy)[1、2]。兩者之間的關係可用圖一表示。回饋策略包含了擁塞偵測(congestion detection)及回饋產生(feedbckdbk generation)。擁塞偵測依據輸出線使用率(output link utilization)或緩衝區使用率(buffer utilization)來決定網路是否發生擁塞;回饋產生根據偵測結果來設定回饋訊息(feedback information)。控制策略則包括*決策函數(decision function)*與*流量調整(load adaption)*。發送端收到回饋訊息後,決策函數參考回饋訊息決定增加或減少流量,流量調整再決定增減量。簡言之,網路透過回饋策略將交通狀況回饋給發送端,發送端再參考回饋訊息作為調整流量的依據。發送端可以速率(rate-based)(例如:秒/封包)或視窗大小(window-based)(即網路中所允許的最多封包數)的方式送出封包。

當網路的交通流量超過所能承擔的負荷量時,便會造成擁塞。依據調整時機的不同,流量控制演算法分為被動的(reactive)與主動的(preventive)兩類[1、3]。被動式的流量控制演算法是等到網路已經發生擁塞,造成封包流失,這時才減少交通流量,使網路脫離擁塞狀態,例如TCP Tahoe[4]與CUTE[5]等便屬於



圖一 流量控制演算法組成要件

此種流量控制演算法;主動式的流量控制演算法是透過控制交通流量,避免網路造成擁塞,例如DECbit[6、7]便是一種主動的流量控制演算法。

在 TCP Tahoe 演算法中,回饋策略將網路分為擁塞狀態與非擁塞狀態。它並不使用明確的回饋訊息,而是以發送端發現封包流失,導致計時器(timer)中斷作為間接的回饋訊息,表示網路處於擁塞狀態,此時將視窗大小設為 1;否則便認為網路並未發生擁塞。每當收到回應(acknowledgement)時便增加視窗大小。DECbit演算法在封包中加入一個擁塞指示位元(congestion indication bit)作為回饋訊息。此位元初始值為0,發送端送出封包後,若網路任何一個交換節點的平均佇列長度(queue length)大於或等於1,便將擁塞指示位元設為1,封包到達目的端後,此位元被複製到回應中,隨回應傳回發送端,發送端收集兩個視窗大小的封包後,若超過一半的擁塞指示位元值為1,便降低視窗;否則便增加視窗長度。

近年來有不少探討關於 TCP Tahoe 演算法的論文發表[8、9、10]。我們發現,由於 TCP Tahoe 演算法的調整策略,導致計時器週期性的發生中斷,使視窗大小、佇列長度週期性的振盪,造成整體產能(throughput)下降及封包間的來回旅行時間(Round Trip Time, RTT)過長。因此我們提出新的流量控制演算法,讓交換節點緩衝區維持一定的使用率,達到充分利用網路資源,獲致最大的整體產能,且維持合理的來回旅程時間。

在本篇文章第二節裡,將介紹我們所提出的演算法。第三節中我們透過模擬實驗將本文方法與TCP Tahoe作一比較。最後我們提出結論。

2. 演算法模型

我們希望藉由使交換節點始終處於忙碌狀態，達到充分利用網路資源，獲致最大的整體產能(maximal global throughput)，因此必須讓交換節點緩衝區保持封包存在的狀態；同時控制交換節點緩衝區的使用率，令緩衝區內封包數不致過多，使得封包的來回旅行時間太長，甚至發生封包因交換節點緩衝區溢滿(overflow)而流失，造成產能的下降，所以必須將緩衝區的使用率控制在一定的範圍內，如此便能維持良好的網路傳輸效能及合理的來回旅行時間，並避免網路因擁塞而造成封包流失。

我們提出的演算法主要分成兩部分：交換節點的回饋策略及發送端的控制策略。我們在封包中加入兩個擁塞位元，藉以作為網路交通狀況的參考依據。我們希望網路中的每位使用者都能公平的使用頻寬，所以將交換節點緩衝區虛擬地平均分給每條連線(實際上緩衝區是所有連線共享的)。在發送端與目的端建立連線(connection)後，每當封包流經交換節點時，交換節點便依據連線緩衝區(per-connection buffer)的使用率來設定擁塞位元。在封包到達目的端後，目的端便將擁塞位元複製到回應封包送回發送端，發送端便以擁塞位元來作為調整視窗大小的依據(如圖二)。所以當交通流量過多時，網路會通知超用頻寬的使用者減少流量，而不影響其它使用者，達到選擇性回饋(selective feedback)的效果。接著先介紹交換節點的回饋策略，再介紹發送端的控制策略。

依據連線緩衝區使用率的高低，將其由高至低區分成4個階段來設定擁塞位元值。由於n個位元最多可表示2^n種狀態，因此利用兩個位元來表示交換節點緩衝區的使用率。我們以C語言演算法來說明回饋策略及控制策略：

發送端送出封包時，封包的擁塞位元初始化設定

```
pkt->venbit_i = 0;
```

每當封包流經交換節點時：

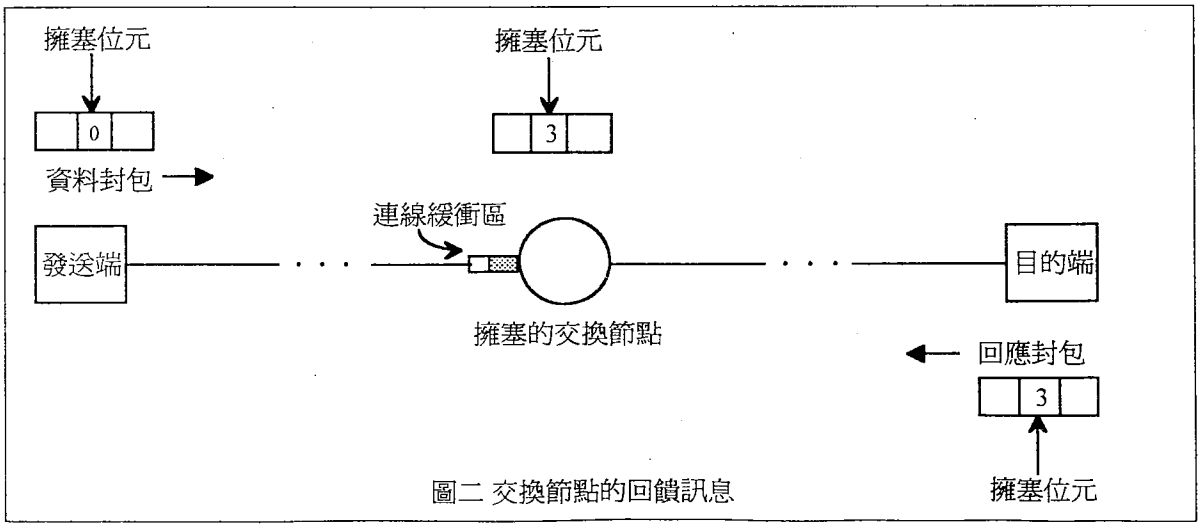
```
if(bytes_in_q_i ≥ aver_conv_size_i * S_3)
  pkt->venbit_i = 3;
else
  if(bytes_in_q_i ≥ aver_conv_size_i * S_2)
    pkt->venbit_i = 2;
  else
    if(bytes_in_q_i ≥ aver_conv_size_i * S_1)
      pkt->venbit_i = 1;
```

其中，pkt->vnebit$_i$連線i封包的擁塞位元，bytes_in_q$_i$為連線i緩衝區內所存位元組數，aver_conv_size$_i$為連線i所配置緩衝區大小，S_1、S_2及S_3為連線緩衝區使用率。

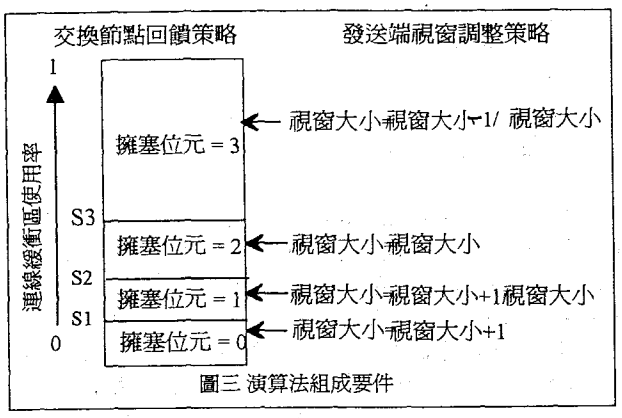
當目的端將擁塞位元拷貝至回應封包送回發送端後，發送端依據擁塞位元值作視窗大小調整(如圖三)：

```
if(pkt->venbit==0)
/* window = window + 1 */
  {window+=1;
   counter1=0;
   counter2=0;}
else
  if(pkt->venbit==1)
/* window = window + 1/window */
    if(++counter1 ≥ window)
      {window+=1;
       counter1=0;
       counter2=0;}
    else
      if(pkt->venbit==2)
/* window size unchanged: operating region */
        {window=window;
         counter1=0;
         counter2=0;}
      else
        if(pkt->venbit==3)
/* window = window - 1/window */
          if(++counter2 ≥ window)
            if(window>1)
              {window-=1;
               counter1=0;
               counter2=0;}
```

其中，window為視窗大小，counter1、counter2分別為記錄視窗增加及減少的參數，當counter1或



圖二 交換節點的回饋訊息



圖三 演算法組成要件

counter2等於視窗大小時，分別將視窗大小加1或減1。此外，每次調整視窗時，便將counter1及counter2重設為0，避免過度累積，產生不適當的調整。

增加的調整策略類似 TCP Tahoe，我們可以看出，當連線緩衝區使用率低時(擁塞位元=0)，發送端每收到回應時便將視窗大小加1，讓視窗呈指數增加，連線緩衝區使用率較高時(擁塞位元=1)，我們便減緩增加的速度，讓視窗呈線性增加；當連線緩衝區使用率到達某一階段時(擁塞位元=3)，我們便減少視窗大小。因網路傳輸延遲的緣故，擁塞位元所傳回的網路訊息並非目前的狀況，所以調整視窗長度的策略除了增加與減少之外，還包含一部分不作任何調整(擁塞位元=2)，這部分是作為緩衝地帶，使連線緩衝區內封包維持在合理的範圍內，避免視窗因過度調整而造成擁塞或未充分利用頻寬。如此便能減小視窗振盪幅度，甚至維持不變，使網路處於穩定狀態，獲致較佳的產能及來回旅行時間。

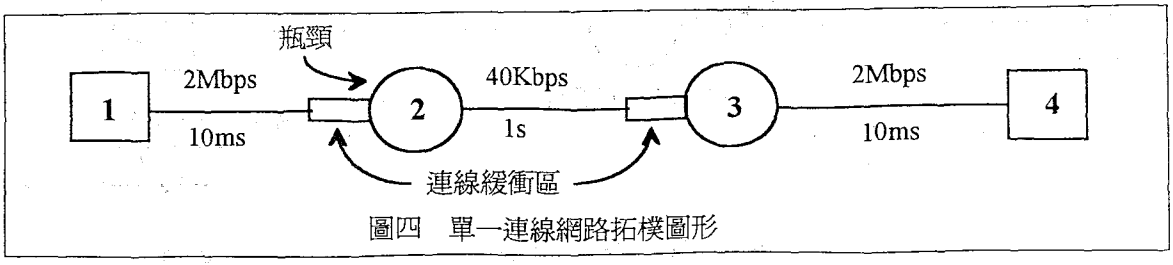
3. 模擬結果與比較

為了能較為清楚的觀察本文方法與TCP Tahoe兩者之間的差異，我們透過模擬實驗將本文所提出的演算法與 TCP Tahoe 演算法作一比較。本文以 U.C. Bekerley 所發展出的 REAL 4.0[11]作為網路模擬實驗的軟體。透過單一連線實驗(single connection experiment)，我們可以更加瞭解演算法運作的行為；公平性實驗可測試演算法在不同路徑時，是否能達到公平性；而橫截交通實驗(cross-traffic connection experiment)則較能模擬眞實網路中的交通行為模式，因此在本節中，我們分別設計了這三種網路模擬實驗。實驗中假設只有當交換節點緩衝區溢滿的時候，封包才會流失，而刪除封包時選擇最後一個封包刪除。交換節點緩衝區的排班策略為先到先服務(FCFS)。當封包的到達速率(input rate)大於輸出速率(output rate)時，節點便會發生擁塞，我們稱此節點為瓶頸(bottleneck)。為了評估演算法的效率，在每個實驗中均有瓶頸存在。此外，對於連線緩衝區使用率S_1、S_2與S_3的設定分別為1/10、1/5及1/3，這是因為當交換節點緩衝區較小時，設定太高容易造成緩衝區溢滿，設定太低則無法充分利用頻寬，且模擬實驗顯示，我們的設定甚為適當。基本上，S_1、S_2、S_3值的設定與交換節點緩衝區大小及連線個數有關，因此我們認為應由系統管理者決定，在何種情況下的設定會有較好的效果。

圖四為單一連線網路拓樸圖形，節點1為發送端，節點2與3為交換節點，節點4為目的端。發送端與交換節點及交換節點與目的端之間傳輸線的頻寬為每秒2M位元，傳輸延遲為10微秒；交換節點間的頻寬為每秒40K位元，傳輸延遲為1秒，為本實驗的瓶頸，每個交換節點緩衝區大小為5000位元組。本實驗發送端將傳送500K位元組，每個資料封包大小均為500位元組，回應封包大小為40位元組，實驗模擬時間為300秒。

圖五是TCP Tahoe與本文方法瓶頸佇列(queue)的變化狀況。由圖五(a)可看出封包因緩衝區週期性的溢滿而被刪除，佇列呈現週期性振盪。在圖五(b)中，因緩衝區的使用率被控制在一定的範圍內，佇列則呈現較小振盪。圖六則是視窗大小的變化情形。圖六(a)為TCP Tahoe的視窗變化，由此圖得知，因封包被刪除，使計時器週期性的發生中斷，視窗降低為1並重新開始運作，導致視窗呈現週期性的振盪。在圖六(b)中，因緩衝區的使用率控制在一定的範圍內，所以視窗大小逐漸趨於一固定值，最後當資料即將傳送完畢時，視窗長度隨著瓶頸佇列長度減少而增加。圖七為本文方法與TCP Tahoe的產能比較。我們每隔20秒統計封包傳輸量。由於瓶頸的頻寬為每秒40K位元，因此連線每20秒最多可傳送200個封包。我們可以看出，TCP Tahoe因計時器週期性的發生中斷，視窗呈現巨幅振盪，傳輸量並不穩定，並且未能充分利用網路資源。本文方法由於緩衝區內始終存有封包，穩定的維持每20秒傳送200個封包的最大傳輸量，因此在120秒時就已經傳輸完成，而TCP Tahoe在160秒時才傳完。

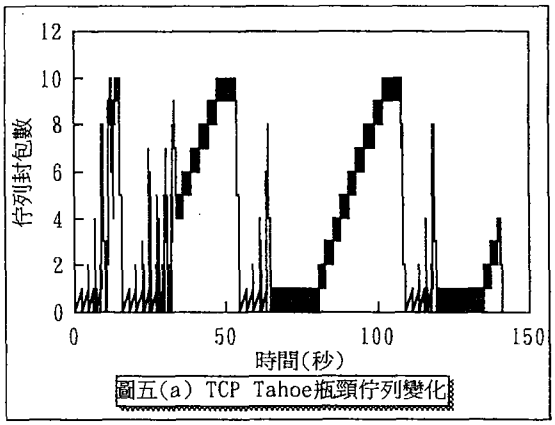
第二個模擬實驗測試當連線間的距離不同時，演算法能否達到公平。實驗模型如圖八。圖九為本文方法的視窗變化。最初連線1的視窗持續增加，連線2加入後，連線1因連線緩衝區長度減半，連線緩衝區使用率相對增加，隨後的擁塞位元均被設定為3，視窗長度因而降低，將頻寬分給連線2，此一現象為選擇性回饋的效果。當緩衝區使用率穩定後，兩者視窗均維持不變，因連線2的延遲-頻寬乘積(delay-bandwidth product)較大，所以視窗比連線1大。TCP Tahoe的視窗變化如圖十。在圖中可看出，兩者視窗均巨幅振



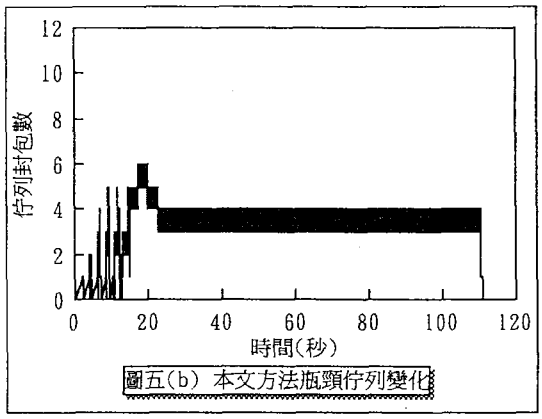
圖四 單一連線網路拓樸圖形

盪，且長度差距頗大，連線2因為視窗較大，由於FCFS的結果，使得它的封包比連線1較易被服務，而佔用大部分頻寬，因此300秒前便傳送完畢。圖十一為每20秒統計連線間的產能比較圖。瓶頸的頻寬為每秒20K位元，因此連線每20秒最多可傳送100個封包。從圖十一(a)可知本文方法不但達到最大整體產能，並且維持公平性；圖十一(b)中顯示兩者傳輸量極不公平。

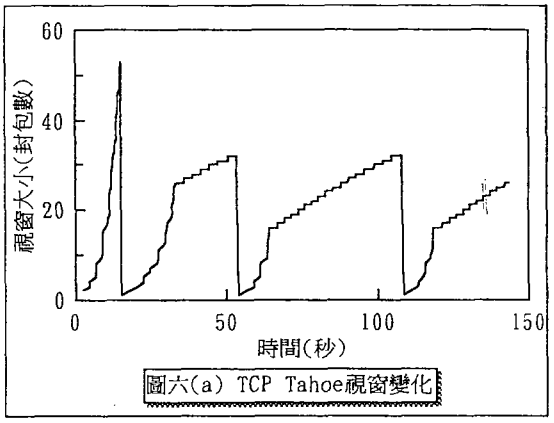
為了模擬眞實網路中的交通流量，我們令整體橫截交通的資料量呈現逐漸增加後再逐漸減少，來觀察當連線個數增加及減少時，演算法是否能如單一連線般的運作。圖十二為橫截交通連線網路拓樸圖形。此實驗是當主連線(primary connection)1開始傳送資料後，橫截交通連線隨後加入。連線1在0秒時開始傳送，資料量為750K位元組。橫截交通連線2至8則從5秒開始，每隔5秒陸續加入傳送。每條橫截交通連線所傳送的資料量均為250K位元組。每個交換節點緩



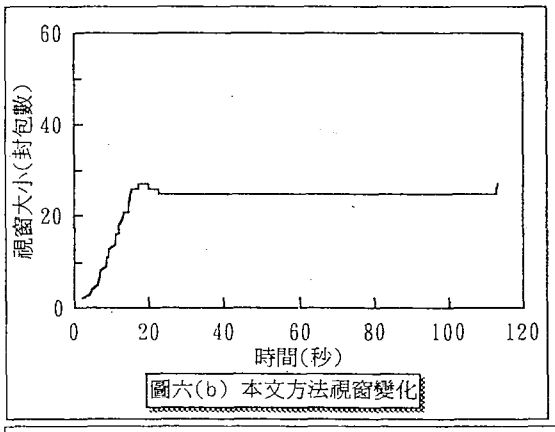
圖五(a) TCP Tahoe瓶頸佇列變化



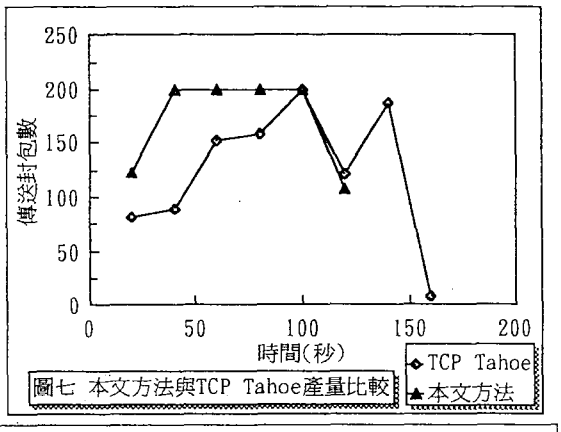
圖五(b) 本文方法瓶頸佇列變化



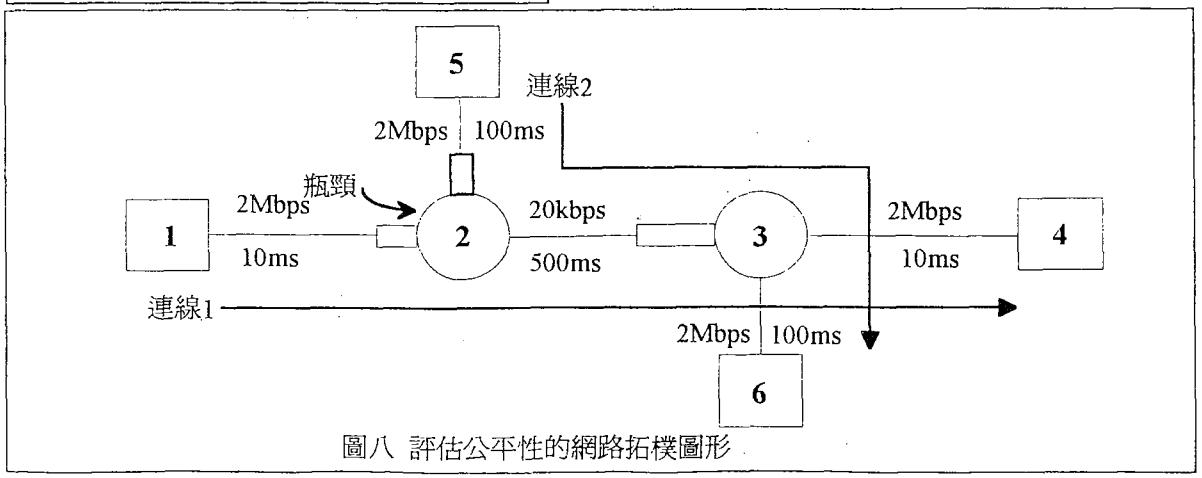
圖六(a) TCP Tahoe視窗變化



圖六(b) 本文方法視窗變化



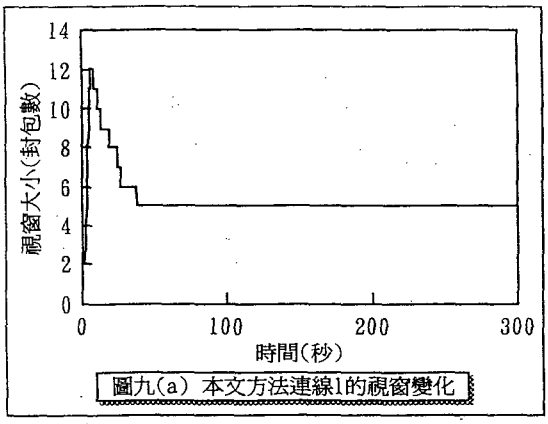
圖七 本文方法與TCP Tahoe產量比較



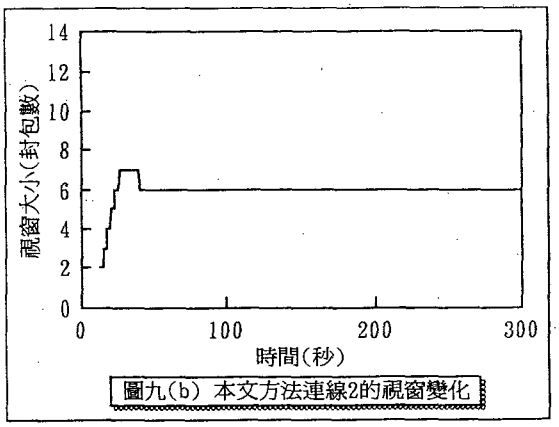
圖八 評估公平性的網路拓樸圖形

衝區大小為40000位元組，實驗模擬時間為1100秒，其餘實驗環境設定如單一連線實驗。

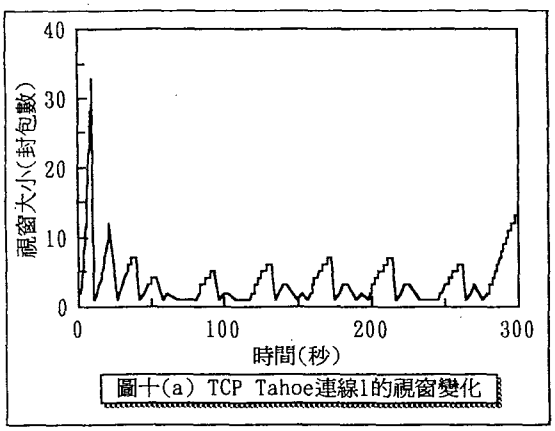
圖十三為連線1的瓶頸佇列變化情形。TCP Tahoe連線1的佇列變化如圖十三(a)，由於剛開始時



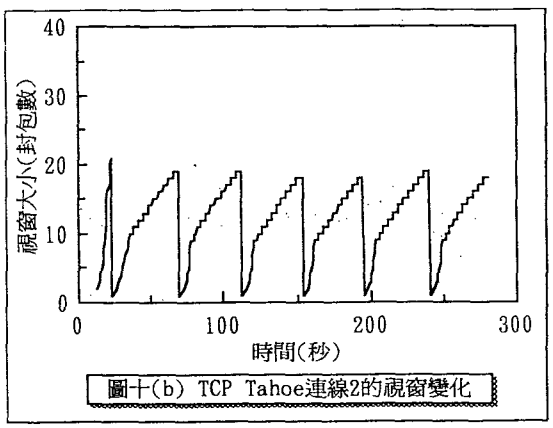
圖九(a) 本文方法連線1的視窗變化



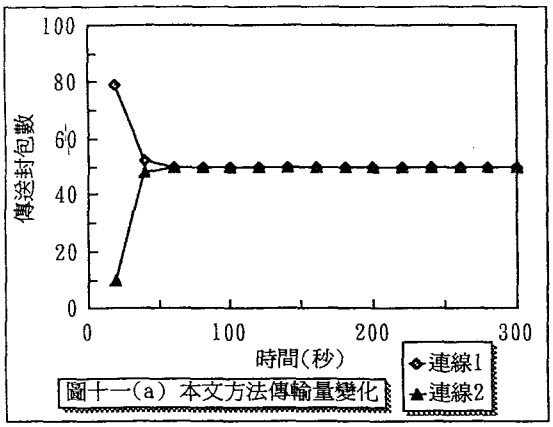
圖九(b) 本文方法連線2的視窗變化



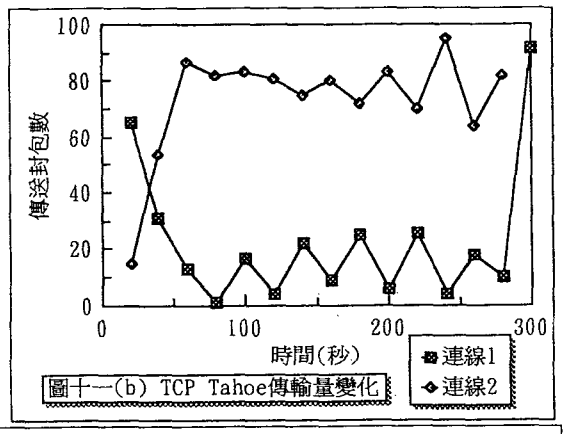
圖十(a) TCP Tahoe連線1的視窗變化



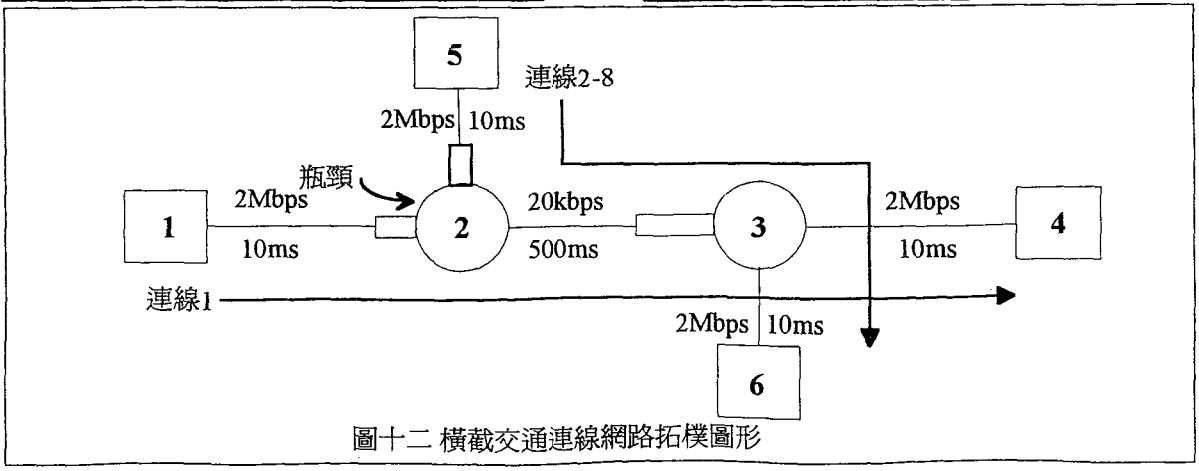
圖十(b) TCP Tahoe連線2的視窗變化



圖十一(a) 本文方法傳輸量變化

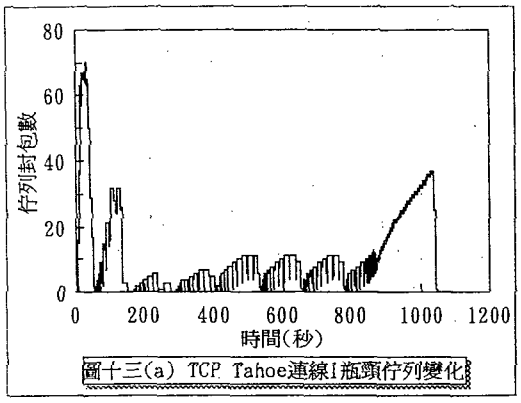


圖十一(b) TCP Tahoe傳輸量變化

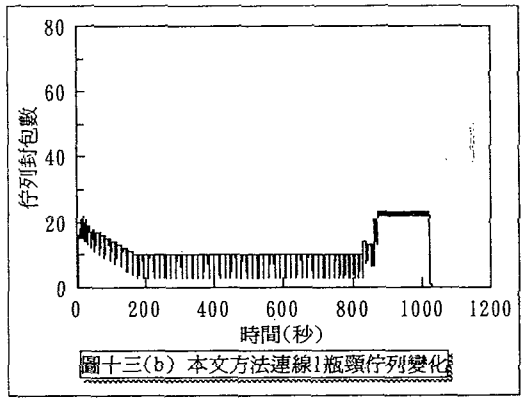


圖十二 橫截交通連線網路拓樸圖形

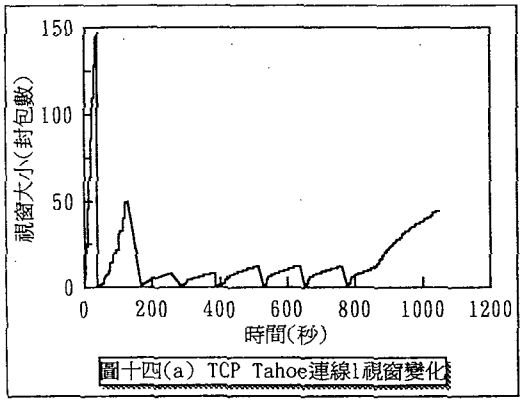
只有少數橫截交通流量加入，瓶頸緩衝區幾乎均被連線1所佔用，因此連線1的佇列最多高達70個封包，隨後因封包被刪除，視窗降為1及橫截交通陸續加入，佇列長度減小，並呈現週期性的振盪，直到830秒後，橫截交通流量逐漸減少，佇列長度才逐漸增加。圖十三(b)則是本文方法連線1的瓶頸佇列變化。因擁塞位元的設定是依據連線緩衝區的使用率，當連線個數增加時，每條連線的緩衝區相對變小，減少視窗長度的機率隨之提高。一開始受到橫截交通陸續加入的影響，發送端減少視窗長度，佔用瓶頸緩衝區的封包數逐漸減少，等到連線個數不再增加，佇列長度便被控制在一定的範圍內，當橫截交通離開後，佇列長度才又增加。連線1的視窗變化如圖十四。圖十四(a)是TCP Tahoe連線1視窗變化情形，由圖可看出，連線1的視窗長度最初高達147，隨後呈週期性劇烈振盪，待橫截交通離開後，此時因緩慢啓動門檻值(slow-start threshold)很小，視窗以線性的方式遞增。本文連線1的視窗變化如圖十四(b)，最初因為只有少數橫截交通連線，視窗便快速增加，而後其視窗長度始終維持為10，期間雖仍有部分封包的擁塞位元被設定為3，但由於演算法中的counter2尚未累積到足以降低視窗時便被歸零(代表回饋訊息不足以反應網路的擁塞)，因而視窗大小保持不變。連線6的瓶頸佇列變化情形如圖十五所示。圖十五(a)是TCP Tahoe的瓶頸佇列變化，當連線6加入時，網路交通呈現極度擁塞，交換節點緩衝區已被填滿，封包因此被刪除，而後佇列長度逐漸增加，並呈週期性振盪。相對的，本文方法有效控制連線緩衝區使用率，所以交換節點緩衝區仍有空間容納封包。圖十五(b)顯示本文方法瓶頸佇列很快的趨於穩定。圖十六是連線6視窗調整的變化情形。在圖十六(a)中，我們可明顯的看出，連線6一加入後便因封包流失，導致計時器中斷，緩慢啓動門檻值被設定為 1 ，所以視窗以線性的方式遞增，並呈週期性劇烈振盪。圖十六(b)為本文方法視窗調整的變



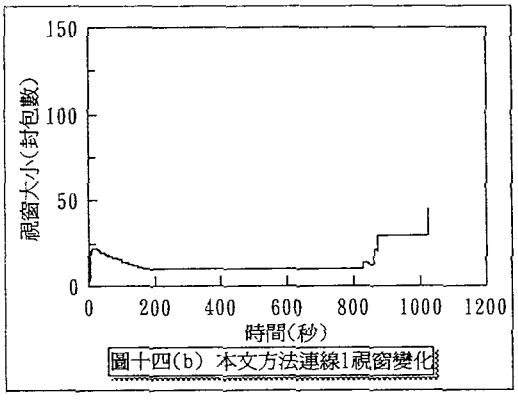
圖十三(a) TCP Tahoe連線1瓶頸佇列變化



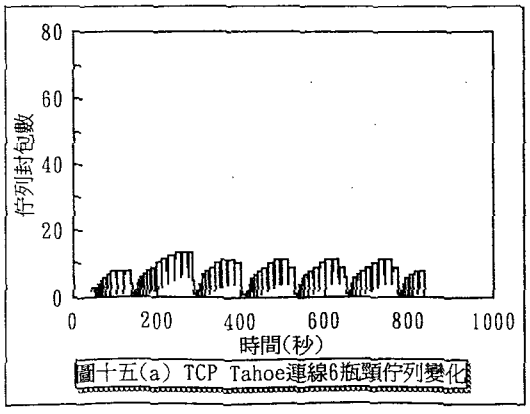
圖十三(b) 本文方法連線1瓶頸佇列變化



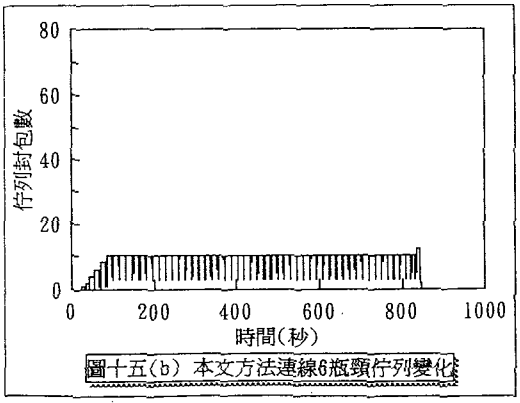
圖十四(a) TCP Tahoe連線1視窗變化



圖十四(b) 本文方法連線1視窗變化



圖十五(a) TCP Tahoe連線6瓶頸佇列變化



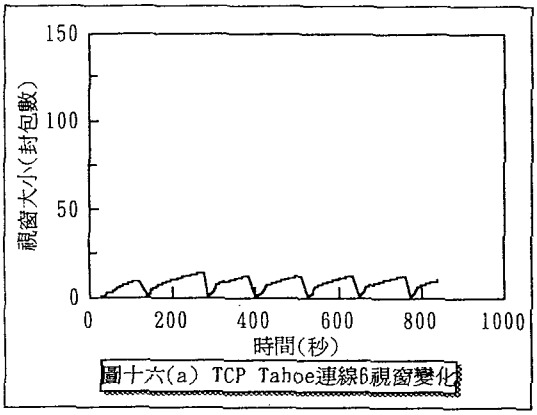
圖十五(b) 本文方法連線6瓶頸佇列變化

化情形。由於瓶頸佇列長度便被控制在一定的範圍內，因此視窗長度逐漸趨於穩定，與連線1有相同的大小，達到公平使用網路資源。圖十七為每隔100秒統計的整體產能比較圖。瓶頸的頻寬為每秒20K位元，因此連線每100秒最多可傳送500個封包。由圖中我們可以看出，本文方法的產能明顯的優於TCP Tahoe。

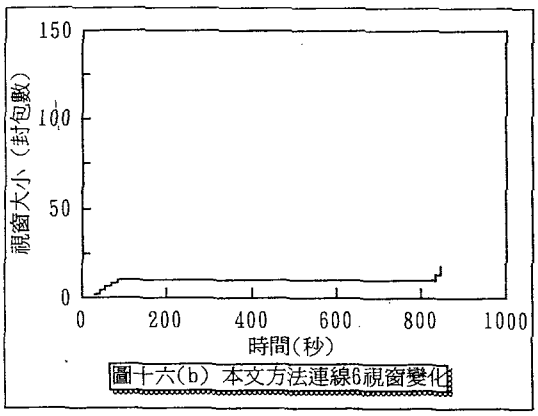
4. 結論

在本文中，我們提出了一個新的流量控制演算法，經由在封包中加入兩個擁塞位元，將網路的交通狀況區分成四種狀態。交換節點根據連線緩衝區使用率的高低來設定擁塞位元值，當此一訊息隨回應傳回發送端後，發送端便依據回饋訊息調整視窗長度。

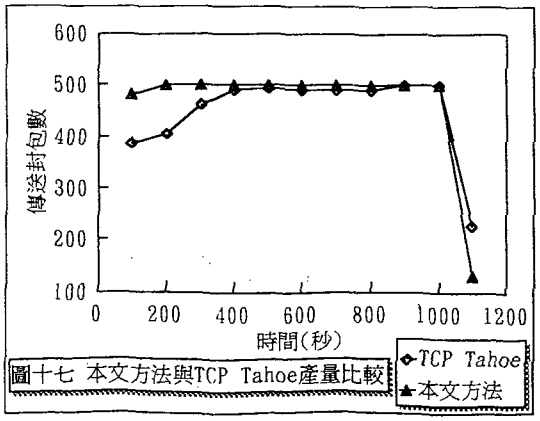
經由模擬實驗結果顯示，本文方法藉由控制連線緩衝區使用率，使其維持在一定的範圍內，如此不僅在傳輸過程中較TCP Tahoe有較高的產能、能夠充分利用網路資源、傳輸量較穩定外，同時不需借助交換節點的排程(queueing scheduling)演算法亦能達到公平性(fairness)，公平的分配網路上的資源。



圖十六(a) TCP Tahoe連線6視窗變化



圖十六(b) 本文方法連線6視窗變化



圖十七 本文方法與TCP Tahoe產量比較

參考文獻：

[1] O. Rose,"The Q-bit Scheme,"*ACM SIGCOMM* Vol. 22, No. 2,April,1992.

[2] Raj Jain and K. K. Ramakrishnan, "Congestion Avoidance in Computer Networks with a Connectionless Network Layer: Concepts, Goals and Methodology,"*Proc.Computer Networking Symposium, Washington, DC, 1988;*

[3] Yannis A. Korilis and aurel A. Lazar, "Why is Flow Control Hard: Optimality, Fairness, Partial and Delayed Information," *IEEE INFOCOM'93.*

[4] V. Jacobson,"Congestion Avoidance and Control," *Proc. of ACM SIGCOMM'88*, Stanford, CA, pp. 314-329, Aug. 1988.

[5] R. Jain, " A Timeout-Based Congestion Control Scheme for Window Flow-Controlled Networks," *IEEE Joural on Selected Areas of Communication*, Vol. SAC-4, No. 7, pp. 1162-1167, Oct. 1986.

[6] D. M. Chiu and Raj Jain, "Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks," *Comp. Networks and ISDN Syst.*, Vol. 17, pp. 1-14, 1989.

[7] K. K. Ramakrishnan and R. Jain, "A Binary Feedback Scheme for Congestion Avoidance in Computer Networks," *ACM TOCS* Vol. 8, No. 2, pp. 158-181. May, 1990.

[8] S. Shenker, L. Zhang and D. Clark, "Some Observations on the Dynamics of a Congestion Control Algorithm," *ACM Computer Communication Review*, Vol. 20, No. 5, Oct. 1990.

[9] Z. Wang and J. Crowcroft,"Eliminating Periodic Packet Losses in the 4.3-Tahoe BSD TCP Congestion Control Algorithm," *ACM SIGCOMM*, April, 1992.

[10] A. Mankin and K. Thompson,"Limiting factors in the performance of the slow-start TCP algorithms," *Proc. of USENIX* Win. Conference'89, pp. 219-228, Jan. 1989.

[11] S. Keshav,"REAL:A Network Simulator," Computer Science Dept. TR 88/472. Uni. of California, Berkeley, Dec. 1988.