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

一種植基於公平排班法則的速率回饋流量控制方法

A RATE FEEDBACK FLOW CONTROL SCHEME BASED ON FAIR QUEUEING DISCIPLINE

朱延平　　黃一泓　　張明湘
Y.P. Chu　　E.H. Hwang　　M.H. Chang

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

摘要

本篇文章提出一個新的預防式速率回饋流量控制法，適用於無預留的網路，網路中交換節點採用公平排班法則或其他類似循環佇列服務的排班法則。我們在封包表頭中增加一位元，作爲網路中緩衝區利用率之回饋訊息，當緩衝區利用率到達操作點時，交換節點便設定此位元，這個回饋訊息透過傳輸層的認可到達傳送端。我們採用一個適於類似循環佇列服務排班法則的網路模型，基於這個模型，我們採用封包對來探測瓶頸服務率，並利用相鄰被設定位元的認可來探測新的瓶頸服務率，作爲調整傳送速率的依據。模擬實驗結果顯示，本文方法在產能、瓶頸佇列長度、對網路交通量改變的反應能力、公平性和小檔案的傳送時間都相當良好。

關鍵字：速率回饋流量控制法、網路擁塞、封包對流量控制法、公平排班法則、速率分配伺服器

ABSTRACT

This paper presents a new preventive rate feedback flow control scheme in reservationless network, in which switching nodes use Fair Queueing or other round-robin-like queue service discipline. We add a bit in the packet header to be the feedback information of buffer utilization in the network. The switching nodes set the bit when the buffer utilization achieve the operating point. This feedback information is communicated back to the sources through the transport-level acknowledge. We use a network model for round-robin-like queue service discipline. Based on this network model, we use packet-pair to probe the bottleneck service rate and use immediate acknowledges whose bits are set to probe new bottleneck service rate to be the basis of the adjust of sending rate. Simulation results show that our scheme is very good for throughput, bottleneck queue length, responsiveness to the change of network traffic, fairness and the transfer time for small files.

Keywords: Rate Feedback Flow Control , Network Congestion, Packet-Pair Flow Control, Fair Queueing, Rate Allocation Server.

一、簡介

由於網路通訊的普及化，網路的使用者越來越多，可以同時提供多種交通型態的整合式的通訊電腦網路（integrated telecommunication network）[1，6]的出現，滿足了各類使用者的需求。整合式的通訊電腦網路中，各種交通型態有其個別的特性和不同的服務品質（quality of service，QOS）需求。在預留導向（reservation-oriented）的網路中，網路會爲每個被網路接受的呼叫保留足夠的網路資源（頻寬、緩衝區等）以容納尖峰交通負載量。而在無預留（reservation-less）的網路中，網路並不爲傳送端保留任何網路資源。資源共享（resource sharing）提高了網路的利用率（utilization），但是網路和使用者必須採用更複雜的控制方法，來適應網路情況的變化，以達到所需的服務品質。許可控制（admission control）、及緩衝區管理（buffer management）等，都是爲了協調在不同的交通型態和連線間複雜的網路資源共享。雖然採用這些資源管理方法，網路仍會有使用者傳送了超過網路可以承受的負載量，使網路無法滿足使用者所需的服務品質，這樣的情況即爲擁塞（congestion）[6]。擁塞會引起佇列延遲時間（queueing delay time）變長、緩衝區溢滿（buffer overflow）、封包遺失（packet loss）和計時器終了（timeout）等。解決網路擁塞狀況可由兩方面來達成：一方面可由傳送端（source）根據網路狀態的回饋訊息調整他們傳送資料的交通量，此種方法就是回饋流量控制（feedback flow control）；另一方面可由網路中的交換節點（switching node）藉著路徑選擇和排班（queueing scheduling）來達成擁塞控制的目的。

流量控制的目標是以在高網路利用率和服務品質兩者間維持有效率的妥協（trade-off）方法來調整進入網路的交通量。一般而言，網路利用率是以平均產能（throughput）來度量，而服務品質則用平均時間延遲（time-delay）來度量。此外，網路資源的公平（fairness）使用則爲另一個流量控制的目標[6]。回饋流量控制可分爲兩類：被動

式（reactive）的和預防式的（preventive）：前者是等到擁塞被偵測到時才採取行動以減輕擁塞的狀況；而後者則是試著讓網路不會進入擁塞的狀態[9]。一個回饋流量控制方法包含兩部份：（一）資訊取得機制（information acquisition mechanism）或路由器策略（router policy），負責提供網路狀態的資訊給傳送端；（二）控制決策機制（control decision mechanism）或使用者策略（user policy），是傳送端根據資訊取得機制提供的資訊，來調整交通負載的方法。資訊取得機制有明顯的（explicit）和不明顯的（implicit）兩類[7]：明顯的回饋訊息是由網路或其他使用者直接傳送給傳送端，著名的例子爲抑制（choke）封包[18]、傳送端壓抑（source quench）封包[19]、DECbit[8]等；而不明顯的回饋訊息則是將網路視爲一個黑箱（black box），使用者輸入具探測性質的交通量，並觀察網路的回應，來獲得網路的狀態，可利用的網路回應訊息有封包遺失引發的計時器終了、產能降低、產能斜率（throughput gradient）變小、及延遲增加等[12]，著名的例子有TCP Tahoe[5]、Tri-S[11]、CUTE[13]等。控制決策機制包括下面三個部份：（一）決策頻率（decision frequency）：傳送端調整交通負載的頻率；（二）控制法則（control law）：利用網路的回饋訊息決定交通負載調整的方向；（三）增減演算法（increase/decrease algorithm）：決定交通負載調整量的大小。

現行網路中交換節點的排班法則多採用先到先服務法（First-Come-First-Service），雖然方法簡單，但是不易達成公平分享網路資源的目標，因此我們將焦點集中在循環式（round-robin）的排班法則。循環式服務賦予每個正在傳送的連線一個邏輯的或實際的輸入（incoming）佇列，以虛擬的分時多工（Time Division Multiplexed，TDM）的方式輪流服務資料佇列。像這樣循環服務的交換節點被稱爲速率分配伺服器（Rate-Allocating Server）[1，2，3]，例如公平排班法（Fair Queueing）[14]、權重公平排班法（Weighted Fair Queueing）、封包化的一般化處理器分享（Packetized Generalized Processor Sharing）[15]、虛擬時鐘（Virtual Clock）[16]等。

近年來高速網路蓬勃發展，傳統的視窗流量控制法（window flow control）不足以應付高速網路中較高的延遲頻寬乘積（high delay-bandwidth product）；因此，發展一個適用於高速網路的流量控制法顯得格外重要，而控制封包傳送間距（inter-packet time）的速率流量控制法（rate flow control）正符合高速網路的需求。所以，我們提出一個適用於無預留、預防式的、交換節點採用公平排班法（或其他速率分配伺服器）的速率回饋流量控制法。在本文的第二節中我們提出一個決定性的（deterministic）網路模型，並簡介一個著名的速率回饋流量控制法——封包對（Packet-Pair，PP）流量控制法[1]；第三節則描述我們提出的速率回饋流量控制法；接著在第四節以模擬實驗來觀察我們的流量控制法的行爲，並和封包對流量控制法在產能、瓶頸佇列長度、來回旅行時間及公平性做比較；最後一節提出我們的結論。

二、網路模型和封包對流量控制法

在這一節中，我們先提出一個決定性的網路模型，這個模型適用於我們的方法，也適用於封包對流量控制法；接著對封包對流量控制法做一簡單的描述。

2.1 網路模型

我們把一條連線（connection）上的伺服器從1開始按照1，2，3，…，n的順序編號，傳送端是第0號，接收端是第n號，接收端會爲每一個封包傳送認可（acknowledge）訊息給傳送端。當封包在時間t到達第i個伺服器，而此伺服器正好是停滯（idle）狀態時，伺服器服務封包所需的服務時間（service time）爲$s_i(t) = 1/\mu_i(t)$，其中s_i是伺服器輪流服務所有連線依次所需的時間，而μ_i是第i個伺服器的服務率（service rate），也就是同一連線相鄰兩個封包被服務的時間間隔的倒數。傳送端的傳送率（sending rate）爲λ，而且傳送端傳送封包的時間間隔爲$s_0 = 1/\lambda$。

瓶頸（bottleneck）是傳送路徑中服務時間最長，也就是速度最慢的伺服器。我們定義整條連線的瓶頸服務時間爲$s_b(t) = max\ (s_i(t) \ / \ 0 \le i \le n)$，其中$b$是瓶頸伺服器的編號，而瓶頸服務率$\mu_b(t)$被定義爲$\mu_b(t) = 1/s_b(t)$。若在傳送的連線數目有變化，$\mu_i(t)$也隨之改變。如果傳送端的傳送率$\lambda$大於瓶頸服務率$\mu_b$，則在瓶頸中會形成資料佇列，並且傳送端收到認可的時間間隔爲s_b；反之，如果傳送端的傳送率小於或等於瓶頸服務率，則不會在瓶頸中形成資料佇列，而傳送端收到認可的時間間隔即爲傳送速率[3，4，10]。

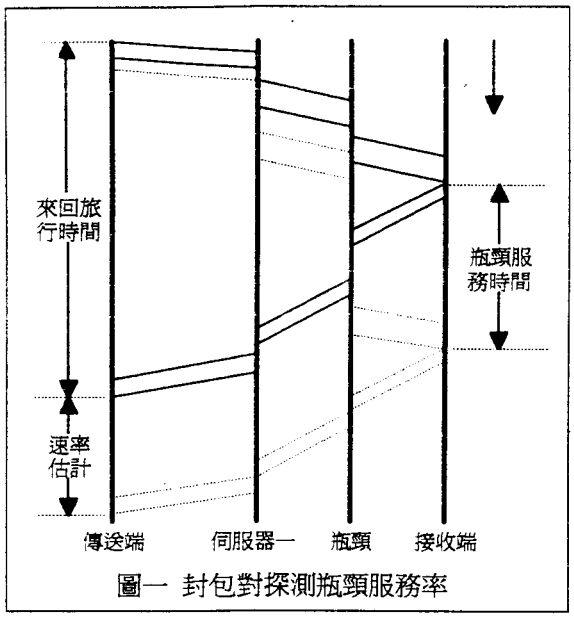
2.2 封包對流量控制法

封包對流量控制法是由Keshav提出的速率回饋流量控制法，其適用的網路排班法則爲公平排班法。此法假設傳送路徑上，每個交換節點分配給每一連線的緩衝區大小均爲B，而其演算法的目標是讓瓶頸緩衝區佇列半滿，也就是其操作點爲B/2。

傳送端利用封包對來探測出瓶頸服務率，如圖一：傳送端連續送出兩個資料封包，盡可能的讓這兩個封包的間隔時間最小，由於網路中交換節點爲速率分配伺服器，因此，這兩個封包的認可的到達時間間距（interarrival time）即相當於s_b，也就是瓶頸服務率的倒數。每當有一對認可到達傳送端，傳送端便調整傳送速率；傳送端估計瓶頸佇列中的封包數，若小於B/2就增加傳送速率，反之則減少其傳送速率。

三、本文方法

和封包對流量控制法一樣，我們也選擇以網路交換節點中的緩衝區利用率（buffer utilization）作爲操作點，也就是我們希望讓瓶頸緩衝區利用率保持在一定的範圍內，當瓶頸緩衝區利用率超



圖一 封包對探測瓶頸服務率

過此範圍時，減少傳送速率，反之則增加傳送速率。

計算交換節點中緩衝區利用率時，爲了讓每一條通過此交換節點的連線能公平的分享緩衝區空間，我們將整個緩衝區的大小，除以目前交換節點中處於可動式（active）的連線數，所得的商，即爲交換節點虛擬分配給每一連線之緩衝區空間。而選擇緩衝區利用率的操作點，必須考慮以下三點：平均封包延遲、封包遺失及頻寬漏失（bandwidth loss），在三者之間取得妥協。所謂頻寬漏失，就是輪到交換節點爲此連線服務時，連線卻沒有封包可以傳送的情況。如果操作點偏低，平均封包延遲和封包遺失的機率會降低，但頻寬漏失的機率則會增加；反之，如果選擇的操作點較高，頻寬漏失的機率減少，然而，平均封包延遲和封包遺失的機率則會增大。

傳送端如何得知瓶頸緩衝區利用率呢？在封包對流量控制法中，假定傳送路徑上每個交換節點分配給一條連線的緩衝區大小皆爲B，但是由於每個交換節點的緩衝區大小不一定會相同，所以實際上這是不可能的，而且通過交換節點的連線數也往往各不相同。因此，爲了能確切的知道瓶頸緩衝區利用率，我們選擇了明顯的回饋策略——當交換節點的緩衝區利用率超過 δ（δ<1）時，交換節點在封包的表頭（header）中設定一bu位元（buffer utilization bit，bubit），這就是我們的網路策略。bu位元隨著封包到達接收端，接收端將此位元拷貝到認可的表頭中，再隨著認可回到傳送端，用以通知傳送端，瓶頸緩衝區利用率已超過操作點。由於速率分配伺服器循環運作的方式，使得交換節點緩衝區中，同一連線相鄰封包的認可到達傳送端的時間間隔的倒數，即爲瓶頸服務率s_b；因此，由兩個相鄰的bu位元被設定而且它們的序列號碼（sequence number）也相鄰的認可的到達時間間距，可以得到瓶頸服務率，見圖二。事實上，當傳送端檢查出bu位元被設定，具有兩種意義：一是傳送端的傳送速率超過瓶頸服務率太多；一是有新的連線加入，分享了交換節點的緩衝區。無論是那一個原因，此時傳送端都應該減低其傳送速率。

我們的增減演算法，在增加策略方面，我們採用加法增加（additive increase）方式，將原本的傳送速率加上一個大於0的值 α，α代表每秒要增加傳送的封包數；而在減少策略方面，我們使用乘法減少（multiplicative decrease），也就是將瓶頸服務率乘以一個小於1的有理數β作爲新的傳送速率。加法增加/乘法減少的增減演算法已經被證明爲最適當的網路流量調整方式[17]。

我們的流量控制法有三個模式：起始模式（initial mode）、增加模式（increase mode）和減少模式（decrease mode）。

傳送端在開始傳送資料封包時進入起始模式，先傳送一對緊鄰送出的封包以探測瓶頸服務率，待一對認可回來後，認可的到達時間間距爲瓶頸服務時間s_b，其倒數即爲瓶頸服務率μ_b，以瓶頸服務率當作傳送速率來傳送資料封包，此時進入增加模式。

在增加模式中，我們採用不明顯的回饋訊息，以來回旅行時間（round trip time）作爲調整傳送速率的依據。來回旅行時間是自封包由傳送端傳送進入網路，到傳送端收到其對應的認可的時間區間，這段時間包括交換節點的處理時間（processing time）、佇列延遲時間（queueing delay time）及傳輸延遲時間（propagation delay time），當傳送端以瓶頸服務率來傳送封包時，由2.1節的網路模型可知在瓶頸中並不會形成資料佇列，所以來回旅行時間只包含交換節點的處理時間及傳輸延遲時間，而這兩個值爲定值，故來回旅行時間亦爲定值，而且此時的來回旅行時間最小。爲了讓瓶頸緩衝區內有封包停留，傳送端每收到一個認可，便檢查其來回旅行時間，若和前一個封包的來回旅行時間相同，則增加傳送速率α_1（α_1>0）。當瓶頸緩衝區開始有封包累積，來回旅行時間加入了佇列延遲時間，便會比前一個封包的來回旅行時間大，此時，傳送速率便不再增加。由於這時候的傳送速率比瓶頸服務率大，緩衝區佇列會越來越長，當緩衝區利用率到達δ，交換節點便設定封包表頭的bu位元。

傳送端檢查認可的bu位元，若bu位元被設定，則進入減少模式。若相鄰兩個認可的bu位元皆被設定而且它們的序列號碼也相鄰，則這兩個認可的到達時間間距的倒數即爲瓶頸服務率（若有新的連線加入，則爲新的瓶頸服務率。），此時我們以低於瓶頸服務率的傳送速率（$\beta \times \mu_b$）使瓶頸緩衝區的利用率降回操作點之下。當傳送端發現認可表頭的bu位元不再被設定，則進入增加模式。此後便在增加模式和減少模式之間循環。

另外，考慮下面的情況：連線一先開始傳送資料封包，連線二比連線一晚加入傳送封包，因此連線二所得到的最小來回旅行時間並非眞的最小值。當連線一先傳完封包，結束連線，連線二便可能量得比原先的最小來回旅行時間更小的值，爲了迅速吸收連線一所釋放的頻寬，連線二的傳送速率以比α_1更快的增加速度α_2來傳送封包。

以下是我們所提出的詳細演算法：

VARIABLE DEFINITION：

```
send_rate : sending rate
inter_ack : inter-ack time
rtt : round trip time
rtt_min : minimal rtt
last_rtt : rtt of last packet
first_of_pair : seq. # of first ack of pair
time_of_last_ack : time when last ack recvd
active_flow : # of active conversation
aver_conv_size : buffer allocated to a conv.
op_qsize : buffer size of a router
bytes_in_q : bytes in queue of a conv.
```

USER POLICY：

Initial mode

```
send packet-pair;
/* bottleneck service rate probing */
receive two acks of packet-pair;
send_rate = 1 / inter_ack;
/* send rate = bottleneck service rate */
rtt_min = last_rtt = rtt;
```

Increase mode

```
receive an ack，check pkt->bubit;
if (bubit = 0)
/* buffer utilization < δ */
  compute rtt;
  if (rtt < rtt_min)
    send_rate = send_rate + α2;
    /* send rate increases α2 */
    rtt_min = rtt;
  else if (rtt ≧ rtt_min and rtt ≦ last_rtt)
    send_rate = send_rate + α1;
    /* send rate increases α1 */
  else send_rate = send_rate;
  /* send rate unchanged */
  last_rtt = rtt;
```

Decrease mode

```
if (bubit = 1)
/* buffer utilization ≧ δ */
  if (pkt->id = first_of_pair + 1)
  /* get bottleneck service rate */
    send_rate = β / inter_ack;
    /* send rate < bottleneck service rate */
  time_of_last_ack = now;
  first_of_pair = pkt->id;
```

NETWORK POLICY：

```
if (active_flow > 0)
  aver_conv_size = op_qsize / active_flow;
  bytes_in_q = get_num_bytes_in_q(conv_id);
  if (bytes_in_q/ δ ≧aver_conv_size)
  /* buffer utilization ≧ δ */
    pkt->bubit = 1;
  else
    pkt->bubit = 0;
```

四、模擬實驗與比較

爲了觀察我們所提出的回饋流量控制法在實際網路中的行爲，驗證此法在網路中的可行性，我們以U.C. Bekerley 所發展出來的模擬實驗軟體 REAL 4.0[21]來進行模擬測試。我們選擇了五個具代表性的模型（scenarios）：（一）單一連線，（二）兩條連線先後加入網路，（三）十條連線先後加入網路，（四）兩條路徑長度不同的連線同時傳送資料封包，（五）單一連線傳送小檔案[20]。實驗所用的網路拓樸形狀有四個，如圖三~六，所有的封包大小均爲500位元組，認可訊息的大小爲40位元組；主機（host）和交換節點間的連線頻寬爲2Mbit/sec，傳輸延遲（latency）爲1ms；交換節點間的連線頻寬爲40Kbit/sec，傳輸延遲爲200ms（圖三、四）或500ms（圖五，六）；每個交換節點的緩衝區大小爲30個封包（圖三、四）或100個封包（圖五，六）。所有的網路中，由於頻寬的差異，均在第一個交換節點形成瓶頸。實驗中假設沒有資料傳輸錯誤（error-free），只有當交換節點的緩衝區溢滿時才會刪除封包，交換節點的排班策略爲公平排班法。對於每一個模型，都以本文方法和封包對流量控制法做測試。在模擬實驗結果中，傳送速率是以封包/秒爲單位，佇列長度是指第一個交換節點針對單一連線的輸入緩衝區佇列大小，產能是度量每秒所能傳送的封包數，但不包括重送。

模擬實驗中，演算法中參數值的設定如下：α_1=2/3，α_2=4/5，β=7/8，δ= 1/5。α_1和α_2是增加模式中傳送速率的增加速度，若設定的值大雖然反應性（responsiveness）較佳，能很快達到操作點，但造成的傳送速率震盪也較大；反之，若設定的值小，造成的傳送速率震盪較小，但較慢到達操作點。β的功能是爲了要將瓶頸緩衝區中超過操作點的部份降回操作點之下，若設定的值太小，會造成網路資源的浪費，而太大的值則無法達到應有的功能，當網路中的連線很多或交換節點緩衝區很小時，容易造成緩衝區溢滿和封包遺失。δ爲緩衝區利用率的操作點，若設定值太小，容易造成瓶頸的停滯現象，浪費寶貴的網路資源；另一方面，若設定值太大，則發生緩衝區溢滿和封包遺失的頻率會很高，且佇列延遲時間會很長。而以上的設定值，經模擬實驗顯示，效果相當良好。

4.1 單一連線模擬實驗

這個實驗是要觀察本文方法的實際運作情形。實驗用的網路拓樸形狀如圖三。連線一自0秒開始，從第一個主機經兩個交換節點傳送1Mega位元組的資料至第二個主機，模擬時間爲200秒。本文方法和PP的傳送速率如圖七和圖八，而本文方法和PP的瓶頸佇列長度變化如圖九和圖十。由圖七可知本文方法的傳送速率會有近似週期性的震盪情形，這是傳送速率在增加模式和減少模式間循環的緣故，但圖九則顯示出本文方法在控制瓶頸佇列長度的優越效果，增加模式使瓶頸佇列長度到達並超過操作點，而減少模式則使瓶頸佇列長度降回操作點之下，所以，瓶頸佇列長度一直維持在5至10個封包；反之，PP的傳送速率雖然較爲平穩，但造成瓶頸佇列長度超過20個封包，較本文方法高出許多，而且佇列長度的變化亦較本文方法的變化大，甚至有緩衝區溢滿和隨之而來的封包遺失現象。由第三節中對來回旅行時間的討論可知，穩定且較小的佇列長度可得到變化較

小且短的來回旅行時間，由瓶頸佇列長度變化可知，本文方法所獲得的來回旅行時間，無論是平均值或標準差都優於PP。圖十一是兩種方法的產能比較，圖中顯示本文方法的產能較爲穩定，且略佳於PP。

4.2 兩條連線模擬實驗

這個實驗是要測試本文方法在穩定狀態下，對於新的連線加入和舊的連線離開的反應。實驗用的網路拓樸形狀如圖四。連線一自0秒開始，從第一個主機經兩個交換節點傳送1Mega位元組的資料至第二個主機，連線二自30秒開始，從第三個主機經兩個交換節點傳送2.5K位元組的資料至第四個主機，模擬時間爲200秒。本文方法和PP之連線一和連線二的傳送速率及瓶頸佇列長度變化分別見圖十二~十九。由圖十二和十三可知，本文方法能立刻偵測出瓶頸服務率的變化，馬上有適當的反應；當新的連線加入，瓶頸服務率變小，連線一就根據瓶頸服務率的變化量降低其傳送速率；當連線二傳送完畢，連線一也能根據瓶頸服務率的變化調高其傳送速率至沒有連線加入前的狀態，立即吸收連線二所釋放出來的頻寬。在瓶頸佇列長度變化方面，也有和傳送速率一致的穩定變化。圖十四、十五、十八、十九則顯示PP的傳送速率和瓶頸佇列長度變化非常劇烈，且兩條連線並沒有很公平的分享頻寬和瓶頸緩衝區。圖二十和二十一是兩種方法的產能比較，當新的連線加入後，本文方法的兩條連線產能皆相同，而PP兩條連線的產能則呈現大幅震盪。

4.3 十條連線模擬實驗

這個實驗是要測試本文方法在複雜的網路狀況下的運作情形，實驗用的網路拓樸形狀如圖五。連線一自0秒開始，從第一個主機經兩個交換節點傳送1Mega位元組的資料至第二個主機，連線二至連線十自30秒開始，每隔二十秒加入一條新的連線，從第三個主機經兩個交換節點傳送250K位元組的資料至第四個主機，模擬時間爲660秒。本文方法和PP的連線一傳送速率變化如圖二十二和二十三。由圖二十二可知，本文方法的主連線隨著其他連線的增加，瓶頸服務率減少，而適度的降低其傳送速率，釋放頻寬給其他連線使用；隨著橫截交通量逐漸減少，主連線的傳送速率也逐漸增加；當橫截交通結束，則迅速恢復原先的傳送速率，這個結果和兩條連線時的結果完全符合。此外，在前面單一連線和兩條連線的模擬實驗中，本文方法傳送速率的振盪在網路中有十條連線時有明顯減少振幅的趨勢。由以上分析可知，本文方法足以勝任實際的複雜網路環境中流量控制的工作。PP的傳送速率則呈現不規則的大幅振盪，並有較多的封包遺失和重送，因而影響它的產能。圖二十四中針對連線一的產能做比較，本文方法的產能相當平穩，而PP的產能則和其傳送速率有相似的振盪。

4.4 公平性模擬實驗

這個實驗是要測試在不同的傳送距離下，本文方法是否可以公平的分享頻寬。實驗用的網路拓樸形狀如圖六。連線一自0秒開始，從第一個主機經兩個交換節點傳送1Mega位元組的資料至第二個主機，連線二也自0秒開始，從第五個主機經兩個交換節點傳送1Mega位元組的資料至第二個主機，但連線二的端對端（end-to-end）傳輸延遲比連線一多五分之一；兩條連線遭遇相同的橫截交通——連線三至連線十自30秒開始，每隔二十秒加入一條新的連線，從第三個主機經兩個交換節點傳送250K位元組的資料至第四個主機，橫截交通均採用卜瓦松分配（Poisson distribution）控制封包傳送間隔平均速度爲4.8K位元/秒，模擬時間爲400秒。本文方法和PP的連線一及連線二之傳送速率和瓶頸佇列長度變化分別見圖二十五~三十二。由傳送速率和瓶頸佇列長度的變化圖可知，本文方法和PP都可以達到公平分享網路資源的目標，但PP的瓶頸佇列長度比本文方法大，也就是具有較大的佇列延遲時間和來回旅行時間。圖三十三和三十四是兩種方法兩條連線的產能比較，圖中顯示兩種方法的公平性差異不大，然而，橫截交通量大時，PP的產能稍有振盪的現象。

4.5 小檔案的傳輸時間實驗

這個實驗是要測試當所要傳送的檔案很小時，需要花費的傳輸時間。實驗用的網路拓樸形狀如圖三，我們以單一連線分別傳送10K和100K位元組資料，測試傳送端自開始傳送封包到收到最後一個認可的時間。結果如表一：

傳輸時間(秒)	10Kbytes	100Kbytes
本文方法	2.84	20.84
PP	4.77	22.84

表一 小檔案傳送時間之比較

由上表可知，本文方法在傳送小檔案時較PP快速；另一個意義是，連線開始傳送封包時，本文方法會以較短的時間取得應有的網路資源，並迅速到達操作點。

五、結論

在本文中，我們提出了一個適用於無預留、預防式的、交換節點採用公平排班法的速率回饋流量控制法；開始傳送時以封包對探測連線中的瓶頸服務率，根據此瓶頸服務率和來回旅行時間的變化來增加傳送速率；藉著在封包表頭增加一個位元（bubit），由網路提供緩衝區利用率的資訊，當傳送端檢查到此位元被設定，利用速率分配伺服器循環服務的特性，經由連續bu位元被設定的認可來探知新的瓶頸服務率，再根據此服務率降低其傳送速率。

模擬實驗顯示，本文方法無論在產能、瓶頸佇列長度、來回旅行時間、公平性、對網路環境變化的反應能力及傳送小檔案所需的時間，均優於著名的封包對流量控制法。主要原因是本文方法利用網路的明顯回饋訊息來調整傳送速率，所得到的緩衝區利用率是實際的值；而封包對流量控制法是由傳送端自行根據封包對所得到的不明顯回饋來估計緩衝區利用率，會有誤差存在。本

文方法採用明顯回饋方法，由網路提供回饋訊息，傳送端不用作繁複的計算和估計，所以演算法相當簡單；封包對流量控制法由傳送端自行估計網路狀況，故傳送端需採用複雜的計算和預估。此外，由於我們在計算緩衝區利用率時，將交換節點中的緩衝區，以虛擬的方式平均分配給所有連線，故本文方法的回饋訊息傳送爲選擇性回饋（selective feedback），因而達到公平性。換句話說，若交換節點不採用公平排班法則，而採用其他速率分配伺服器，應仍然可以使用者公平分享網路資源。然而，本文方法必須在使用公平排班法則（或其他速率分配伺服器）的網路才能運作，但現行網路多使用FCFS排班法，而且公平排班法實施不易，如何修改本文方法以適合於在現行網路中運作將是我們未來的研究方向。

參考文獻

[1]S. Keshav, "Packet-Pair Flow Control".

[2]S. Keshav, "A Control Theoretic Approach to Flow Control", Proc. ACM SIGCOMM 1991, September 1991.

[3]S. Keshav, A. K. Agrawala and S. Singh, "Design and Analysis of a Flow Control Algorithm for a Network of Rate Allocating Servers"

[4]S. Singh, A. K. Agrawala and S. Keshav, "Deterministic Analysis of Flow and Congestion Control Policies in Virtual Circuits", Tech. Rpt.-2490, University of Maryland, June 1990.

[5]V. Jacobson, "Congestion Avoidance and Control", Proc. ACM SIGCOMM 1988, August 1988, 314-329.

[6]Y. A. Korilis and A. A. Lazar, "Why is Flow Control Hard: Optimality, Fairness, Partial and Delayed Information", IEEE InfoCom. 1993.

[7] R. Jain and K. K. Ramakrishnan, "Congestion Avoidance in Computer Networks with a Connectionless Network Layer: Concepts, Goals and Methodology", Proc. Computer Networking Symposium, Washington, April 1988, 134-143.

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

[9]O. Rose, "The Q-bit Scheme", ACM SIGCOMM Vol. 22, No. 2, April 1992, 29-42.

[10]J. Bolot and A. U. Shankar, "Dynamical Behavior of Rate-Based Flow Control Mechanisms", ACM Computer Communication Review, Vol. 20, April 1990, 35-49.

[11]Z. Wang and J. Crowcroft, "A New Congestion Control Scheme: Slow Start and Search(Tri-S)", Preprint, University College, London, Uk, October 1990.

[12]R. Jain, "A Delay-Based Approach for Copngestion Avoidance in Interconnected Heterogeneous Computer Networks",Computer Communications Review, October 1989, 56-71.

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

[14]A. Demers, S. Keshav and S. Shenker, "Analysis and simulation of a fair queueing algorithm", Proc. ACM SIGCOMM 1989, 3-12.

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

[16]L. Zhang, "Virtual Clock: A New Traffic Control Algorithm for Packet Switching Networks", Proc. ACM SIGCOMM 1990, Sep 1990, 19-29.

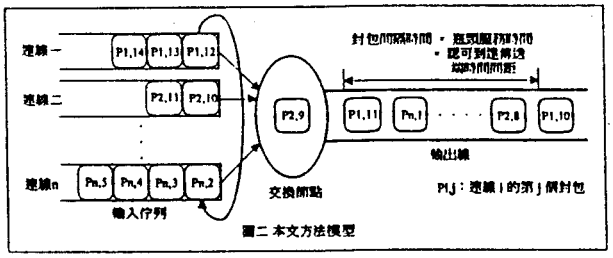
[17] D. M. Chiu and R. Jain, "Analysis of Increase and Decrease Algorithms For Congestion Avoidance in Computer Networks", Computer Networks and ISDN Systems, Vol. 17, 1989, 1-14.

[18]J. C. Majithia, et al., "Experiments in Congestion Control Techniques", Proc. Int. Symp. Flow Control Computer Networks, Versailles, France Feb. 1979.

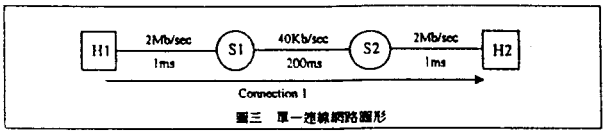
[19]J. Nagle, "Congestion Control in TCP/IP Internetworks", Computer Communication Review, Vol. 14, No. 4, Oct. 1984, 11-17.

[20]H.Kanakia, S. Keshav and P. P. Mishra, "A Comparison of Reactive Host-based Flow Control Schemes in Reservationless Networks",

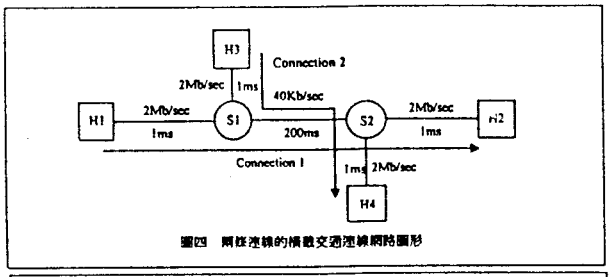
[21]S. Keshav, "REAL: A Network Simulator", Tech. Rep. 8814723, Dep. of Computer Science, UC Berkeley, 1988.



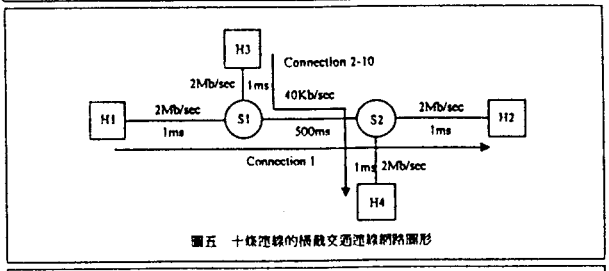
圖二 本文方法模型



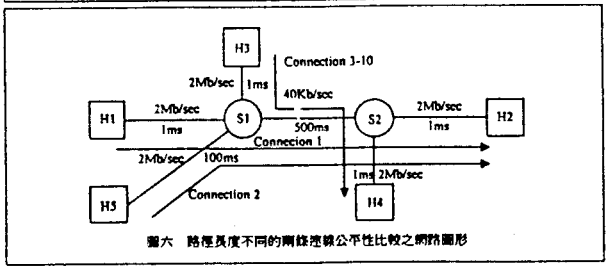
圖三 單一連線網路圖形



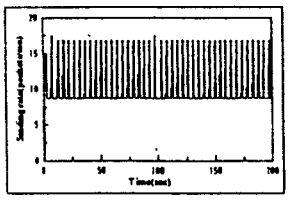
圖四 兩條連線的橫截交通連線網路圖形



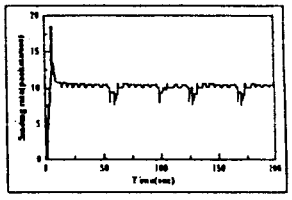
圖五 十條連線的橫截交通連線網路圖形



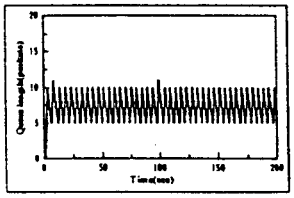
圖六 路徑長度不同的兩條連線公平性比較之網路圖形



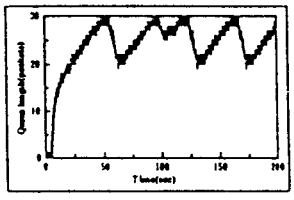
圖七 單一連線本文方法的傳送速率變化



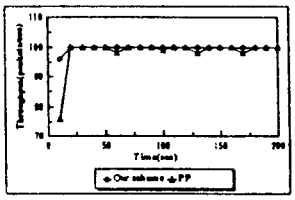
圖八 單一連線PP的傳送速率變化



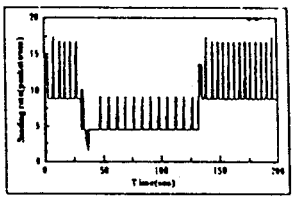
圖九 單一連線本文方法的瓶頸佇列長度變化



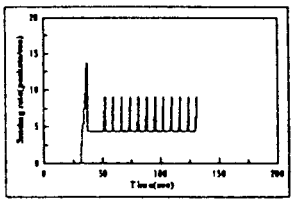
圖十 單一連線PP的瓶頸佇列長度變化



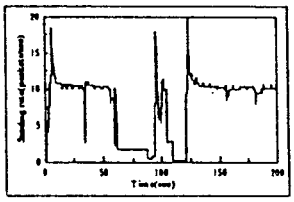
圖十一 單一連線本文方法和PP的產能比較



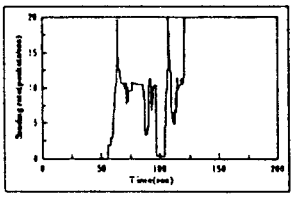
圖十二 兩條連線本文方法連線一的傳送速率變化



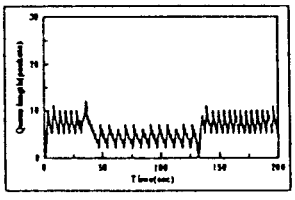
圖十三 兩條連線本文方法連線二的傳送速率變化



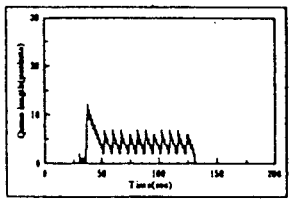
圖十四 兩條連線PP連線一的傳送速率變化



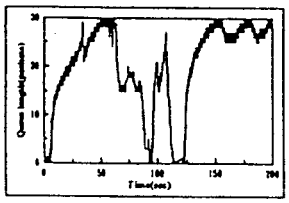
圖十五 兩條連線PP連線二的傳送速率變化



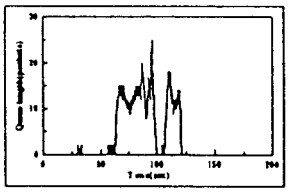
圖十六 兩條連線本文方法連線一的瓶頸佇列長度變化



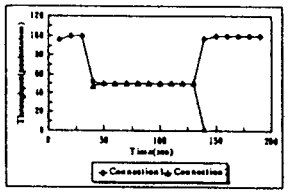
圖十七 兩條連線本文方法連線二的瓶頸佇列長度變化



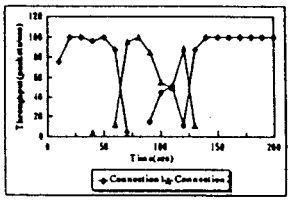
圖十八 兩條連線PP連線一的瓶頸佇列長度變化



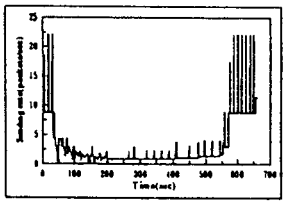
圖十九 兩條連線PP連線二的瓶頸佇列長度變化



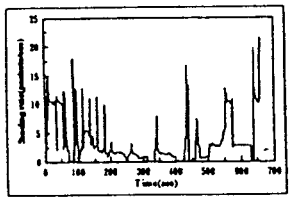
圖二十 兩條連線本文方法的產能變化



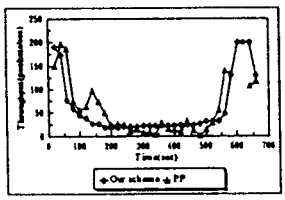
圖二十一 兩條連線PP的產能變化



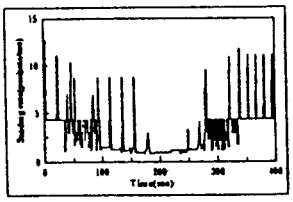
圖二十二 十條連線本文方法連線一的傳送速率變化



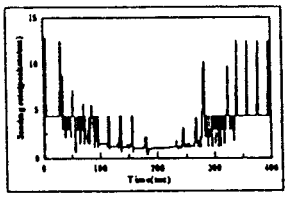
圖二十三 十條連線PP連線一的傳送速率變化



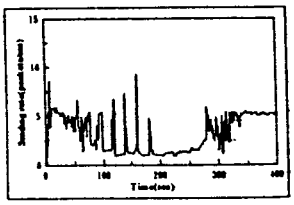
圖二十四 十條連線本文方法和PP連線一的產能比較



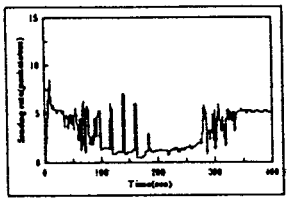
圖二十五 公平性實驗本文方法連線一的傳送速率變化



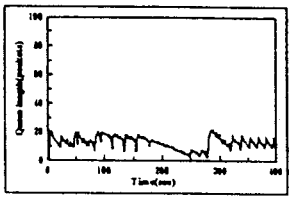
圖二十六 公平性實驗本文方法連線二的傳送速率變化



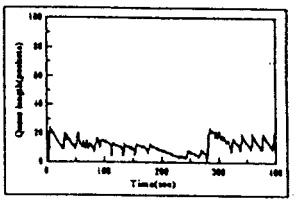
圖二十七 公平性實驗PP連線一的傳送速率變化



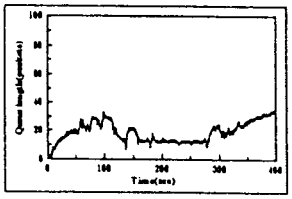
圖二十八 公平性實驗PP連線二的傳送速率變化



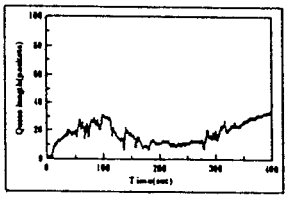
圖二十九 公平性實驗本文方法連線一的瓶頸佇列長度變化



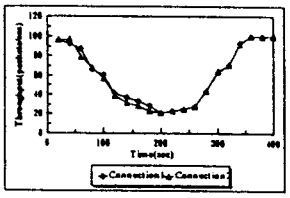
圖三十 公平性實驗本文方法連線二的瓶頸佇列長度變化



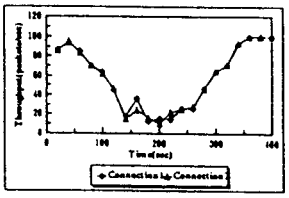
圖三十一 公平性實驗PP連線一的瓶頸佇列長度變化



圖三十二 公平性實驗PP連線二的瓶頸佇列長度變化



圖三十三 公平性實驗本文方法的產能變化



圖三十四 公平性實驗PP的產能變化