路由器輔助的 TCP 擁塞控制技術之設計

摘要

隨著網路訊務流量的快速成長，如何妥善的運用網路資源是一個成功的擁塞控制機制要面對的根本問題。在終端設備上執行的 TCP 是網路上最廣為使用者使用的傳輸層協定，它有很多不同的版本被設計出來改進使用的效能，例如 TCP Reno、TCP Vegas 等。由於 TCP 所棲身的終端設備並未具有網路內部狀態的資訊，大部份的 TCP 擁塞控制機制僅能依賴封包遺失觸發擁塞控制機制，本研究提出 TCP Muzha 協定，藉由路由器協助，提供網路內部資訊給傳送端，在未發生擁塞前不需依賴封包遺失便可進行適度的傳輸速度控制，以減少因為封包遺失所造成劇烈的傳輸速度下降，並可更快速達到最佳傳輸速度。本研究的設計理念是設法尋找傳送路徑中的瓶頸，進而計算出瓶頸提供的可用頻寬，藉由瓶頸所提供的資訊動態的進行流量控制以充份利用頻寬並避免產生擁塞，增進整體的效能。本研究之重點在於路由器應提供何種資訊及如何運用所獲得的資訊進行動態速率調整。我們提出模糊化的多層級速率調整方法，藉著動態所獲得的細膩資訊做擁塞避免。最後於 NS2 平台實驗模擬，評估我們所提出的方法，實驗結果中顯示本方法能有效避免擁塞的產生，降低封包遺失，提升整體效能，和 TCP Reno 共存的環境下不因為 TCP Reno 侵略性的傳輸方式而降低過多的效能並保有較低的封包遺失率。

Design of TCP Congestion Control Techniques by Router-assisted Approach

Abstract

With the tremendous growth of Internet traffic, to utilize network resources efficiently is essential to a successful congestion control protocol. Transmission Control Protocol (TCP) is a widely used end-to-end transport protocol across the Internet. It has several enhencing versions (i.e. TCP Reno, TCP Vegas…) which intend to improve the drawbacks of the initial version of TCP. Most congestion control techniques use trial-and-error-based flow control to handle network congestion. In this paper, we propose a new method (TCP Muzha) that requires routers to feedback their status to the sender. Based on this information, the sender is able to adjust the sending data rate dynamically. Our approach can prevent data rate from decreasing dramatically due to packet loss. It can also help to increase the data rate quickly to where it supposes to be. Our design philosophy is to find out the bottleneck of the path, and its available bandwidth. Our goal is to increase network performance and avoid congestion by using the information obtained from the bottleneck. The design challenges are to determine which information is essential and how to use this information to dynamically adjust the data rate. We also propose the multi-level data rate adjustment method. Congestion can be avoided by dynamically adjusting data rate using this information. Finally, we use NS2 simulator to evaluate the performance of our approaches. The experiment results show that our method can avoid congestion before it actually happens, decrease packet-loss rate and increase the network utilization. In the fairness experiment, our method will only

suffer a minor throughputs degradation when TCP Reno is coexisting.

誌謝

能夠完成論文，通過口試，首先要先感謝連耀南教授這兩年多來的指導，邏輯的思考，表達能力的培養，論文的撰寫，老師嚴謹的研究態度都讓我受益良多。

另外，要感謝廖婉君教授、雷欽隆教授、余孝先主任、張宏慶教授能撥冗參加我的論文口試，並在口試時針對研究的盲點，給予建議及指導。

再來要感謝張宏慶教授、蔡子傑教授這兩年來的指導，從平時的上課到 group meeting，到口試前的指導，皆受益良多。

碩士的兩年，同學之間的腦力激盪，互相砥礪，討論研究，打球娛樂都使碩士生活更充實，感謝 tightsman, hongchi, ferret,小江, addme,小銘,Larry, trying, herb,碩漢, 明翰,逸凡,永全，謝謝所有同學。

鍾永彬 October 13, 2005

目錄

圖目錄

表目錄

第一章

導論

1.1 簡介

近年來由於網際網路快速的發展，網際網路的使用人數也隨著迅速地成長。大量的使用者在有限的資源下進行網路各種應用產生過量的封包，便開始產生了一些問題，其中，擁塞的發生便是一個非常重要的問題。

網路擁塞之最根本原因是需求大於供給，使用者送入網路的封包大於網路資源容量和處理能力，使得延遲時間變長、封包遺失增加、整體效能下降。可以從圖 1.1 看出，最理想的狀況是網路的負載量剛剛好趨近於飽合，但是當發生擁塞後，整體的效能便會大幅度的下降。

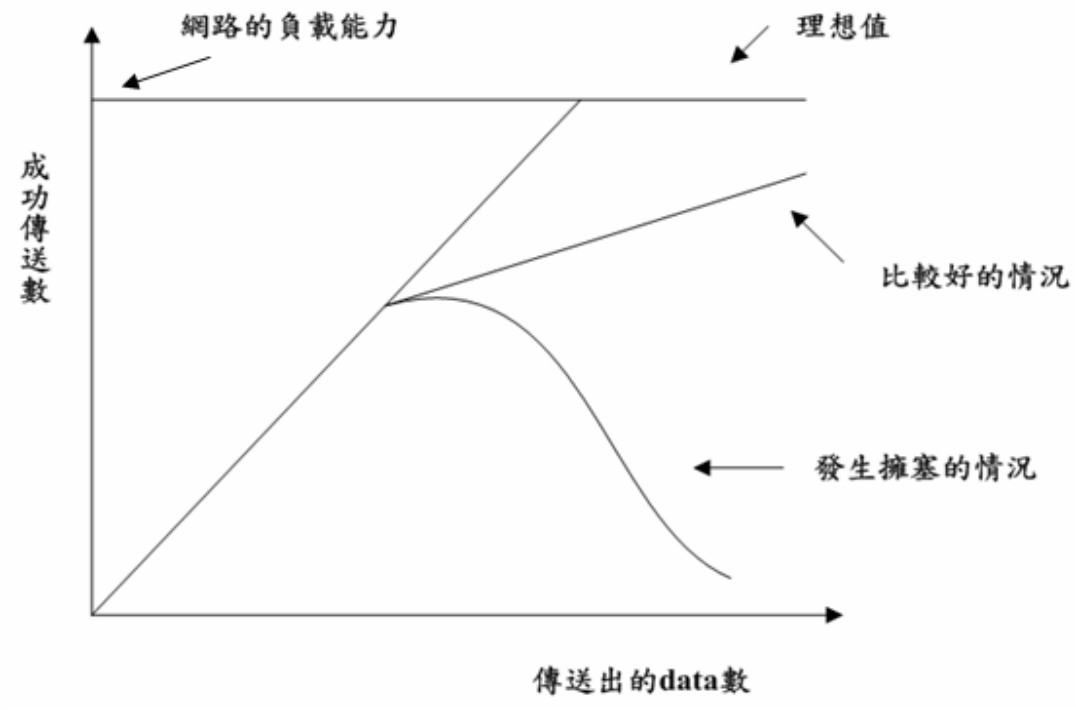

圖 1.1 擁塞狀態示意圖

產生擁塞的原因有可能為 ：

- 暫存的儲存空間不夠。
- 網路頻寬不足。
- 中間節點的處理能力不足。

要解決擁塞問題必須考慮上列可能的問題。

網路是一個開放環境，網路無法根據資源的情況限制用戶的數量，而缺乏中央控管的機制也無法限制使用者使用資源的數量。在網路上各式應用的大幅度成長以及使用者數目快速的增長下，若不使用某種機制來協調資源的使用，必會使網路擁塞，甚至造成崩潰。

雖然擁塞源自於資源的不足，但增加資源卻未必能解決問題，有時還有可能造成更嚴重的擁塞;例如在原本頻寬並未缺乏的路徑加大了頻寬後反而使得瓶頸部分更容易擁塞;又如增加了節點的 buffer 暫存空間時，可能使得封包經過的延遲時間增加，若超過了系統所預估的時間會導致封包重傳，反而加重了擁塞。這幾個例子都剛好反應出網際網路上擁塞控制的複雜性。

擁塞是發生在資源“相對”短缺的地方(如圖 1.2 所示)，因此，一昧的增加網路上的資源並不一定能有效解決問題，這也是擁塞控制困難的原因之一。

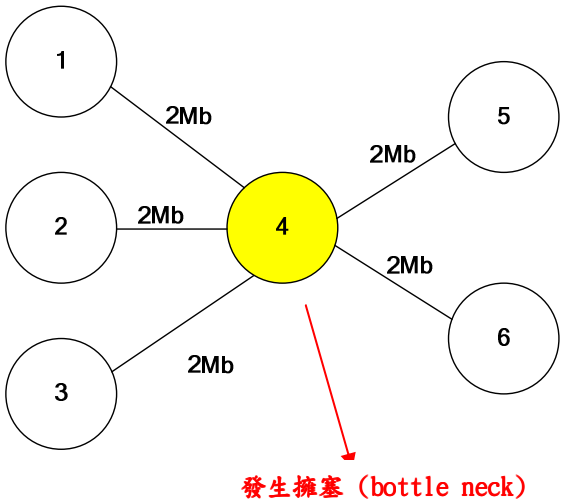

圖 1.2 發生擁塞的位置

擁塞控制是確保網路穩定的關鍵因素之一，也是各種管理控制機制和應用的基

礎。目前 Internet 上的資料傳送最普遍使用的傳輸層協定便是 TCP/IP 協定。而 TCP (Transmission Control Protocol) 內建的擁塞控制 (congestion control) 機制演算法是建構強健及穩定的網路連線所不可或缺的。

1.2 TCP (Transmission Control Protocol) 簡介

TCP[1]是目前網路上最被廣為使用的傳輸層協定是，它可保證封包的運送，它是一個全雙工的協定，這表示每一個TCP連結均支援兩條方向相反的封包流。對每一條封包流而言，TCP具有流量控制機制，允許接收端控制傳送端能傳送的速度。同時，TCP支援解多工機制以便允許同一主機上的多個應用程式能同時與其另一端進行交談。除了上述的特性外，TCP也包含了一個擁塞控制機制 (Congestion Control mechanism)。

TCP的滑動視窗演算法[1][2]扮演三種角色：第一個角色就是如何透過不可靠的連結可靠的運送封包。第二個角色是維持封包的傳送順序。第三個角色是支援流量控制；一種讓接收端能藉此調節傳送端的回饋機制。這個機制被用以防止傳送端造成接收端超載，也就是防止傳送過量的資料使接收端無法處理。這通常是藉由增強滑動視窗協定來達成。使得接收端不僅對所收到的封包作應答，還通知傳送端還有多少空間可以用來接收封包。

TCP靠著滑動視窗以及擁塞控制和擁塞避免的機制來提高整體的效能，TCP利用ACK (acknowledgement)、逾時 (time out) 和重傳的方式來確保可靠性並恢復遺失的封包。

1.3 TCP 的問題

TCP 的擁塞控制[3]是近來一直很關注的研究領域，由於內建不錯的擁塞控制機制 TCP Reno[4]是目前最常用的協定，我們以 Reno 為例來進行解釋現今 TCP 面臨的問題。TCP Reno 的擁塞控制機制主要分成兩個階段，慢啟動 (Slow Start) [5][6] 和擁塞避免 (congestion avoidance) [5]。

其中，慢啟動用於探測適當的頻寬，在連結建立的初始或是逾時發生時啟動，然而，當傳送兩端的距離很遠，路徑的可用頻寬很大，在不容易造成封包遺失的網路環境下，爬升的速度有所不足；而在網路上進行瀏覽網頁等檔案量小的傳輸時，慢啟動的不足，對於效能有很大的影響。

另外，TCP Reno 以封包遺失來判斷擁塞是否發生，當封包遺失或是逾時發生後，TCP Reno 便啟動擁塞控制的機制。但是，增加速度直到封包遺失的方式，容易週期性的產生擁塞，而這種只判斷是否發生擁塞的機制，並無法做精確的速度調控，當每一次認定擁塞產生，便快速的降低速度，而不斷的擁塞產生易使得整體效能震盪下滑，品質維持不易。

1.4 研究動機

採用慢啟動來探知 TCP 連結初始的可用頻寬有時會超過了網路的使用率，有時又對網路使用率的利用不足；增加速度直到擁塞產生的方法，無法避免會有擁塞的產生，而運用這種方式來調整速率，由於無法知道當下的網路狀況，因此只能採用保守的方式來做擁塞避免，顯得不足。

因此，我們想設計一個能夠在不依賴封包遺失下便做適當反應的 TCP 擁塞控制的方法，並依據網路的狀況，能夠更積極的增加速度或是積極的擁塞避免。

1.5 研究目標

我們以避免產生擁塞為主要目標，減少擁塞的次數能讓因為擁塞所造成的速度降低以及封包遺失的相關問題降低，對於整體的 QoS (Quality of Service) 也有所助益。另一方面也希望藉由適時的動態調整傳輸的速率，在不發生封包遺失的情況下，讓傳送端快速的達到最佳傳輸速率，使得 TCP 和整體網路的效能提升。並希望能在多協定共存的環境下，能保持穩定的效能。

1.6 論文架構

在第二章中我們會介紹相關的背景知識以及相關的研究工作，第三章

介紹我們的擁塞控制方法，在第四章以實驗模擬評估，第五章為結論以及未來展望

第二章

背景與相關研究

網路上的各個節點藉由擁塞控制的機制對網路上的擁塞行為進行反應，以確保整個網路的穩定度，否則整個網路容易因擁塞產生而導致崩潰 (collapse)。TCP (Transmission Control Protocol) 是現今網際網路上最被廣泛使用的傳輸層協定。它在兩個終端節點之間提供可靠的資料傳輸，現今很多的網路應用程式便是以 TCP 當作其通訊協定的基礎。

TCP 基本的策略是不斷的送出封包到網路中，並依事件的發生做反應。原始的 TCP 擁有簡單的滑動視窗(sliding window)的流量控制(flow control)機制，但是並沒有擁塞控制的能力[1]。在觀察到一連串的擁塞所造成的網路崩潰後，Jacabson 在 1988 年介紹了一些創新的 TCP 擁塞控制機制[3]，這個版本叫做 TCP Tahoe，包含了 Slow Start、 additive increase and multiplicative decrease (AIMD) 還有 Fast Retransmit algorithms。而在兩年後，把 Fast Recovery algorithm[4]加入後的版本，TCP Reno，成為現在最被使用的 TCP 版本 。

過去數十年來，TCP 的傳送端與接收端的擁塞控制機制主要由兩個擁塞控制演算法在真實網路中採用，一個是 TCP Reno，另一個則是 TCP Vegas。TCP Reno 被廣泛的使用在真實的網際網路中，RFC 中也有許多的文件在探討如何實行 Reno，表 2.1 列出了這些相關的 RFCs。接下來的段落，將摘要說明擁塞控制的特性並介紹一些擁塞控制機制像是 TCP Reno、TCP Vegas，除此之外，並介紹一些採用路由器輔助的擁塞控制技術像是 RED、ECN、TCP Rovegas 以及和本研究有關的一些相關資訊。

2.1 擁塞控制機制

2.1.1 擁塞的產生

產生擁塞的原因是當網路上的負載(load)大於網路的容量(capacity)、包括：暫存

的儲存空間不夠、節點的處理能力不夠。

擁塞主要發生在資源相對短缺的地方(瓶頸點)，並不能光靠一昧的增加網路資源來解決問題，這也是在面臨擁塞問題上困難的地方。

2.1.2 擁塞控制

擁塞控制主要可以區分為兩大類別，一種是從網路內部解決擁塞狀況(亦即在路由器或交換器)，另一種則從網路週邊解決擁塞問題(亦即在端點的傳輸協定中)。

在以路由器為中心的設計中，路由器決定傳送或丟棄那個封包，並知會那些產生網路流量的端點加速或減速。在一個以端點為中心的設計中，終端程式觀察本身在網路上成功傳送的封包數，並據以調整其傳輸率。以上兩種控制是同時運作的，彼此不一定互相協調，而是各自運作的。

應用程式如要避免擁塞，可以採用預留資源之方式，終端主機在建立連結時會向網路要求某特定量的資源。而每個路由器會配置足夠的資源以滿足此要求。如果在某些路由器上的資源不足，無法滿足此要求，路由器將拒絕此資料流。這種情況類似於撥電話時得到忙線中的回應。由於資源保留方式太複雜，很多應用程式並不採用，而使用回饋 (Feedback) 式降低擁塞之影響，終端主機在未預先保留任何資源的情況下開始傳送資料，然後再根據收到的回饋訊息而調整傳送速率。此回饋訊息可以是明示的：一個擁塞的路由器送回「請減慢速度」的訊息給主機，也可能是暗示的：終端主機根據觀察到的網路現象而調整其傳送速率，例如封包遺失。而在傳送端調整傳送速度之主要機制是滑動視窗。

產生擁塞的結果造成了頻寬的浪費、產生了重傳、服務品質受損、延遲時間拉長、使用者的吞吐量 (throughput) 降低、增加了負載率、並可能因此導致整個網路效能降低，因此我們需要做擁塞控制。擁塞控制的主要目的就是避免產生擁塞，使得網路負載 (load) 低於容納量 (capacity)。擁塞控制的複雜度在於無法預知使用者的傾向、網路資源的配置情形、整個網路架構所伴隨的問題等等。然而因其極為重要，因此一直有很多人在研究，以下我們介紹TCP擁塞控制機制。

2.2 TCP的擁塞控制

TCP 是以視窗為基準的傳輸協定，它藉由調整視窗的大小來做流量的控制，利用 ACK 的收到來判斷是否成功傳輸。其擁塞控制主要是藉由擁塞視窗 (congestion window)來做速度的控制而非直接以速率來做控制，大部份的 TCP 所採用的方式是藉由封包的遺失或是逾時來做為擁塞的產生與否的判斷並啟動擁塞控制。

2.2.1 TCP Tahoe and TCP Reno 的擁塞控制

表2.1列出了一些和TCP有關的RFC文件。

表 2.1:TCP 實作相關的 RFC 文件

RFC number	Topic
793	Transmission Control Protocol
1323	TCP Extensions for High Performance
2018	TCP Selective Acknowledgement Options
2581	TCP Congestion Control
2914	Congestion Control Principles
3168	The Addition of Explicit Congestion Notification (ECN) to IP
3390	Increasing TCP's Initial Window
3782	The NewReno Modification to TCP's Fast Recovery Algorithm

TCP 擁塞控制最早於 1988 年提出[3]，目前網路中最常使用的版本就是 TCP Reno[4]。TCP Reno 是以視窗為基礎的擁塞控制機制，依據 RFC 2001[5]，其主要分為四個狀態，Slow Start、Congestion Avoidance、Fast Retransmit、Fast Recovery。而它的擁塞視窗調整的演算法包含了三個部份，Slow Start、AIMD (additive increase/multiplicative decrease) [6]、Fast Retransmit and Fast Recovery。

TCP 擁塞控制主要分為兩大階段，一個是慢啟動 (Slow Start) ，另一個則是擁塞避免 (congestion avoidance) :

2.2.1.1 慢啟動(Slow Start)

在 TCP 連結建立的開始或是逾時產生後啟動慢啟動機制，接收端每收到一個封包，便回送一個 ACK，而傳送端在接收到每一個 ACK 後便把擁塞試窗大小增加，以倍數的方式遞增直到產生封包遺失而進入擁塞避免的階段。這個方法是希望緩慢的初始速度藉由倍數的成長來加快以達到最適頻寬。

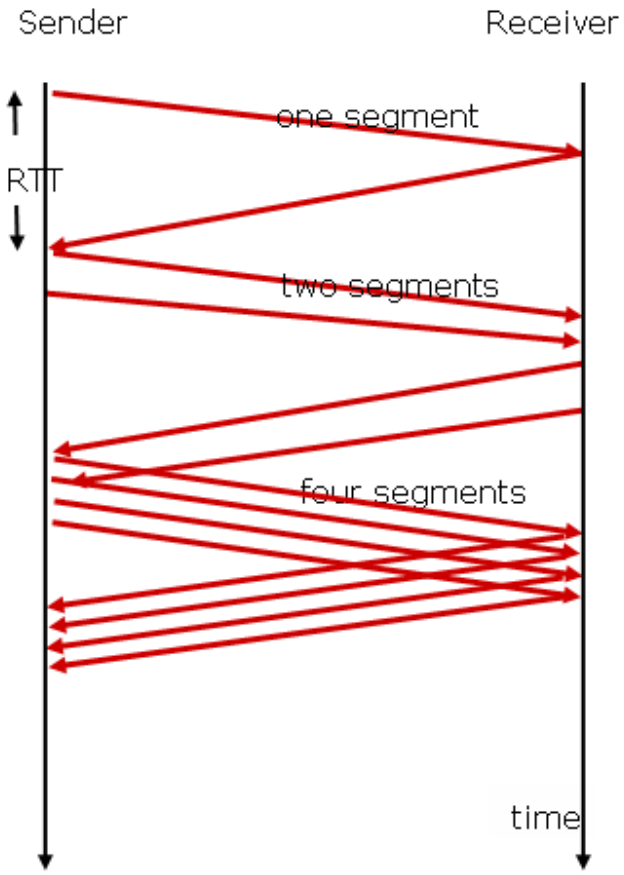

圖 2.1 慢啟動圖示

2.2.1.2 擁塞避免(congestion avoidance)

擁塞避免階段是在擁塞發生之後的速率控制機制，一則必須設法解除擁塞，二則既已偵知速率之極限，理當調整更恰當的速率。不同的 TCP 會做不同的方式，像是 AIMD、Fast Recovery、Fast Recovery 等等[4][5][6]。

AIMD 含有兩個階段:

- additive increase : 為了避免快速的又產生擁塞，Congestion Window Size

(CWND) 在每一個 Round Trip Time (RTT) 一次加一個 Maximum Segment Size (MSS)

- multiplicative decrease : 當擁塞產生後，CWND 之大小減半

其它較為詳盡的描述會在後面一一說明。

當一個 TCP 的連線建立後便進入 Slow Start[5]階段，這個階段的目的便是讓 TCP 藉由不斷的增加 CWND 的大小使得注入網路的資料量越來越大以間接的探索網路的可用頻寬。每接收到一個回覆的 ACK，CWND 便增加一個單位，如圖 2.1 所示，初始的時候，傳送端送出一個 CWND，當第一個 ACK 接收到以後，CWND 變增加成 2，當下一個 RTT 的 ACK 又收到以後，則增加成 4，以此類推，每一個 RTT，CWND 大小便以兩倍遞增。

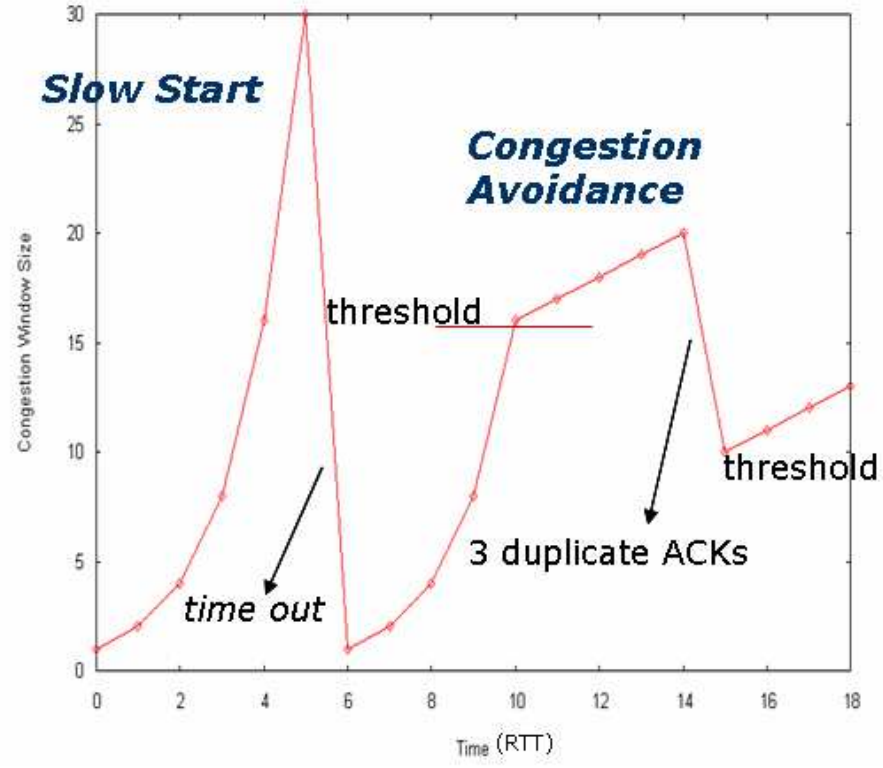

圖 2.2 TCP 擁塞控制機制示意圖

由於這種快速增加速率的方法，必定會使傳送速率超過網路容量，而導致網路的擁塞發生。在 AIMD 階段，CWND 在每接到一個 ACK 後以線性的方式增加 CWND

的大小，這個過程持續不斷的進行，直到擁塞產生後而 threshold 的大小被減半。

TCP Reno 藉由兩種方法偵查封包之遺失，一是 retransmission time out 的產生，另一個則是接收到三個重覆的 ACKs。當傳送端接到了三個重覆的 ACKs，Fast Recovery 和 Fast Retransmit 演算法便開始執行。

由於等到 timeout 發生在進行重傳常常會耗費過久的時間，造成效能高度的下滑，因此有了 Fast Retransmit 機制，當傳送端收到三個重覆的 ACK 後，便假設封包已經遺失，立即進入重傳的動作。

每當 TCP 接收端收到跳序的封包時，它會送回一個 duplicate ACK 給傳送端，在收到第一及第二個 duplicate ACK 時，接收端會先視為是網路上的延遲，但是如果繼續收到第三個或是更多的 duplicate ACK 時，那麼傳送端就當作有封包遺失了，在還沒發生 timeout 之前，就馬上重送遺失的封包，這就是 Fast Retransmission。

在 Fast Retransmit 之後，所進入的狀態可以是擁塞避免階段，而非慢啟動階段，此機制稱為 Fast Recovery。Fast Recovery 則是基於封包守恆的原則 (conservation of packets principle) [7]，同一時刻在網路中傳輸的封包數量是恆定的，只有在封包離開網路後，才能發送新的封包進入網路。因此，當 TCP 傳送端收到三個重覆的確認訊息，意謂著緊接在遺失封包之後已經有三個封包成功的到達接收端，也就是網路擁塞情況並不那麼嚴重，因此它不需要把 CWND 大小縮到 1，而大幅度降低傳輸量。所以 Fast Recovery 便是在快速重傳後，將 threshold 設為目前 CWND 的一半然後將 CWND 設為 threshold 加三(因為有三個封包到達接收端)。在這個階段，CWND 增加的方式是以每收到一個 duplicate ACK 時，CWND 的值就加一。如此一來，就不用因此大幅度降低傳輸速度，因此可以提高效能。

然而 TCP Reno 和其之前的版本都有個問題，就是 TCP 傳送端在 Round Trip Time 之內僅能重傳一個封包，因此在遭遇多封包遺失時容易導致逾時。因此後來衍生了一些版本如 TCP NewReno[7]、TCP Sack[8][9]均可處理多封包遺失的問題。

2.2.2 TCP Vegas

在 1994 年，L.S.Brakmo 等提出了一種新的擁塞控制策略 TCP Vegas [10][11][12]，由於 RTT 值與網路運行情況有密切關係，因此，TCP Vegas 通過觀察

TCP 連接中 RTT 值的改變來探知網路是否發生擁塞，從而控制擁塞視窗大小。它偵測網路的擁擠狀況以避免像 TCP Reno 的週期性封包遺失。其擁塞視窗並非週期性的增減，而是會達到一個穩定值。TCP Vegas 的評價是較為穩定且公平。如果發現 RTT 值變大，Vegas 就認爲網路正在發生擁塞，於是開始減小擁塞視窗；另一方面，如果 RTT 變小，Vegas 就認爲網路擁塞正在解除，於是再次增加擁塞視窗。這樣，擁塞視窗在理想情況下就會穩定在一個合適的值上。由於 TCP Vegas 不是利用封包遺失來判斷網路可用頻寬，而是以 RTT 的變化來判斷，因此能更精確地預測網路的可利用頻寬，維持高度的效能。

但是，TCP Vegas 之所以未能在網路上大規模使用，主要是因爲使用 TCP Vegas 的頻寬競爭能力方面不足，在跟 TCP Reno 之類競爭會導致網路資源享用不公平 [13][14][15][16]，效能大幅度下降。

2.3 採用 Router-Assisted 的擁塞控制方法

2.3.1 TCP RoVegas

RoVegas[17]是一個藉由 router 輔助來改良 Vegas 的方法。它主要在解決非對稱網路下 TCP acknowledge 的問題，當 TCP 協定在運作時，接收方在收到資料之後，必須回傳一個 ACK 給傳送端以告知資料送達，如果 ACK 因為返程的網路擁塞而導致無法順利回傳時，TCP 傳送端會誤以為正向網路擁塞，因而啟動擁塞控制機制，主動降低傳送速度。但是事實上，正向網路其實並沒有產生擁塞，這種情況下，用一般的方式處理會有問題。

RoVegas 利用路由器輔助的方法，在 TCP 中的 IP header 定義了一個新的欄位 AQT (accumulate queuing time) 藉由偵測的封包沿著整個 round-trip 路徑來收集正向和反向的 queuing time，使得 RoVegas 的發送端可以判斷正向是否產生擁塞。另外，RoVegas 也解決了重新繞路時，當重新繞路 (rerouting) 後，新的 route 有較長的 fixed delay，Vegas 無法判斷是因為較擁塞還是新的 route，因而誤判降低傳輸的速度大小的問題。

2.3.2 RED (Random Early Detection)

RED[18]擁塞控制機制的基本思想是透過監控路由器佇列的平均長度來探測擁塞，一旦發現擁塞逼近，就隨機的選擇某些經過該 Router 之 session 來通知擁塞，使他們在佇列溢出導致丟棄封包之前減小 CWND，降低發送資料速度，從而減少網路擁塞。由於 RED 是基於 FIFO 佇列調度策略的，並且只是丟棄正進入路由器的資料包，因此實施起來也較為簡單。

RED 演算法主要包含兩個部分:其一為計算平均佇列長度，以此作為對擁塞程度的估計；另一個就是計算丟棄封包的機率。

2.3.3 ECN (Explicit Congestion Notification)

ECN[19]最早為 Sally Floyd 所提出，相較於現行的 TCP Reno 以 Timer 和 ACK 來作為評估擁塞的依據，TCP 利用附加在 IP header 中的 ECN bits 來作為判斷網路是否發生擁塞的依據，TCP 在 IP header 設置一個兩位元的 ECN 欄位，一個顯示傳輸協定是否支援 ECN；另一個則由路由器負責註記，用以顯示是否發生了擁塞。

它利用了路由器的幫忙，當傳送端接收到了 ECN 標注的擁塞訊息，則啟動了擁塞避免的演算法。

將 RED 和 ECN 結合起來，如果擁塞是在佇列滿之前偵測到的，那麼除了用丟棄封包作為擁塞通知之手段外，對支援 ECN 的傳送端和接收端，還可以在 IP header 設置擁塞的通知。這樣可以避免不必要的丟包，特別是對短的 TCP 連接和對延遲敏感的 TCP 連接而言；另外也避免了不必要的 TCP 超時重傳。

2.4 New Net

前面的章節曾提到說，現在會發生的擁塞狀況跟當初網路的設計有密切的關聯，而過去的研究在做擁塞控制時大部份沒有運用路由器來做為輔助是因為實施上的困難。

Intel 和 HP 等公司最近合作的 PlanetLab[20]計畫推出了一個 New Net[21]的新想法，目前的架構下，應用端在看整個網路時就像一個黑盒子。然而應用程式和服務都需要網路來告知或提供一些資訊，以增進效能、可靠度、網路安全性、存取能力(accessibility)。要所有網路上所有的路由器是幾乎不可能的事。導致無法設計更好的 TCP 擁塞控制機制。

New Net 的想法，就是希望能得到網路的狀態用來幫助使用者，此種企圖與我們的理念不謀而合 。

2.5 小結

目前 TCP 擁塞控制的方法包含了 Fast Retransmit、Fast Recovery、Slow Start 等等，而目前的研究針對上面的幾種方式提出不少修改或微調的方法，也有利用 RED、ECN、RTT 等來做為判斷擁塞，調整速度的依據，另外也有一些針對特殊環境下做的改進，像是在高速網路下、無線網路下等等。然而，大多數的 TCP 擁塞控制技術因為看不到網路的內部狀態，因此有的時候爬升速度不夠快，有時後降速又太超過，因此，若是我們能獲得更細膩的網路資訊，便能改善現有的擁塞控制機制，適度的調整速度，能夠自主性的調控，並減少擁塞的產生。

在下一章中，我們將提出基於這個出發點的 TCP 擁塞控制技術。

第三章

TCP Muzha 擁塞控制技術

擁塞不會隨著網路處理能力的提高而消除，網路的複雜性和對擁塞控制演算法的性能要求，使得擁塞控制演算法的設計具有很高的難度，到目前為止，擁塞問題還沒有得到很好的解決。本研究提出的方法和之前學者所研究的方向並不相同，而是藉由路由器所提供的資訊協助 TCP 進行擁塞控制，經實驗後發現其的確有可行性，在後面的章節中將介紹我們所使用藉由路由器輔助的擁塞控制方法，TCP Muzha。

3.1 設計理念

大多數的 TCP 擁塞控制技術因為看不到網路的內部狀態，因此有時頻寬使用率過低，有時頻寬使用率過高，造成擁塞，封包遺失。因此，若是我們能獲得更豐富的網路內部資訊，應能有效改善現有的擁塞控制機制，減少擁塞的產生。

對一個路徑而言，擁塞是發生在瓶頸點，而這個瓶頸點往往是延遲時間增長、封包遺失的位置。

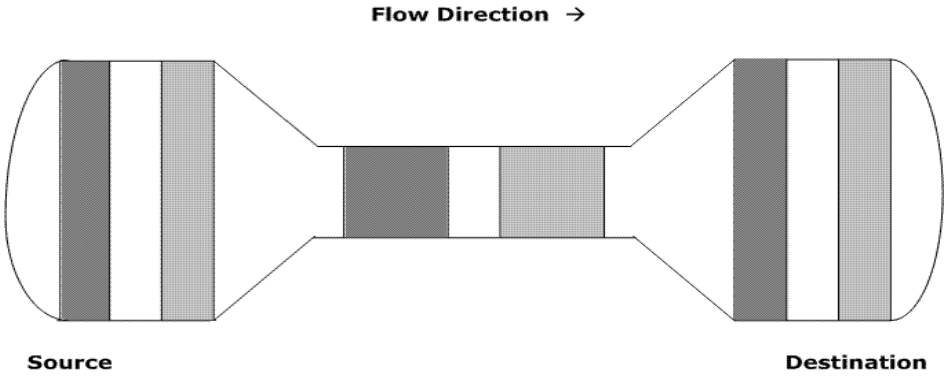

圖 3. 1 瓶頸示意圖

我們若能設法依據瓶頸點的狀況來動態調整傳送的速率，則可以有效降低擁塞的發生。因此，我們提出了 TCP Muzha 擁塞控制機制，希望能藉由路由器的協助以

(1) 偵測路徑中的瓶頸點，

(2) 估計瓶頸點可用的網路資源，及

(3) 利用瓶頸點提供的資訊做流量控制，

以達到高度的頻寬使用率以及減少擁塞的產生。

3.2 設計目標

我們的研究主要期望能達到 ：

(1) 減少擁塞的產生。

(2) 讓傳送端的傳送速度快速的達到適當的值。

在每一個新的 TCP 連結建立以後，傳送速度便會由初始的擁塞視窗開始慢慢的增加，一般的做法是採用慢啟動的方式 (Slow Start) 以倍數的方法爬升速度直到擁塞的產生。TCP Muzha 希望在不產生擁塞的情況下儘可能的快速達到適當的速度值。若能有效的降低擁塞之發生，便能降低因為重傳或是封包遺失所造成的網路資源浪費，進一步的提升整體網路資源的利用，提高整體效能。

(3) 在與 TCP Reno 等協定共存的環境下，仍能有穩定的效能。

目前最被廣泛使用的 TCP 協定是 TCP Reno，Reno 的侵略性擁塞控制的技術使得像 Vegas 在和它共存時，因為競爭不到頻寬，造成 Vegas 效能大幅度的降低，我們希望能維持穩定的效能。

3.3 設計重點

3.3.1 決定路由器的可用頻寬

如圖 3.2 所示，路由器可以藉由佇列長度、佇列等待時間、buffer size

的大小、剩餘的佇列長度等等參數，計算剩餘頻寬。各個路由器計算出剩餘頻寬後，接下來的問題是如何比較一個路徑上各路由器的剩餘頻寬。因為每一個路由器所採用的計算方式不一定相同，不能直接比較。

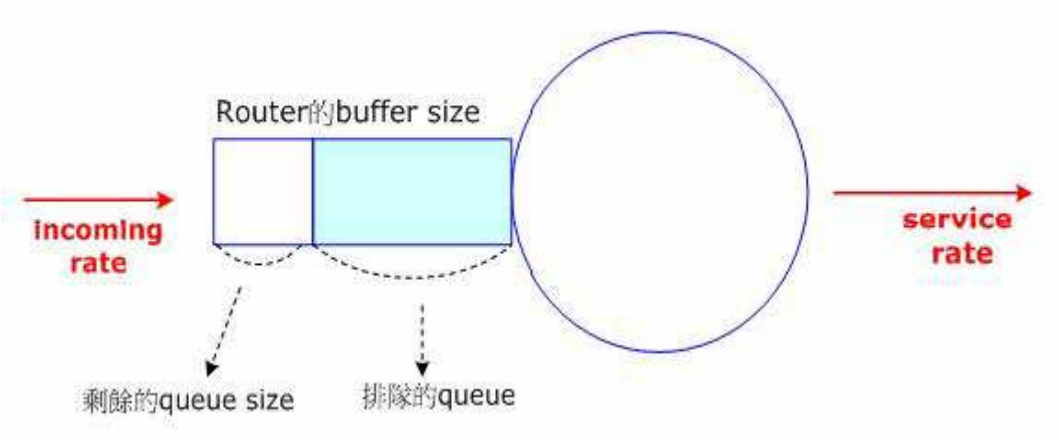

圖 3.2 路由器可用參數之圖示

3.3.2 找到瓶頸所在

TCP Muzha 定義了一個新的 IP option 叫做 AVBW-S (available bandwidth status) (圖 3.3)，沿著整個路徑收集各個 router 公佈的可用頻寬狀態。

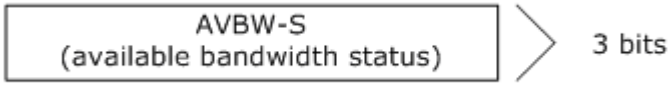

圖 3.3 AVBW 欄位

TCP 封包 (data 或 ACK)包含了 AVBW option 在它的 IP header 內。當具有此項功能的 TCP 傳送端建立連結並送出資料時，它把 AVBW 欄位設定為一極大值，在它經過的整條路徑中，每個支援此功能的 router 便把各自算出來的速率調整索引值 (data rate adjustment index) 跟該欄位的現值相比較，若是小於現值便取代之，經不斷的比較和取代之後，最後接收端將會取得最小的速率調整值，並傳回給傳送端。我們把這個最小的速率調整索引值稱為最小速率調整值，這是我們所想要獲取的資訊，我們猜測發生最小值的地方即為瓶頸。

請注意，每一條 TCP 連結會產生擁塞的節點位置都不盡相同，瓶頸點的位

置伴隨著每條 TCP 的連結而改變著。

傳送端則依據速率調整指示值動態的調整傳送的速率，以期達到避免產生擁塞並快速的達到最適速率的目標。

3.3.3 運用可用頻寬值調整速率

3.3.3.1 主控權的決定

當我們知道了路徑上的可用頻寬的資訊時，可以選擇兩種處理方式:

(1)直接傳送給傳送端，由傳送端決定如何使用

傳送端可以將此值和之前的記錄值比較，來決定速率增加、減少或是不變。

(2)路由器決定如何運用可用頻寬的值

我們選擇採用第二種方式，由路由器決定怎麼運用可用頻寬的值。理由是因為 TCP 之端點無法將眾多 TCP 連結在瓶頸路由器的狀況給考量進去。每個 TCP 連結由於進入的時間點不同，所面臨的網路狀況也不一樣，只有路由器本身知道有多少 TCP 連結經過瓶頸點，TCP 的傳送端並不了解整個的狀況，因此，由路由器來決定如何使用此值較為恰當。

3.3.3.2 解決共用頻寬的問題

一個路由器如果把剩餘頻寬的絕對值直接提供出來會產生一個嚴重的問題: 因為所提供的資訊是被多個通過同一節點的 TCP 聯結所分享的，網路上在同一時間內所建立的連結可能非常的多(圖 3.4)，這些 TCP 聯結很可能同時來搶這剩餘頻寬，很容易因此造成瞬間流量大減或爆增。

TCP Muzha 讓每一個路由器依據各別的網路狀況,各自決定一個模糊化的可用頻寬狀態值，提供給通過該路由器的 TCP 傳送端作為加速或減速之參考。一個路徑中各路由器間的最小值為最小速率調整值 (minimum data rate adjustment index , MRAI) ，這是我們所要獲得的資訊，下節有更詳細的說明。

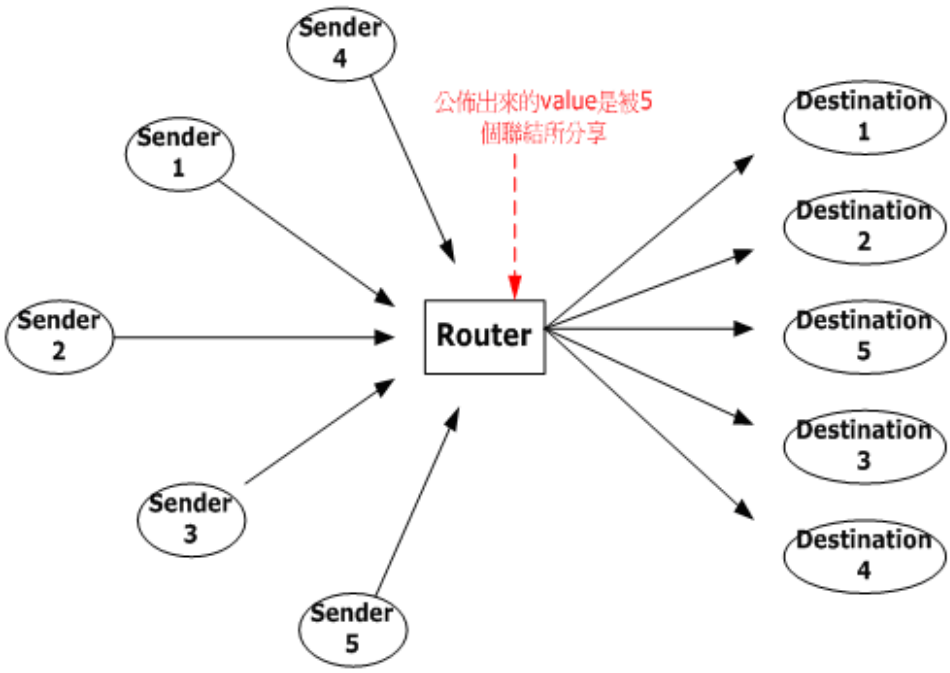

圖 3.4 多聯結共享同一頻寬

3.3.3.3 多級的速率調整

藉由路由器的協助，我們可以依據獲得的資訊(最小速率調整值)做調整，而不必依賴擁塞發生後才啟動控制的機制。

以 ECN 為例，它是藉由路由器協助的一個最簡單的例子，ECN 利用 RED 的佇列管理機制，當佇列大小超過一個臨界區間後，將 ECN 值設為 1，告知傳送端網路即將發生擁塞。此項資訊太簡略，傳送端並無法知道網路內部的細緻狀況，只能消極的採用減半降速的方式 (AIMD) 做擁塞避免。

ECN 這種兩元式的速率調整，無可避免的有不少控制上的缺失，也容易過度的調控傳送速率。如果 Router 提供的資訊較為充足，便可以設計更細膩的控制機制，因此我們提出了模糊化的多級速率調整方法。

由於我們獲得的資訊較為充足，為了避免直接公佈剩餘頻寬，我們要求路由器將剩餘頻寬以分級的方式轉換成 MRAI。但由於目前並沒有相關的研究理論依據能協助我們決定如何分級，我們將以實驗的方式來提供一些參考資訊。

3.3.3.4 多級速率調整之設計

為了加強控制的細緻度，在頻寬較不充裕的區段給予細密的間隔，而在頻寬充足的區段則給予較寬廣的間隔。頻寬充裕與否由路由器根據自己的情況自行決定，我們會在實驗中提供數據作為參考。

分級原則如下 ：

- 頻寬充裕時 ：選擇非常積極的加速。
- 頻寬較不充裕時 ：維持穩定或是積極的擁塞避免。

我們訂定積極加速、維持穩定、積極擁塞避免三階段，構成了我們的分級設計初步想法，最後在由實驗模擬得到進一步的分級和相對應的解決方案。

3.4 擁塞控制機制

3.4.1 TCP Reno 的擁塞控制機制

TCP Reno 在做擁塞控制的時候分為三個階段[4][5](圖 3.5)，

(1)慢啟動 (Slow Start) :

這個階段在連結剛建立以及在發生逾時的時候啟動，擁塞控制視窗以倍數成長的方式成長直到產生了擁塞。

(2)擁塞避免 (congestion avoidance) :

這個階段包含了 AIMD、Fast Retransmit、Fast Recovery。

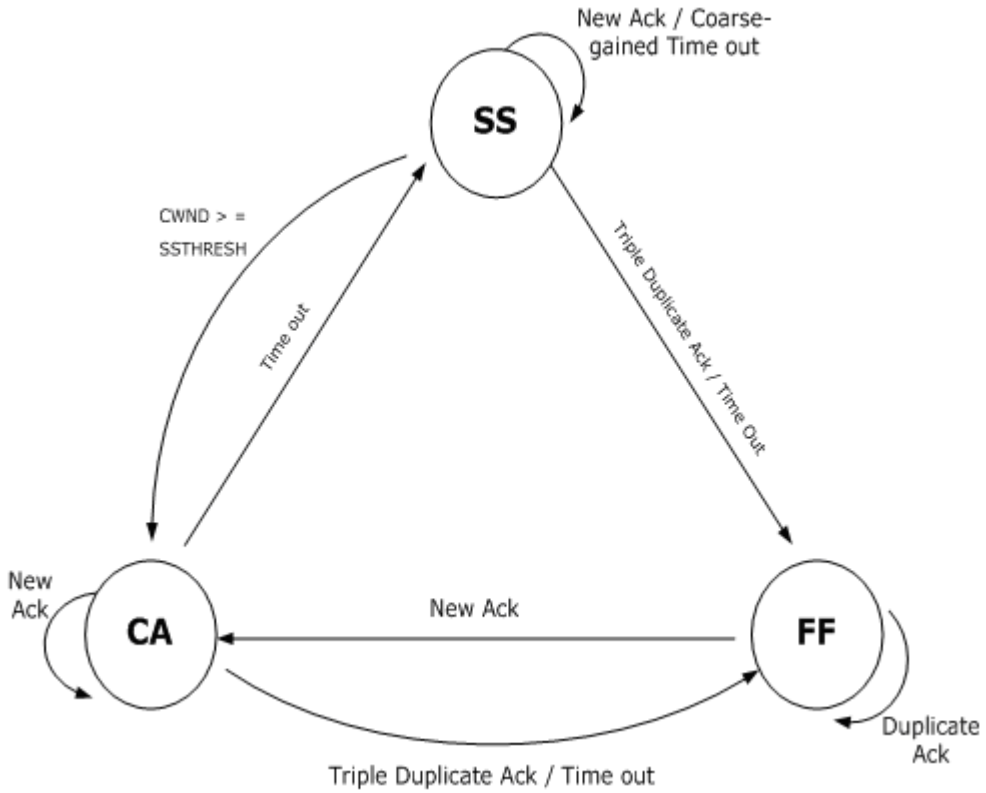

圖 3.5　TCP Reno 的狀態轉移圖

TCP Reno 和大部份的 TCP 所採用的擁塞控制研究歸納在表 3.1 中。

表 3.1 TCP Reno 擁塞控制設計的機制

事件	狀態	TCP 傳送端的行為	說明
接收到先前還未收到的 ACK	慢啟動階段 (SS)	(1) CWND=CWND*2 (2) 當 CWND > threshold 時，進入 congestion avoidance 階段	在每一個 RTT 便增加一倍
接收到先前還未收到的 ACK	擁塞避免階段 (CA)	CWND=CWND+1	在每一個 RTT，CWND 呈線性增加
由於三個重複的 ACK 所形成的封包遺失判斷	(SS & CA)	(1) threshold = CWND*(1/2) (2) CWND = threshold (3) 進入 congestion avoidance 階段	快速回覆以及減半降速
逾期(time out)	(SS & CA)	(1) threshold = CWND*(1/2) (2) CWND = 1 (3) 進入 Slow Start 階段	進入 Slow Start 階段

3.4.2 TCP Muzha 的擁塞控制機制

靠著路由器的協助，TCP Muzha 在未發生擁塞前不需依賴封包遺失便可進行適度動態的傳輸速度控制，因此，TCP Muzha 是一直處於擁塞避免的階段，而 TCP 原有的慢啟動階段則可以被我們隱含在擁塞避免階段，成為其中的一部份。因此，TCP Muzha 把 TCP Reno 採用的三階段狀態簡化成兩個(圖 3.6)，分為擁塞避免階段，CA (Congestion Avoidance)，並保有原有的 FF (Fast Recovery & Fast Retransmit) 快速回覆和快速重傳階段。

當 TCP 的連結開始後，便直接進入擁塞避免階段，當收到了新的 ACK 後會依據即時的最小速率調整值動態的調整 CWND 大小，若重覆收到了三個 ACKs 則進入快速回覆和快速重傳階段，並做發現擁塞時的降半速行為，若發生了逾時，則回到最開始的狀態重新開使進入擁塞避免階段。

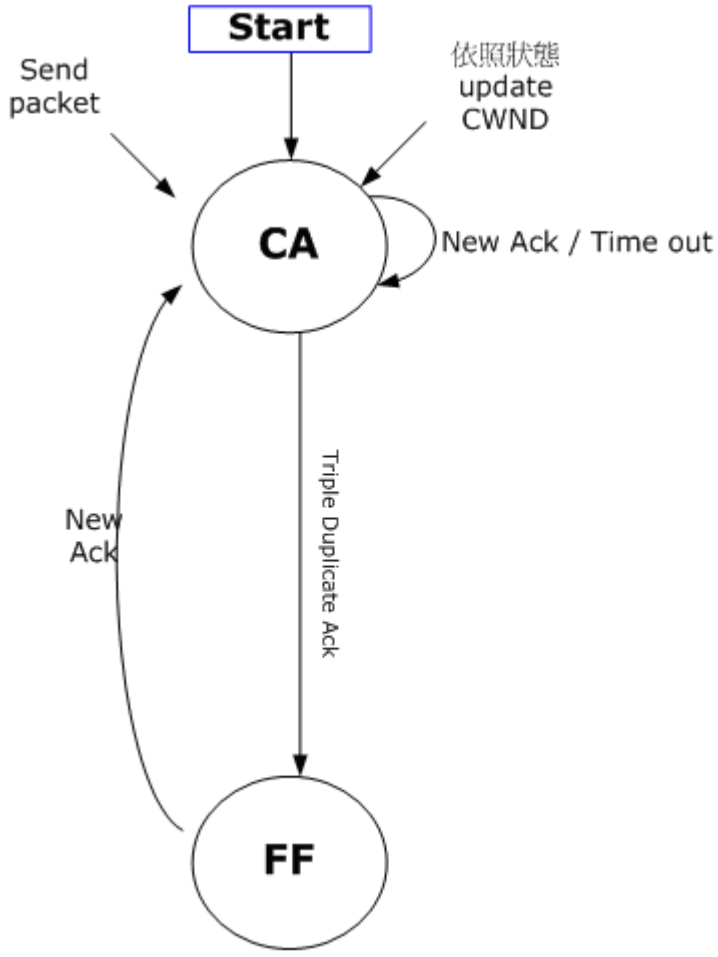

圖 3.6 TCP Muzha 的狀態轉移圖

當建立起 TCP 的連結以後，便開始進入擁塞避免的階段，這時候的 CWND

大小的調整是透過了帶有速率調整指示的 ACK 值而做動態的調整，有可能是高速的增加傳輸速度、緩步上升、維持不變、緩步的下降、幅度較大的下降等等，直到了無可避免的擁塞產生，造成了 3 個的重覆 ACK 或是逾時 (timeout) 的發生。

當遇到了 3 個重覆的 ACK 後，TCP Muzha 把 CWND 的大小直接減半，並進入 FF 狀態，以這個減半後的 CWND 當作初始速度繼續的傳輸，直到接受到新的 ACK 後，依據傳回的資訊再做調整。若是因為 time out 的原因，則 CWND 大小設回初始值 1，重新開始傳送。

遇到擁塞產生後所執行的動作 TCP Muzha 沿用了大多數 TCP 擁塞控制演算法，以最快速的下降速度來做當機立斷的處置，而由路由器所提供的資訊則讓 TCP Muzha 可以更有彈性的調整。

TCP Muzha 的機制歸納於表 3.2。

表 3.2 TCP Muzha 的擁塞控制設計機制

事件	狀態	TCP 傳送端的行為	說明
接收到先前還未收到的 ACK	擁塞避免階段 (CA)	依據傳回來的速率調整指示值動態的做 CWND 的調整	在每一個 RTT，CWND 適度的調整
由於三個重複的 ACK 所形成的封包遺失判斷	擁塞避免階段 (CA)	(1)CWND = CWND * (1/2) (2)進入 FF 階段	快速回覆以及減半降速
逾期(time out)	擁塞避免階段 (CA)	(1)CWND = 1 (2)重新進入 congestion avoidance 階段	重新進入擁塞避免階段

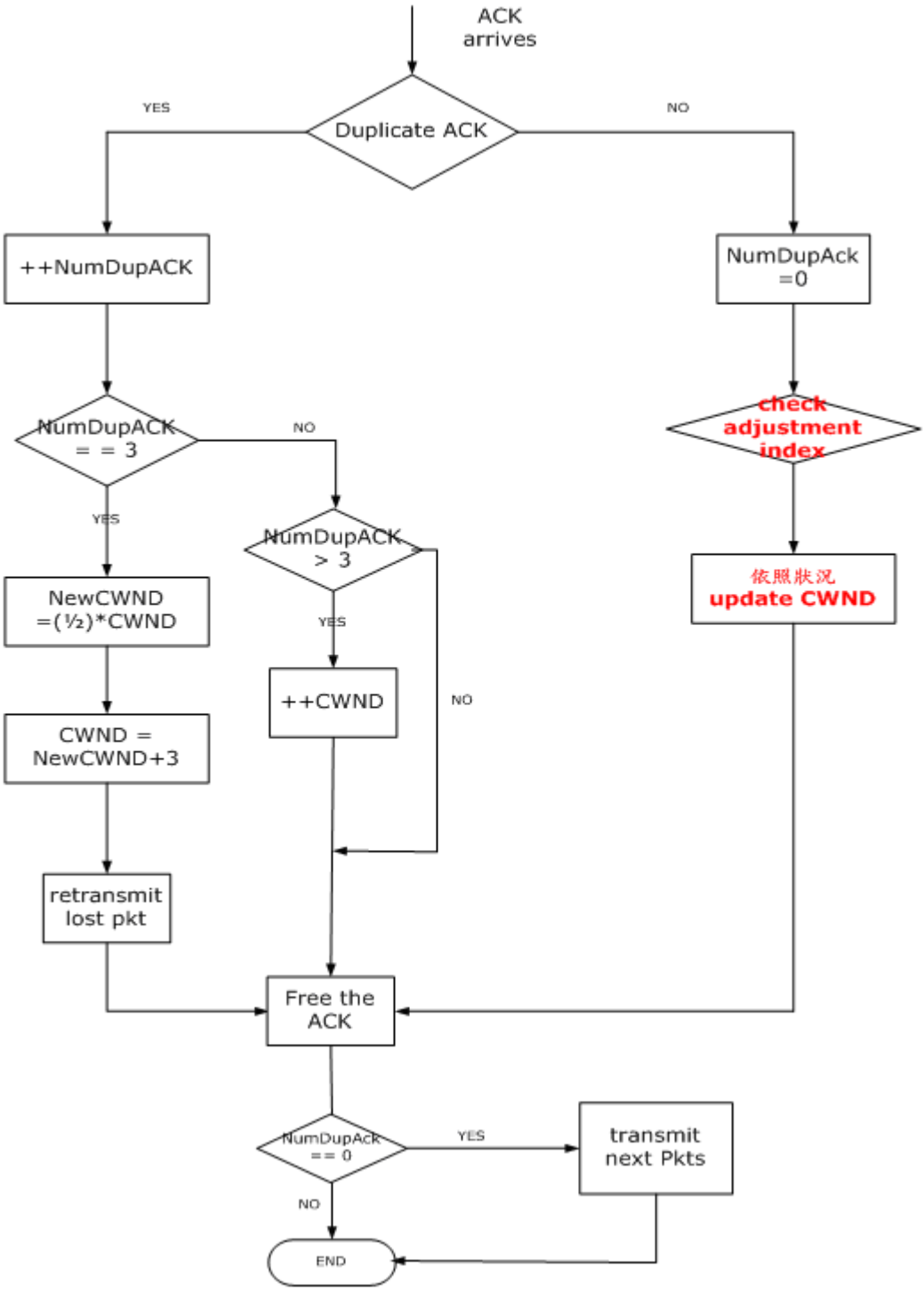

圖 3.7 TCP Muzha 接收到 ACK 後之流程圖

一般 TCP 的傳送端接到一個 ACK 時等同於接到一個 bit 的訊息，告知以前傳送的封包是否成功(並不擁塞)、如果是重複的 ACK 則告知另一種情況。透過這個訊息傳達了降速或是昇速的指示。在 TCP Muzha 中，我們可以使用較多個

bits 以表示更細緻的狀態。

3.4.2.1 TCP Muzha 的分級速率調整

要如何分級(包括各級之分隔值，各級之間隔等)，必須考量實際網路環境，頻寬的大小，連結的延遲狀況，buffer size 的大小，traffic load 的情形等諸多因素。由於要考量的因素很多，無法在有限時間內以實驗找出所有狀況應有的分級，所以在本文中以簡化的實驗模擬，粗略推出以六個分級的方式。在第四章中會針對其他的參數做出複雜的變化以評估方法。

圖 3.8 簡化的分級實驗拓樸

參數	範圍
節點數目	3
連結頻寬	1~5Mbps
連結延遲	7ms
buffer size	5 、 20 、 50
traffic load	100Kbps

圖 3.9 實驗參數

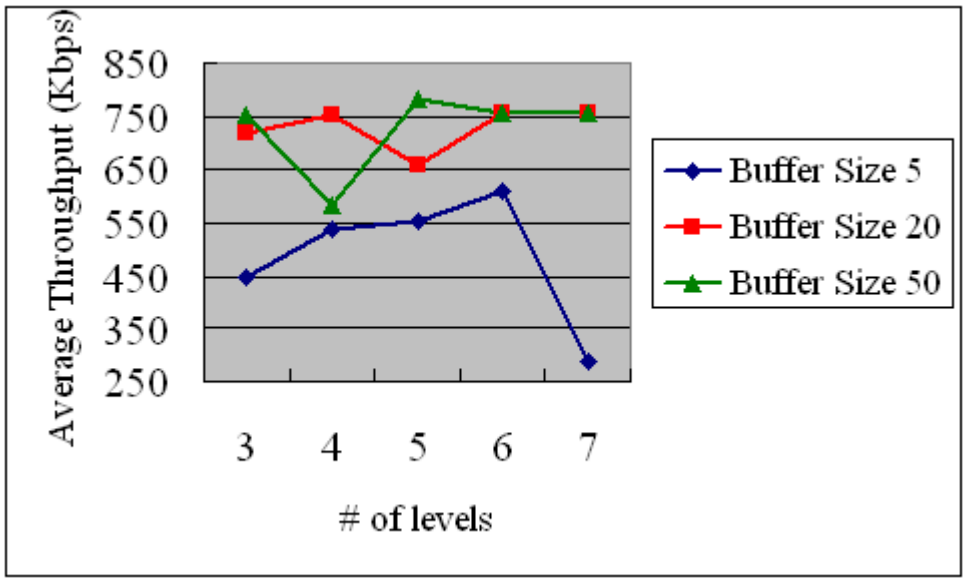

圖 3.10 不同層級在不同 buffer size 下的 average throughput

經過實驗模擬後，TCP Muzha 將 MRAI 分成三段六級。三段為積極加速，維持穩定，積極擁塞避免。

(1)積極加速

包含 5，6 兩級，以快速的提供傳輸速度為主要的目標，在小量但是頻繁的網路應用中，能否更快速的提升速度是影響效能很大的關鍵，而在遠距離或是高頻寬的連結中，越快的加速也越能提高整體吞吐量 (throughput)。

(2)維持穩定

包含 3，4 兩級，當獲知的網路可用資源並不充裕時，考量多個 TCP 連結共同分享的問題，而採用穩定的維持或是漸增的方式以慢慢的試探網路狀況以減少過快的產生擁塞。

(3)積極避免擁塞

包含 1，2 兩級，TCP Reno 並沒有這項行為，只有在發生擁塞後減半來做事後的處理，TCP Vegas 則以量測 RTT 的方式來進行調控。TCP Muzha 在這一項類別中，分成小幅度的降速以及 3/4 的降速，分成兩種是為了要能做更細微的微調。

表 3.3 TCP Muzha 之多級速率調整指示圖

AVBW Fields	
1	CWND = CWN * (3/4) + 2/CWND (得到的訊息顯示頻寬嚴重不足)
2	每個 RTT，CWND = CWND+1 (得到的訊息顯示擁塞將產生)
3	每個 RTT，CWND=CWND+1 (得到的訊息顯示處於適當狀態)
4	每個 RTT，CWND = CWND +2 (得到的訊息顯示目前在適當的狀態)
5	每個 RTT，CWND 以 1.75 倍成長 (得到的訊息顯示目前處於充裕的狀態)
6	每個 RTT，CWND 以 3 倍成長 (得到的訊息顯示可用頻寬相當足夠)

如表 3.3，在頻寬在較短缺的情形下，用較細的分類微調來達到我們以不產生擁塞的最高目的；以較廣泛的範圍做保持穩定升速，當資源充裕時，快速爬升速度。

這種多層級的分類及調整，有很多值得研究和探討的地方，本研究主要是提出這種擁塞控制的新想法，因此很細微的支節在本文中並未深入探討。

圖 3.11 多層級分類示意圖

3.5 小結

TCP Muzha 藉由路由器協助，提供網路內部資訊給傳送端，在未發生擁塞前不需依賴封包遺失便可進行適度的傳輸速度控制，以減少因為封包遺失所造成劇烈的傳輸速度下降，並期望能更快速達到最佳傳輸速度。TCP Muzha 的設計理念是設法尋找傳送路徑中的瓶頸，進而計算出瓶頸提供的可用頻寬，藉由瓶頸所提供的資訊動態的進行流量控制以達到充份利用頻寬並避免產生擁塞，增進整體的效能。TCP Muzha 之重點在於如何決定路由器提供的資訊、如何運用所獲得的資訊進行動態速率調整等關鍵問題。我們提出模糊化的多級速率調整方法，藉著動態所獲得的細膩資訊做擁塞避免。在下一章節中我們將於 NS2 平台實驗模擬，評估我們所提出之 TCP Muzha 擁塞控制機制。

第四章

效能評估

4.1 評估指標

本研究採用了路由器輔助幫忙的擁塞控制的機制，主要目標是讓傳送端的傳送速率能快速的達到適當的值，盡可能的減低擁塞的產生，提升 TCP 和整體網路的效能，我們將和現有的 TCP 擁塞控制方法做比較。

評估指標 ：

- CWND (congestion window size) 的變化狀況：由 CWND 的變化狀況可以知道速率爬升的速度，也可以明白產生擁塞的狀況。
- Average Throughput : 用 average throughput 來評估整體的效能。
- 封包遺失率:遺失封包佔成功傳送封包的比例，以觀察擁塞情形。
- 平均延遲的時間:封包平均傳送的延遲。

4.2 實驗設計

本研究提出一個新的 TCP 擁塞控制的機制 (TCP Muzha)，在實驗中除了觀察這個方法的效能外，並在各個不同的模擬環境中和傳統現在被廣泛使用的 TCP 擁塞控制像是 TCP Reno、TCP Vegas、TCP Tahoe 做比較。最後並觀察和 TCP Reno 共存的效能影響以及同步化的問題。

4.2.1 實驗工具

NS2[26]是一套模擬 IP 網路的模擬平台，NS2 (Network Simulator - version 2) 。 利用這套軟體，我們可以比以前更容易去模擬一套完整的實驗。簡單的先

建立起自己的情境模擬、需要的可能網路狀況，然後設定好相關的參數、通訊協定、組態後，交給 NS2 去執行得出一個結果。NS2 內建了不少的 protocols 可以提供我們使用，並可以根據自行發展的理論自行套用到 NS2 上。因此本研究採用 NS2 作為模擬環境，以具公信力之平台配合本研究所提出的理論，觀察實際結果並評論之。

4.2.2 實驗方法

實驗的拓樸環境由 3~16 個節點組成，依各個子實驗分別調整不同的參數觀察其結果，在部份子實驗中以具有 burst 性質的訊務加入其中以增加網路資源的變化，我們將以 TCP Reno、Vegas、 Tahoe 做為對照組，配合著我們的各個實驗觀察評估。

4.2.3 實驗參數

表 4.1 實驗參數

參數	範圍
節點數目	3~16
鏈結頻寬	1~15Mbps
鏈結延遲	5~96 ms
buffer size	15~100 packets
traffic load	300~900Kbps

4.2.4 實驗步驟

我們主要分為四組模擬實驗來評估效能:

1. 觀察單一 TCP Session 下的擁塞視窗變化

2. 觀察多個 TCP Session 下的整體效能

2A. 調整 buffer size 大小來觀察對效能的影響

2B. 調整 traffic load 的大小來觀察對效能的影響

2C. 調整 link delay time 大小來觀察對效能的影響

3. 觀察多協定共存狀態下的公平性 (指和 Reno 共存的情形)

4. 觀察 TCP 同步化的情形

4.3 實驗 1 : TCP 連結的的擁塞視窗變化

4.3.1 實驗目標

觀察在單一 TCP 的情況下，TCP Muzha 擁塞控制機制的擁塞視窗變化。

4.3.2 實驗流程

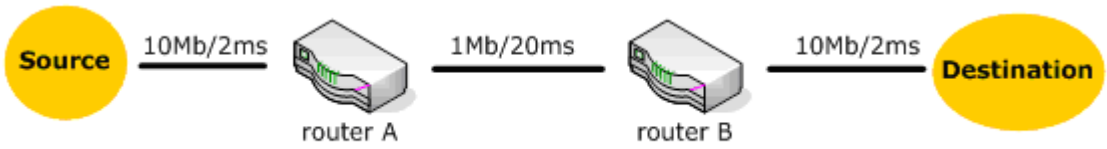

圖 4.1 實驗一的拓樸

圖 4.1 為我們實驗的拓樸，這是一個由兩個路由器組成的簡單架構圖，在這個實驗中我們只建立一個 TCP 資料流，路由器和路由器之間的連結頻寬是 1Mb，延遲時間 20ms，傳送端到路由器以及接收端到路由器的連結頻寬是 10Mb，延遲時間 2ms，路由器佇列的管理機制為 DropTail。我們想透過這個實驗觀察在最簡化的 TCP 資料流下，TCP Muzha 的狀態變化。

我們測試了 buffer size 為 50 和 15 的兩種情形，藉由這兩種狀況來觀察當封包容易遺失時和一般環境下 CWND 的改變。

4.3.3 實驗結果分析

我們由圖 4.2 可以看到 TCP Muzha 在擁塞視窗快速爬升後就不斷進行調控並最後在一狹幅區間內震盪。Vegas 採用的方式使得擁塞視窗維持的很穩定，但是也由於它比較保守的速度調整使得擁塞視窗並沒有爬升到很高；在 Reno 和 Tahoe 中，CWND 則快速的爬升而封包經常遺失並週期性的循環。

由於當路徑中路由器的 buffer size 很小的時後，封包遺失的情形將會提高，因此，接下來我們觀察 CWND 在 buffer size 很小的時後的變化。

由圖 4.3 我們發現，由於 buffer size 很小，TCP Muzha 的 CWND 在快速爬升後仍然無法避免的發生一次擁塞，但之後則動態的調整速度並達到穩定；Reno 和 Tahoe 中的 CWND 則反覆的進行大幅度的震盪行為；Vegas 的 CWND 則維持保守但穩定的策略。

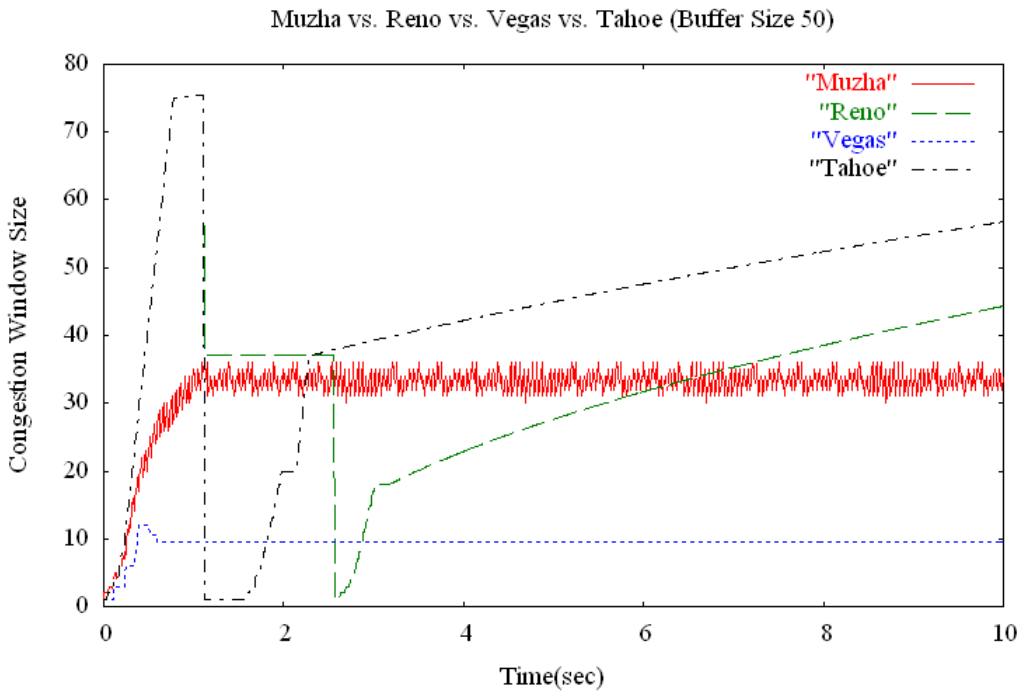

圖 4.2 實驗一中 TCP 資料流在 buffer size = 50 的擁塞視窗變化圖

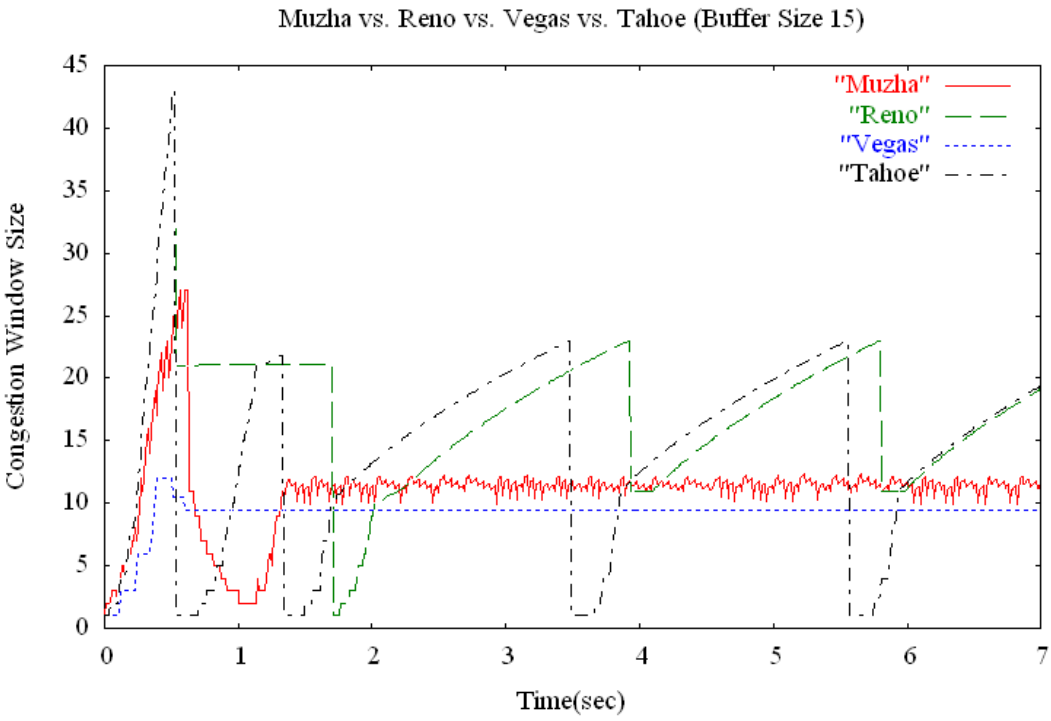

圖 4.3 實驗一中 TCP 資料流在 buffer size =15 時的擁塞視窗變化圖

4.4 實驗 2：探討多個 TCP 鏈結下的整體效能

在較複雜的情景下，分別調整 buffer size、traffic load、delay time 以觀察我們所提的方法的整體效能 (average throughput、封包遺失率、平均延遲時間)的狀況變化。

4.4.1 實驗 2A：探討 buffer size 對效能的影響

4.4.1.1 實驗目標

調整 buffer size 的大小並觀察 TCP Muzha 的整體效能 (average throughput、封包遺失率、平均延遲時間)變化。

4.4.1.2 實驗流程

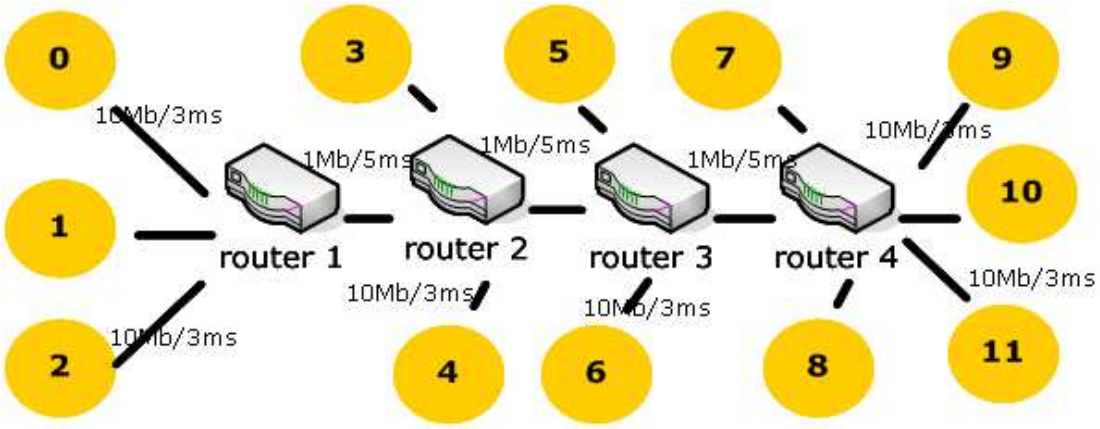

圖 4.4 實驗 2A 的拓樸

圖 4.4 為我們實驗 2A 的拓樸，這是一個由多個路由器和多個節點組成的魚骨狀拓樸圖，在這個實驗中我們建立了數個 TCP 資料流，通過中間的路由器到達右手邊的接收端，並增加 traffic load 100K 具有 burst 性質的訊務以增加頻寬變動的情形，模擬時間為三十秒，路由器和路由器之間的鏈結頻寬是 1Mb，延遲時間 5ms，傳送端到路由器以及接收端到路由器的鏈結頻寬是 10Mb，延遲時間 3ms，路由器佇列的管理機制為 DropTail。

表 4.2 實驗 2A 參數表

實驗 2A	
Buffer Size	5~100 packets
Traffic Load	100K
鏈結頻寬	1~10Mb
延遲時間	11~21ms

我們變化路由器的 buffer size 大小從 5-100 個封包，並觀察整體效能變化。

4.4.1.3 實驗結果分析

我們由圖 4.5、圖 4.6、圖 4.7 可以看出，當 buffer size 極小的情況下，我們的方式效能表現低於 Vegas，因為 buffer size 過小，導致 CWND 在快速爬升的途中 buffer 容易因此產生滿溢，造成效能下降，Vegas 採用的方式是保守但穩定的維持小試窗，並因此讓佇列長度維持一穩定的低點而能夠有良好的延遲時間，因

此有著優越的傳輸效能，TCP Muzha 雖然以避免擁塞為最高目標但仍保有一定的積極性，所以在 buffer size 極小的情況下表現不甚理想，但是當 buffer size 變大後整體效能就漸漸的變好，封包遺失率也能維持在穩定的低點，而平均延遲時間也比 Reno 小，而 Reno 只有在 buffer size 很大時 (buffer size = 100) 靠著積極的特性提高了效能。

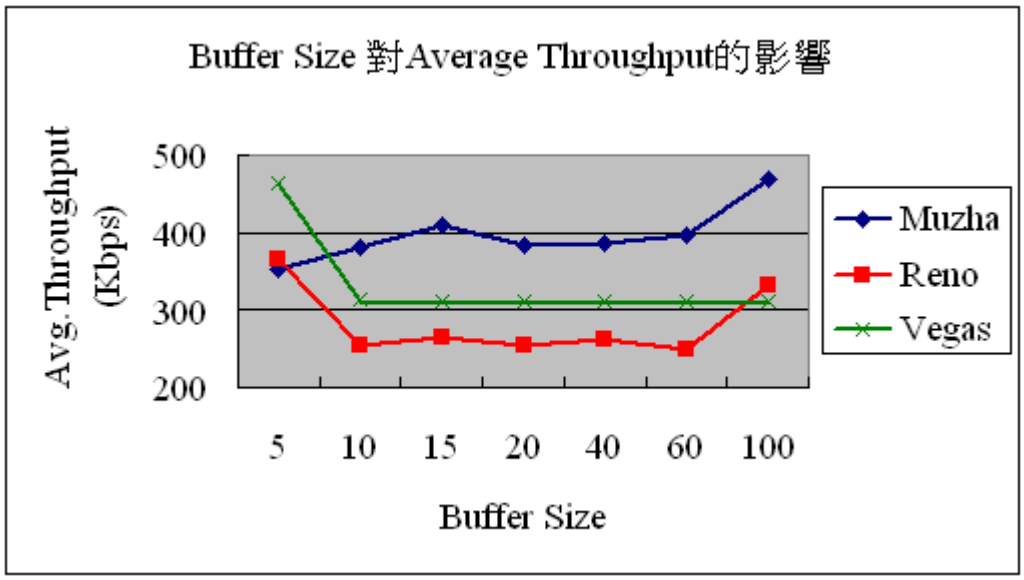

圖 4.5 實驗 2A 變動 buffer size 對 average throughput 的影響

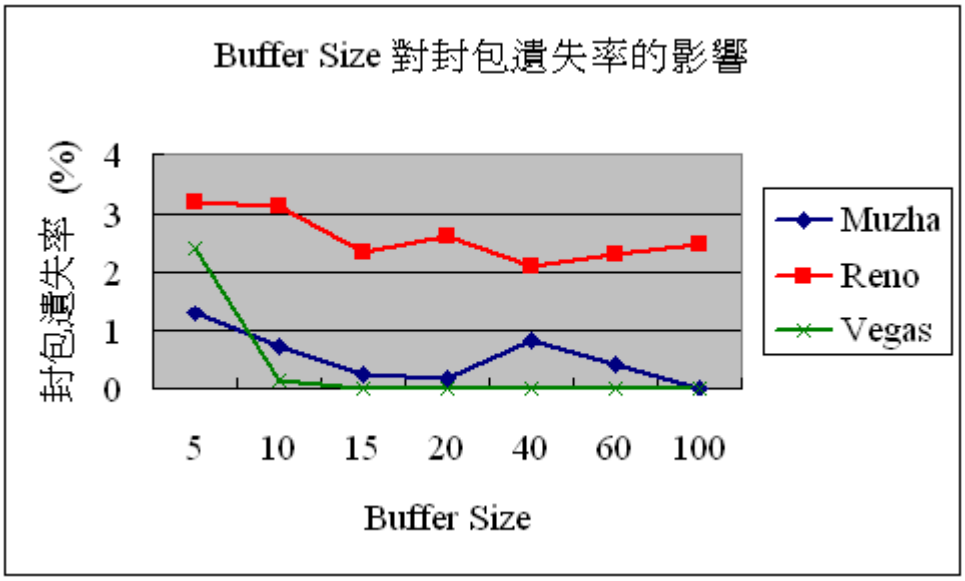

圖 4.6 實驗 2A 變動 buffer size 對封包遺失率的影響

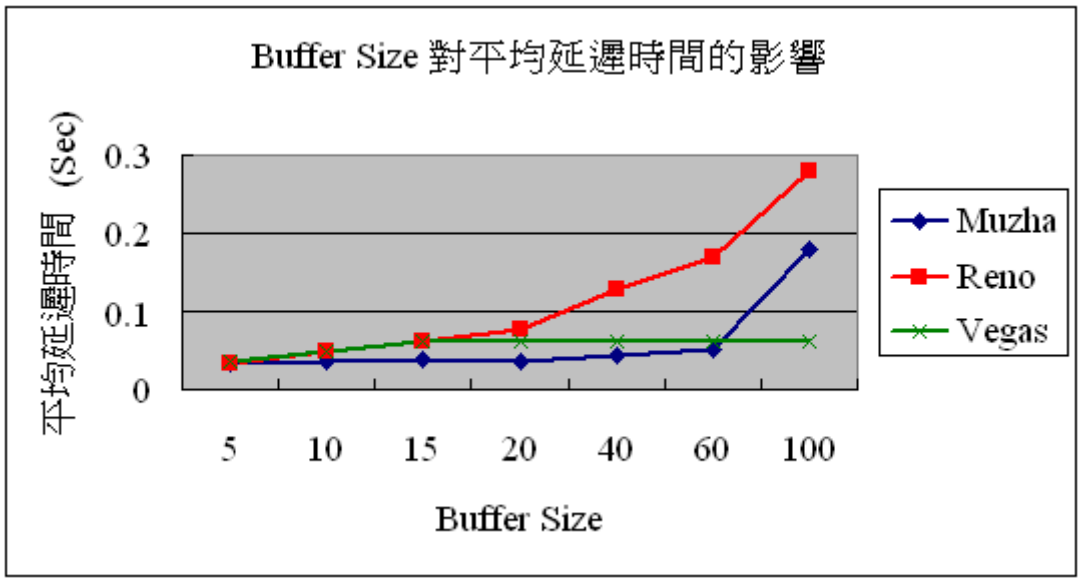

圖 4.7 實驗 2A 變動 buffer size 和平均延遲時間的關係圖

4.4.2 實驗 2B：探討 traffic load 對效能的影響

4.4.2.1 實驗目標

調整 traffic load 的大小並觀察當我們所提的方法面臨網路頻寬變化劇烈的環境下的整體效能 (average throughput、封包遺失率、平均延遲時間) 變化。

4.4.2.2 實驗流程

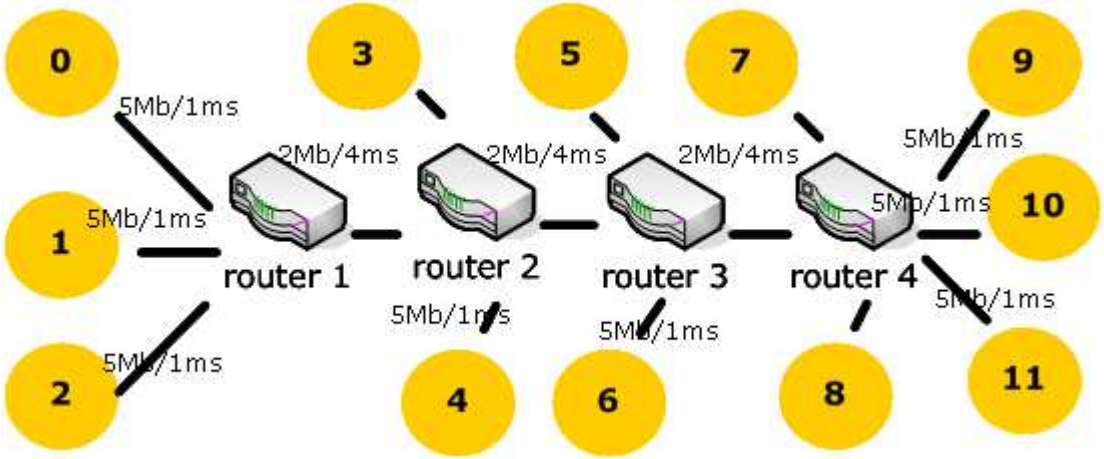

圖 4.8 實驗 2B 的拓樸

圖 4.8 為我們實驗 2B 的拓樸，我們延用了由多個路由器和多個節點組成的魚骨狀拓樸圖並做變化，在這個實驗中我們建立了數個 TCP 資料流，通過中間

的路由器到達右手邊的接收端，主要觀察當增加具有 burst 性質的訊務時變動的情形，模擬時間為三十秒，路由器和路由器之間的鏈結頻寬是 2Mb，延遲時間 4ms，傳送端到路由器以及接收端到路由器的鏈結頻寬是 5Mb，延遲時間 1ms，buffer size 大小為 20 個封包，路由器佇列的管理機制為 DropTail。

表 4.3 實驗 2B 參數表

實驗 2B	
Buffer Size	20~100 packets
Traffic load	300、500、700 、900K
鏈結頻寬	2~10Mb
延遲時間	6~15ms

我們變化路由器的 traffic load 大小從 300K~900K 並觀察整體效能變化。

4.4.2.3 實驗結果分析

我們由圖 4.9、圖 4.10 可以看出，當 traffic load 低於 500K 的情況下，Vegas 效能略高於 TCP Muzha，但是當 traffic load 高過 500K，在 700K 和 900K 的情況下，TCP Muzha 則優於 Vegas，這個原因是因為 Vegas 保守的擁塞避免方式在 traffic load 很高的情況為了避免產生擁塞因此保持很低的擁塞控制視窗，所以整體效能下滑，TCP Muzha，在積極的速度下並維持著擁塞避免，效能在 traffic load 增加下只呈現緩慢的下降，Reno 則因為有 traffic load 而更容易造成擁塞導致效能不佳。

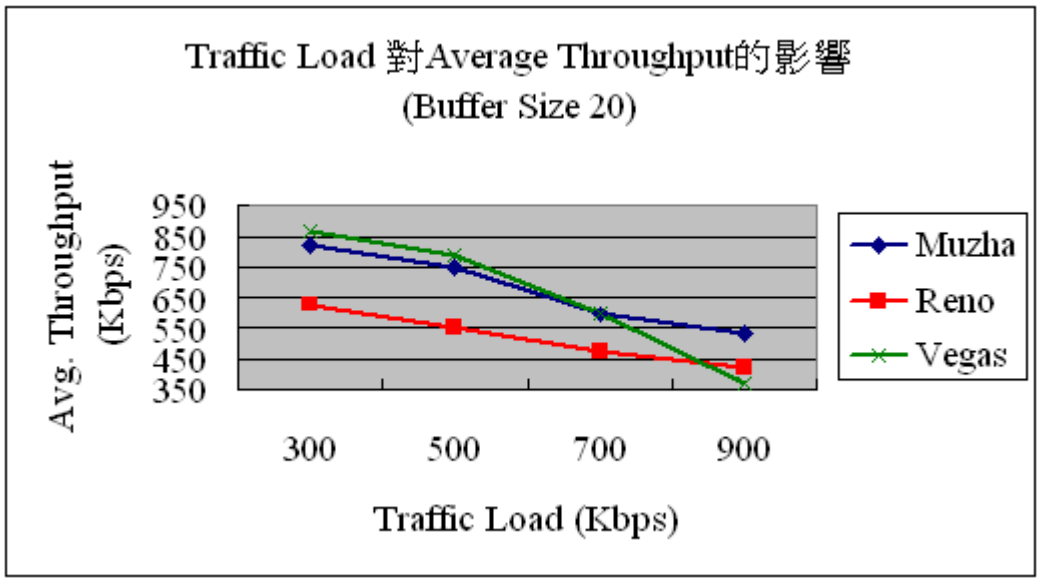

圖 4.9 實驗 2B 變動 traffic load 對 average throughput 的影響 (buffer size = 20)

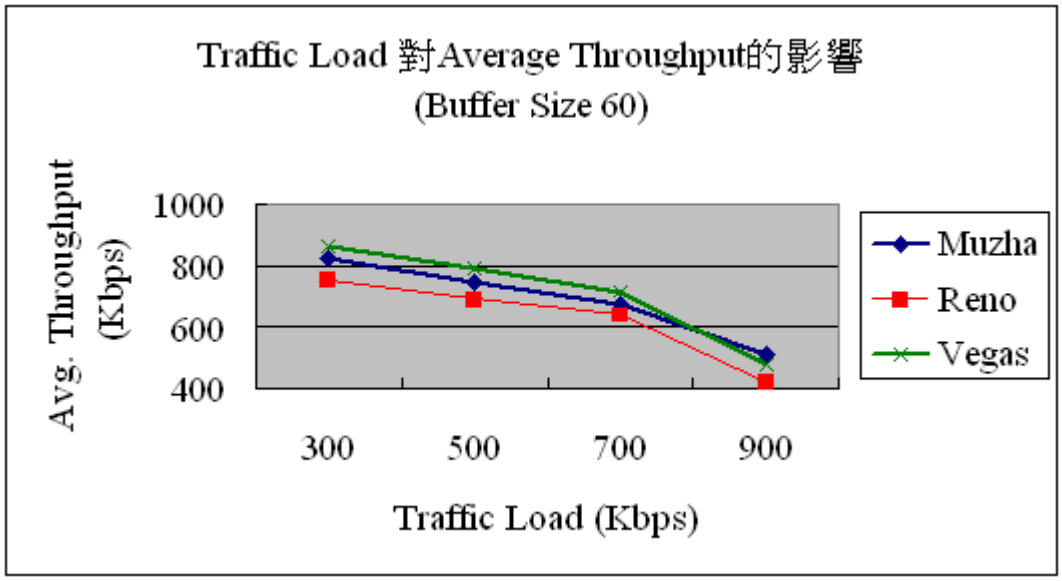

圖 4.10 實驗 2B 變動 traffic load 對 average throughput 的影響 (buffer size = 60)

隨著 traffic load 的不斷提升，TCP Muzha 的封包遺失率則能一直穩定的維持低點，Reno 則因而產生大量的封包遺失

在平均的延遲時間方面，TCP Muzha 則遠低於 Reno，略高於 Vegas。

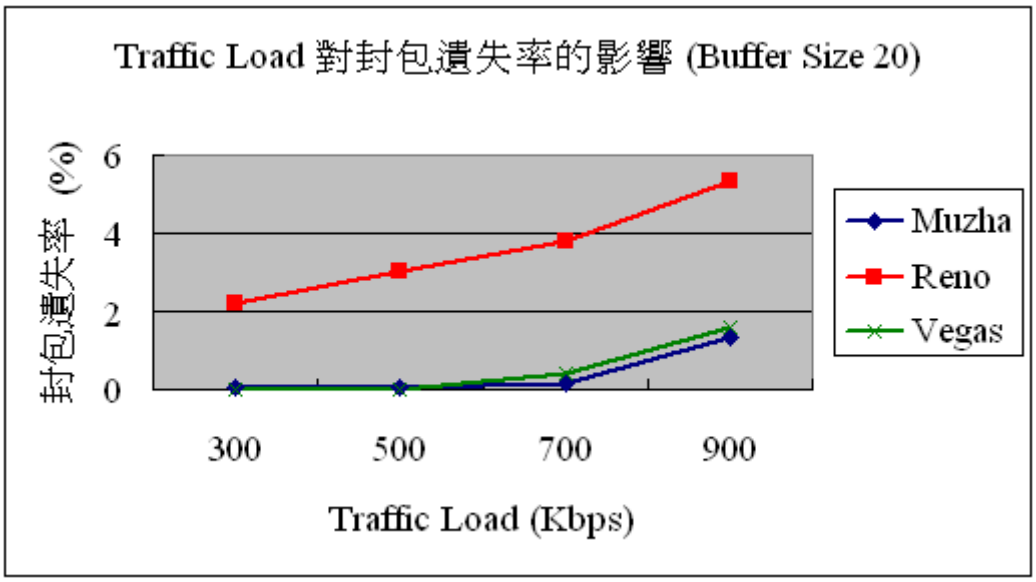

圖 4.11 實驗 2B 變動 traffic load 對封包遺失率的影響 (Buffer Size=20)

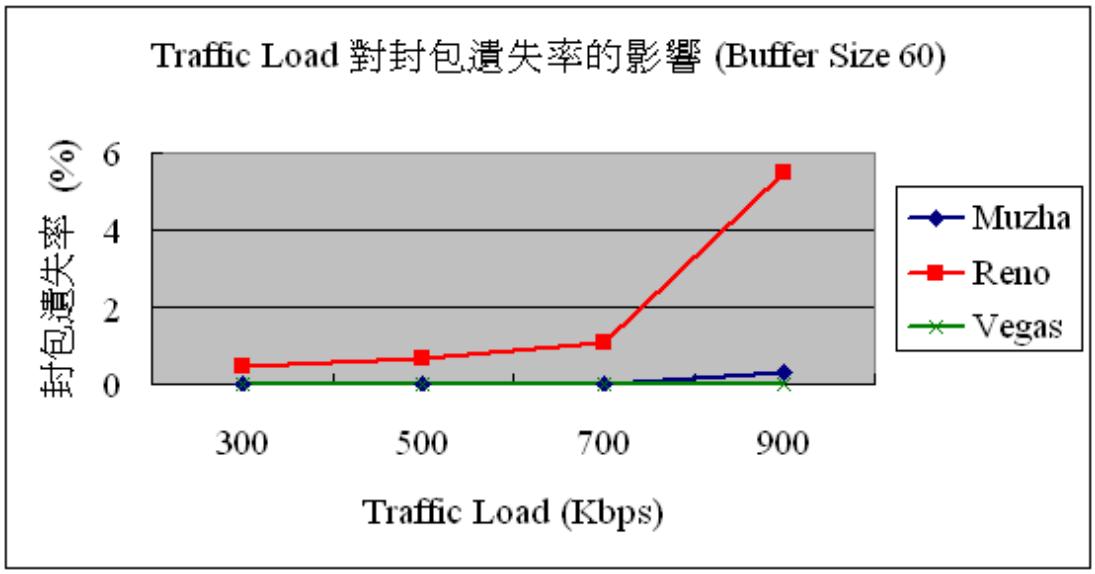

圖 4.12 實驗 2B 變動 traffic load 對封包遺失率的影響 (Buffer Size=60)

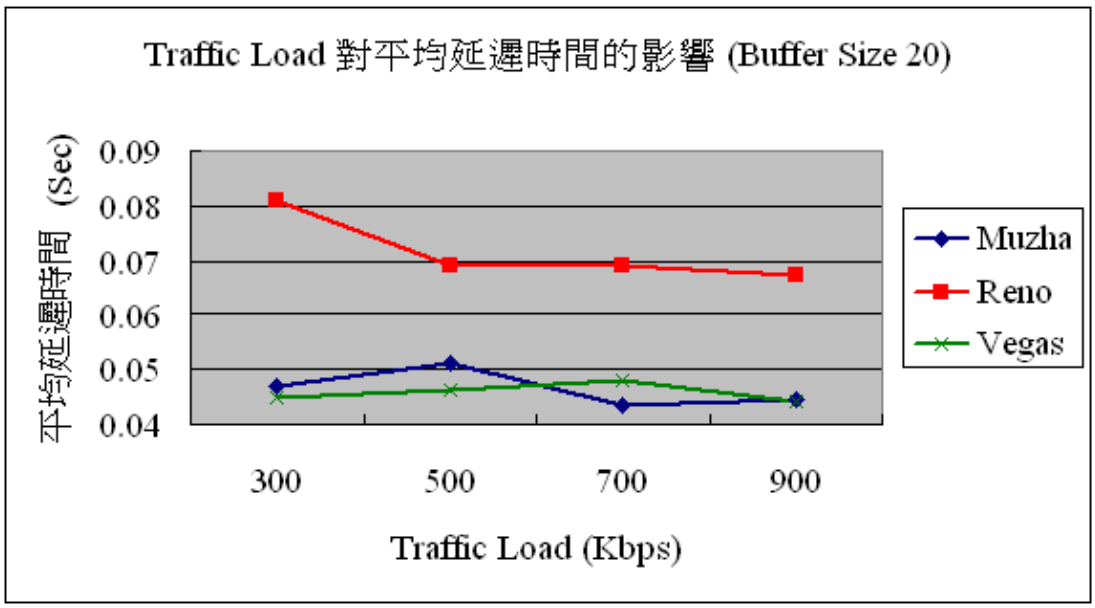

圖 4.13 實驗 2B 變動 traffic load 對平均延遲時間的影響 (buffer size=20)

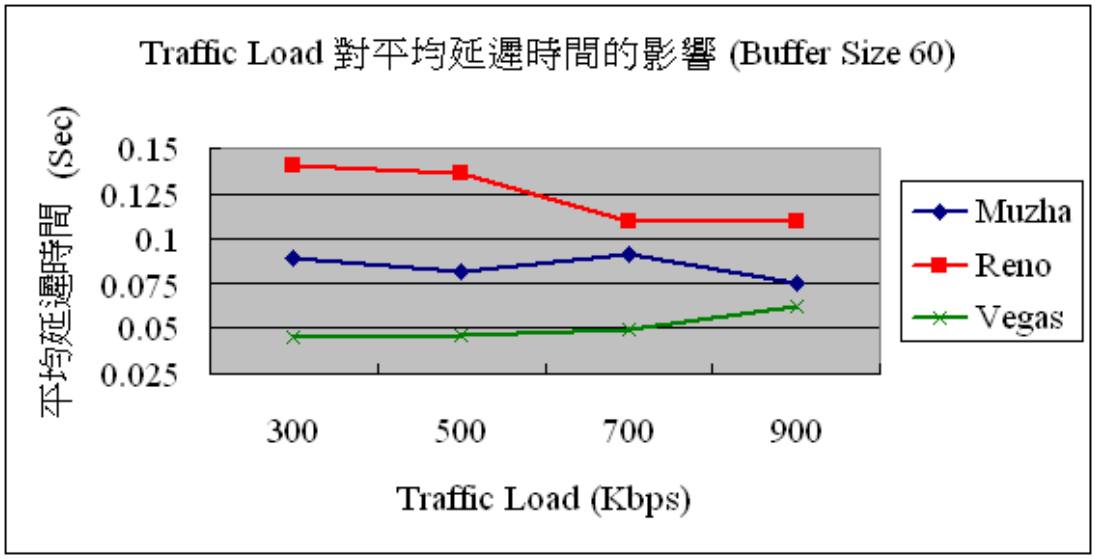

圖 4.14 實驗 2B 變動 traffic load 對平均延遲時間的影響 (buffer size=60)

4.4.3 實驗 2C：探討 link delay time 對效能的影響

4.4.3.1 實驗目標

TCP Muzha 藉由每一次的 ACK 來調整速度，我們這個實驗便是要調整 link delay time 的大小，觀察當我們所提的方法面臨延遲時間增長下的整體效能 (average throughput、封包遺失率、平均延遲時間) 變化。

4.4.3.2 實驗流程

圖 4.15 為我們實驗 2C 的拓樸，我們沿用了由多個路由器和多個節點組成的魚骨狀拓樸圖並做變化，在這個實驗中我們建立了數個 TCP 資料流，通過中間的路由器到達接收端，觀察當中間兩鏈結的延遲時間 *X* 增加時的變動情形，模擬時間為三十秒，我們並維持著具有 burst 性質的 traffic load 100K，以製造變動的網路狀態。路由器和路由器之間的鏈結頻寬是 2Mb，延遲時間 4ms，傳送端到路由器以及接收端到路由器的鏈結頻寬是 5Mb，延遲時間 1ms，buffer size 大小為 30 個封包大小，路由器佇列的管理機制為 DropTail。

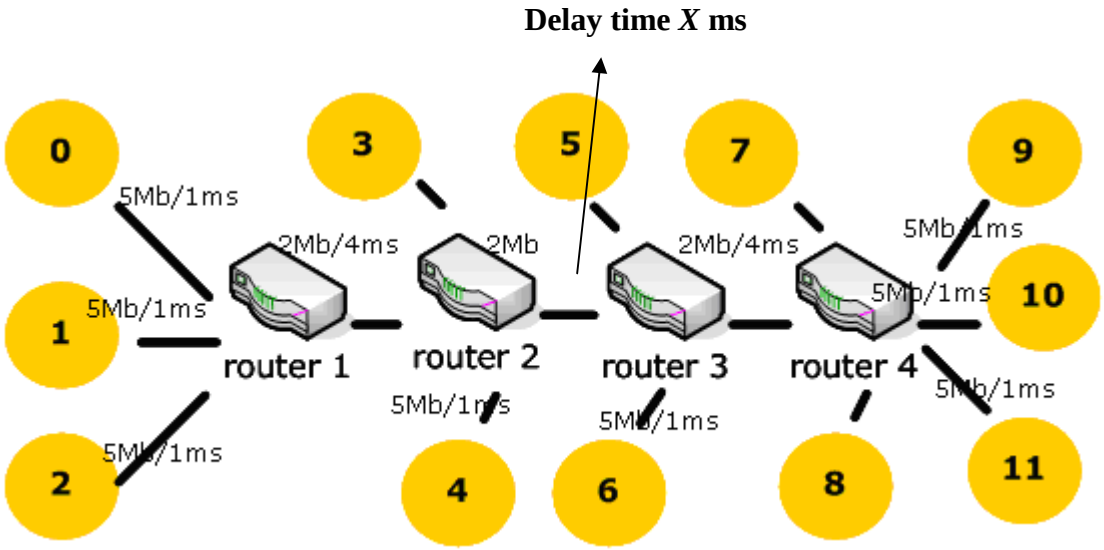

圖 4.15 實驗 2C 的拓樸

表 4.4 實驗 2C 參數表

實驗 2C	
Buffer Size	25-75 packets
Traffic Load	100K
變動的 Link Delay Time	4 、8 、16 、32 、64 、80 ms

我們變化路由器的 delay time 大小以觀察整體效能變化。

4.4.3.3 實驗結果分析

我們由圖 4.16、圖 4.17、圖 4.18 可以看出，由於 Reno 和 TCP Muzha 所採行的方式在接收到每一個回來的 ACK 後皆是用很積極的方式去爬升速度，因此路徑一增長，造成速度爬升變緩慢，雖然這樣也許有可能使得佇列長度能維持較為穩定因而降低了封包遺失的機率，TCP Muzha 由於藉著路由器的輔助，不時的依據最新的網路狀態做適當的反應，因此當收到反應的時間一拉長，傳送端在反應的當時往往也會跟網路狀況有所誤差，所以當路徑拉的很遠的時後，不僅每一次的反應變慢，每一次的動態調整效果也會比較不好掌握，也有可能會因此產生 time out。因此在 link delay time 大於 32ms 以後，整體的封包遺失率就上升了很多。Vegas 雖然也是藉著每一次的 ACK 調整速度，但是它以保守的方式在調整，因此隨著 link delay time 的不斷增長，它效能不斷的緩慢下降，卻能持續的保有良好的封包遺失率並維持著不差的效能。

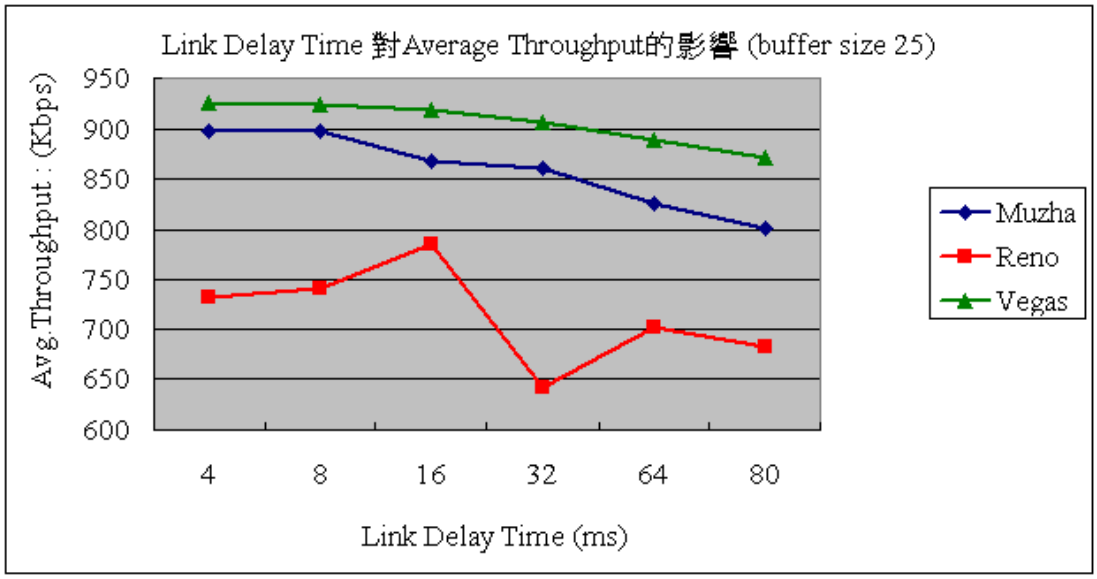

圖 4.16 實驗 2C 變動 link delay time 對 average throughput 的影響

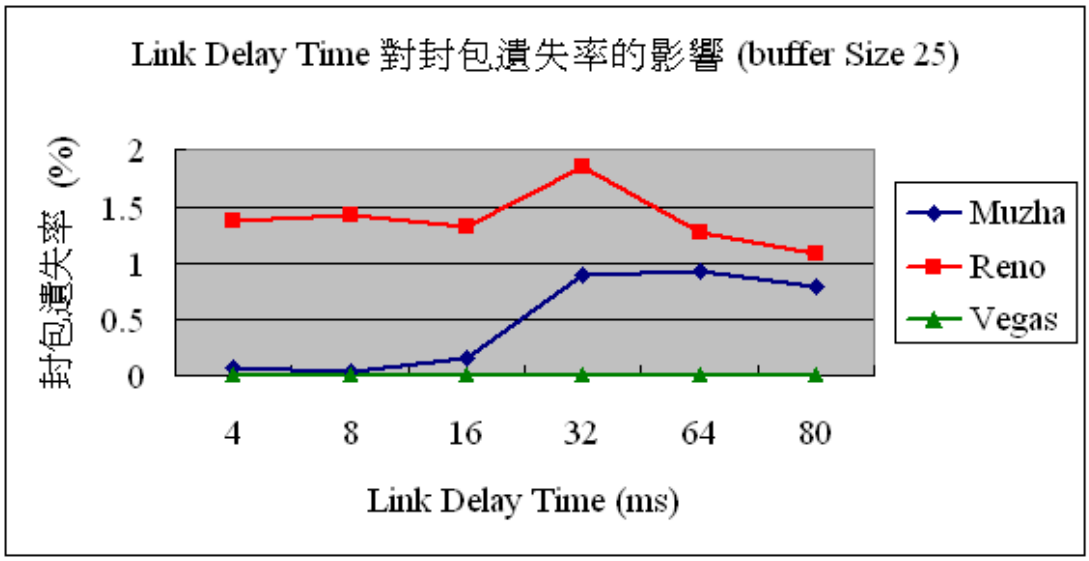

圖 4.17 實驗 2C 變動 link delay time 對封包遺失率的影響

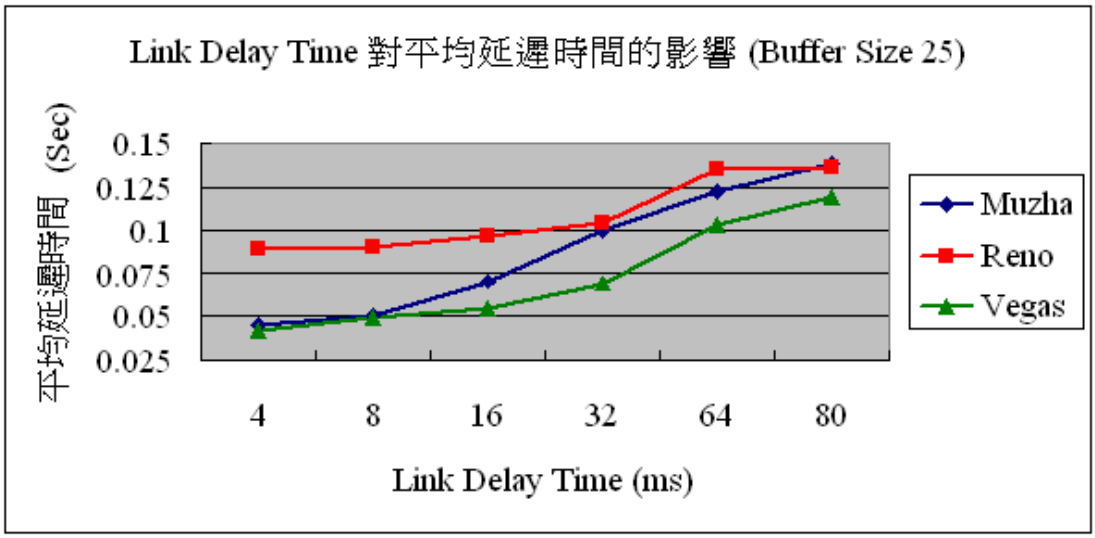

圖 4.18 實驗 2C 變動 link delay time 和平均延遲時間的影響

4.5 實驗 3：多協定共存狀態下的公平性實驗

4.5.1 實驗目標

當 TCP Muzha 和現有的 Reno 共存時，是否能保有一定的效能是我們這個實驗的目地，觀察當我們所提的方法面臨 Reno 侵略性的頻寬爭奪下的整體效能 (average throughput、封包遺失率、平均延遲時間) 變化。

4.5.2 實驗流程

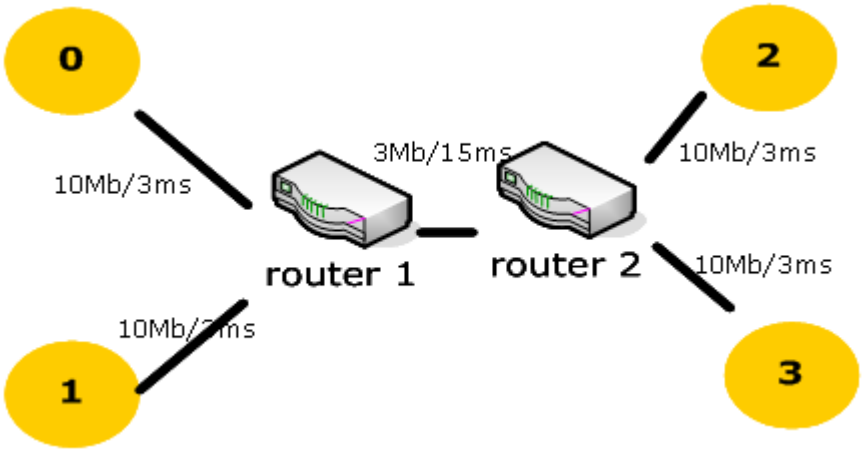

圖 4.19 實驗 3 的拓樸

圖 4.19 為我們實驗 3 的拓樸，我們建立了由兩個路由器和四個節點組成的拓樸圖，在這個實驗中我們建立兩個鏈結傳輸，分別從節點 0 傳到節點 2 以及從節點 1 傳到節點 3，其中一條固定為 Reno，另一條則依我們要實驗的目標而改變(像是 Vegas、TCP Muzha) ，模擬時間為三十秒，路由器和路由器之間的鏈結頻寬是 3Mb，延遲時間 15ms，傳送端到路由器以及接收端到路由器的鏈結頻寬是 10Mb，延遲時間 3ms，路由器佇列的管理機制為 DropTail。

我們主要在觀察和 Reno 共存的狀況，並以變動 buffer size 大小製造容易擁塞的環境來觀察其整體效能 (average throughput、封包遺失率、平均延遲時間)。

表 4.5 實驗 3 參數表

實驗 3	
Buffer Size	3、5 、10 、50 、75 、85 、100
鏈結頻寬	3~10Mb
延遲時間	21ms

我們變化路由器的 buffer size 大小從 3~100 個封包。

4.5.3 實驗結果分析

我們把和 Reno 共存的情形以差異值的狀態呈現出來。由上面的實驗 1，2，我們可以發現 TCP Muzha 和 Vegas 在單一協定的環境下皆能保有優越的效能，而 Vegas 更能保持著低延遲、低封包遺失率的特性。然而，多協定共存時，Vegas 存在和 Reno 等非 Vegas 協定的頻寬爭奪問題，由圖 4.20、4.21、4.22 可以看出，除了在 buffer size 極小的情況下，Vegas 的效能皆明顯的下降。

TCP Muzha 雖然不具有 Reno 那麼侵略的擁塞控制方法，因此效能也有下降，卻仍然能保有穩定的效能，相對於 Vegas，遠高於 Vegas 在多協定共存環境下的表現。

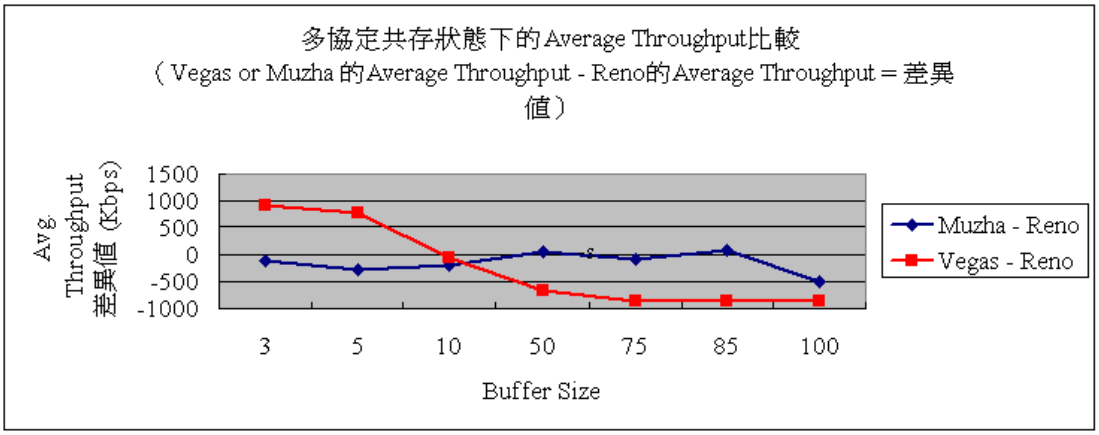

圖 4.20 實驗 3 和 Reno 共存時變動 buffer size 和 average throughput 的關係圖

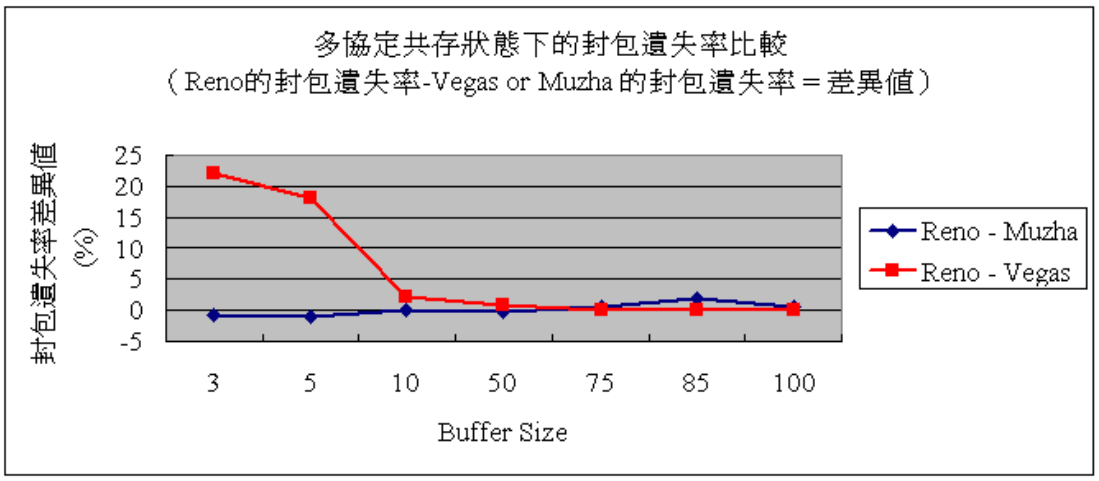

圖 4.21 實驗 3 和 Reno 共存時變動 buffer size 和封包遺失率的關係圖

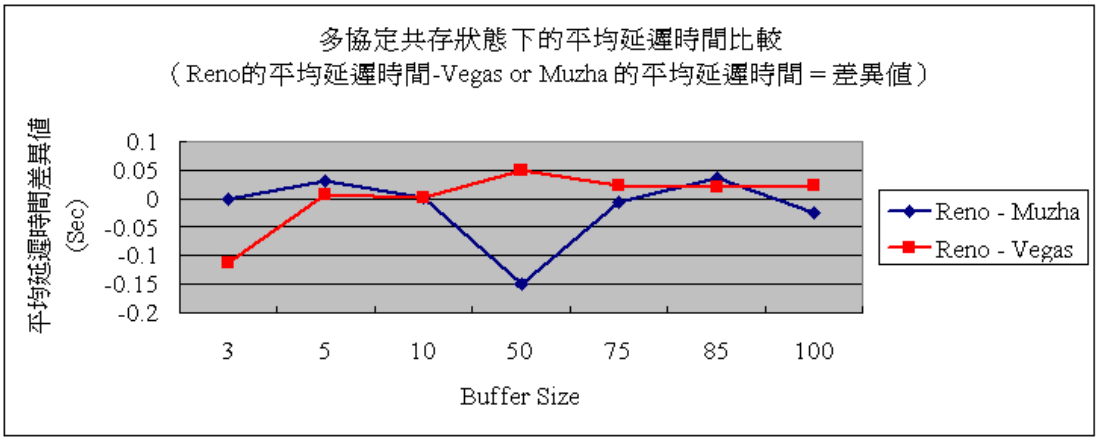

圖 4.22 實驗 3 和 Reno 共存時變動 buffer size 和平均延遲時間的關係圖

4.6 實驗 4 : TCP 同步化的實驗

4.6.1 實驗目標

網路上的 TCP Session 眾多，不同的 TCP Session 會在不同的時間點啟動，當一個新的 Session 要進入網路時，它的擁塞視窗大小和其他的 TCP 鏈結的大小是不相同的，理論上當數條 TCP Session 共享同一頻寬，而且這幾條 Session 從傳送端到目地端的 RTT (round trip time) 也一樣，則幾條 Session 應該要能公平的使用瓶頸頻寬，我們想觀察這個現象。

4.6.2 實驗流程

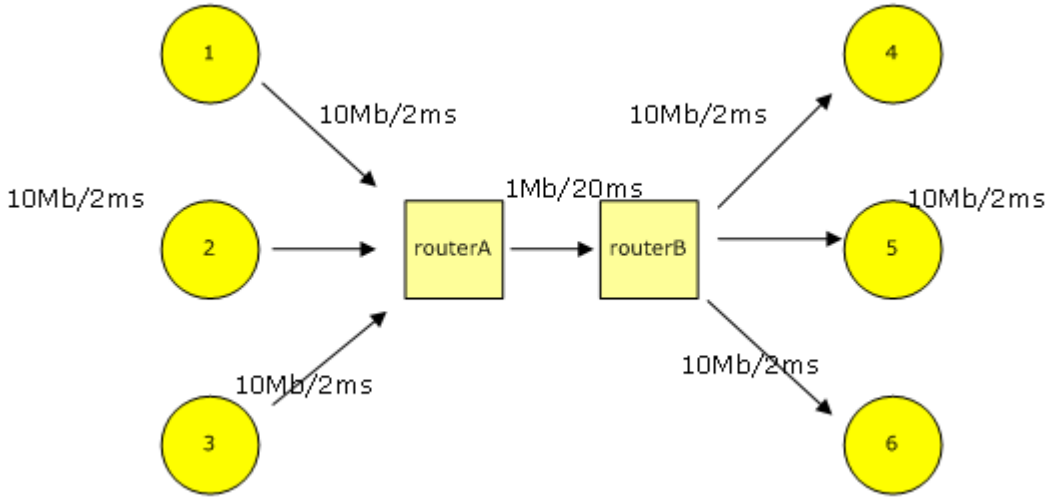

圖 4.23 實驗 4 的拓樸

圖 4.23 為我們實驗 4 的拓樸，我們建立了由兩個路由器和六個節點組成的拓樸圖，在這個實驗中我們建立三個 Session，分別從節點 1 傳到節點 4、節點 2 傳到節點 5、節點 3 傳到節點 6，模擬時間為五十秒，路由器和路由器之間的鏈結頻寬是 3Mb，延遲時間 15ms，傳送端到路由器以及接收端到路由器的鏈結頻寬是 10Mb，延遲時間 3ms，路由器佇列的管理機制為 DropTail。

4.6.3 實驗結果分析

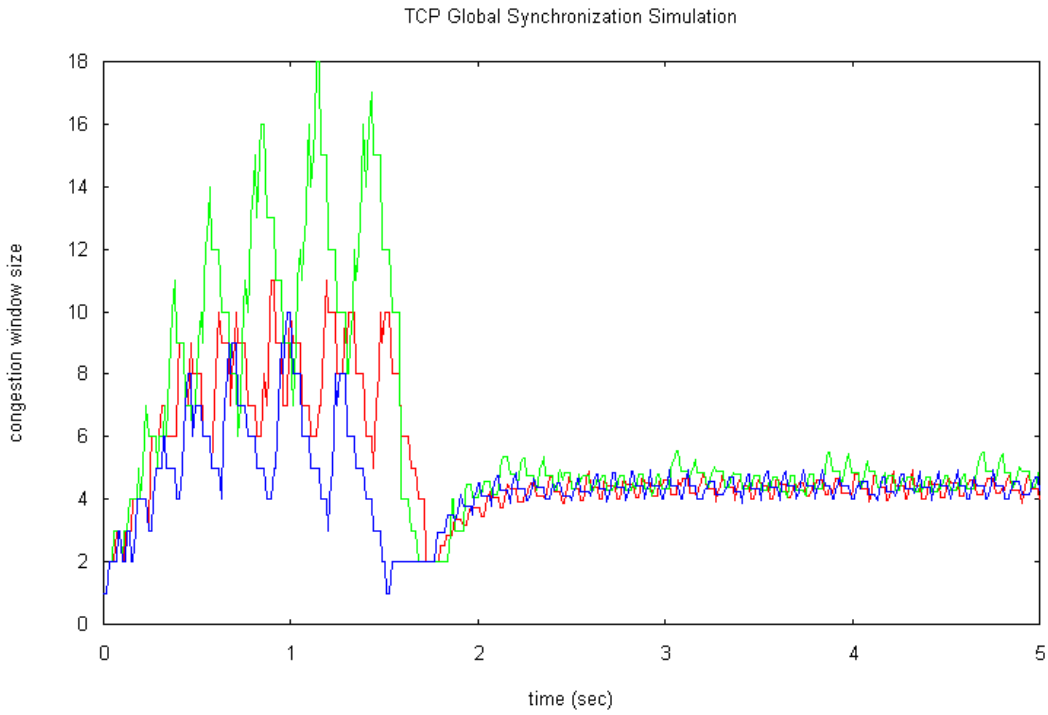

圖 4.24 實驗 4 觀察 TCP Muzha 下的 TCP 同步化狀況

實驗後發現，只有在同時啟動的情況下，TCP Muzha 最後速率才能趨於一致，若是不同時間點進入，我們則會因為依據速率調整指示值做變動，當進入的時間點之可用頻寬高，穩定狀態的速率較多，進入的時間點可用頻寬少，CWND 會較低以避免擁塞的產生，因此會呈現不同群的現象，直到網路資源呈現變動的情況才會改變。

第五章

結論

5.1 結論與未來發展

透過了模擬實驗，我們可以清楚的看到並驗證 TCP Muzha 的可行性，TCP Muzha 可以有效的降低擁塞，並能有效的保有傳輸的 average throughput。跟 Reno 相比，至少高了 25%的效能，相較於 Vegas，則略低於其效能，但是在可用資源高的環境下，由於有著較積極的調速方式，因而效能優於 Vegas，而平均 delay time 則高於 Vegas 低於 Reno。

在多協定網路下與 Reno 共存的環境中，TCP Muzha 則保有穩定的效能，不會因為 Reno 的侵略性傳輸而下降過多的效能比，相對於 Vegas 大幅度的效能下降，改善很多。

分級的想法及調控的細節還有很多可以改進的地方，例如改善擁塞視窗振盪的情況、同步化的加強、針對環境或需求做不同的調控以及將其他可能影響的參數列入考量 (queue size、RTT) 以提升效能

而用 Acknowledgement 來反應的機制的缺失還是會存在，例如當共用同一瓶頸頻寬時，鏈結距離遠的傳送端因為 ACK 的反應慢，在競爭頻寬時比較不利。此外，在非對稱性網路下的問題，也是可以研究改進的地方。

公平性的問題是 TCP 除了效能外，需要額外關注的問題，TCP Vegas 因為無法有效的在多協定的環境下公平的分享頻寬，因而在實際的網路建置下，很少見到採用 Vegas，如何改進 TCP Muzha 讓它更趨於公平，達到同步化，是未來可以研究的方向。

另外，若能有效的把模糊化的速率指示的方法應用在路由器上的設計，也許

不僅僅對於 TCP，甚至整個網路的效能，都能有效提升，這是可以繼續研究的方向。

參考文獻

[1] J. Postel, "Transmission Control Protocol," *IETF RFC 793*, 1981.

[2] D. Clark, "Window and Acknowledgement Strategy in TCP," *IETF RFC 813*, 1982.

[3] V. Jacobson, "Congestion Avoidance and Control," *Proc. of ACM SIGCOMM*, pp. 314-329, Aug. 1988.

[4] V. Jacobson, "Modified TCP Congestion Avoidance Algorithm," Technical report, Apr. 1990.

[5] W. Stevens, "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms," *IETF RFC 2001*, 1997.

[6] D. Chiu and R. Jain, "Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks," *Computer Networks and ISDN Systems*, vol.1, pp. 1-14, 1989.

[7] Sally Floyd, T. Henderson, "The NewReno Modification to TCP's Fast Recovery Algorithm, " *IETF RFC 2582*, 1999.

[8] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP Selective Acknowledgement Options," *IETF RFC 2018*, 1996.

[9] K. Fall and S. Floyd, "Simulation-based Comparisons of Tahoe, Reno, and SACK TCP," *ACM Computer Communication Review*, vol. 26, no.3, pp. 5-21, 1996.

[10] L. S. Brakmo, S. W. O'Malley, and Larry L. Peterson. "TCP Vegas: New Techniques for Congestion Detection and Avoidance, " *Proc. Of ACM SIGCOMM*, pp. 24-35, Aug. 1994.

[11] L. S. Brakmo and L. L. Peterson. "TCP Vegas: End to End Congestion Avoidance on a Global Internet," *IEEE Journal on*

Selected Areas in Communication, vol.13, no.8, pp. 1465-1480, Oct. 1995.

[12] J. S. Ahn, P. B. Danzig, Z. Liu, and L. Yan, "Evaluation of TCP Vegas : Emulation and Experiment," *Proc. of ACM SIGCOMM*, pp. 185-195, Aug. 1995.

[13] C. Barakat, E. Altman, and W. Dabbous, "On TCP Performance in a Heterogeneous Network : A Survey," *IEEE Communications Magazine*, vol.38, no.1, pp. 44-46, Jan. 2000.

[14] E. Altman, C. Barakat, E. Laborde, "Fairness Analysis of TCP/IP," *IEEE Conference on Decision and Control*, Dec. 2000.

[15] G. Hasegawa and M. Murata, "Survey on Fairness Issues in TCP Congestion Control Mechanisms," *IEICE Transactions on Communications*, vol. E84-B, no.6, pp. 1461-1472, Jun. 2001.

[16] R. Denda, A. Banchs and W. Effelsberg, "The Fairness Challenge in Computer Networks," *Proc. of Quality of future Internet Service*, vol. 1922, pp. 208-220, Springer-Verlag Heidelberg, Jun. 2000.

[17] Yi-Cheng Chan, Chia-Tai Chan, and Yaw-Chung Chen, "RoVegas : A Router-based Congestion Avoidance Mechanism for TCP Vegas," *Computer Communications*, vol.27, Issue 16, pp. 1624-1636, Oct. 2004.

[18] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," *IEEE/ACM Transaction on Networking*, vol.1, no.4, pp. 397-413, 1993.

[19] Sally Floyd, "TCP and Explicit Congestion Notification," *ACM Computer Communication Review*, 1994.

[20] Planet Lab, http://www.planet-lab.org/.

[21] L. Peterson, T. Anderson, D. Culler, T. Roscoe, "A Blueprint for Introducing Disruptive Technology into the Internet," *Proc. of the 1st ACM Workshop on Hot Topics in Networks (HotNets)*, Princeton, Oct. 2002.

[22] V. Paxson, "End-to-end Internet Packet Dynamics," *IEEE/ACM Transactions on Networking*, pp. 277-292, 1999.

[23] R. Carter and M. Crovella, "Measuring Bottleneck Link Speed in Packet-Switched Networks," *International Journal on Performance Evaluation*, pp. 27-28, 1996.

[24] A. S. Tanenbaum, "Computer Networks," 4th edition, Prentice Hall, 2002.

[25] L. L. Peterson, B. S. Davie "Computer Network : A Systems Approach," 3rd edition, Morgan Kaufmann, 2003.

[26] "The Network Simulator - ns-2", http://www.isi.edu/nsnam/ns/.

[27] S. Ryu, C. Rump, C. Qiao, "Advances in Internet Congestion Control," *IEEE Communications Surveys and Tutorials*, vol 5. no.2, 2003.

[28] J. Postel, "Internet Protocol," *IETF RFC 791*, Sep. 1981.

[29] W. Stevens, "TCP/IP Illustrated, Volume 1: The Protocols," Addison-Wesley, 1994.

[30] D. Bertsekas and Robert Gallager, "Data Networks," Prentice-Hall, 1992.