國立政治大學資訊科學系
Department of Computer Science
National Chengchi University

碩士論文

Master's Thesis

適用於無線隨意式網路之逐節點 TCP 傳輸協定

Hop-by-Hop TCP over MANET

研 究 生：游逸帆

指導教授：連耀南

中華民國九十六年十二月

December 2007

適用於無線隨意式網路之逐節點 TCP 傳輸協定

Hop-by-Hop TCP over MANET

研 究 生： 游逸帆　　Student：Yi-Fan Yu

指導教授： 連耀南　　Advisor：Yao-Nan Lien

國立政治大學

資訊科學系

碩士論文

A Thesis

Submitted to Department of Computer Science

National Chengchi University

in partial fulfillment of the Requirements

for the degree of

Master

in

Computer Science

中華民國九十六年十二月

December 2007

摘要

行動隨意式網路(MANET)是一種具有高度動態拓撲結構的網路。每一個行動隨意式網路由一組移動節點(Node)組成，彼此之間互相支援轉送封包可以不依靠基地台建構成Intranet。此種網路中，因節點移動之緣故，連線不穩定、頻寬較窄，錯誤率亦較高。傳統的TCP傳輸協定在行動隨意式網路上的效能不免遭受重創。

傳統的TCP在封包遺失時，只能從傳送端進行重傳，而行動隨意式網路傳輸品質極不穩定，常常重送多次才可到達目的地，導致要耗費極長的時間才能將封包送達目的地，然而，在行動隨意式網路中，大量傳輸資料的需求並不大，反而是封包的快速送達更為重要，因此加速封包的送達成為比增大傳送量更為重要的目標。

為了使封包較快送達目的地端，我們提出了Hop-by-Hop TCP的方法，使每個節點使用當地重傳以保證封包成功的傳到下一個節點，遺失的封包不必重新由傳送端重傳，能更快反應封包遺失，並且提昇傳輸可靠度，使封包在高遺失率的情形之下能順利且較為快速的送達目的地端。

我們利用 NS-2 網路模擬器進行實驗，驗證我們的機制，實驗在不同的

拓樸及負載等參數下進行，觀察傳輸成功率及封包傳輸時間，以及公平性。實驗結果指出，本方法在網路環境不穩定時吞吐量能有 25.7%以上的提昇，而延遲時間也能有 25%的提昇，亦有相當好的公平性。

Abstract

A Mobile Ad hoc Network (MANET) MANET is composed of a group of mobile computing devices (nodes) that are equipped with Wireless LAN (WLAN) capability. Nodes can transmit packets to each other to construct Intranet without any base station. In an MANET environment, the communication links are unstable due to various reasons. Error rate is higher and bandwidth is smaller than fixed networks. Running regular TCP protocol on MANET will suffer from serious performance degradation in MANET.

To handle packet lost, regular TCP can only retransmit lost packets from the source. However, when error rate is high, several retransmissions may be needed to transmit a packet to its destination successfully. As a result, the effective bandwidth is much lower and the average time to transmit a packet will be much longer.

Considering that most applications on MANET prefer shorter transmission time to higher bandwidth, this thesis proposes Hop-by-Hop TCP protocol aiming to accelerate the transmission of packets. Hop-by-Hop TCP makes every intermediate node in the transmission path running a local TCP to guarantee the transmission of each packet on each link. The retransmission of a lost packet is right at the transmitting end of the link where the packet is lost. It doesn't need to retransmit a lost packet from its source node. It takes less time in average to transmit a packet to its destination in a high error rate environment.

We evaluate the performance of our approach by simulation using NS-2 simulator. Our experiments show that our proposed protocol outperforms TCP Reno by 25.7% in throughput and 25% reduction in average transmission time.

The fairness requirement is also achieved while our proposed protocol coexists with other major TCP variants.

誌謝

能夠完成本論文，首先最重要的就是要先感謝連耀南教授兩年多來的辛勤指導，讓我在做研究方法，甚至做事情的態度都有十足的長進，另外，要感謝政治大學行動通訊實驗室的蔡子傑教授、張宏慶教授，從平時的上課、group meeting，皆讓我獲益良多。另外，還要感謝吳曉光教授、雷欽隆教授、陳耀宗教授、蔡子傑教授能撥冗參加我的論文口試，並在口試時針對研究的問題給予改進的建議以及指導。

在政治大學的這兩年多以來，我接觸到了許多不同的人事物，比起在大學時更加有不曾遇過的挑戰，兩年多之中我學習了如何去發現、描述、解決問題，能更加嚴謹的處理事情，讓我能以更成熟的態度面對未來的人生。

另外我也要感謝在政治大學給過我幫助的所有學長、同學以及學弟，特別是實驗室的所有成員，包括了教導我許多的宗銘、Larry 等學長，明翰以及永全，以及三位學弟妹，怡萱、威中及小冷，他們給我許多幫助，真的非常高興能認識各位！

最後我要感謝我的家人給予我的支持，讓我可以無後顧之憂的攻讀碩士，在我遭遇到困難的時候，能夠勇敢的堅持；遇到壓力的時候，給我力量直到跨越各個難關。

其實還有許多人曾經幫過我的忙，在此可能沒辦法全部寫上，我想要不是因為這麼多朋友同學伸出援手，我想在我唸書的這段日子一定會非常無助且無趣，真的非常謝謝大家！

游逸帆在台北，政治大學 Jan 10, 2008

目錄

圖目錄

表目錄

第一章

導論

1.1 簡介

在這個資訊爆炸的時代，網際網路的快速興起改變了人們的生活。我們可以透過網際網路的技術進行互動，使我們的生活更加便利。科技不斷的進步，網際網路技術也隨著時間逐漸地改變。原本的網際網路都是有線的，我們必須透過一條網路線才能連上網路，然而因為科技的發展，使得我們不再總是受限於這項要求， 如今我們可以使用無線的技術連上網際網路。

由於傳統TCP是在有線網路環境下所設計，當傳統TCP在無線網路中運行時就會遭遇到許多不同的問題；行動隨意式網路(MANET)是一種具有高度動態拓撲結構的網路。每一個行動隨意式網路由一組移動節點(Node)組成，彼此之間互相支援轉送封包可以不依靠基地台建構成Intranet。在MANET 中，節點受到其傳輸範圍的限制，傳送端節點要傳送資料到目的端節點時需要經過多跳(Multi-hop)的路徑才可以到達目的端節點，封包在多重跳躍所經過的長路徑過程中，每個節點間需經過MAC層的頻道競爭，競爭到頻道的節點才能完成封包的傳送。此種網路中，因節點移動之緣故，連線不穩定、頻寬較窄，錯誤率亦較高。傳統的TCP傳輸協定在行動隨意式網路上的效能不免遭受重創，封包送達目的地的時間亦較緩慢。

傳統的TCP在封包遺失時，只能從傳送端利用快速回復(Fast Recovery)以及逾時(Timeout)重傳進行重傳的動作，而行動隨意式網路傳輸品質極不穩定，重送封包也要經

過同樣不穩定的環境，因此封包常常需要重送多次才可到達目的地；多次重送導致需要耗費極長的時間，然而，在行動隨意式網路中，大量傳輸資料的需求並不大，反而是封包的快速送達更為重要，因此加速封包的送達成為比增大傳送量更為重要的目標。本論文要求TCP傳遞路徑的中間節點也利用重傳機制提高封包在中間節點的存活率，如此可減少傳送端的重送次數，進而減少由傳送端重送所導致的時間延遲。

1.2 無線隨意式網路簡介

現今最重要的一種無線通訊網路就是802.11[1]。IEEE 802.11制訂了一套適合在無線網路環境下作業的MAC層（Media Access Control sublayer）和實體層之通訊協定。

主要特性為：

- 傳輸媒介為無線電波。
- 通訊協定是CSMA/CA，能提供優先權服務。
- 訊框為IEEE 802.11 CSMA/CA訊框。
- 頻寬使用不保證公平。每個工作站實際使用的頻寬量可能不同。

IEEE 802.11制訂出兩種不同類型的無線區域網路基本架構：

- 有基礎架構的無線區域網路(Infrastructure Wireless LAN)
- 無基礎架構的無線區域網路(Ad Hoc Wireless LAN)

有基礎架構通常指的就是現存的有線網路系統，在此類型的網路架構中，有一種特別的節點，稱之為擷取點(Access points)，它的功用就是將一個或多個的無線區域網路和有線網路相連結，提供某個無線區域網路中的節點和較遠距離的其他無線區域網路的節點通訊。此種類型的無線網路的通訊範圍，常在同一棟建築物中出現，例如某棟樓層、商店等等。

無基礎架構的無線區域網路主要能提供不限數目的節點，快速架設起無線通訊網路，在

此種架構中，任二個節點間都可以直接通訊，此種的無線網路架構在會議室裡經常用得上。IEEE 802.11制訂的架構允許「有基礎架構的無線區域網路」和「無基礎架構的無線區域網路」同時使用同一套的擷取協定。而無基礎架構的無線區域網路其中的行動隨意式網路(MANET，Mobile Ad-hoc Network)正被各方所矚目。行動隨意式網路是一種具有高度動態拓樸結構的網路。每一個行動隨意式網路由一組移動節點(Node)組成。起先行動隨意式網路是作為軍事領域之用，之後也運用於在民用的行動通信中，特別是在一些較特殊的環境，像是用於救災或是需要做資料交換時，由於缺乏已建設好之設備，因此可作為臨時部署的用途。行動隨意式網路被認為是下一代行動通信技術解決方案中，可能會被採用的末端網路(Last mile)，所以受到不小的重視。

1.3 TCP (Transmission Control Protocol) 簡介

TCP[27]是目前網路上最被廣為使用的一種端對端傳輸層協定。這是因為它在兩部電腦之間的連線扮演了重要的角色：當一部電腦需要與另外一部電腦建立連線時，TCP協定會為它們建立一個連線、傳送和接收資料以及終止連線。TCP利用重傳和擁塞控制機制(Congestion Control mechanism)對應用程式提供可靠的連線。即使在網路暫時出現擁塞的情況下，TCP也能保證通訊的可靠。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支援多工(Multiplexing)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。

TCP的擁塞控制機制中，滑動視窗演算法[27, 10]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及擁塞控制；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，或塞爆網路。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)、逾時和重傳的方式來確保可靠性並恢復遺失的封包。然而傳統TCP是針對有線網路的環境中設計，當傳統TCP在無線網路環境執行時，會遭遇到不同以往的問題。

1.4 TCP在無線網路下的問題

TCP 的設計目標是確保網路從傳送端可以可靠的傳輸至接收端，保證packet到達，在不把網路塞爆的情況下盡量利用剩餘頻寬，而傳統TCP係針對有線網路環境下設計的，相較於無線網路，有線網路的頻寬較大，且不容易有傳輸上的錯誤產生，因此傳統TCP認為封包遺失的主要原因為網路擁塞，並據此而設計，因此TCP把封包遺失當作是網路壅塞的指標(indicator)。傳統的TCP如果發現封包遺失，便會降低封包的發送速率。

然而，在無線網路下造成封包遺失的原因不只是網路擁塞，尚有傳輸介質不穩定而造成的問題[19]，傳統 TCP 對於所有的封包遺失皆視為網路擁塞來處理，因此也導致了效能不佳及傳輸延遲過長等等的問題，如圖 1.1。

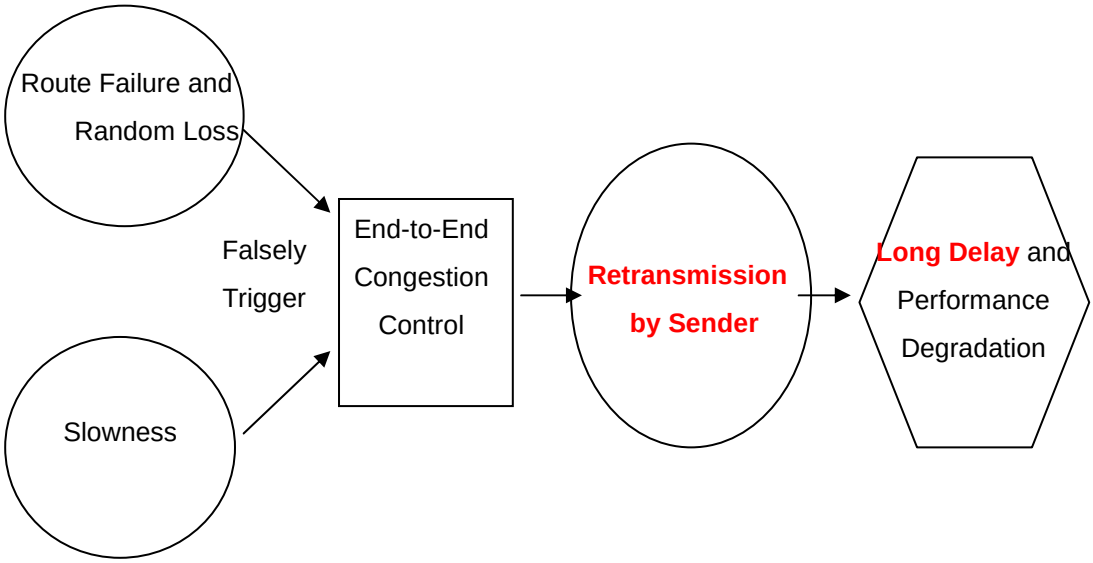


圖1.1：The Effect of Mis-triggering Congestion Control

無線網路下的挑戰

行動隨意式網路具有以下的特性：高的錯誤率(error rate)、節點移動、有限的頻寬及傳輸頻道。

由於這些特性，使得我們容易遇到下面的情況：

- 隨機遺失(random loss)：過多的傳送錯誤(transmission error)會造成許多封包遺失(packet loss)。在MANET中由於高錯誤率，且節點會移動而造成鏈結失效(link failure)，使得傳輸品質相當不穩定，造成封包不斷移失，而不斷從傳送端重送，除了效能低落之外，也造成封包送達速度非常緩慢，封包成功送達的延遲時間(delay time)就會拉長；
- 緩慢(slowness)：節點競爭頻道及變動的延遲時間會使得封包在節點中停留的時間延長。在MANET中，節點需要競爭頻道才能進行傳送，如圖1.2所示，節點A,B,C皆要進行傳送，A與C皆在B的傳輸範圍內，當B進行傳送時，A與C就必須要等待B傳輸完畢才能競爭頻道，這使得封包在節點中等待的時間拉長，每個封包的延遲時間變化量大，長短不一，較容易引起逾時，造成傳送端進入慢

啟動(Slow Start)程序，使得擁塞視窗降到最小，降低效能(throughput)，重傳封包，並且拉長了RTO (Retransmission Timeout)。

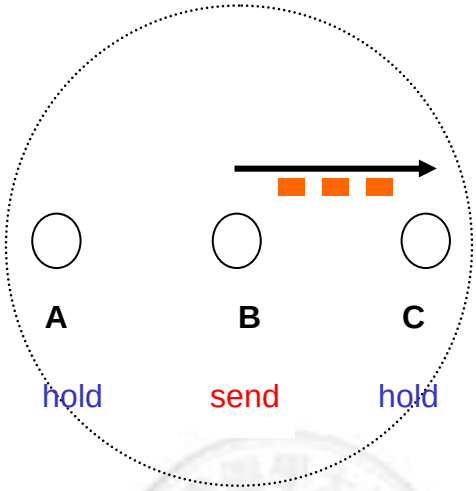


圖1.2：Slowness問題示意圖

Link重傳封包的影響

MAC Layer 與擁塞控制機制的交互影響，也是影響效能的因素，在[16]中提到，若沒有鏈結回覆(link ACK)且擁塞視窗大於 1 個封包時，封包遺失率大為提升，效能非常低；而[29]指出，當鏈結在 MAC 層重傳次數過多時，會導致 TCP 進入逾時重傳，以至於同一個資料封包在路徑上的兩個節點競爭頻道，反而會降低整體性能(Performance)(如圖 1.3 所示)。

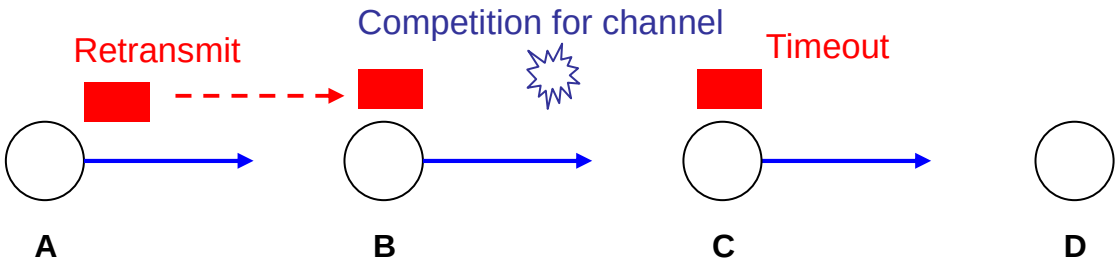


圖1.3：Link重傳影響示意圖

1.5 研究動機

在無線網路的環境之下，由於傳輸環境本身的不穩定而容易造成封包遺失的情形發生，而延遲時間的不固定也更容易造成許多的逾時，必須從傳送端重傳，如圖1.4，使得封包成功傳送的延遲時間拉長，當網路品質很差時，整體傳輸效能更會非常低落。

在MANET中，大部分的應用大量傳輸資料的需求並不大，反而是封包的快速送達更為重要。如果我們能從每個中間節點進行重傳，兩個節點間所需等待的RTO較短，能避免傳統版本TCP從傳送端重傳所需耗費較長的時間，則反應封包遺失的速度可以提升，如圖1.5。

1.6 研究目的

我們的研究目標就是在網路狀況不好時傳送端也能可靠的傳送至接收端，並且能更快速的送達封包(縮短傳送時間)，並且期望能提升傳送吞吐量。

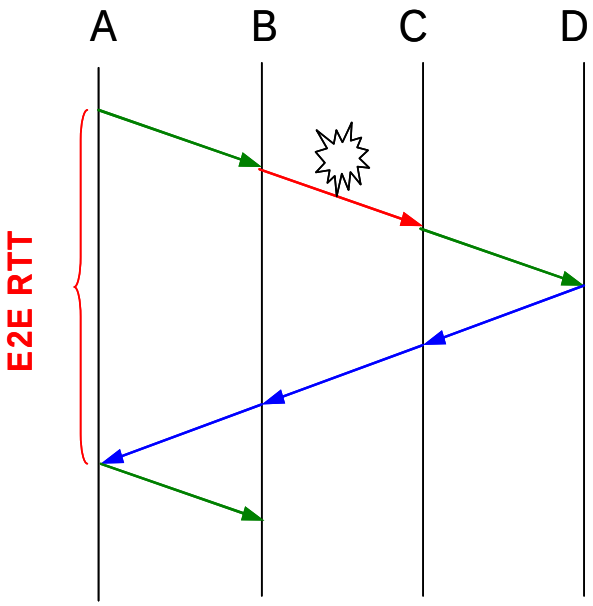


圖1.4：從Sender重傳

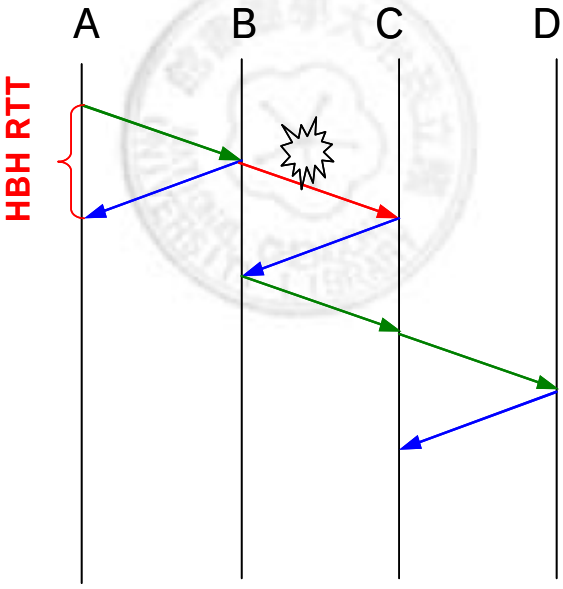


圖1.5：從Local重傳

1.7 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Hop-by-Hop TCP，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

2.1 傳輸層通訊協定

傳輸層通訊協定在傳輸路徑的兩端點執行，負責將應用層交付的資料送達目的地，傳輸層協定必須負責將所交付的資料包裝成封包，交付給網路層傳送至接收端，而在目的地重組為原來的資料。傳輸層協定必須負責決定最適當的傳送速率，必要時，並須保證資料完整無缺的送達目的地。

最風行之傳輸協定有二，一為 UDP，二為 TCP，UDP 協定非常簡單，它並不保證資料之完整，它負責將應用程式所交付的資料裝成封包後依指定的速度交付給網路層，由網路層送達目的地，而目的地的 UDP 則將所收到的封包交付給上層的應用程式，對於封包之遺失及亂序之封包則置之不理，也不會依網路壅塞狀況調整傳送速率。由於不須重送遺失的封包之故，整體傳送速率較快，因此對於時效性要求較高且不計較封包遺失的多媒體通訊最常使用。其弊病則因不會依網路狀況而調整傳送速率，而在網路壅塞時，不但遺失太多封包，且對網路有火上加油之弊。此外，如果遺失太多封包，不免影響所傳送資料之品質，例如網路電話(VOIP)可容忍某種程度之封包遺失，但過度遺失則不免讓通話品質下降，如果要保證資料之完整送達，最常使用的傳輸協定則非 TCP 莫屬。

TCP (Transmission Control Protocol)是一個可靠的點對點(End-to-End)傳輸層協定。如圖2.1，確保網路從傳送端可以可靠的傳輸至接收端，封包在網路上兩個端點(End point)

之間的傳送過程中，可能會因為網路壅塞(congestion)或毀損而遺失，TCP使用回應(ACK)與重傳(retransmit)以確保兩端點間封包能正確的傳達，如圖2.2所示。TCP是目前廣泛使用的可靠的傳輸層協定。

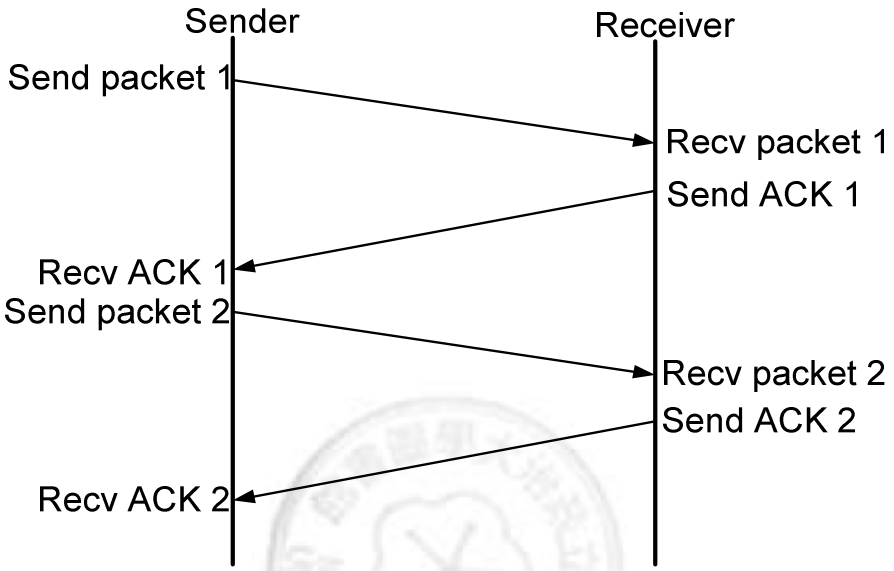


圖2.1：TCP端對端傳送示意圖

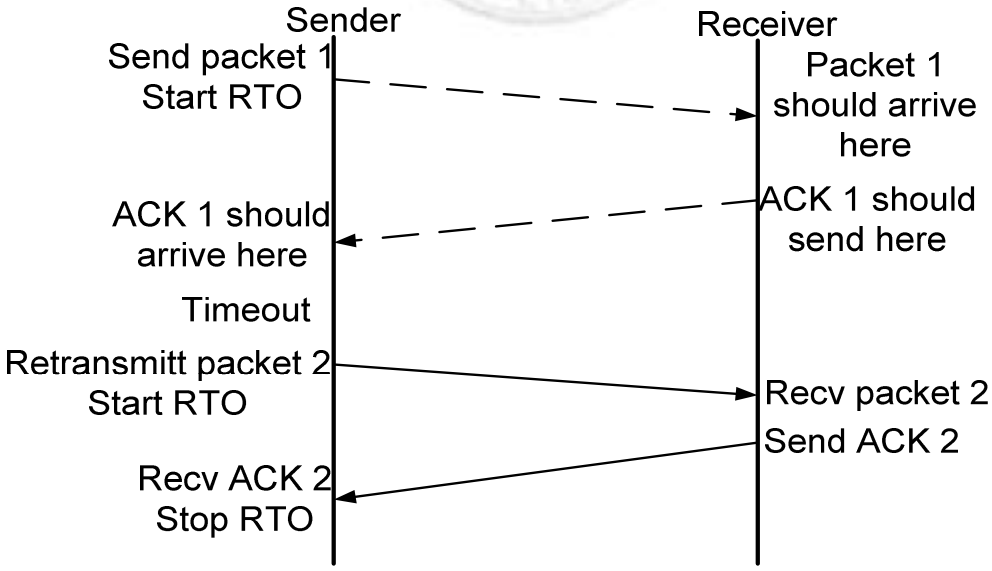


圖2.2：TCP重傳機制

如圖2.3所示：傳送端傳送資料給接收端，但是網路中節點的轉送只經過實體層(Physical Layer)、資料鏈結層(Data-Link Layer)、網路層(Network Layer)，但是這三層的轉送可能會造成封包遺失，必須依賴傳送端與接收端中的傳輸層TCP之回應與重傳以做到可靠的傳輸。

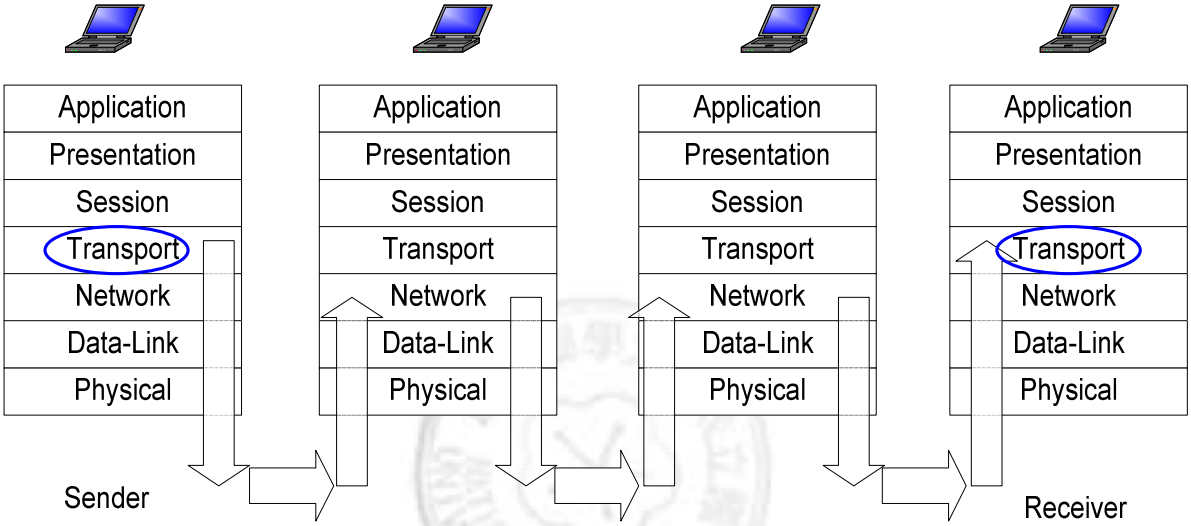


圖2.3：網路傳輸架構示意圖

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[10]。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[20]，這個版本叫做 TCP Tahoe，包含了慢啟動(Slow Start)、additive increase and multiplicative decrease (AIMD) 還有快速重傳(Fast Retransmit algorithms)。而在兩年後，把快速回復(Fast Recovery algorithm) [28]加入後的版本，TCP

Reno，成為現在最被使用的 TCP 版本 。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 擁塞控制機制

大部分的 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，利用是否收到 ACK 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傳送速率來做控制，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程序。

2.2.1 TCP Tahoe and TCP Reno 的擁塞控制

下面的表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擁塞控制的機制最早於1988年提出，然而隨著時間的演進，逐漸出現了許多新的版本。從最初的版本TCP Tahoe [20]，到目前最常使用的TCP Reno [28]，和對TCP Reno改良的TCP NewReno [14]等等。

TCP Tahoe提出了慢啟動，快速重傳和擁塞避免的觀念。TCP一開始是在慢啟動階段，擁塞視窗從1開始成長，直到擁塞視窗超過threshold則進入擁塞避免階段。在慢啟動階段中，傳送端每收到一個ACK則擁塞視窗加1；在擁塞避免階段則是每個RTT (Round Trip Time)才將擁塞視窗加1。TCP Tahoe在發生逾時的時候，擁塞視窗會降為1，而threshold會降為一半；而快速重傳則是當傳送端收到3個的重複ACK的時候，會將擁塞視窗大小降為一半，並將threshold設為降為一半之後的擁塞視窗的大小。

TCP Reno則是增加了快速回復的階段。當在快速重傳階段重傳遺失的封包時，會進入快速回復階段，直到收到重傳封包的ACK時才離開快速回復階段，回到擁塞避免階段。在快速回復階段，TCP傳送端和慢啟動階段一樣，每收到一個ACK，擁塞視窗就加1。

TCP Reno 是以視窗為基礎的擁塞控制機制，依據 RFC 2001 [28]，其主要分為四個狀態，慢啟動、擁塞避免、快速回復、快速重傳，如圖 2.4 所示。其他較為詳盡的描述會在後面一一說明。

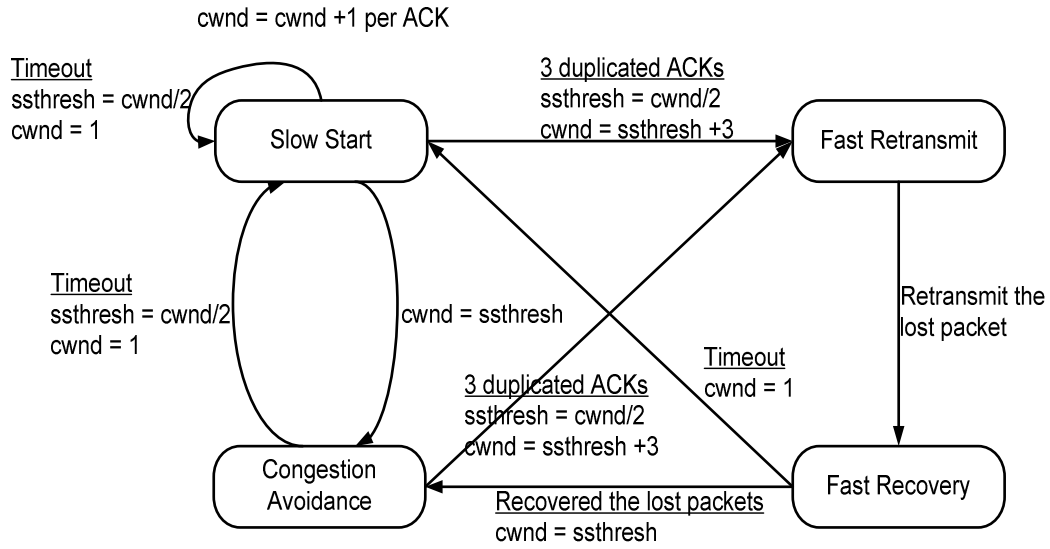


圖2.4：TCP Reno 執行流程圖

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)

當一個 TCP 的連線建立後便進入 Slow Start [28]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 2.5 所示，初始的時候，傳送端送出一個 CWND 的封包，當第一個 ACK 接收到以後，CWND 便增加成為 2，當下一個 RTT 送出封包的 ACK 又收到以後，則增加成 4，依此類推，每一個 RTT 之後，若無封包遺失，CWND 大小便以兩倍遞增；當 TCP 連線發生逾時的情形，也會進入 Slow Start 階段，並且把 CWND 降到最小，重新開始成長；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，如圖 2.6 所示。

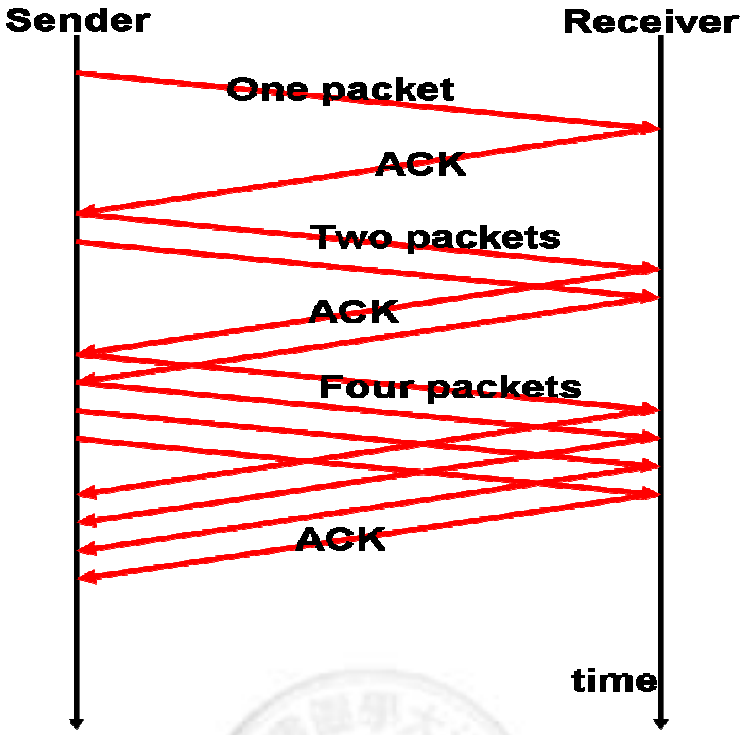


圖2.5：慢啟動圖示

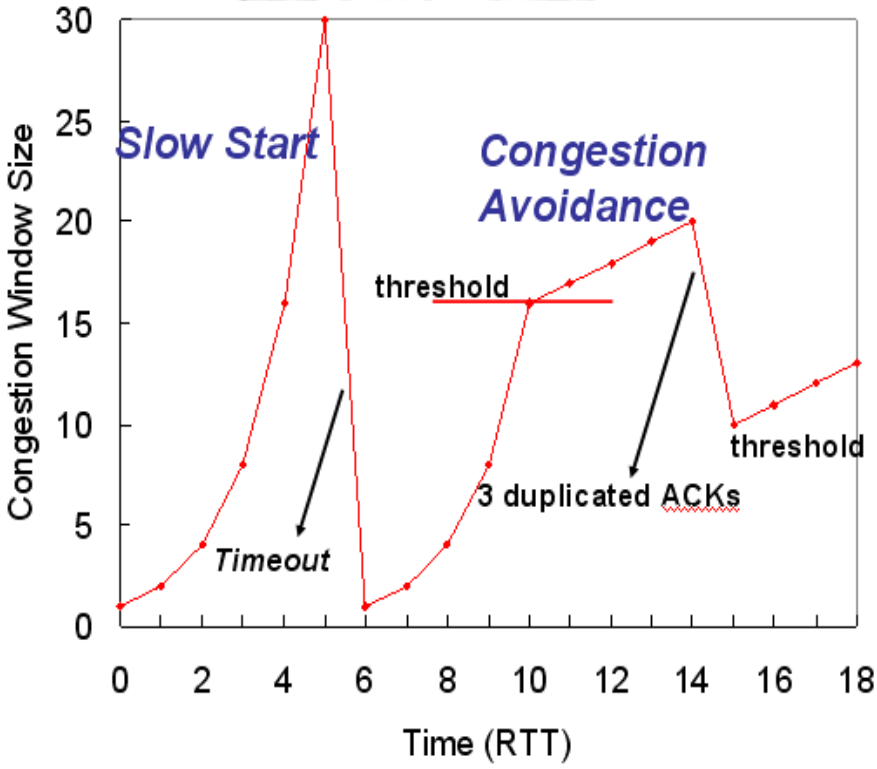


圖2.6：TCP擁塞控制機制示意圖

由於等到 Timeout 發生後再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複的 ACK 後，便假設此重複 ACK 號碼之下一個封包已經遺失，立即進入重傳的動作。

在 Fast Retransmit 階段重傳遺失的封包時，TCP 將會進入 Fast Recovery 階段，Fast Recovery 是基於封包守恆的原則 (conservation of packets principle) [14]，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 ACK 時，CWND 的值就加一。如此一來，就不用大幅度降低傳輸速度，因此可以提高效能。

2.2.1.2 擁塞避免(Congestion Avoidance)

當擁塞視窗大小超過 threshold 時，TCP 便會進入擁塞避免階段。在 Fast Recovery 階段收到遺失封包的 ACK 之後，離開 Fast Recovery 所進入的狀態是擁塞避免階段，而非慢啟動階段，這是因為在回復遺失封包之後，擁塞視窗大小會降為一半，而 threshold 則設成擁塞視窗的大小。擁塞避免階段是在擁塞發生之後的速率控制機制，一則必須設法解除網路擁塞，二則既已偵知速率之極限，理當調整更恰當的速率[9, 28]，因此在擁塞避免階段中，TCP 傳送端會在每個 RTT 將擁塞視窗加 1，相對於慢啟動階段則以較緩慢的方式成長。

然而 TCP Reno 和其之前的版本都有個問題，就是 TCP 傳送端在 Fast Recovery 階段每個 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 [14]、TCP Sack [26, 13]均可處理多封包遺失的問題。

2.2.2 TCP SACK

TCP SACK [26]的主要目的是要改進TCP Multiple Loss的問題，傳統的TCP擁塞控制方法在一個RTT內只能復原一個封包遺失的錯誤，SACK則是可以允許TCP可以在一個RTT之內復原多個封包遺失的方法。SACK的接收端可以告知接收端已經收到了哪些封包，利用這些資訊，傳送端只要重送那些沒有被收到的封包即可。然而在[4]中指出，如果有多個連續的ACK遺失，傳送端無法得知接收端回傳的資訊，那麼傳送端可能會不知道有某些區段的封包沒有被接收，只能等到逾時之後才能重傳遺失的封包。尤其在無線網路這種高封包遺失率的環境之下，將無法避免逾時重送的情形發生。

2.2.3 TCP Vegas

在 1994 年，L. S. Brakmo 等提出了一種新的擁塞控制策略 TCP Vegas [2, 5, 6]，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 比起其他版本的 TCP 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，於是自動開始減小擁塞視窗；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，因此能在發生擁塞之前即時降低傳送速率，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，在與 TCP Reno 之類的高積極性協定共存時會導致網路資源享

用不公平[3, 4, 11, 17]，效能大幅度下降。

2.3 Ad-hoc Network 下改進效能的相關研究

如同前面章節所述，TCP的機制，在無線網路的環境中，會因為無線網路的種種特性，使得TCP的整體效能很差，為了提升TCP在無線網路下的效能，有許多不同方法被提出來。

2.3.1 Split TCP for MANET

Split TCP[23]設計的目的主要是處理在 MANET 環境下跳數(hop count)很多時會頻繁地發生路由失效(route failure)所產生之問題；這是因為長的路徑(connection)比短的路徑更容易發生封包遺失的情形。如圖 2.7 所示。因此為了提升效能， Split TCP 便以此概念將一個長的連結分為較短的多個區段(segments)，兩個區段間的節點稱為代理節點(proxy node)，代理節點收到封包需存在緩衝區(buffer)中，並且回傳 ACK 給前個代理節點，代理節點需負責控制傳送給下個代理節點的速度，而控制速度的方式就是使用每個代理節點中的擁塞視窗。

為了確保傳送端到接收端的可靠傳輸，接收端仍要跟傳統 TCP 一樣傳送 ACK 到傳送端。在傳送端有兩個傳送視窗(transmission window)：擁塞視窗(congestion window) 以及端對端視窗(end-to-end window)。

擁塞視窗依據下個代理節點所回傳的 LACK(Local ACK，每個代理節點回傳至前一個代理節點的 ACK)而改變，端對端視窗則是依據接收端所回傳的 end-to-end ACK 而變化，在每個代理節點中會有擁塞視窗以便控制兩個代理節點之間的傳送速率。

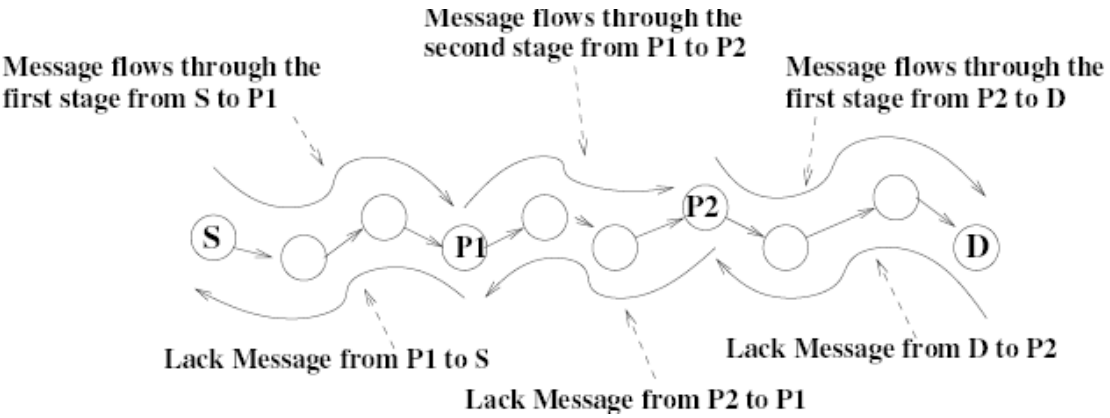


圖2.7：Split TCP示意圖

2.3.2 Transport Layer Revisited

此篇論文[18]指出，在 End-to-End route 並非總是可用的環境下，使用中間節點的幫助能達成有效率的資料傳送，然而到目前為止只有少數這樣的傳輸協定被提出，其中一個就是 Split TCP [23]。這篇論文就是基於這個原理所提出。

他們所提出的方法分為 hop-by-hop layer 以及 end-to-end layer。Hop-by-hop layer 執行於每個節點，負責流量控制(flow control)及擁塞控制(congestion control)，在 hop-by-hop layer 資料傳送的單位是片段(fragment)，由數個封包組成(8 個)，而 end-to-end layer 則是執行於連結的端點，用於確保端對端的資料傳輸，在這個 layer，資料傳送的單位是區段(segment)，由數個片段所組成(4 個)。

他們的方法是在每個鏈結的傳送節點使用 rate-based 的流量控制，利用對每個片段測量的傳送時間以估計傳送速率(sending rate)。

若是傳送出去的資料沒有被 ACK，則從傳送端(source)重傳，如果一個區段在一段時間內都沒被 ACK，傳送端需要重傳這個區段，由於重傳整個區段會為網路帶來相當重的負擔，因此傳送端使用 RTO 指數後退法(exponential back-off)讓 RTO 以指數方式成長，減少逾時次數，以避免過多次的不必要重傳。

這篇論文提出的 hop-by-hop protocol，能夠處理封包遺失，也就是說，當一個封包

在任何一個中間鏈結遺失，接收此封包的節點便不會 ACK 此片段，並且前一個 hop 就會重傳此片段。

此篇論文於 IEEE COMSEARE’07 提出，與我們的方法較為類似，然而此法於逾時時需重傳整個區段，導致 overhead 過大，平均延遲時間拉長許多，若是區段的 ACK 遺失那麼浪費的網路資源更為可觀。

2.3.3 ATCP

ATCP (Ad-hoc TCP) [25]是基於回饋方法來辨識封包遺失的原因及網路的狀況。ATCP 在網路層與傳輸層中間多加一層協定，它不改變上層TCP 的運作邏輯，因此可搭配上層不同版本TCP 一起運作。ATCP 是假設網路會傳出壅塞與斷線訊號，它於發送端過濾網路傳上來的訊號，以分辨封包遺失是網路壅塞、無線傳輸錯誤或斷線所造成。ATCP 並因此而瞭解網路狀況，調整封包傳送速度。若是因為無線傳輸錯誤，則不改變壅塞視窗，不改變封包發送速度。若是斷線，則壅塞視窗由1重新開始，封包發送速度也由慢速重新開始。

2.3.4 TCP-F

TCP-F (TCP-Feedback)[7]是一種根據網路層回饋的機制，當 TCP 傳送端接收到中間節點傳來的路由失敗通知，RFN (Route Failure Notification)，告知封包遺失是因為 Route Failure 而不是擁塞所造成。當收到 RFN 通知後，傳送端會認為到接收端沒有可使用的路徑，便會凍結計時器和擁塞視窗的大小，不再發送任何資料。如果收到路由協議(Routing Protocol)所攜帶的路由重建通知，RRN（Route Re-establishment Notification），傳送端會認為到接收端之間已有可用的路徑，便會恢復中斷前所有和此連結有關的變數並且開始發送資料。同時，設置了路由失敗計時器防止傳送端無止盡地等待 RRN 通知。傳送端一旦收到 RFN 通知，路由失敗計時器就會被觸發。如果路由失敗計時器到期仍

未收到 RRN，傳送端就認為是由於網路擁塞造成的封包，而啟動 TCP 擁塞控制機制。

2.3.5 TCP-BuS

TCP-BuS (TCP Buffering capability and Sequence information)[22]是使用類似於 TCP-F 的檢測機制，當路由失效和重新建立時，Pivoting Node(發生路由失效的節點)會使用兩個控制訊息 ERDN (Explicit Route Disconnection Notification)和 ERSN (Explicit Route Successful Notification)告知傳送端，傳送端收到 ERDN 則停止傳輸、收到 ERSN 則恢復傳輸。Pivoting Node 並在控制訊息發送之後監聽頻道，由於下個節點在發送控制訊息時，Pivoting Node 也能收到下個節點所發送之訊息，如果沒有聽到控制訊息被下個節點發送，則重發控制訊息；而在路由重建過程中，已被傳送的封包將被暫存起來，當路由重建之後便可繼續傳送。

2.3.6 Fixed-RTO

Fixed-RTO [12]這種方法的理念就是認為所有鏈結的中斷都是暫時的，如此出現路由失敗時可以迅速恢復，不需啟動 TCP 的指數後退機制以免造成不必要的長時間的延遲。當連續兩次重傳都逾時，傳送端認為路由出現問題，停止 TCP 指數後退機制，並且改用固定的時間間隔而不是使用指數逐漸增加的時間間隔重傳封包。

若是遺失的封包在第二次 RTO 到期後仍未被 ACK，發送端將 RTO 的值加倍，然後不斷重傳遺失的封包，但保持固定的 RTO 值直到重傳的封包被 ACK 為止。

2.3.7 TCP-Muzha

TCP-Muzha [24]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。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充份利用頻寬並避免產生擁塞，並可區分出封包遺失的原因是否為網路擁塞所引起，增進整體的效能。使用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。

2.4 小結

目前在無線隨意式網路中大部分的改進 TCP 的方法，大都著重在於利用分辨封包遺失的原因，以減少 TCP 不必要的降低傳送速率的機率，較少利用中間節點以幫助確保傳送的方法。然而在無線隨意式網路環境下，不穩定的環境(如節點移動、競爭及碰撞等)使封包容易遺失；傳統的 TCP 只能從傳送端重傳封包，然而每次從傳送端重傳封包會遭遇到相同的情況，造成惡性循環，不斷的從傳送端重送封包。

TCP 在 MANET 環境下，越多跳數時傳輸效能越差，因此實際的 MANET 的跳數不可能太多，過長的跳數意義不大；且在 MANET 中，封包的快速送達較大量傳輸資料更為重要。我們的研究針對 MANET 中每個節點皆可能是一部電腦而可以方便的參與 TCP 運作的特點，提出效率更好的 TCP。我們讓 TCP 所經過的每一個節點都成為代理節點執行 Local TCP，每個相鄰節點之間的封包採用一去一回的 ACK 方式，因此不需要擁塞視窗，而為了其傳輸效率，使用搭順風車(piggybacking)的方法減少過多的 overhead。

第三章

Hop-by-Hop TCP

MANET的不穩定環境使得封包往往容易在傳送過程中遺失，傳統TCP只能由傳送端進行重傳，因此使得封包在從傳送端至接收端的長路徑中遺失的可能性提升，我們建議讓網路中的每個中間節點能夠進行重傳動作，讓每個節點都幫忙保留住封包，當發現封包遺失時，只需從Local端重新發送，而不需回到傳送端重送。

3.1 設計理念

傳統的TCP只會在兩個端點上執行，這是因為有線網路中封包較不易因為轉送的介質不穩定而遺失，因此在兩個端點執行TCP以控制傳輸速度，確保封包能夠送達接收端。然而MANET的不穩定環境容易造成封包遺失，傳統TCP在不穩定的環境中效能非常低落，且因MANET中每個節點都可能是一部電腦，因此我們可以執行protocol在每個節點中，而不像一般而言網路的中間節點都是路由器(router)，若要在路由器上執行TCP協定則需要將每個路由器升級才能支援新的協定。因此我們設計了**Hop-by-Hop TCP (HBH TCP)**，有別於其他TCP版本，Hop-by-Hop TCP除了有傳統TCP中兩端點間的TCP協定之外，並在每個節點間運行TCP協定，在MANET這種不穩定的環境中，當發生封包遺失時不必從傳送端重傳封包。在網路狀況很差的情形下，我們期望能得到更好的效能。

3.2 設計目標

1. 降低平均延遲時間使封包能快速送達。
2. 減少重傳次數。
3. 提升整體效能。
4. 和其它協定共存時，仍能保有相當的效能。

3.3 TCP 設計議題

設計一個的TCP 協定時，有以下必須考慮的議題：

1. 速度調控 ：如何不造成擁塞並有效利用頻寬
2. 公平性
3. 重傳機制
4. 暫存區控制
5. 如何處理封包遺失 (端對端的封包遺失如何處理)
6. 如何處理ACK遺失
7. 重送次數上限

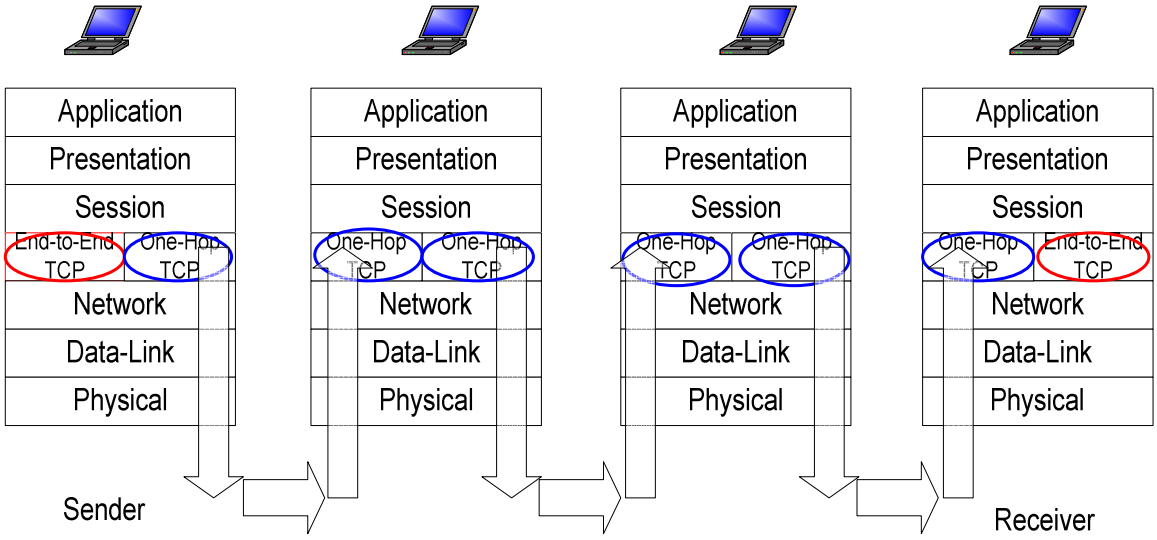


圖3.1：Hop-by-Hop TCP示意圖

3.4 Hop-by-Hop TCP

如圖3.1所示，Hop-by-Hop TCP是由在傳送端及接收端之間的End-to-End TCP及位於每個相鄰節點之間的One-Hop TCP所組成，利用無線隨意式網路特性，使封包在不穩定環境中能縮短傳輸時間、更快速送達目的地端的協定。

當Hop-by-Hop TCP建立連結(connection)時，如同傳統TCP的三方交握(3-way handshake)一樣，傳送端的End-to-End TCP須要發送SYN封包，每個收到SYN封包的節點立刻回覆一個SYN的Local ACK給上一個節點，此時便啟動了兩節點間的One-Hop TCP連結，SYN封包一站一站確保傳送至接收端，接收端的End-to-End TCP需回覆SYN/ACK封包至傳送端，如同傳統TCP，傳送端傳送一個ACK封包至接收端完成三方交握之後End-to-End TCP連結也建立成功，便可開始進行傳送。

傳送完成之後，結束連結時傳送端的End-to-End TCP發送FIN封包至接受端，如同傳統TCP的三方交握，最後傳送端須傳送一個ACK封包至接收端，收到ACK封包的節點一站一站確保將ACK封包轉送至接收端，同時每個節點結束One-Hop TCP連結，收到

ACK封包的兩個端點也結束End-to-End TCP連結。

本節我們將解釋何謂Hop-by-Hop TCP以及Hop-by-Hop TCP的運作方式。

3.4.1 End-to-End TCP

End-to-End TCP是執行於網路的兩個端點，確保封包能從傳送端成功傳輸至接收端的協定。整個End-to-End TCP承襲了TCP Reno的機制，如同我們在前面所解釋的，分為慢啟動及擁塞避免兩大階段，傳送端根據End-to-End ACK以調整擁塞視窗大小，不同之處為：

1. 傳送端的擁塞視窗設定一個上限值：

   在[8, 15]中指出，在MANET中小的擁塞視窗有較好的性能，且由於我們的方法可以降低封包遺失率，可能會造成擁塞視窗過度成長，因此我們將傳送端的擁塞視窗設定一個上限值，使其不會過度的成長。

2. 在逾時重傳機制中則採取較大的End-to-End RTO 計算方式：

   為了避免不必要的逾時重傳而影響效能，我們將採取較大的RTO計算方式。

3.4.2 One-Hop TCP

One-Hop TCP 是執行於相鄰兩個節點之間，用以保證將封包送達下游節點的傳輸層協定。One-Hop TCP主要功能是：(1)暫存封包，(2)進行遺失封包的重傳。One-Hop TCP的主要執行方式如下：

1. 速度調控：

   - 上游端一次傳送一個資料封包給下游端，上游端收到下游端所回覆的Local ACK封包之後，再送下個資料封包；換言之，擁塞視窗固定為一。

2. 重傳：

   - 如果在One-Hop RTO (Retransmission Timeout) 逾時之後仍未收到下游端

回覆的Local ACK封包，則上游端重傳未被回覆的資料封包。

- 若封包超過重傳次數門檻(Retransmission Threshold)則不再重傳。

傳統TCP用於一般的網路時，如果一次只送一個封包，則因路徑太長導致來回的等待時間太長，效能不彰，所以採用一次送多個封包並用擁塞視窗控制。然而One-Hop TCP是使用於相鄰兩節點之間，因此等待時間較短，且如果傳送封包的節點要繼續傳送下一個封包時，下游端也正要回覆Local ACK，兩邊都要傳送時則可能會造成衝撞；因此將擁塞視窗大小固定為一。而One-Hop RTO之內若沒收到下一站回覆的Local ACK，則視為資料封包遺失，就重傳此資料封包；當重傳次數超過門檻值之後，判斷此時很可能是發生路徑鏈結失效，或是下游節點的暫存區滿溢，因此傳送數次仍無法送達下一站。為了避免不停地重傳此封包而造成更大擁塞及頻寬浪費，我們選擇丟棄超過重傳次數門檻值的封包，由傳送端降速重傳，流程如圖3.2、圖3.3所示。

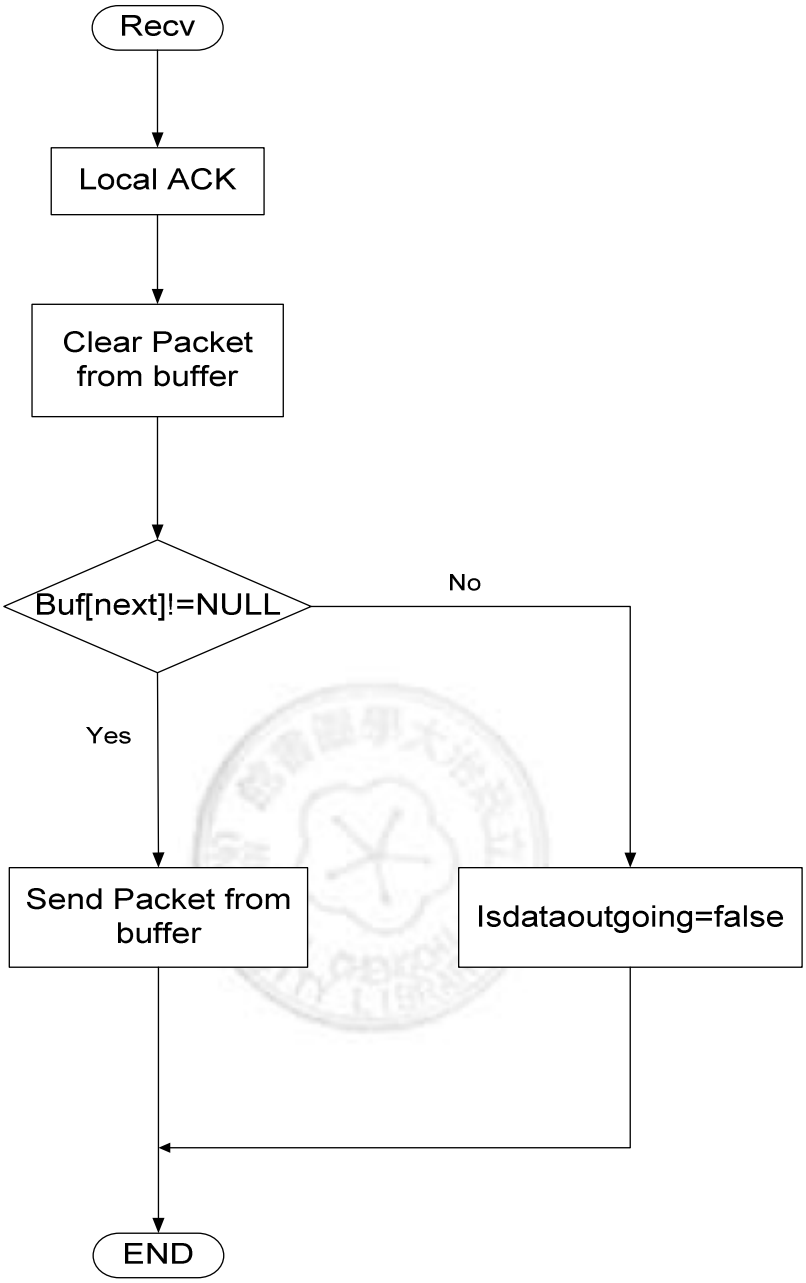


圖3.2：One-Hop TCP接收Local ACK流程圖

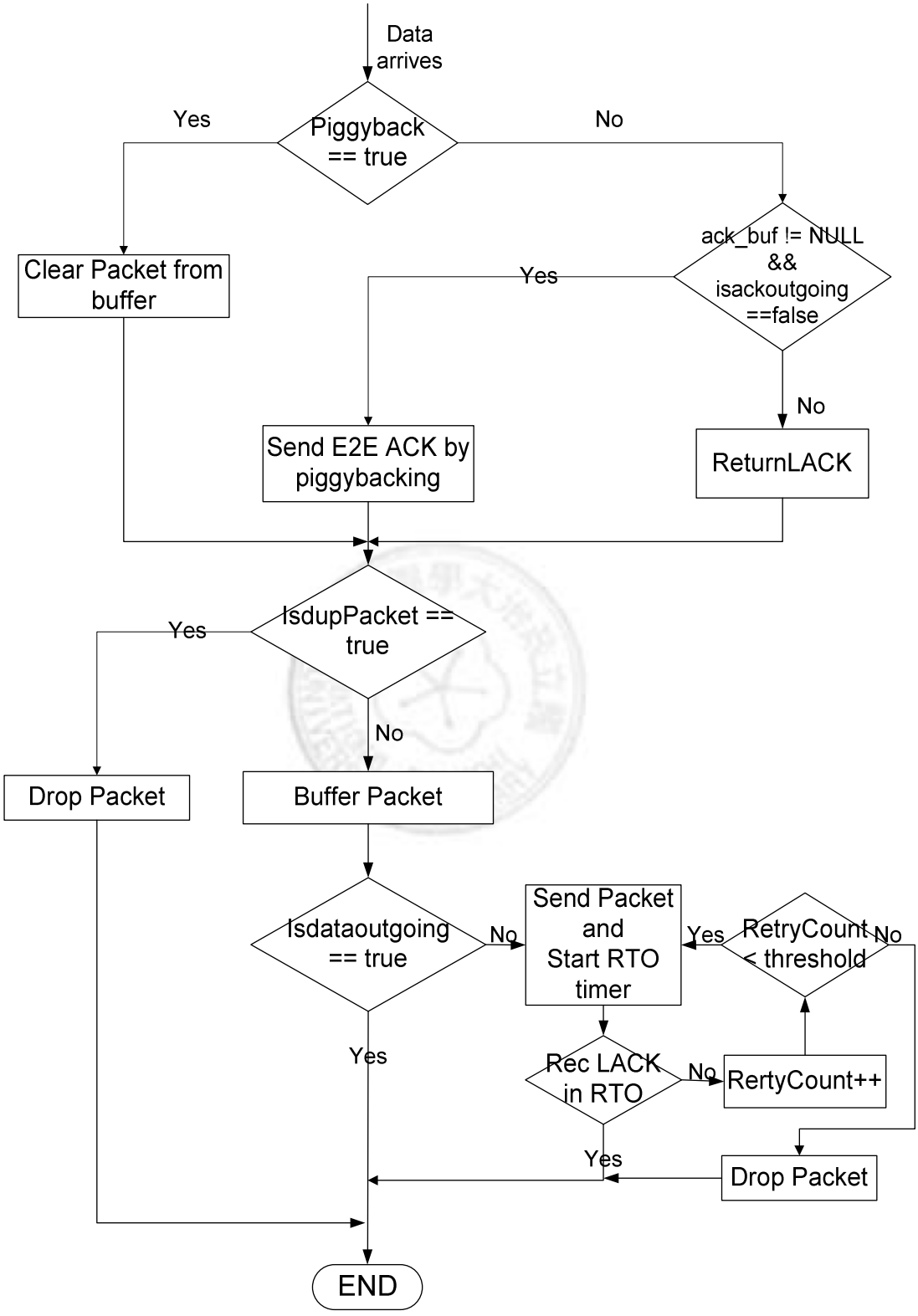


圖3.3：One-Hop TCP接收資料封包流程圖

3.4.2.1 遺失封包的處理

我們希望在MANET的不穩環境中能夠提昇傳輸可靠度並縮短延遲時間，為了能夠確保成功的傳輸，復原遺失的封包，在此我們討論One-Hop TCP的重傳機制。在MANET下可能會發生封包遺失的原因有：

1. 雜訊干擾。
2. 競爭者太多搶不到頻道，節點無法接收到下一站回傳之Local ACK。
3. 暫存區滿溢。
4. 鏈結斷線(Link failure)。

如同傳統TCP一樣，傳送端中仍需對資料封包維持RTO，我們稱呼為End-to-End RTO，而每個節點中需要對每個封包設定One-Hop RTO。

若發生了傳輸錯誤而遺失封包，One-Hop RTO逾時，傳送節點就會重送此封包給下一站的節點，提昇傳輸可靠度，不必回到傳送端進行重傳。

若是鏈結失效，則發生斷線的中間節點將無法成功傳送封包，而我們的方法提供了能從此節點就地尋找另一個路徑重建以成功傳送的機會；為了避免無法從發生斷線的節點找到可用路徑而不斷地傳送失敗，我們在One-Hop重送次數到達一個門檻值之後，便將此封包丟棄，由下層的機制(如路由協定)決定如何處理路由失效，之後由傳送端的End-to-End TCP進行重傳。

若是發生暫存區滿溢而無法成功傳送，到達我們預定的One-Hop重送次數限制，就丟棄封包，以降低擁塞程度，同樣仍由傳送端的End-to-End TCP降速重送。

3.4.2.2 One-Hop TCP執行流程

如圖3.4所示，One-Hop TCP的執行流程分為三階段狀態，準備階段(Ready)、等待階段(Wait)及完成階段(Done)。當One-Hop TCP建立連結之後，就進入準備階段，收到封包之後若能進行傳送，則送出封包並且進入等待階段，開始此封包之One-Hop RTO計時器，如果發生逾時並且重傳次數未超過預定重傳門檻，便重傳此封包，持續在等待階段，等待下一站所回覆之Local ACK，若超過了重傳門檻則丟棄此無法成功傳送之封包，並進入完成階段；當收到了此封包之Local ACK，表示此封包已成功傳輸至下一站，進入完成階段、並從暫存器中清除此封包；如果此時暫存器中有其他封包等待傳送，便送出此封包、並進入等待階段，若無則停止計時器、進入準備階段。

Hop-by-Hop TCP機制歸納於表3.1，演算法虛擬碼見圖3.5。

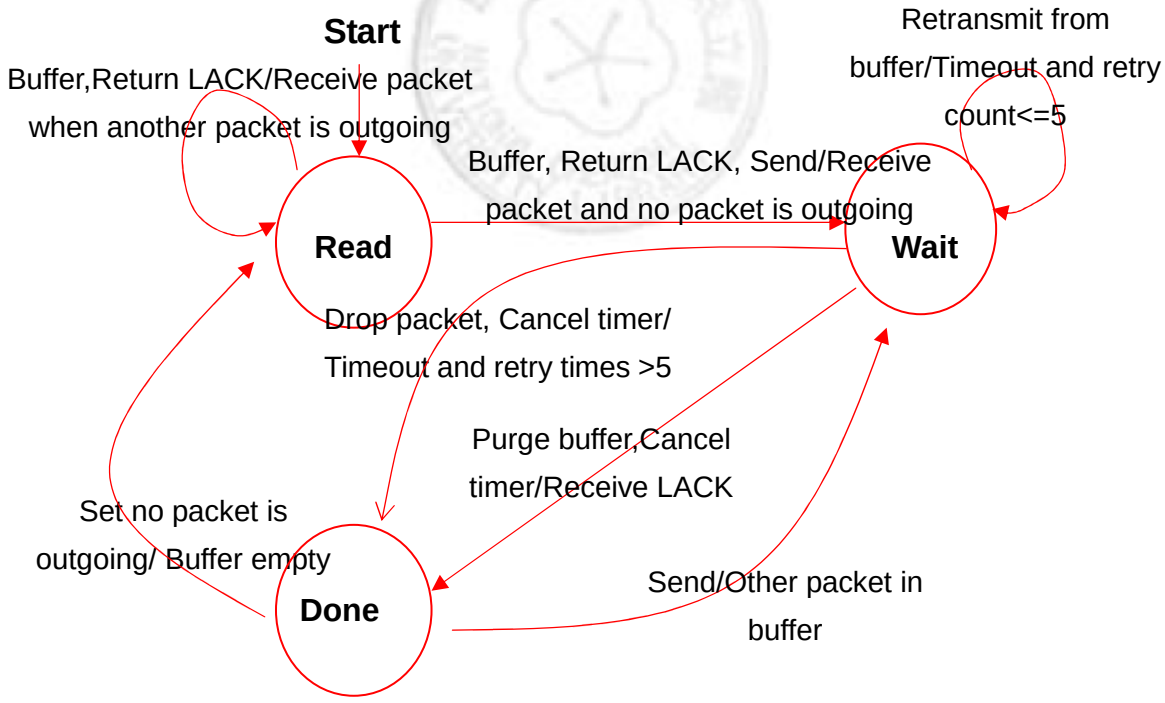


圖3.4：One-Hop TCP狀態轉移圖

表3.1：One-Hop TCP機制

Event	Status	Actions taken in the receiving node	Next state
Receive a packet when flag no_packet_outgoing is false	Ready	檢查是否重複封包，若不是則暫存此封包；回覆此封包之 Local ACK，若反方向有封包欲傳送，則 piggyback 上去 Buffer the new packet; if there is any packet to be sent in another direction, piggyback it, if not, return LACK	Ready
Receive a packet and no packet is outgoing	Ready	檢查是否重複封包，若不是則暫存此封包；回覆此封包之 Local ACK，若反方向有封包欲傳送，則 piggyback 上去；送出封包、啟動計時器 Buffer the new packet; if there is any packet to be sent in another direction, piggyback it, if not, return LACK, send the packet, and start timer	Wait
Local Timeout and retry count < 6	Wait	重傳逾時的封包；重新啟動計時器 Retransmit the Timeout packet; restart timer	Wait
Local Timeout and retry count > 5	Wait	從暫存器清除重傳超過五次的封包；停止計時器 Purge the packet that retransmits over five times from buffer; stop timer	Done
Receive a LACK from downstream node	Wait	清除暫存器內相對應封包；停止計時器 Purge the packet from buffer; stop timer	Done
Other Packet in Buffer	Done	送出暫存器內的下一個封包，並啟動計時器 Send the next packet from buffer; start timer	Wait

Buffer empty	Done	設定為無封包正在傳送，等待接收下一個封包 Set flag no_packet_outgoing to true; wait the next packet	Ready

```
Receive ( Packet ){
   if ( Data )
   {
      ReturtLACK ( ) ;
      if ( NewPacket_ == True )
         { BufferPacket ( ) ; }
      if ( isdataoutgoing_ == False )
         {
            SendPacket ( ) ;
            Isdataoutgoing == True;
         }
   }
   if ( LACK )
   {
      PurgeBuffer ( ) ;
      if ( Buffer == NULL )
         { set isdataoutgoing == False ; }
      else
         { SendPacketInBuffer ( ) ; }
   }
}
```

圖3.5：One-Hop TCP演算法虛擬碼(Pseudo code)

3.4.3 提昇End-to-End ACK存活率

End-to-End ACK就是傳統TCP中，由接收端回傳至傳送端的ACK封包，在傳輸中也會面臨高遺失率的問題，如果End-to-End ACK遺失，我們對於資料封包所做的保留重送等動作幾乎全部浪費，因此我們將End-to-End ACK也使用Hop-by-Hop的方式傳送，將End-to-End ACK封包逐步傳送至傳送端。

3.4.4 降低overhead之方法

使用我們的方法可能會產生相當高的overhead，因此我們使用下面介紹的方式以降低overhead。

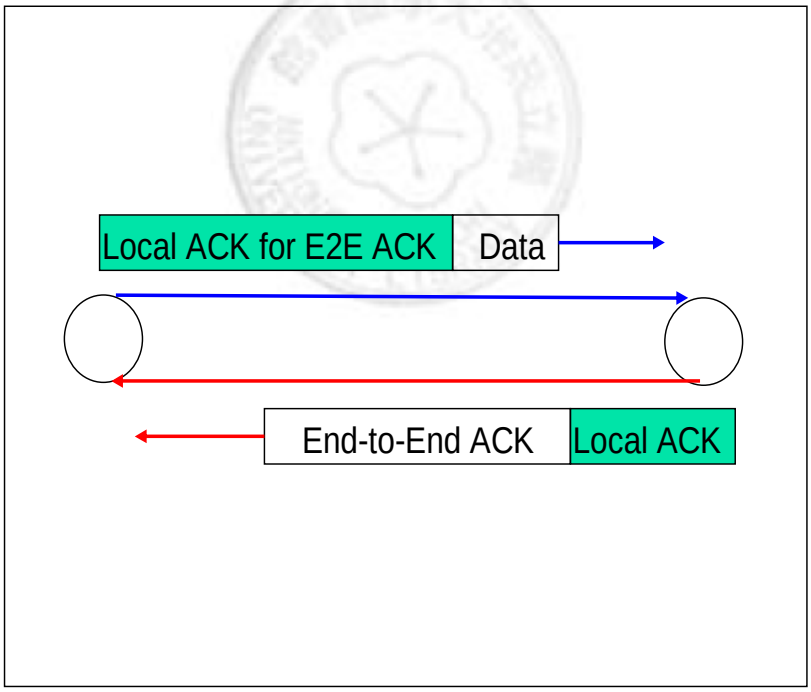


圖3.6：Piggybacking 機制

3.4.4.1 Piggybacking 機制

如圖3.6所示，我們可以看出兩個方向的不同封包，可分為：

1. Forward packet stream (由左至右) : 資料封包以及Local ACK for End-to-End ACK
2. Backward packet stream (由右至左) : End-to-End ACK以及Local ACK

為了要減少過多的Local ACK封包，我們將Local ACK for End-to-End ACK封包搭順風車在資料封包上、Local ACK封包搭順風車在End-to-End ACK封包上。我們將資料封包以及End-to-End ACK封包的封包表頭(packet header)中新增一個欄位，將要回覆的Local ACK號碼直接放進去，利用piggybacking機制減少overhead。

3.4.4.2 重複封包的處理

由於下游節點所回覆之Local ACK亦有遺失的可能，即使傳送的節點成功將封包送至下游節點，傳送封包的節點因為沒接收到Local ACK，會判斷為封包遺失，並重傳此封包至下游節點；當下游節點收到重複的封包時，若是直接就暫存起來並且轉送，則會造成不必要的資源浪費，因此我們在封包中加入一個buf_rtx_欄位，當節點發生One-Hop逾時的時候，就會在這個重傳的封包中註明，讓下一站知道這是重傳的封包；而每個節點收到封包的時候，便會記住它所收到的這個封包號碼，若是收到的封包中buf_rtx_欄位有被註記，則檢查此封包之封包序列號碼是否與剛剛收到的封包號碼相同，若是則直接回覆Local ACK，並且不暫存此封包，因為下游節點已經接收過此重複之封包。

3.5 與MAC層之關連及配合方法

如圖3.7所示，IEEE 802.11 MAC層的4-way handshake，在資料訊框傳送之後，接收到的節點需回覆一個ACK訊框，以確認有收到此資料訊框，若傳送節點沒有收到此回覆之ACK訊框，則需要重傳此資料訊框4~7次，此為MAC層之重傳，在[16]中提到，沒有MAC ACK，則傳輸吞吐量將會很低。然而，即使有MAC層之重傳，也是會發生傳送失敗遺失封包的情形，當網路狀況較不穩定或是傳送至下個節點的鏈結失效時，封包非常容易傳送失敗。當MAC層傳送成功，上層的One-Hop TCP也等於傳送成功，因此就不需要One-Hop TCP進行重傳；若MAC層的重傳仍無法成功，我們可由此中間節點進行傳輸層的重傳，再交由下層(MAC層)進行傳送，當網路品質不穩定時，利用此法能夠達到提昇傳輸可靠度的目的。而若要提昇傳輸可靠度，可修改MAC層之重傳次數到無限多次亦能達到效果，但由於修改MAC層之重傳次數需要升級所有硬體，因此實作上不易，我們的方式使用的是TCP層之重傳，實作上難度較低，且每個節點對於傳送之封包仍採取端對端之概念，以接收端之地址為傳送目的地，即便下個節點之鏈結失效，仍可主動尋找另一條至接收端至路徑繼續傳送，這是MAC層只有一跳(One hop)，只能知道傳輸範圍內的節點的方式所無法做到的。

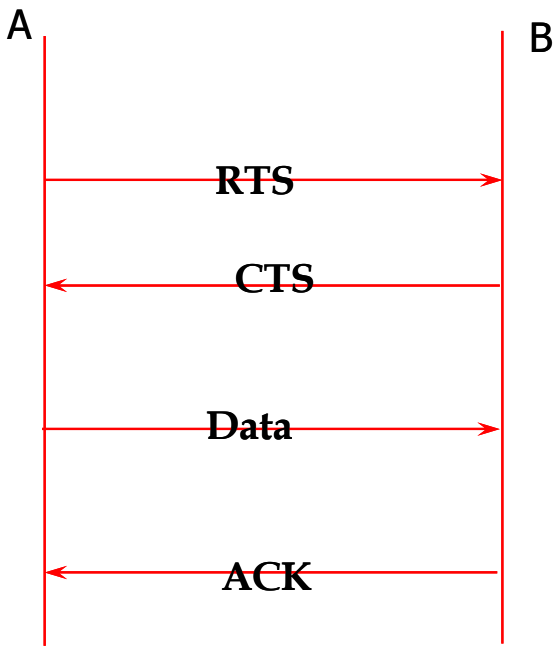


圖3.7：802.11 MAC Layer 4-Way Handshake

3.6 小結

Hop-by-Hop TCP藉由中間節點協助，在每個節點之間執行One-Hop TCP，讓封包能更可靠的一站一站傳輸至目的地，以避免在MANET不穩定的環境中頻繁地遺失封包，能夠更有效率的執行封包的傳送，以減少因為封包遺失所造成劇烈的傳輸速度下降，縮短封包成功送達目的地的時間。在下一章節中我們將於NS-2平台實驗模擬，評估我們所提出之Hop-by-Hop TCP擁塞控制機制。

第四章

效能評估

本研究將以多組實驗來模擬評估我們的Hop-by-Hop TCP機制，在不穩定的環境中藉由觀察數個效能指標了解我們方法的效能，我們將和現有的TCP 機制做比較。

4.1 實驗目的

本實驗的目的在於驗證我們提出的方法之可行性，在網路不穩定的環境中能夠提升傳輸之效能。

4.2 實驗設計

本研究提出一個MANET下的TCP機制 (Hop-by-Hop TCP)，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現在被廣泛使用的TCP擁塞控制例如 TCP NewReno、TCP Vegas、TCP SACK等協定做比較。最後並觀察和TCP NewReno共存時的效能影響以及同步化的問題。

我們進行以下兩組實驗：

1. Performance Test

   觀察 Hop-by-Hop TCP 在 chain topology 下各種變因對效能的影響

   1A. 可靠度

   1B. 路徑長度

   1C. 頻寬

2. Fairness Test

   2A. 觀察多協定共存狀態下的公平性 (與 NewReno 共存的情形)

   2B. 觀察有多個 Hop-by-Hop TCP flow 並存時的公平性變化

4.3 實驗 1 : Hop-by-Hop TCP 的效能測試

4.3.1 實驗目標

觀察在單一 TCP 的情況下，Hop-by-Hop TCP 機制的整體效能。

4.3.2 評估指標

- 平均延遲的時間:封包從傳送端至接收端平均傳送的延遲。
- 平均 throughput :用平均 throughput (kbps/sec)來評估整體的效能。
- 封包重傳次數:傳送過程中發生重傳的次數。
- CWND (congestion window size) 的變化。由 CWND 的變化狀況可以知道速率的變化。

4.3.3 實驗流程

實驗的拓樸環境由 4~12 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，我們將以 TCP NewReno、TCP Vegas、TCP SACK 做為對照組，配合著我們的各個實驗觀察評估。

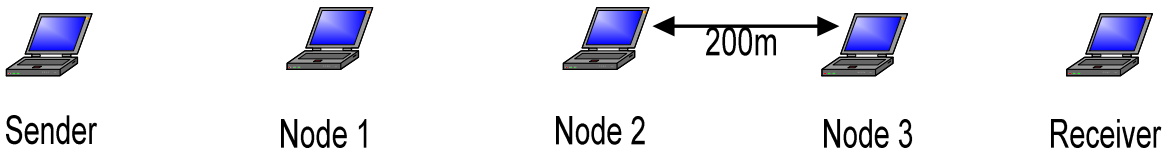


圖4.1：實驗一拓樸

在這個實驗中，我們將研究Hop-by-Hop TCP在4-10 hop的chain topology在不同error rate情形下，觀察Hop-by-Hop TCP的整體效能 (average throughput、重傳次數、平均延遲時間) 的狀況變化。圖4.1是第一個實驗的網路拓樸範例，包含了傳送端、接收端及不同數目的中間節點，在這個實驗中我們只建立一個TCP資料流，每個鏈結間的頻寬是1Mbps，路由器佇列的管理機制為DropTail，節點的傳輸半徑為250公尺，MAC Protocol為802.11，路由協定為DSR，封包大小設定為1000 bytes，我們將觀察Hop-by-Hop TCP的CWND變化、平均throughput以及平均延遲時間，並且與其他的TCP版本比較。

表4.1：實驗1參數

Parameter	Range
Buffer Size	50 packets
CWND Upper Bound	4
Link Bandwidth	0.5~1 Mbps
Error Rate	0.0~0.2
Number of Nodes	5~11
MAC Protocol	802.11
Routing Algorithm	DSR

4.3.4 實驗結果分析

我們從圖 4.2、圖 4.3 可以看出，Hop-by-Hop TCP 在擁塞視窗爬升至我們所設定的上限值之後，就停在上限值內不再震盪。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 NewReno 和 SACK 中，CWND 則快速的爬升而封包經常遺失並週期性的循環。在網路較不穩定時，Vegas 保守的擁塞視窗控制機制，讓擁塞視窗降到較低的位置，而 NewReno 及 SACK 則更容易大幅震盪，無法維持平穩的 CWND；Hop-by-Hop TCP 的擁塞視窗依然能維持固定的上限值，不因環境的不穩造成擁塞視窗的震盪，這是因為我們的方式封包能在封包遺失的時候從遺失的節點復原，因此不容易觸發 TCP 的逾時以及快速重傳等機制，使得我們能以平穩的速度傳送。

我們由圖4.4、圖4.5、圖4.6可以看出，當遺失率不高時，Hop-by-Hop TCP的延遲時間與NewReno等TCP的表現相當接近，但是當遺失率高時，Hop-by-Hop TCP的延遲時間則能顯著降低，從圖4.5可以看出，Hop-by-Hop TCP至少縮短25%的延遲時間，這是因為在封包遺失的時候，我們的Hop-by-Hop TCP能夠從封包遺失的節點直接當地重傳，而不必回到傳送端等到逾時重送，不需要經過長路徑才能重傳，使得平均延遲時間降低。

從圖4.7、圖4.8、圖4.9可以發現，在error rate為0的環境中，Hop-by-Hop TCP的平均

throughput略低於TCP Vegas、NewReno、SACK，這是因為Hop-by-Hop TCP有Local ACK，會有較多的overhead，因此會使得throughput表現稍不如Vegas、NewReno、SACK。而在圖4.8的不穩定環境中，平均throughput則比Vegas等TCP高25.7%，這是因為中間節點一站一站確保傳送成功，使得封包到達的機率增加，從圖4.11中顯示，Hop-by-Hop TCP從傳送端重傳機率較Vegas、NewReno、SACK低，由於重傳機率下降，使得資源不會在從傳送端重傳的長距離中過度浪費，因此也連帶讓效能提升；而從圖4.10可看出，使用搭順風車的方法之後，可以有效減少Local ACK的數量，降低overhead，可進一步提升Hop-by-Hop TCP的效能。

從圖4.6及圖4.9可以看出，當網路環境非常惡劣時(error rate = 0.2)，Hop-by-Hop TCP的延遲時間及throughput仍然比Vegas、NewReno、SACK優秀，但是在此種封包難以傳送成功的環境中，各種TCP的效能表現皆非常低落，因此我們之後的實驗便不以此種網路環境來當作參考；我們將以error rate = 0.1進行實驗，以觀察Hop-by-Hop TCP的表現。

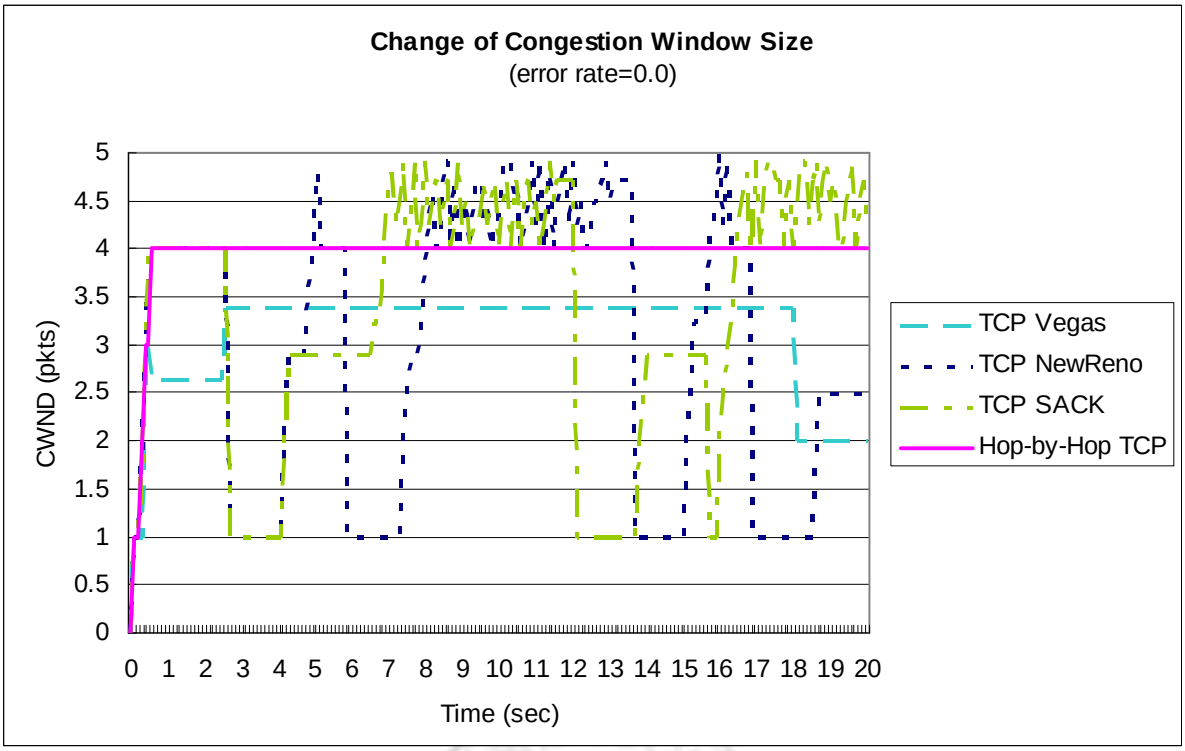


圖4.2：Change of Congestion Window Size (error rate = 0.0)

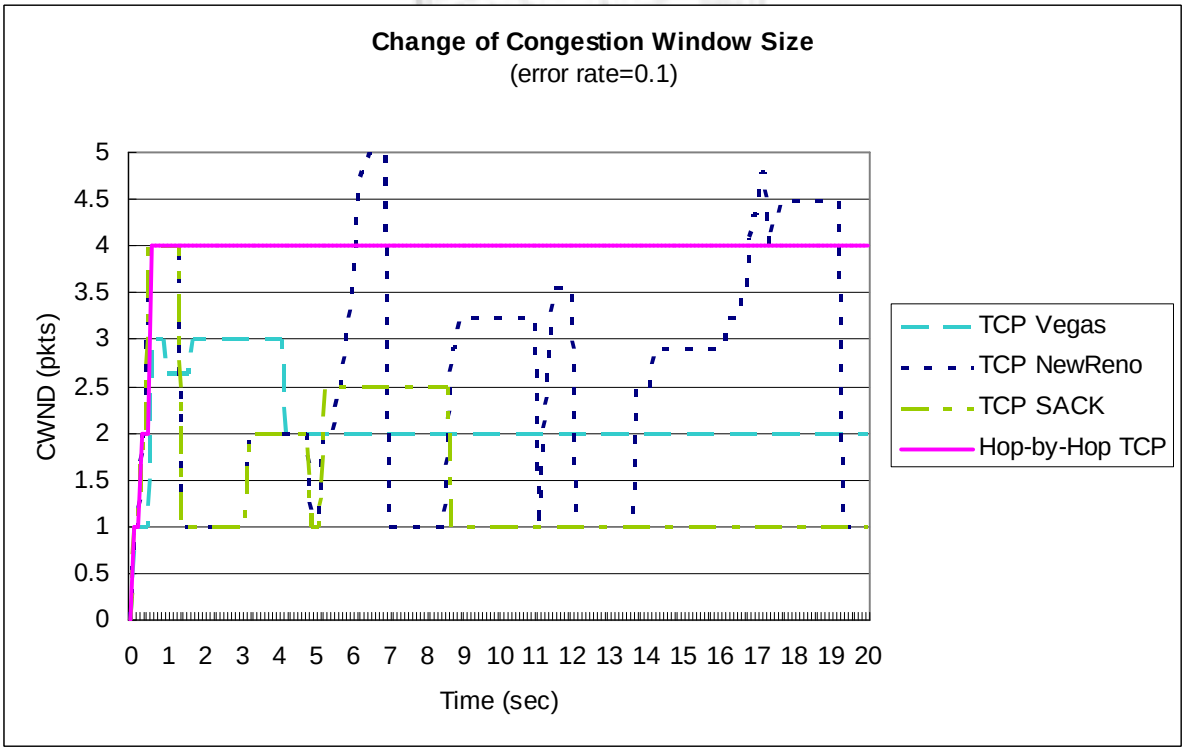


圖4.3：Change of Congestion Window Size (error rate = 0.1)

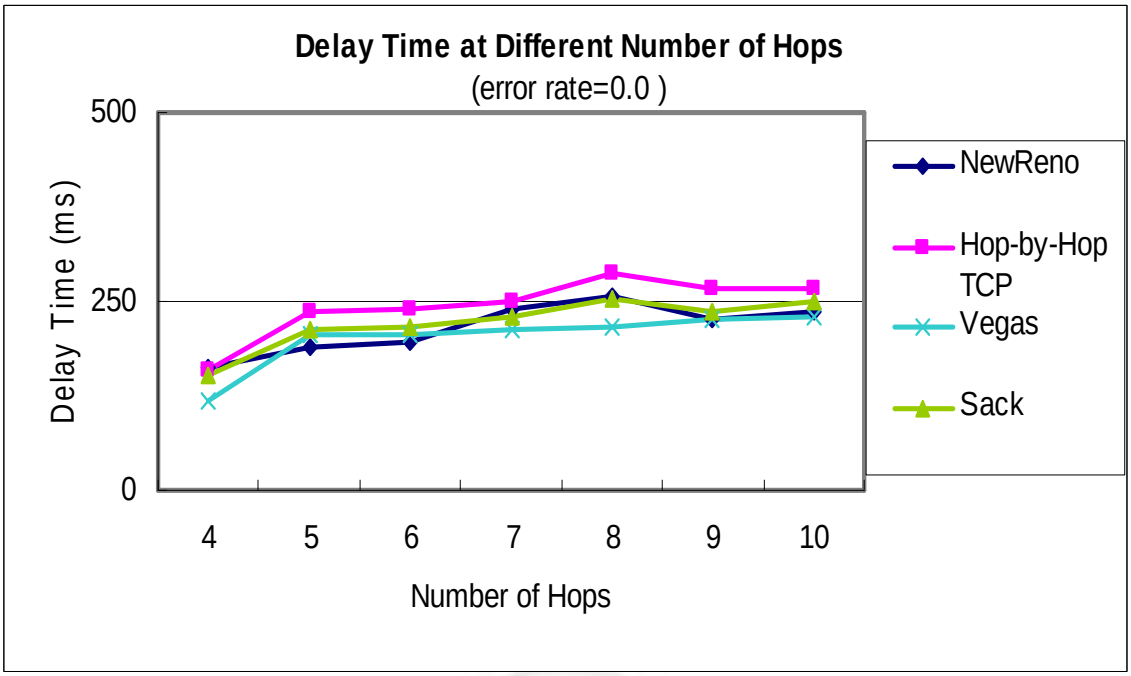


圖4.4：Delay time at different number of hops (error rate = 0.0)

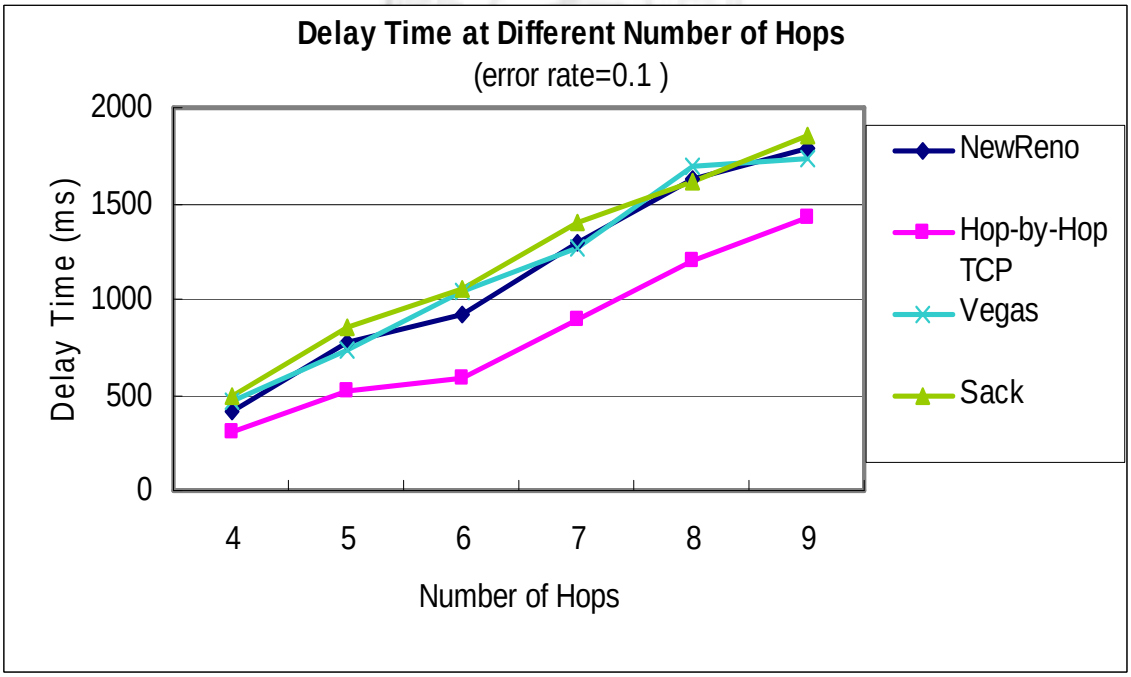


圖4.5：Delay time at different number of hops (error rate = 0.1)

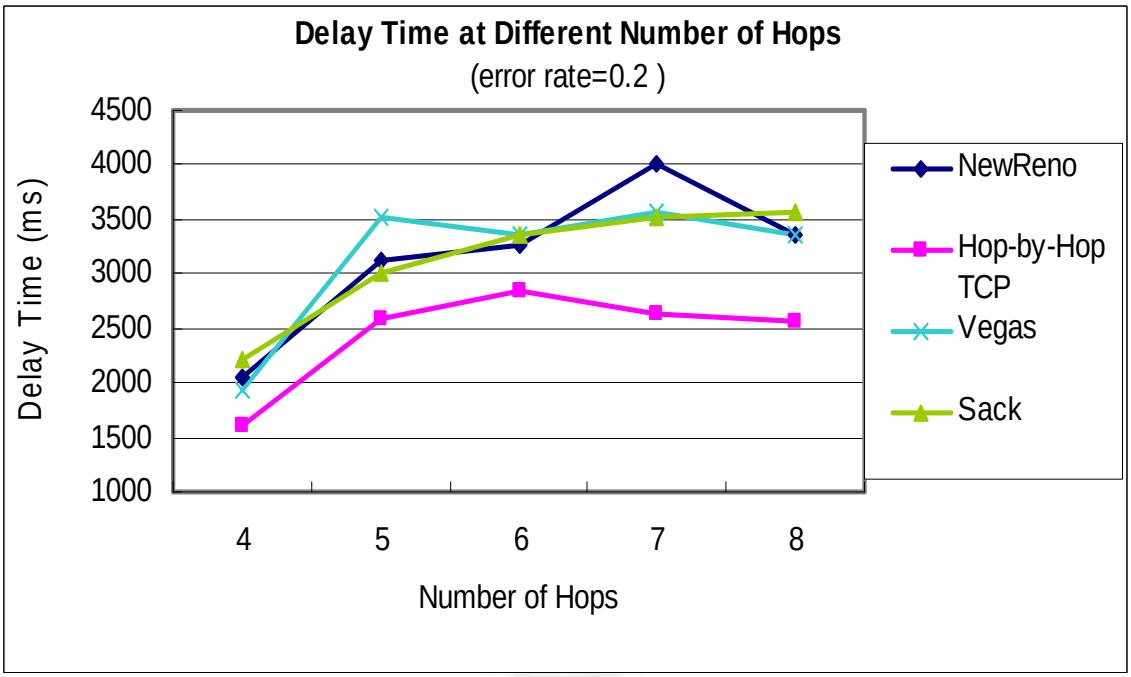


圖4.6：Delay time at different number of hops (error rate = 0.2)

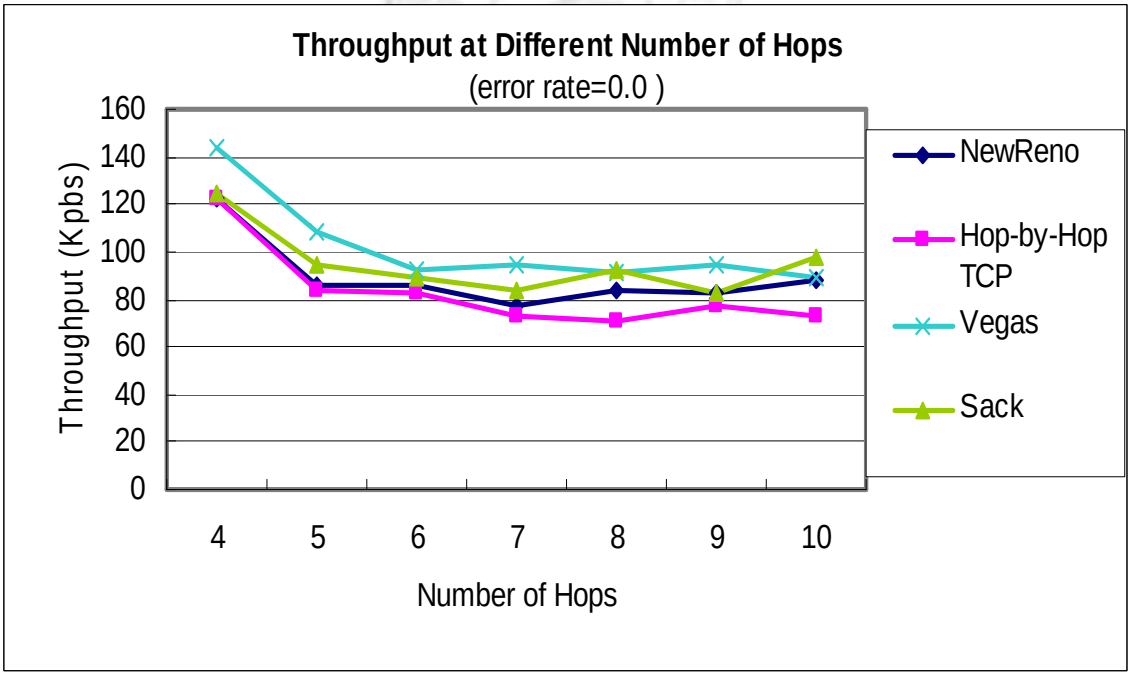


圖4.7：Throughput at different number of hops (error rate = 0.0)

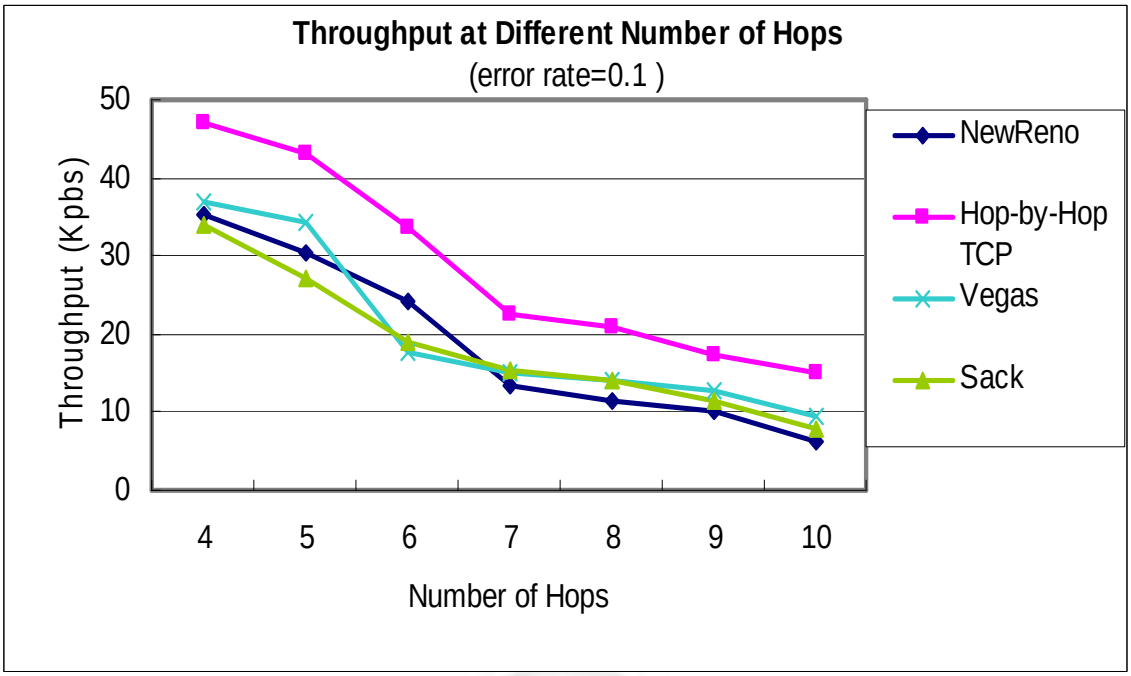


圖4.8：Throughput at different number of hops (error rate = 0.1)

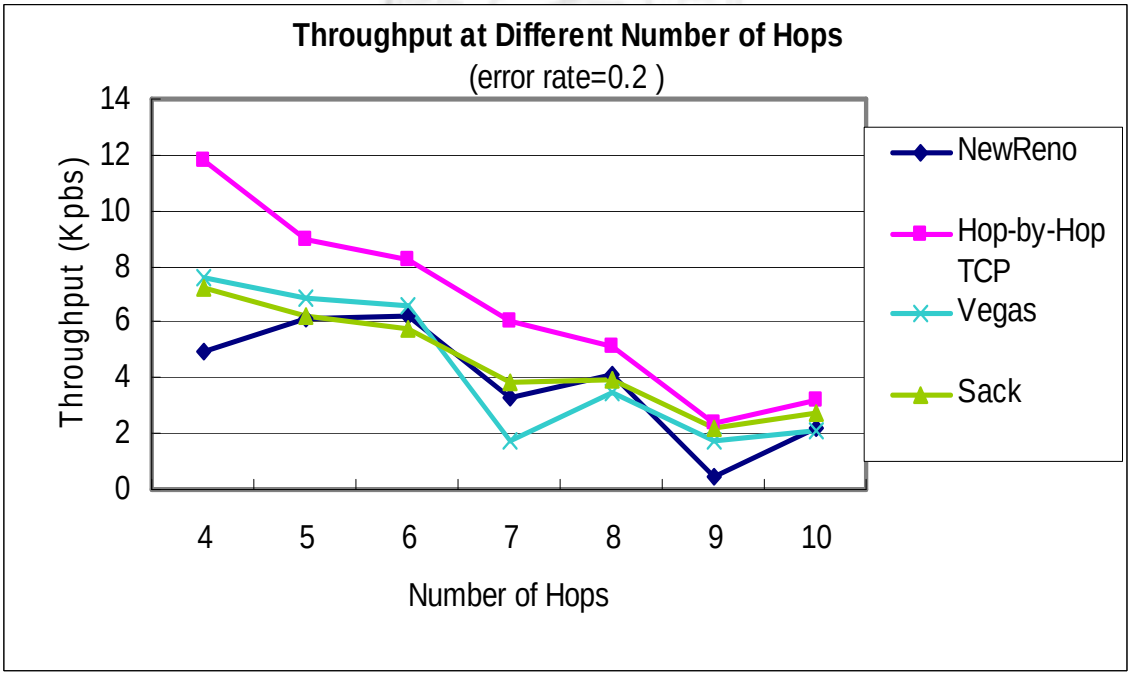


圖4.9：Throughput at different number of hops (error rate = 0.2)

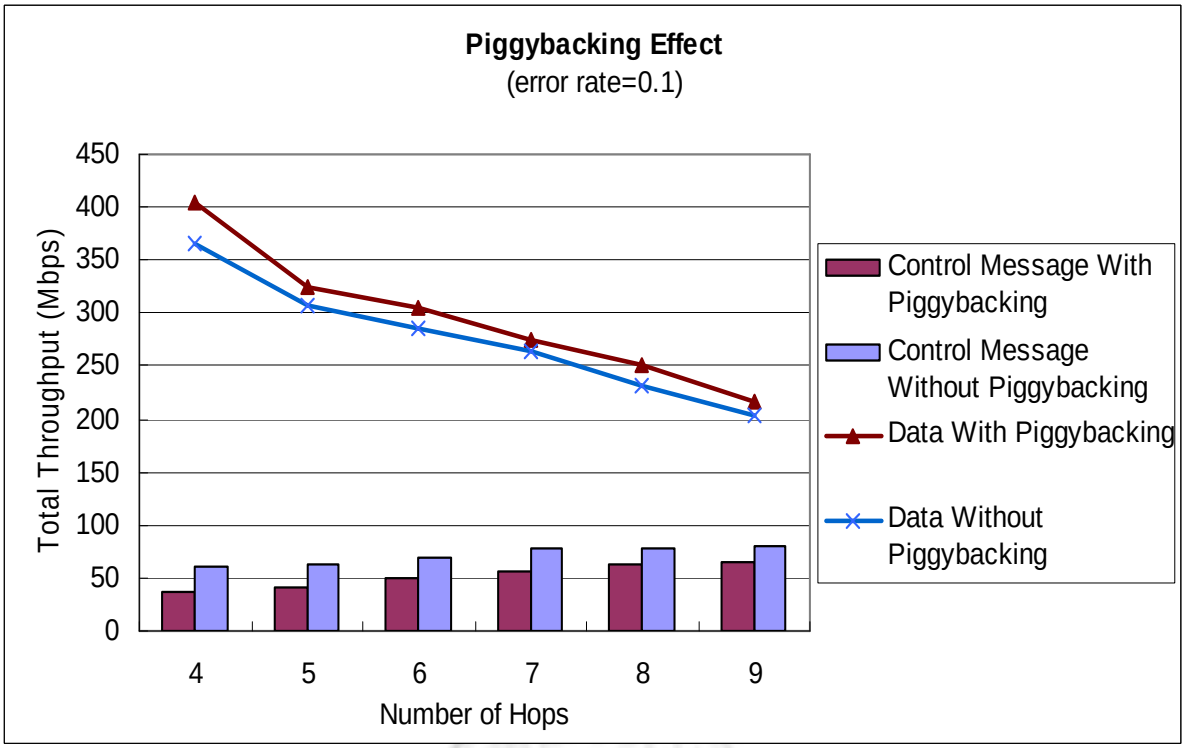
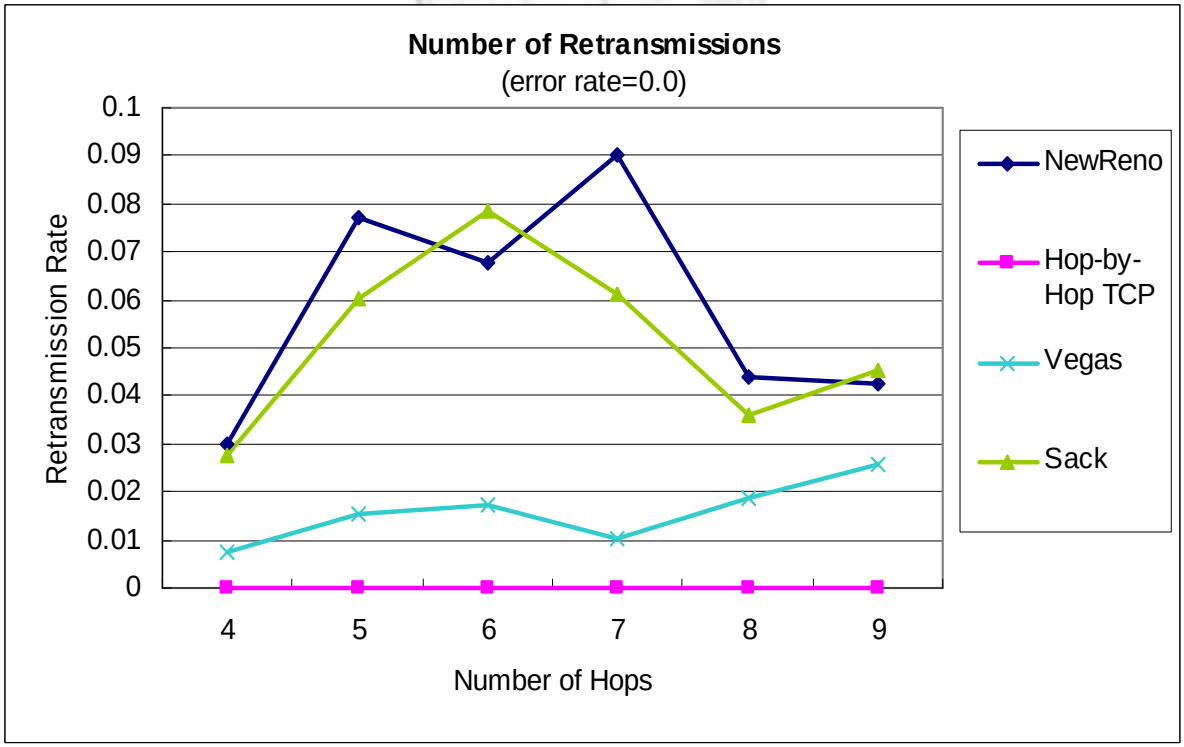
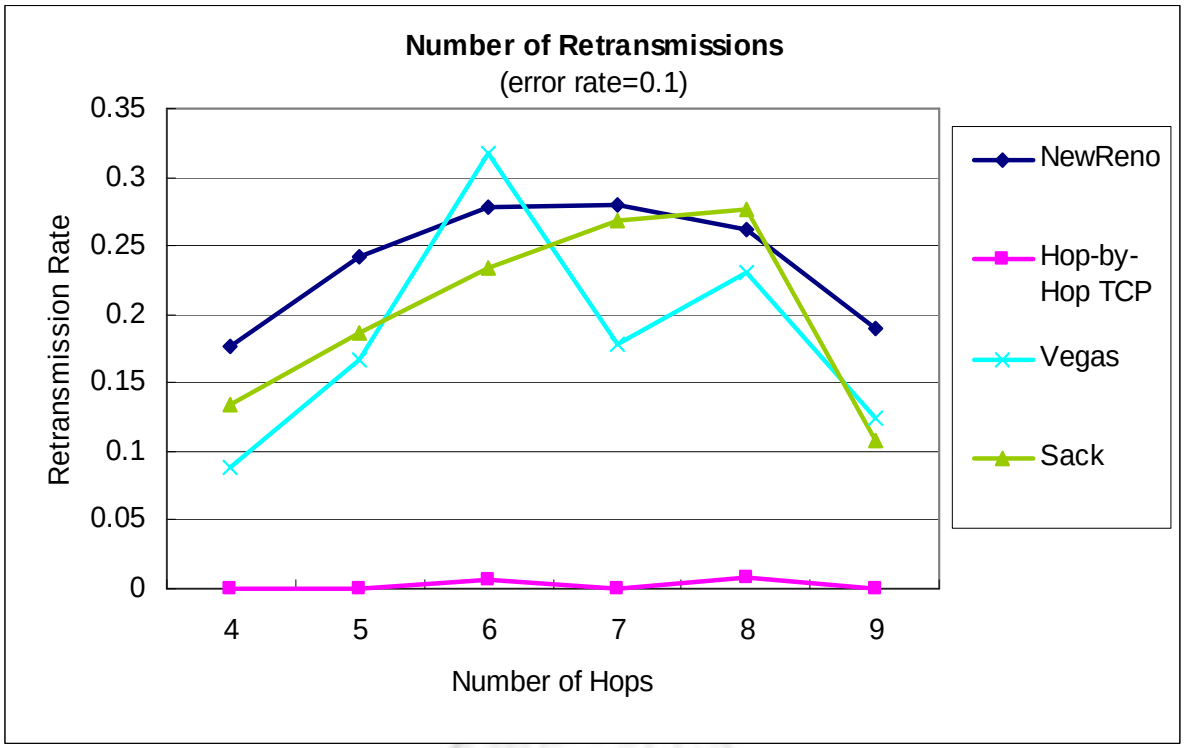


圖4.10：使用piggybacking機制及不使用piggybacking機制的throughput比較



(a)



(b)

圖4.11：不同版本TCP在傳送端重傳比率之比較

(a) error rate為0.0 (b) error rate為0.1

4.4 實驗 2 : Fairness test

4.4.1 實驗目標

網路上通常會有多條不同版本的 TCP flow 共存，彼此之間的擁塞控制會互相影響，會導致頻寬的不當競爭，例如最被廣為使用的 TCP Reno 就會排擠其他版本 TCP 的使用頻寬，本實驗的目的是要觀察當 Hop-by-Hop TCP 與 Reno 共同存在時，以及有多條 Hop-by-Hop TCP flow 共存時的效能表現。

因此我們的實驗分為 2A：觀察多個協定共存的情形下的公平性以及 2B：觀察有多個 Hop-by-Hop TCP flow 並存時的 throughput 表現。

4.4.2 評估指標

我們使用[21]中定義的公平性指標來評估公平性，每 n 個 flow 間的公平性指標計算方式為：

$$\left(\sum_{i=1}^{n} x_i \right)^2 \Big/ n \sum_{i=1}^{n} x_i^2 \qquad (1)$$

其中 x_i 代表第 i 個 flow 的 throughput，當公平性指標數值愈接近 1 即表示公平性愈高。

4.4.3 實驗 2A: 多協定共存狀態下的公平性實驗

目前網路上最被廣為使用的 TCP 協定為 TCP Reno，因此在使用其他版本的 TCP 協定時，便難免要和 Reno 共存，TCP Vegas 就是因為和 Reno 等較具侵略性的協定共存時會被排擠，無法維持其效能的穩定而不被採用。這個實驗就是要測試 Hop-by-Hop TCP 和 Reno 協定共存時的效能表現。

4.4.3.1 實驗環境

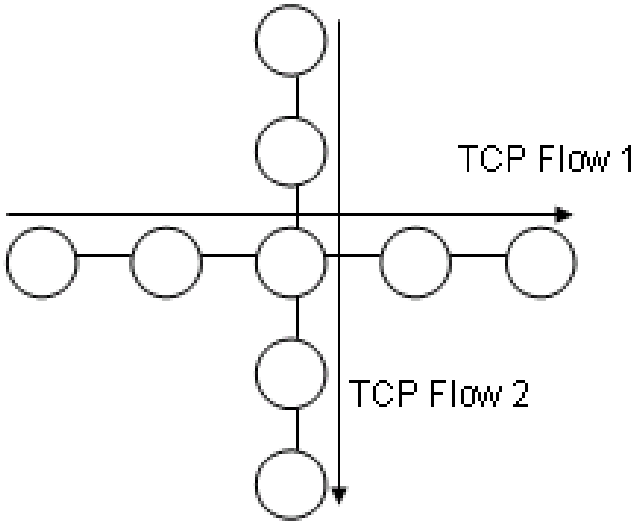


圖4.12：4-hop Cross Topology with 9 Nodes and 2 TCP flows

圖 4.12 為實驗 2A 的十字型拓樸，鏈結數分別為 4, 6, 8，在這個實驗中我們建立兩個互相交叉的 TCP flow，分別使用不同的 TCP protocol，其中一條固定為 NewReno，另一條則為 Vegas 或 Hop-by-Hop TCP，模擬時間為 50 秒，每個鏈結間的頻寬是 1Mbps，路由器佇列的管理機制為 DropTail，節點的傳輸半徑為 250 公尺，MAC Protocol 為 802.11，路由協定為 DSR，封包大小設定為 1000 bytes。實驗結果及公平性如圖 4.13 到圖 4.16 所示。

表4.2：實驗2A參數

Parameter	Range
Buffer Size	50
Error Rate	0.1
Link Bandwidth	1 Mbps
Number of hops	4, 6, 8
MAC Protocol	802.11
Routing Algorithms	DSR

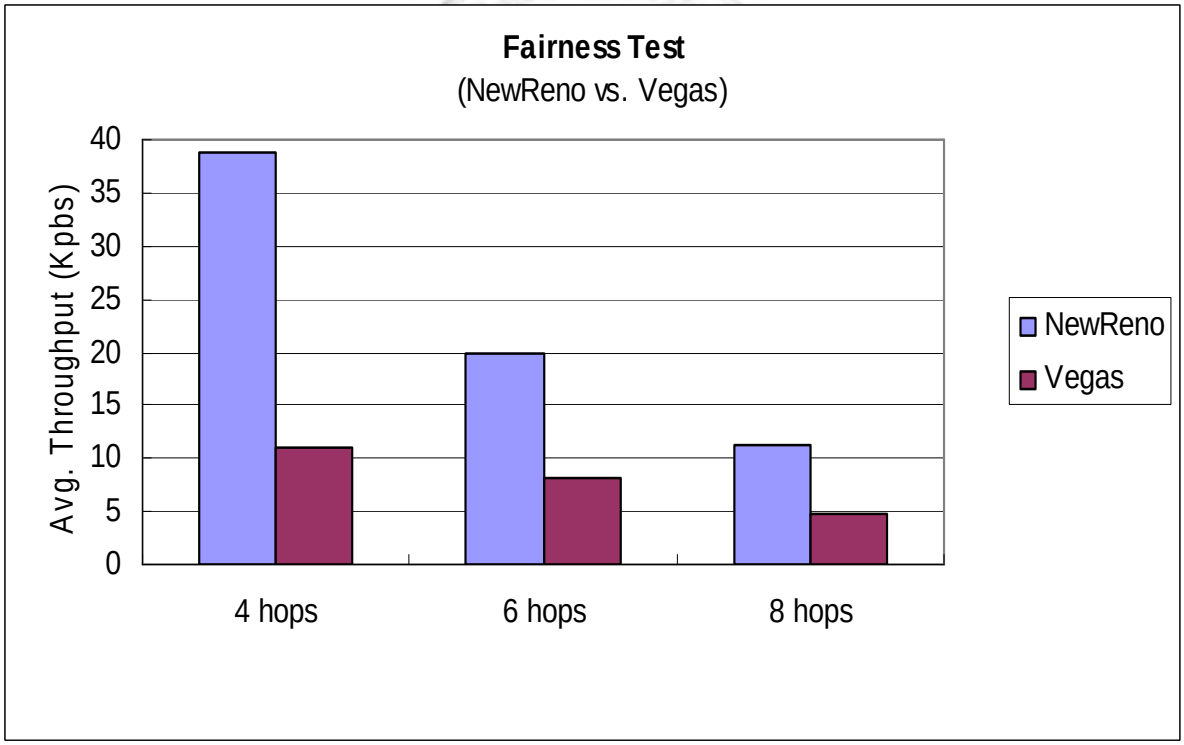


圖4.13：Throughput in Fairness Test 2A at diff. num. of hops (TCP NewReno vs. Vegas)

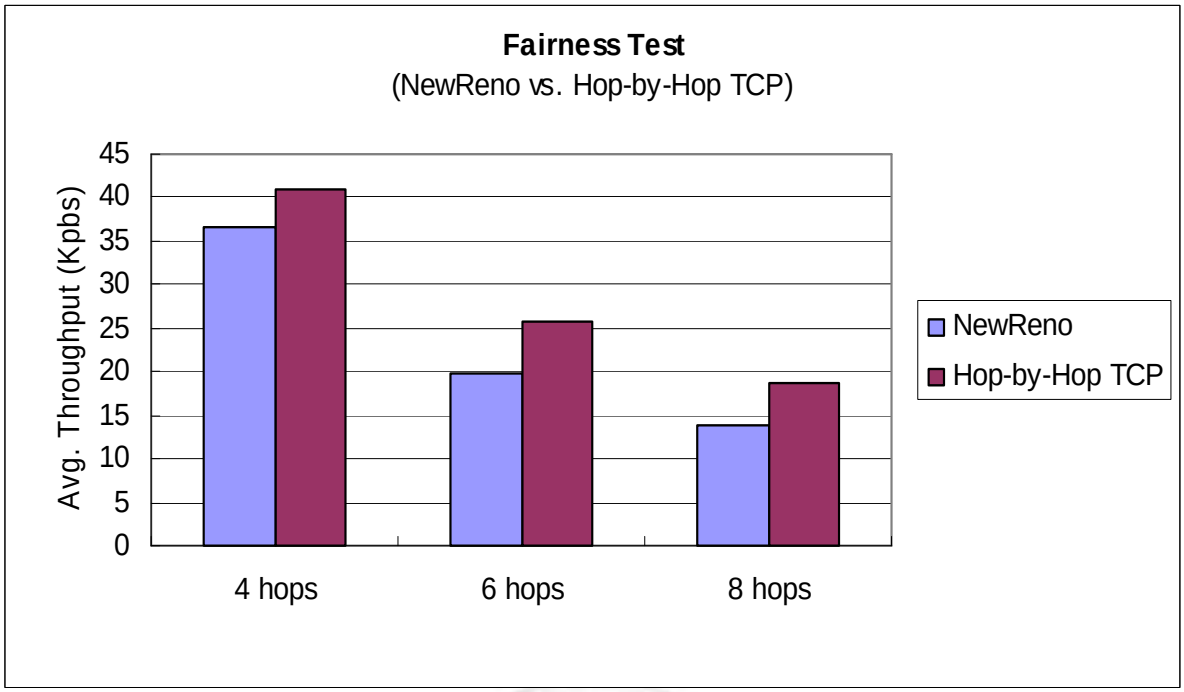


圖4.14：Throughput in Fairness Test 2A at diff. num. of hops (TCP NewReno vs. Hop-by-Hop TCP)

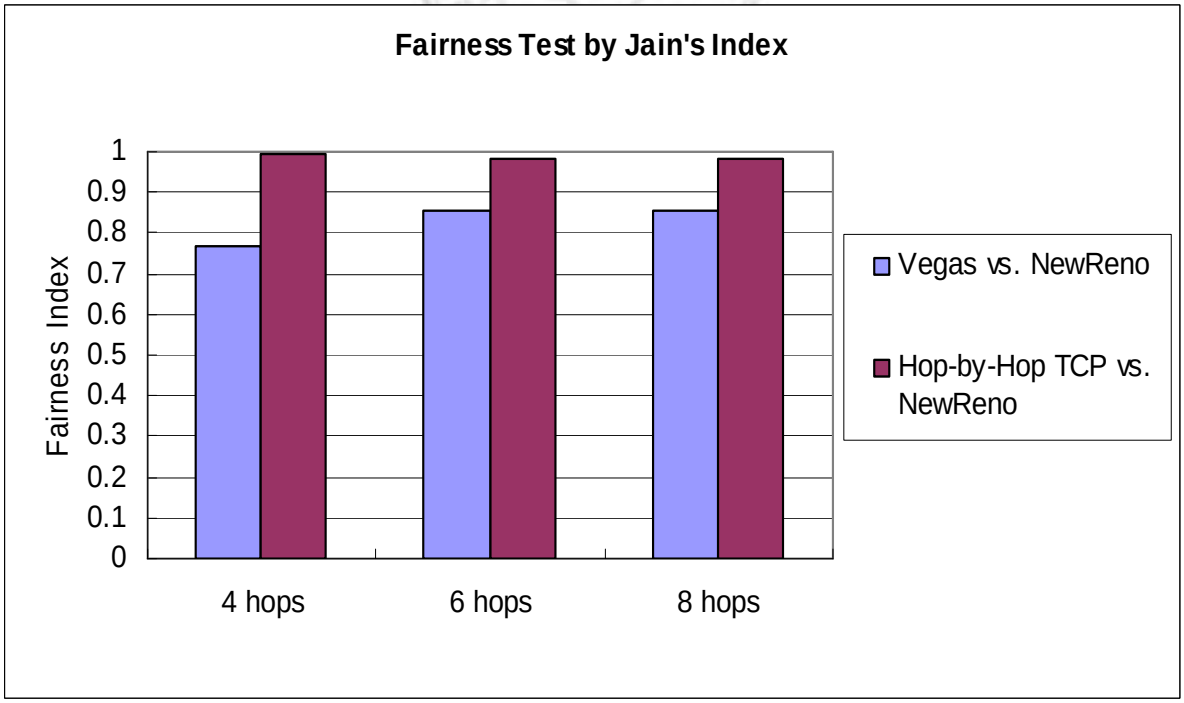


圖4.15：Comparison of Fairness Index in Fairness Test 2A

4.4.3.2 實驗結果分析

從圖 4.13、4.14 可以看出在不穩定環境中，Vegas 效能因為頻寬被 NewReno 搶奪而更為低落，Hop-by-Hop TCP 由於承襲了 Reno 的許多傳送機制，雖然效能也有下降，卻仍然能保有穩定的效能，圖 4.15 中顯示，在和 NewReno 此種高侵略性的擁塞控制協定共存時，Hop-by-Hop TCP 不致於像 Vegas 那樣被排擠，且能擁有更高的公平性，即使網路環境不穩定，也能有好的表現。

4.4.4 實驗 2B：TCP 同步化的實驗

4.4.4.1 實驗目標

使用相同 TCP 協定的不同 TCP flow 在不同時間進入網路時，擁塞控制機制也會影響這些 flow 的頻寬分配。網路營運者常希望即使進入的時間不同，所有 TCP Session 應該要能公平的使用頻寬，本實驗想觀察這個現象。

4.4.4.2 實驗步驟

在實驗 2B 中，我們建立一個 4-hop 單線式拓樸的網路，三個 flow，分別在 0 秒，10 秒，20 秒時進入網路，每個鏈結間的頻寬是 1Mbps，路由器佇列的管理機制為 DropTail，節點的傳輸半徑為 250 公尺，MAC Protocol 為 802.11，路由協定為 DSR，封包大小設定為 1000 bytes。如圖 4.16、圖 4.18、圖 4.20 所示，NewReno 的擁塞視窗變化非常大、Vegas 則是較為平穩而保守，雖然最終都可達到公平狀態，但耗時較長；而 Hop-by-Hop TCP 中各 flow 都可以很快地到達我們所預設之上限值，這是因為我們的方式確保封包能一站一站到達傳送，每個 flow 的機會較公平，因此擁塞視窗大小能維持平穩。

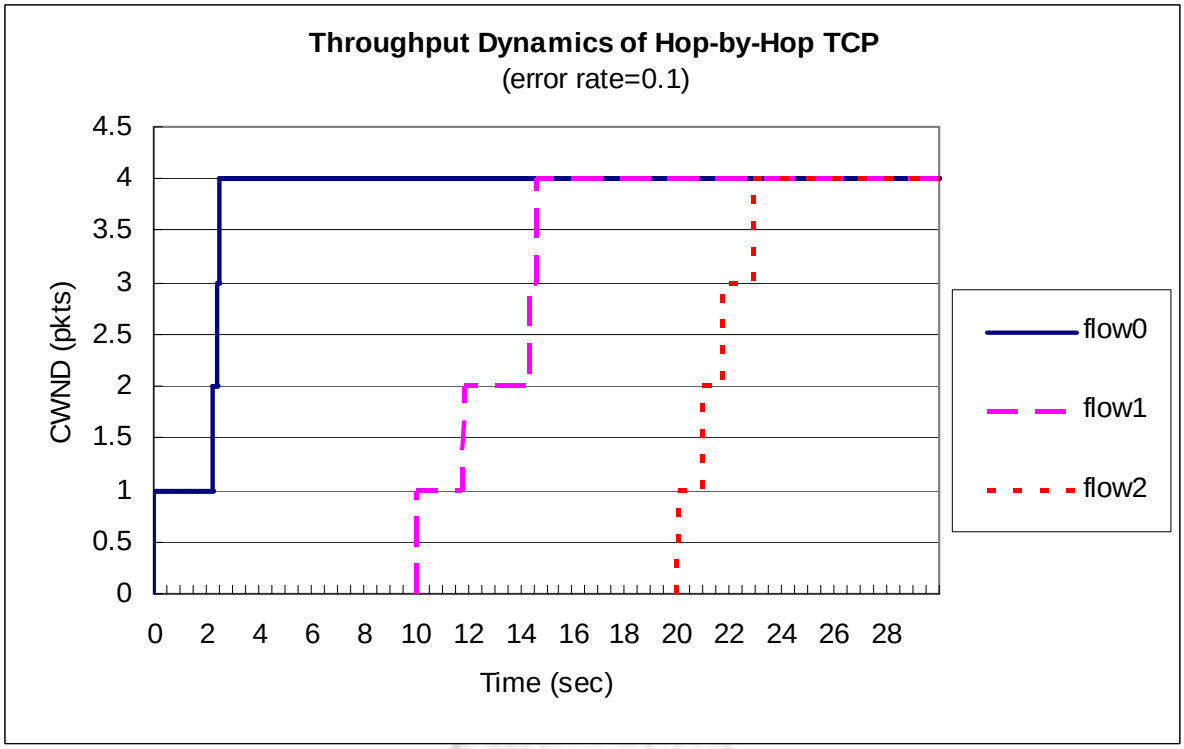


圖4.16：Throughout Dynamics in Fairness Test 2B at diff. num. of hops (Hop-by-Hop TCP)

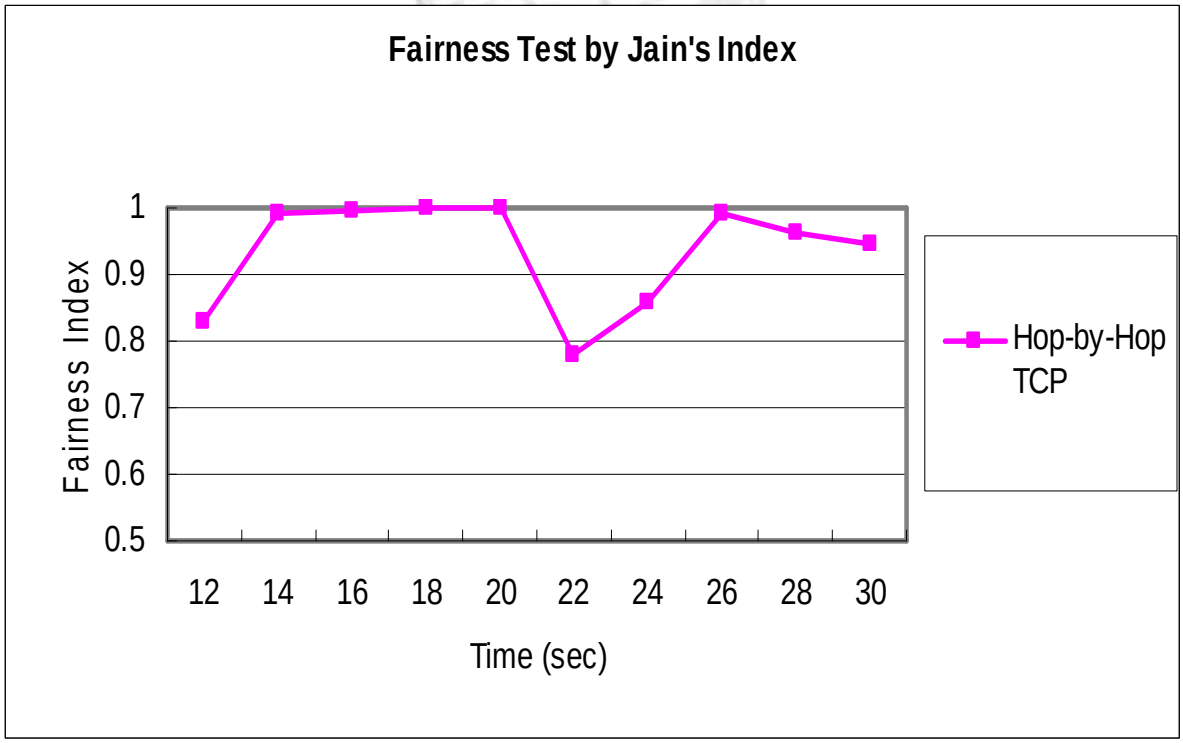


圖4.17：Fairness Index Dynamics in Fairness Test 2B (Hop-by-Hop TCP)

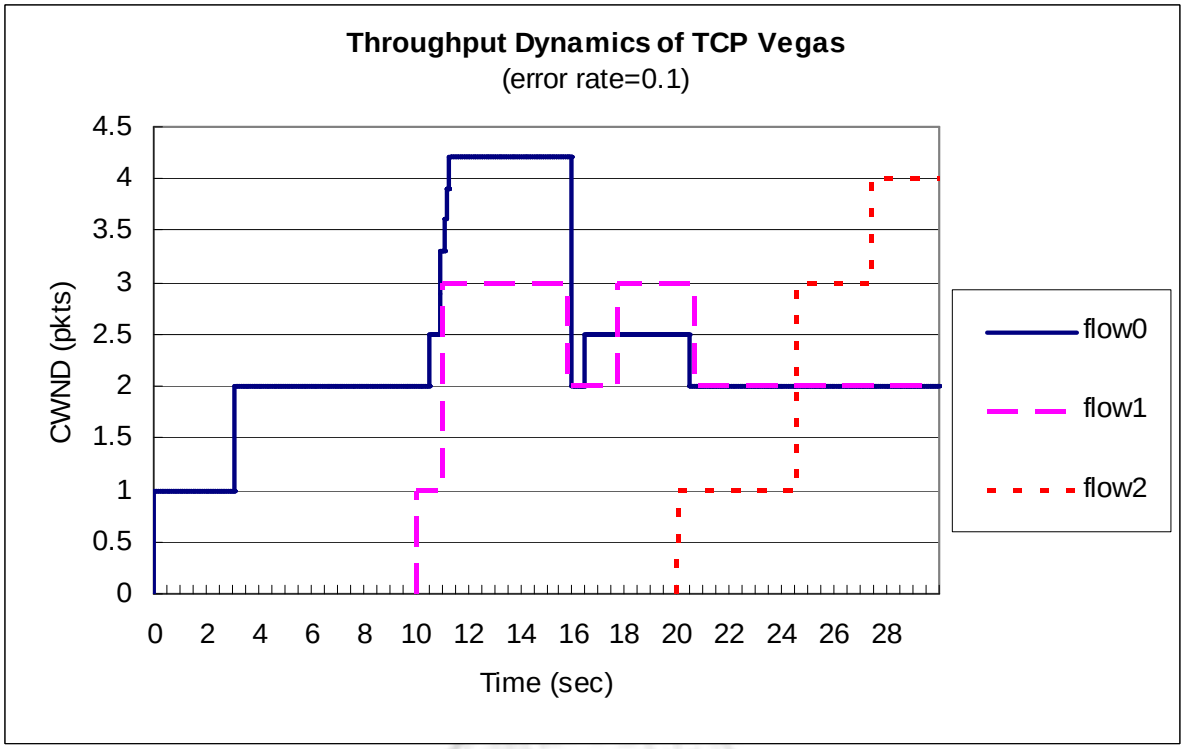


圖4.18：Throughout Dynamics in Fairness Test 2B (TCP Vegas)

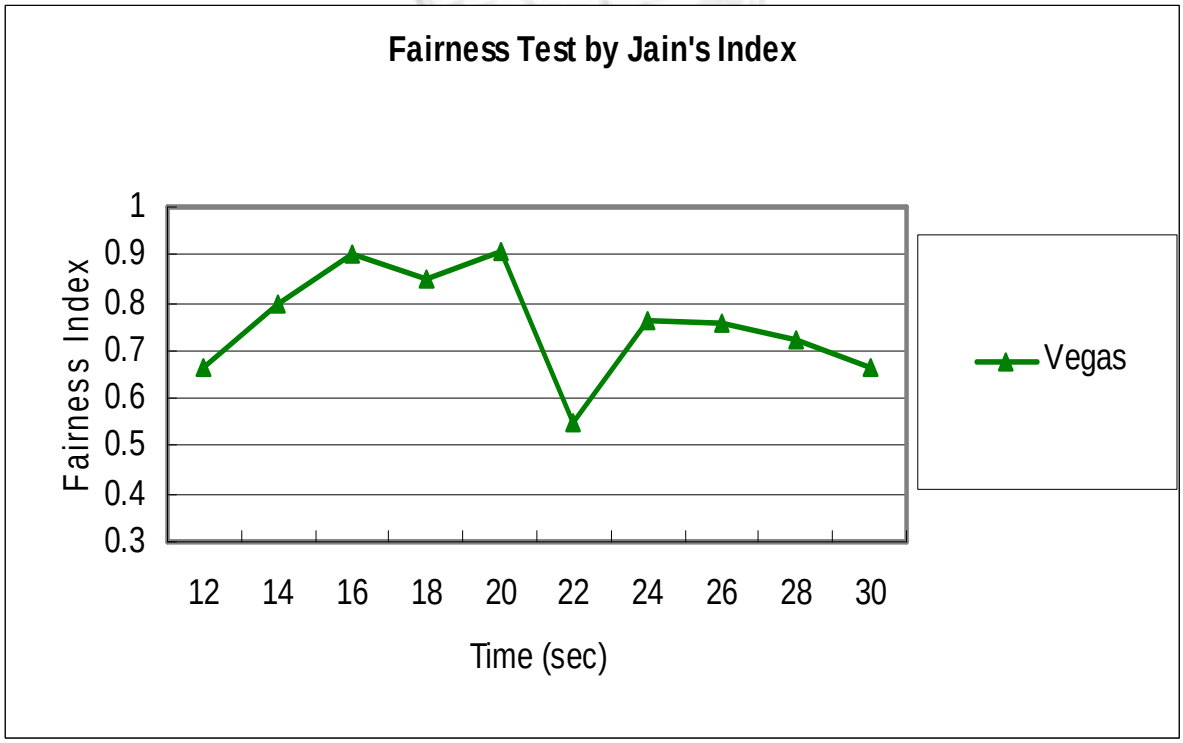


圖4.19：Fairness Index Dynamics in Fairness Test 2B (TCP Vegas)

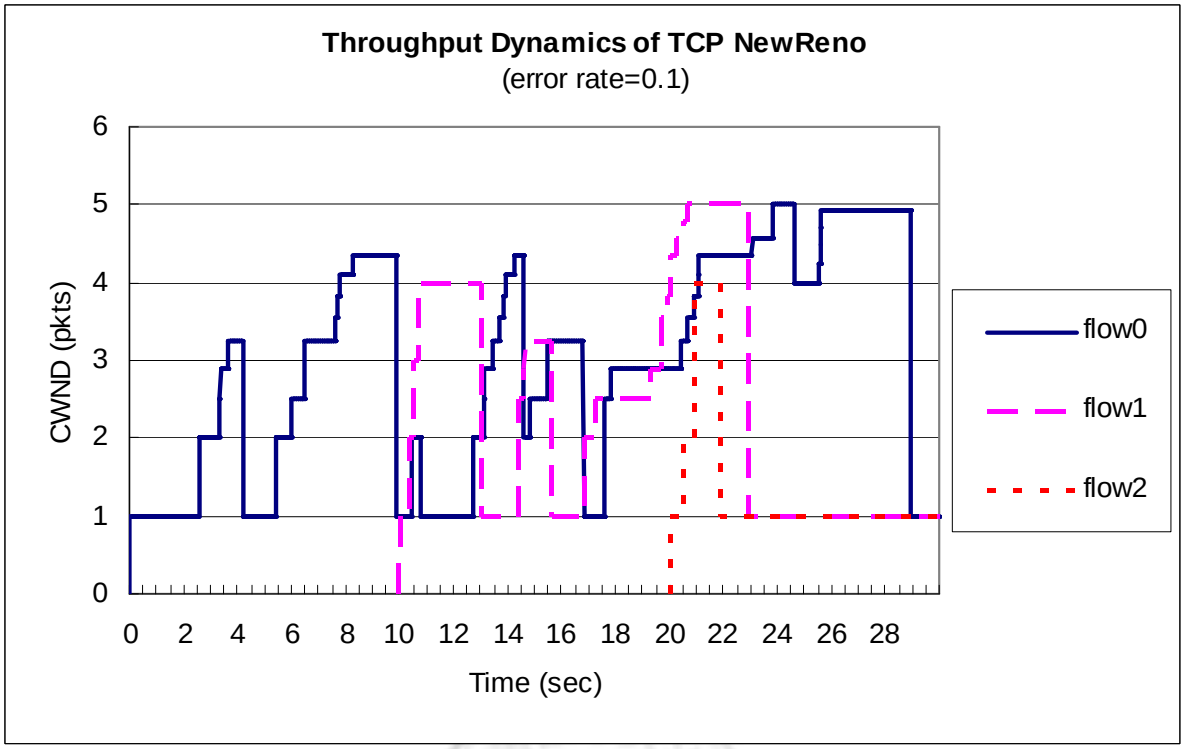


圖4.20：Throughout Dynamics in Fairness Test 2B (TCP NewReno)

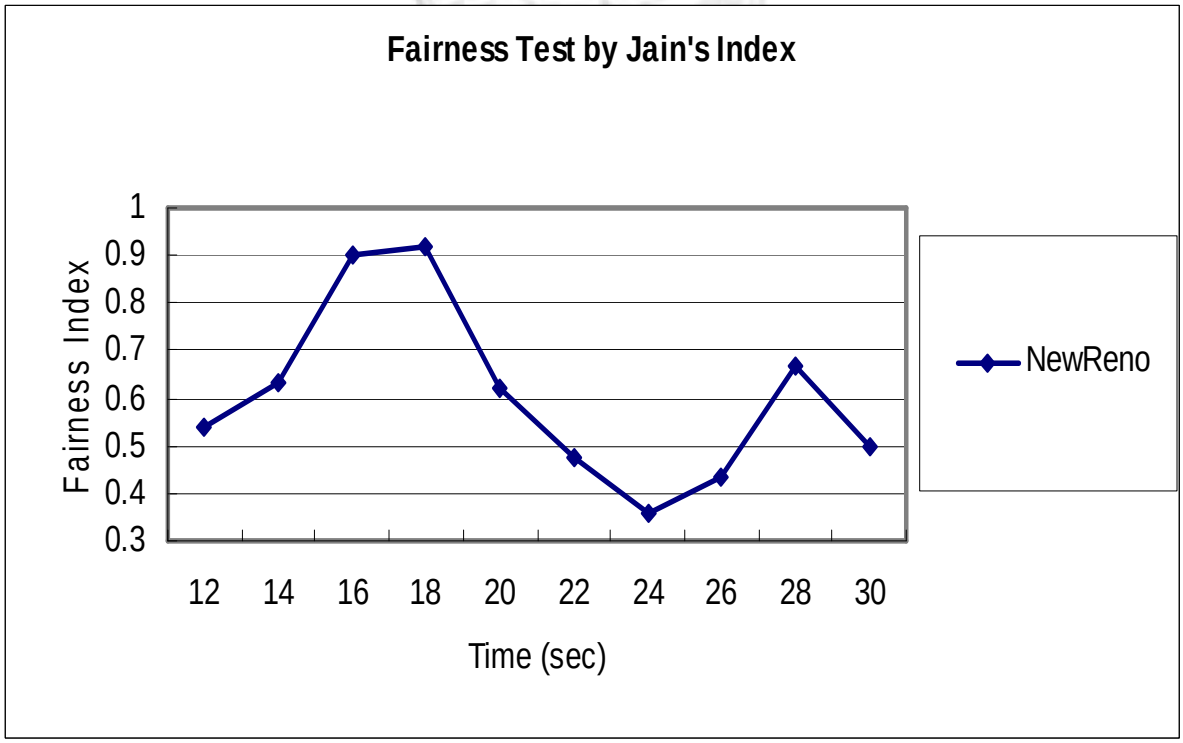


圖4.21：Fairness Index Dynamics in Fairness Test 2B (TCP NewReno)

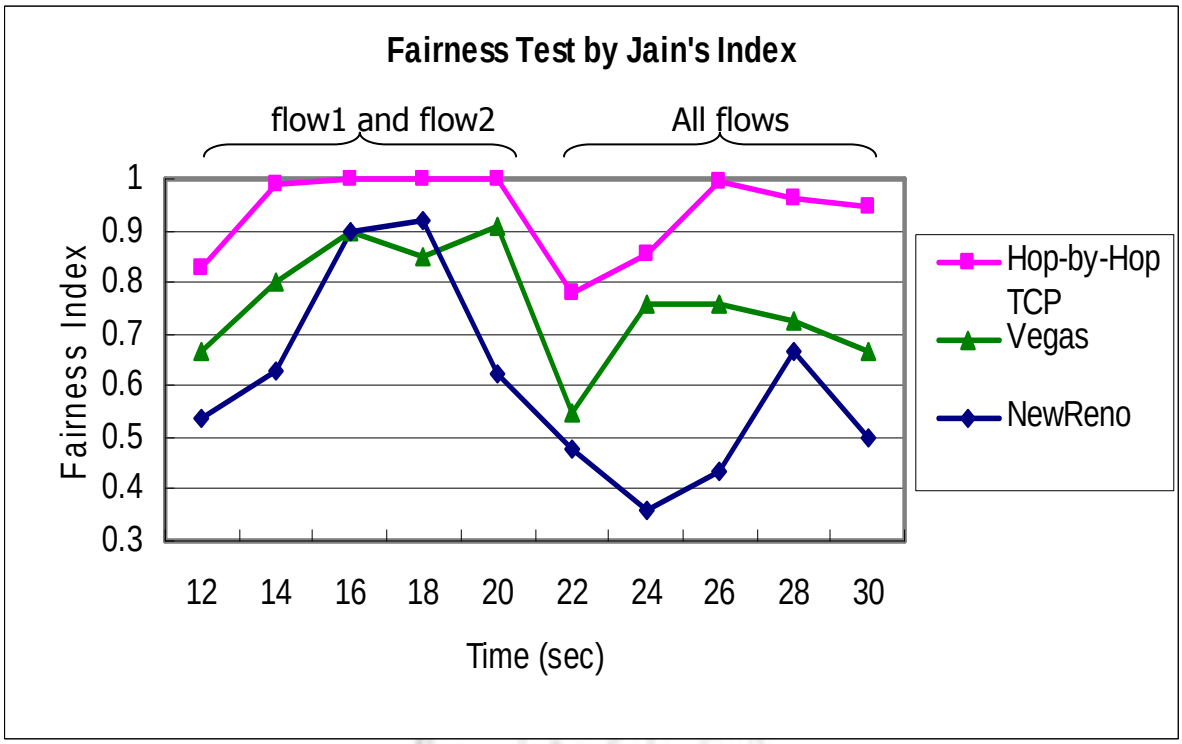


圖4.22：Comparison of Fairness Index in Fairness Test 2B (Hop-by-Hop TCP vs. Vegas vs. NewReno)

4.4.4.3 實驗結果分析

從圖4.16~4.22可以看出，Hop-by-Hop TCP在同時存在多條flow時，公平性的表現優於NewReno及Vegas。NewReno及Vegas的Fairness Index在有新的flow進入時(10sec~12sec，20sec~22sec)大幅降低，而Hop-by-Hop TCP的Fairness Index僅稍微降低。這是因為NewReno較具侵略性的擁塞視窗控制策略，使得先到的flow先搶得較多頻寬，因此會讓較晚進入的flow在先前的flow進入Slow Start時才有機會爭取到頻寬，因此造成不公平的情形；Vegas則因為採取較保守的頻寬競爭策略，較晚到的flow不容易搶到頻寬，使得公平性亦較差；而Hop-by-Hop TCP的每個flow則不論先來後到，在每一站的

CWND皆為1，當其中一個flow在等待下一站回覆的Local ACK時，其他flow也能夠傳送封包，較晚到的flow不會因為被先到的flow佔滿了頻寬而難以競爭，因此Hop-by-Hop TCP多個flow共存的公平性較佳。

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 Hop-by-Hop TCP 的可行性。在本實驗中，網路環境不穩定時，Hop-by-Hop TCP 可以保有傳輸的 average throughput，跟 NewReno 相比，至少高了 25.7%的效能，並且擁有較短的傳輸延遲時間，能夠縮短 25%以上的傳輸延遲時間。在封包容易遺失的環境中，可以有效提升整體效能，這是因為傳統的 TCP (NewReno、Vegas、SACK)只能從傳送端重傳遺失的封包，而行動隨意式網路不穩定的環境中，常常重送多次才可送達接收端，使得封包到達目的地的時間拉長。為了使封包較快送至接收端，我們提出的 Hop-by-Hop TCP 方法，使每個節點使用當地重傳以保證封包成功的傳到下一個節點，當封包遺失時可以重當地端直接重傳，不必重新由傳送端重傳，因此能更快反應封包遺失，使封包在高遺失率的情形之下能順利且較為快速的送達目的地端，並使 average throughput 也獲得了提升。

在多協定網路下，與 NewReno 共存的環境中，Hop-by-Hop TCP 則保有穩定的效能，不會因為 NewReno 的侵略性傳輸而受到排擠，相對於 Vegas 大幅度的效能下降，改善很多。

使用中間節點幫忙傳輸的想法及細節還有很多可以改進的地方，例如與路由協定的結合、封包走不同路徑造成接收端收到亂序的封包、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，是未來可以研究的方向。

參考文獻

[1] "IEEE Standards for Information Technology- Telecommunications and Information Exchange between Systems- Local and Metropolitan Area Network- Specific Requirements- Part 11: Wireless LAN Medium Access Control and Physical Layer Specifications," *ANSI / IEEE Std 802. 11* , 1999.

[2] 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.

[3] E. Altman, C. Barakat, and E. Laborde, "Fairness Analysis of TCP/IP," *Proc. of IEEE Conference on Decision and Control*, pp. 61-66, Dec. 2000.

[4] 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.

[5] 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.

[6] 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.

[7] K. Chandran, S. Raghunathan, *S.* Venkatesan, and R. Prakash, "A Feedback Based Scheme For Improving TCP Performance In Ad-Hoc Wireless

Networks", *IEEE ICDCS*, vol. 8, no. 1, pp. 34-39, Feb. 2001.

[8] K. Chen, Y. Xue, and K. Nahrstedt, "On setting TCP's congestion window limit in mobile ad hoc networks," *Proc. IEEE ICC 2003*, Anchorage, Alaska, May. 2003.

[9] 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.

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

[11] R. Denda, A. Banchs, and W. Effelsberg, "The Fairness Challenge in Computer Networks," *Lecture Notes in Computer Science*, vol. 1922, pp. 208-220, Jun. 2000.

[12] T. Dyer and R. Boppana, "A comparison of TCP performance over three routing protocols for mobile ad hoc networks," *Proc. of the 2nd ACM Int'l Symp*, pp. 56−66, 2001.

[13] 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.

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

[15] Z. Fu, P. Zerfos, H. Luo, S. Lu, L. Zhang, and M. Gerla, "The impact of multihop wireless channel on TCP throughput and loss," *Proc. IEEE INFOCOM*, Mar. 2003.

[16] M. Gerla, K. Tang, and R. Bagrodia, “TCP performance in wireless multi-hop networks,” *Proc. of the IEEE WMCSA*, pp. 25-26, Feb. 1999.

[17] 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.

[18] S. Heimlicher, R. Baumann, M. Martin, and B. Plattner, “The Transport Layer Revisited,” *Proc. of 2nd IEEE International Conference on Communication Systems Software and Middleware*, 2007.

[19] G. Holland and N. Vaidya, “Impact of routing and link layers on TCP performance in mobile ad hoc networks,” *IEEE Wireless Communications and Networking*, vol. 3, pp. 1323-1327, 1999.

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

[21] R. Jain, D-M. Chiu, and W. Hawe, “A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Computer Systems,” *Technical Report, DEC Research Report TR-301*, Sept. 1984.

[22] D. Kim, C. Toh, and Y. Choi, “TCP-BuS: Improving TCP performance in wireless ad hoc networks,” *Journal of Communications and Networks*, vol. 3, no. 2, pp. 175–186, Jun. 2001.

[23] S. Kopparty, S. Krishnamurthy, M. Faloutous, and S. Tripathi, “Split TCP for mobile ad hoc networks,” *Proc. of IEEE GLOBECOM*, Nov. 2002.

[24] Yao-Nan Lien and Ho-Cheng Hsiao, “A New TCP Congestion Control Mechanism over Wireless Ad Hoc Networks by Router-Assisted Approach,” *Proc. of IEEE Workshop on Specialized Ad Hoc Networks and*

Systems, Jun. 2007.

[25] J. Liu and S.Singh, “ATCP: TCP for Mobile Ad Hoc Network,” *IEEE Journal on Selective Areas of Communication*, vol19, no.7, July. 2001.

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

[27] J. Postel, “Transmission Control Protocol,” *IETF RFC 793*, 1981.

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

[29] S. Xu and T. Saadawi, “Does the IEEE 802.11 MAC protocol work well in multihop wireless ad hoc networks?” *IEEE Communication Magazine*, vol. 39, no.6, P130-137, Jun. 2001.