東海大學資訊工程研究所

碩士論文

指導教授：朱延平　博士

基於高頻寬延遲乘積網路中 TCP Vegas 效能改進

An enhanced performance of TCP Vegas based on high bandwidth-delay product network

研究生：羅勝暉

中華民國一○ 一年七月

東海大學碩士學位論文考試審定書

東海大學資訊工程學系 研究所

研究生 羅 勝 暉 所提之論文

基於高頻寬延遲乘積網路中 TCP Vegas 效能改進

經本委員會審查，符合碩士學位論文標準。

學位考試委員會
召 集 人 陳錦杏 簽章

委 員 朱延平
林冠成
許元誠

指 導 教 授 朱延平 簽章

中華民國 101 年 6 月 25 日

摘要

隨著網路技術的演進，網路頻寬與封包的傳輸距離均有所大幅度的增長。網路環境逐漸演進成高頻寬延遲乘積的環境。在高頻寬延遲乘積網路環境中，適用於低頻寬的 TCP 協定，已經無法勝任當下的網路環境。因此關於 TCP 在高頻寬延遲乘積網路環境的傳輸效能，已經成為一個研究重點。

雖然 TCP Vegas 在穩定性、頻寬利用率與吞吐量方面均擁有較佳的表現。但是應用在高頻寬延遲乘積網路環境中， TCP Vegas 卻有著提早離開緩啟動階段的問題與壅塞視窗調整速度太慢的問題。上述的問題將嚴重降低 TCP Vegas 掌握網路頻寬的能力與吞吐量。

因此本文則提出一個基於 TCP Vegas 的改進機制，High Bandwidth-Delay Product Vegas（HBDP Vegas）。

HBDP Vegas 在評估壅塞程度的方法部分，則是提出往返時間與最小往返時間的比例值作為衡量壅塞程度的依據。在緩啟動階段部分，HBDP Vegas 藉由修改緩啟動階段中壅塞視窗的成長行為，來加強掌握網路頻寬的能力。同時在壅塞避免階段部分，也藉由動態調整 TCP Vegas 的 α 與 β 值與修改壅塞視窗的控制方式，來強化適應網路環境變化的能力。

最後經觀察 HBDP Vegas 的緩啟動階段之表現、收斂時間、和其他連線競爭情形與公平性指數，均證實 HBDP Vegas 在掌握網路頻寬的能力、適應網路環境變化的速度與吞吐量均有所改進。

關鍵詞：頻寬延遲乘積、壅塞控制、TCP Vegas、往返時間、緩啟動、壅塞避免

Abstract

With the evolution of network technology, the network bandwidth and packet transmission distance have been greatly increased. The network environment has gradually evolved into high bandwidth-delay product environment. Low bandwidth TCP protocol has been unable to support the current network environment in high-bandwidth delay product network environment. Therefore, how to enhance TCP transmission performance with high bandwidth-delay product network environment has become a research focus now.

Although TCP Vegas has better performance in stability, bandwidth utilization and throughput, it has the problems of early leaving in slow-start phase and slow adjusting speed in congestion window in high-bandwidth delay product network environment. This phenomenon will severely reduce the throughput of TCP Vegas and its ability of grasping network bandwidth.

In this paper, we proposed a new enhanced mechanism based on TCP Vegas, High Bandwidth-Delay Product Vegas（HBDP Vegas）.

HBDP Vegas assessed the degree of congestion by the ratio of round-trip time and minimum round-trip time. In slow-start phase, HBDP Vegas modified the growth of behaviors of congestion window to enhance the ability of grasping network bandwidth. In congestion avoidance phase, HBDP Vegas dynamically adjusted the value of α and β of TCP Vegas and modified the control method of congestion window to strengthen the ability of adapting to the fast change of network environments.

In conclusion, simulation results showed that in slow-start phase, convergence time, competitive situation with other connections and fairness index, HBDP Vegas effectively improved the ability of grasping bandwidth, the response speed of adapting to the change of network environments, and the throughput.

Keywords： Bandwidth Delay Product、Congestion control、TCP Vegas、Round Trip Time、Slow-Start、Congestion Avoidance

誌謝

隨著時間的推移，我在東海大學的求學生活也暫時告一段落。在離開東海前我經常一直在回想第一天踏上東海大學的校園的的記憶，以及在求學過程中遇到的人、事、物。雖然不可能再次回到過去，但是在求學的過程中我仍得到不少無價的知識與經驗。

首先我要感謝我的家人，我的父母給了我一個避風港，即一個可以遮風避雨的家，同時也給了我一個可靠的支援，讓我航行在一個名為東海的汪洋中沒有後顧之憂。

我必須要感謝我的指導教授朱延平博士，在我的求學過程中除了給我知識之外，也給了我做人處事的道理，同時老師在我的求學過程中猶如航海過程中的羅盤，告訴我正確的方向，不讓我迷航於東海。

我也要感謝陳錦杏教授與林冠成教授和許玟斌教授在百忙之中仍然抽空去審查與指正論文中的錯誤，讓我能更堅定的航行在正確的方向。

同時我必須感謝昭佑學長在我碩一學習的期間，幫我建立在往後學習過程中與探索未知領域過程中的基礎。同時也要感謝鑫一學長，在撰寫論文的過程中幫我檢查論文的內容與教我撰寫論文的技巧，有了學長的幫忙則幫我度過了在航行過程中的礁石。

雖然我成功的度過名為東海的大海，但這只是其中一片汪洋，我還必須再度航行於一個名為未來的汪洋，故最後引用一句話，「讓我們出發吧，大海正等著我們。大海和巨浪，正在呼喚著我們。」

目錄

圖目錄

表格目錄

第一章　序論

1.1. 簡介

近年來隨著科技的演進，人與人之間的溝通方式已從原始的書信演進到傳統的電話網路，接著再從電話網路的基礎上發展了網際網路。迄今網際網路的相關技術仍在發展之中。故由此可見，網際網路是人類歷史上的重大發明之一。

綜觀網際網路發展的歷史，其一切的開端則可以回溯到美國國防部的國防高等研究計劃署（Defense Advanced Research Project Agency，DARPA）在 1960 年後期所進行的研究項目。其中，關於網路項目的開發成果，美國高等研究規劃網路（Advanced Research Project Agency NETwork，ARPANET）可以是現今網路系統的原形。接著於 1970 年，開發了 ARPANET 所使用的通訊協定，網路控制協定（Network Control Protocol，NCP），此通訊協定即是後來的傳輸控制協議（Transmission Control Protocol，TCP）的雛形。但是，因為 NCP 自身機制的缺陷，造成傳輸效能的瓶頸，因此在 1973 年，Bob Kahn 著手開始發展 TCP 協定，並與 DARPA、史丹佛大學、及 Vinton Cerf 接手繼續開發。1974 年，TCP 已經完成了大部分的研發，並以 TCP / IP 的名稱浮上世界檯面。時至 1980 年，美國國防部確立 TCP / IP 為主要的傳輸協定，並

於 1983 停止 NCP 的使用，同年 DARPA 資助柏克萊大學把 TCP / IP 整合到 UNIX 上，最終使得 TCP / IP 成為公訂的標準。1995 年，網際網路服務提供商（Internet Service Provider，ISP）開始向個人和企業提供網路服務。綜觀上述關於網際網路的發展過程，除了發現網路已經逐漸融入人們的生活中之外，同時 TCP 在網路的發展史中則有著舉足輕重的地位。

1.2. 研究動機

現今的網路環境已經演進至具有高頻寬與高往返時間兩種特性。因此，具有以上兩種特性的網路環境，即是所謂的高頻寬延遲乘積（Bandwidth-Delay Product，BDP）網路。

由於此類型的網路環境漸漸已成為當下十分普遍的網路環境，所以導致目前最普遍的 TCP / IP 傳輸層協定 TCP Reno，已經在相關的研究中證實，在高頻寬延遲乘積網路環境中，其傳輸性能並不理想【21】。

TCP Reno【14】【20】【22】的壅塞控制的方式是通過觀測封包是否有遺失來評估目前網路壅塞狀況及估計可用頻寬，來調整壅塞視窗（Congestion window，CWND），但是此種壅塞控制的方式在運作過程中會造成壅塞視窗產生周期性的起伏的現象。具有此現象的壅塞控制法若應用在高頻寬延遲乘積網路中，則會因為高往返時間使得壅塞

視窗調節速度過於緩慢使得傳輸效率下降。

另一種有別於 TCP Reno 的壅塞控制的方式且同時是近年來發展起來的 TCP Vegas【9】【13】，則是一種依據連線的往返時間來衡量網路狀況與控制壅塞視窗大小的壅塞控制機制。在相關的研究指出，TCP Vegas 比 TCP Reno 擁有更好的頻寬利用率，且吞吐量也更大更穩定【9】【11】【13】。

但是 TCP Vegas 在高頻寬延遲乘積網路環境下的運作仍然有一些問題。在緩啟動階段中，則有著因緩啟動階段結束時當下的壅塞視窗大小過小，導致在進入壅塞避免階段之後的一段時間內，壅塞視窗會呈現線性成長，造成必須花費大量時間才能達到對網路頻寬的完全利用之現象，此現象即是由所謂的提前離開緩啟動階段問題所引起的。在壅塞避免階段中，原本 TCP Vegas 的設計會使得壅塞視窗的調整速度過於緩慢，此狀況會嚴重降低 TCP Vegas 的傳輸效率與吞吐量。

1.3. 論文貢獻

本文將針對 1.2 節所提到關於 TCP Vegas 在高頻寬延遲乘積網路環境下的緩啟動階段與壅塞避免階段的問題，提出了一個基於 TCP Vegas 且可以應用在高頻寬延遲乘積網路環境中的改良機制來解決以上的問題。

我們提出的改良機制可分成兩個部分，分別是衡量網路壅塞的程度與壅塞控制機制兩個部分。

在衡量網路壅塞的程度方面，我們則是提出藉由計算出最小往返時間（bass Round Trip Time，bass RTT）與傳輸過程中所量測的往返時間（Round Trip Time，RTT）的比例值，並藉由此比例值來衡量網路壅塞的程度。同時，我們也會將新的衡量網路壅塞程度的方法運用在改良的壅塞控制機制上面。

在改進的壅塞控制機制中關於緩啟動階段部分，我們必須修改原始的 TCP Vegas 在緩啟動階段對於壅塞視窗的成長方式，來解決提前離開緩啟動階段的問題，同時也增強在緩啟動階段掌握頻寬的能力。

接著在改進的壅塞控制機制中關於壅塞避免階段部分，我們則是藉由修改原始的 TCP Vegas 對於壅塞視窗大小的控制方式與使用動態修改 α 與 β 值來取代原始 TCP Vegas 固定不變的 α 與 β 值的構想，來改善原始的 TCP Vegas 在壅塞避免階段中對於壅塞視窗的調整能力。

1.4. 章節簡介

本文總共有六個章節，接下來的章節中，本文的第二章將介紹 TCP 與 TCP 的壅塞控制技術與運作原理，以及其他學者所提出改良過的壅塞控制技術。第三章將說明 TCP Vegas 在高頻寬延遲乘積網路環

境下緩啟動階段與壅塞避免階段的問題。第四章將介紹一種以 TCP Vegas 為基礎的改進機制 HBDP Vegas （High Bandwidth-Delay Product Vegas，HBDP Vegas），此章節中將說明 HBDP Vegas 衡量網路壅塞程度的方法與壅塞控制機制。接下來，第五章將會利用網路模擬軟體 NS-2 來驗證在第四章中所提出的 HBDP Vegas，並檢視 HBDP Vegas 的運作過程。最後在第六章則是本論文的結論與未來的改進目標。

第二章　文獻探討

在本章節為針對本研究中所使用的參考文獻與使用技術的探討，首先將會簡單的介紹TCP，接著並說明TCP與壅塞控制的關聯，接下來則會開始介紹基本TCP版本的壅塞控制技術，即TCP Tahoe、TCP Reno與TCP Vegas所使用的壅塞控制技術，同時也依據所使用的壅塞控制技術做比較。接下來也會再介紹其它以TCP Vegas為基礎並在高頻寬延遲乘積網路環境上運作的TCP版本與其自身的壅塞控制機制，最後本章節也將會解釋為什麼要使用TCP Vegas作為本次研究中的基礎機制。

2.1. TCP

TCP是一種端點對端點的傳輸層協定。TCP在網路上的功能已在發表於1981年的RFC793【12】中已經規劃出來，在RFC793中總共規劃了六個功能，分別是：基本的資料傳輸（Basic data transfer）、可靠度（Reliability）、連結（Connections）、多工（Multiplexing）、流量控制（Flow control）、安全性（Precedence and security）。

TCP 在發送資料的流程方面，在開始發送資料之前，會基於 TCP 的連接導向（Connection-oriented）特性，會事先在發送端與接收端之間建立一個連結。當兩端的連結已經建立起來後則會開始進行發送

資料的工作，如圖一，首先發送端會將要發送的資料切割成封包，同時並將以已分割的封包賦予編號，最後則是把封包交給網路層並傳送到接收端。當接收端收到來自發送端的封包時，除了依照封包上的序號將收到的封包重組成為原來的資料之外，同時也會發送一個代表已收到封包的確訊號封包（Acknowledgement，ACK），此舉是為了告知發送端，封包已成功的被接收。因此，當發送端收到來自接收端發送的 ACK 封包後則會立刻再發送封包。

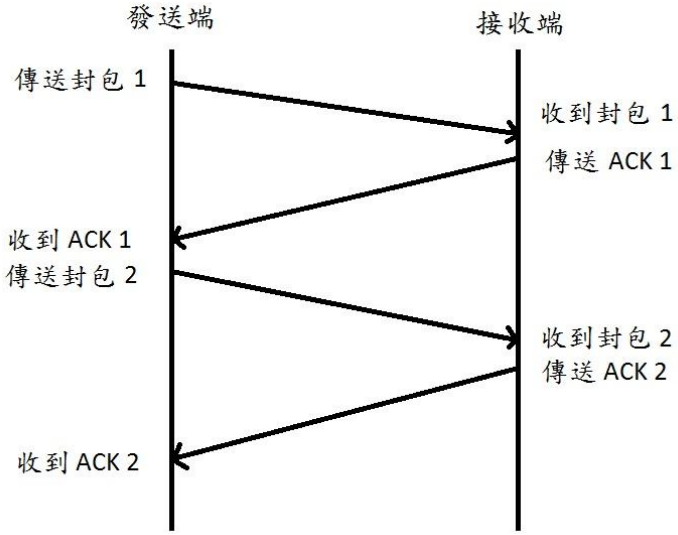


圖 一：TCP 運作示意圖

2.2. TCP 與壅塞控制

隨著網路技術逐漸成熟，人們在網路上傳輸的資料量逐年增加，雖然由2.1節中可以得知，TCP是目前主流的傳輸層通訊協定，它提供了具有可靠的傳輸服務。但是，觀察2.1節中RFC793所規劃的TCP的六

個功能中，很顯然得可以發現RFC793只有考量到接收端設備的接收資料的能力並沒有考慮到網路即將發生的新問題，壅塞現象。

所謂的壅塞現象是指當傳輸的資料量超過網路傳輸容量的負荷上限時，導致發生傳輸失敗的現象。若網路發生了壅塞現象，將會嚴重的影響網路傳輸的效能。因此，為了要避免發生壅塞現象，所以人們開發了壅塞控制機制，同時也將壅塞控制機制納入TCP的功能之中。因此目前的RFC2581【14】已經定義了基本TCP的壅塞控制機制於其中。

壅塞控制機制在2.1節中所提的TCP發送資料流程中扮演了關鍵的角色，壅塞控制機制的原理是評估網路壅塞狀況並透過控制壅塞視窗大小來管制每一次發送的封包數量，以達到控制資料的傳輸量與傳輸速度以避免發生壅塞現象。其中壅塞視窗類似滑動式窗，可以允許多個封包同時被發送，其中被發送出去的封包數量即是壅塞視窗大小，其運作過程可參考圖二。

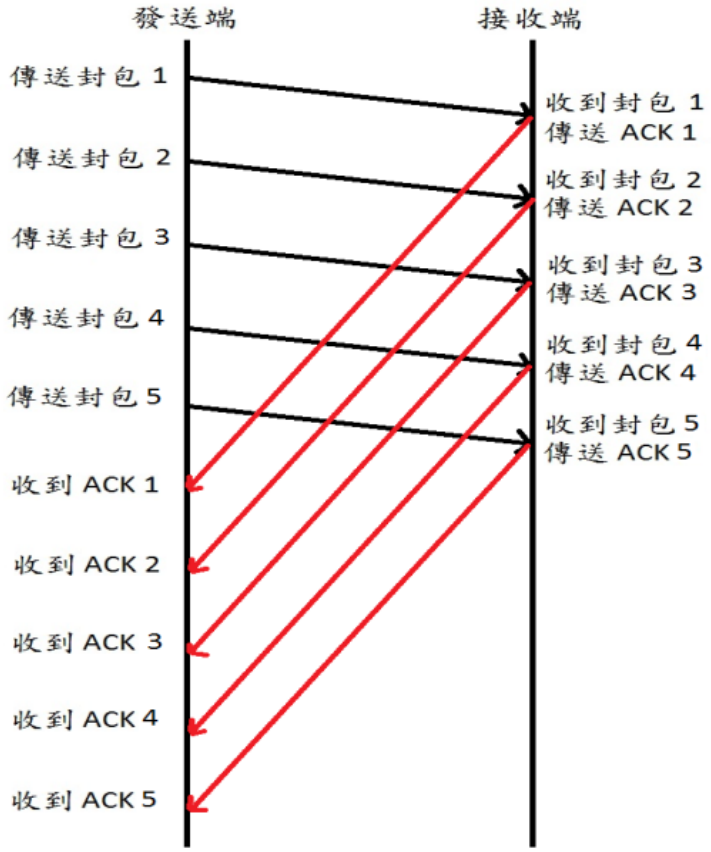


圖 二：壅塞視窗運作示意圖

然而由於判定壅塞現象是否發生的方式與壅塞控制機制運作方式的差異也產生了許多不同的TCP版本，因此接下來的小節中將會對其他的TCP衍生版本做介紹：

2.2.1. TCP Tahoe

TCP Tahoe【12】【19】判定壅塞現象的方式是藉由觀測是否有封包遺失來評估目前網路壅塞狀況。TCP Tahoe 的壅塞控制機制運作的方式是搭配三個階段來控制壅塞現象，分別為緩啟動階段、壅塞避免階段和快速重傳機制。

緩啟動階段（Slow-start phase）：當一個連結被建立起來之後，就進入緩啟動階段。首先發送端將初始化壅塞視窗為一個封包大小，接著就以此大小的壅塞視窗來發送資料。若接收端收到封包，會發送一個 ACK，若此 ACK 在被發送端判定是在發生逾時（timeout）以前到達發送端，則壅塞視窗的大小則增加為目前壅塞視窗大小的兩倍，接著再以此大小的壅塞視窗來發送資料。接下來收發雙方將重複上述動作，直到發生壅塞視窗超過門檻值（slow-start threshold，ssthresh）值。以上的過程被稱為緩啟動階段或是以壅塞視窗成長的型態稱為指數成長階段。

壅塞避免階段（Congestion avoidance phase）：當壅塞視窗超過 ssthresh 值之後，就進入壅塞避免階段中。在壅塞避免階段中，壅塞視窗的成長是以線性的方式來增加壅塞視窗，每收到一個 ACK 就將壅塞視窗增加（$\frac{1}{\text{cwnd}}$），直到發生 timeout 或是偵測到有封包遺失。若最後發生 timeout 時，ssthresh 值會被設定成發生 timeout 當下視窗大小的一半。此時壅塞視窗會恢復成初始大小，再次進入緩啟動階段。反之，若是偵測到有封包遺失時，則進入快速重傳機制。由於此階段的壅塞視窗的成長是以線性的成長方式，故壅塞避免階段又稱為線性成長階段。

快速重傳機制（Fast retransmit）：在上述壅塞避免階段中，若最後偵測到有封包遺失時，TCP Tahoe 就會認為網路發生壅塞了，並進入快速重傳機制。 TCP Tahoe 判定封包遺失的方式，是利用接收端所發送的重複的確認訊息（Duplicate ACK）當依據。當有發生封包於傳輸過程中遺失時，發送端會收到接收端所發送的 Duplicate ACK 。當發送端收到三個 Duplicate ACK ，發送端會將該封包視為遺失，不會等待 timeout ，直接立即重送此封包，並且將 ssthresh 設為偵測到有封包遺失時，當下視窗大小的一半，同時將壅塞視窗大小重新設置為初始大小之後，重新進入緩緩動階段。但是，若在壅塞避免階段中，最後是發生 timeout 則不會進入快速重傳機制，將依照上述壅塞避免階段對於發生 timeout 時的處理方式。

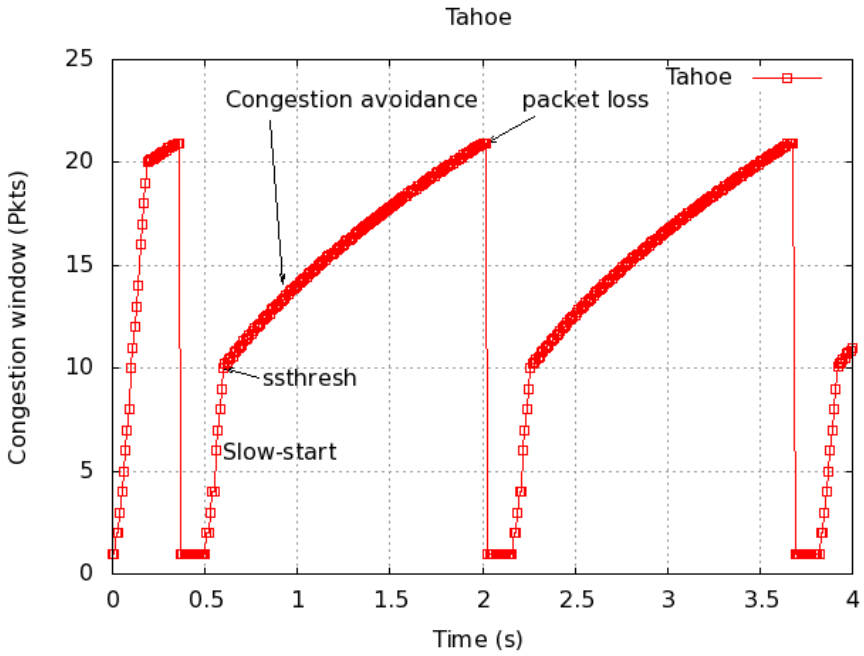


圖 三：TCP Tahoe 的運作

現在我們藉由圖三來觀察 TCP Tahoe 的運作。從連線一被建立起來時，先進入緩啟動階段，當壅塞視窗大小超過 ssthresh 值之後，進入壅塞避免階段，直到最後偵測到封包遺失，進入快速重傳機制，TCP Tahoe 除了重送遺失的封包外，並重新設定 ssthresh 值，同時將壅塞視窗降至為初始大小，接著又進入緩啟動階段。

2.2.2. TCP Reno

TCP Reno【14】【20】【22】在判定壅塞現象的原理與壅塞控制機制運作方式基本上和 TCP Tahoe 的運作方式是一樣的。但是 TCP Reno 在壅塞控制機制運作方式上多增加了一個新的機制為快速復原（Fast recovery）。

TCP Reno 在收到收到三次的 Duplicate ACK 後除了重傳遺失封包並將 ssthresh 值設成偵測到有封包遺失時，當下的視窗大小的一半。完成快速重傳機制後，會直接進入快速復原階段，直到接收到遺失封包的 ACK，才離開快速復原階段並直接進入壅塞避免階段，不再重新進入 Slow Start 階段。

同樣藉由圖四來觀察 TCP Reno 的運作，在進入快速重傳機制之後 TCP Reno 會重送遺失的封包與重新設定 ssthresh 值，與 TCP Tahoe 不一樣的是 TCP Reno 在結束快速重傳機制後，就緊接的進入快速復原，從圖四中我們可以發現，壅塞視窗只下降到當下在快速重傳機制所設定的新 ssthresh 值，接下來就直接進入壅塞避免階段。

圖 四：TCP Reno 的運作

我們比較圖三與圖四的差別，可以很明顯的發現到，兩者的壅塞視窗的震盪狀況有明顯差別，壅塞視窗可以反映 TCP 的傳輸效能，由於 TCP Reno 多了快速復原機制，此機制減輕了壅塞視窗的震盪幅度，相較於 TCP Tahoe，TCP Reno 的傳輸效能會比較高。

但是將 TCP Reno 應用在高頻寬延遲乘積網路環境時，則會有如下所述的兩個問題：

- 由於在高頻寬延遲乘積網路環境中，具有較高的往返時間，造成在固定時間內對壅塞視窗作調整的次數相對變少，導致無法有效利用網路頻寬。
- 若發生封包遺失的現象，從減半的壅塞視窗回復到最大的壅塞視窗來取得最高傳輸頻寬所需時間過久，此問題可藉由觀察圖四得知，由於 TCP Reno 有週期性的封包遺失情況，相對的，也表示 TCP Reno 無法有效利用網路頻寬的情況將會不斷地以週期性的方式出現。

2.2.3. TCP Vegas

TCP Vegas【9】【13】在判定壅塞現象的方式與壅塞控制機制運作的原理，有別於 TCP Tahoe 與 TCP Reno。TCP Vegas 在衡量網路壅塞程度方面則是藉由每一次所量測的 RTT 時間，計算實際傳輸率和預期傳輸率並計算出兩者的差值 Diff，如公式（1），接著利用 Diff 算出停留在網路佇列中的封包數量Δ，並利用Δ來衡量網路壅塞程度，如公式（2）。

$$\text{Diff} = \left(\text{Expected} - \text{Actual}\right) \quad (1)$$

$$\Delta = \left(\text{Expected} - \text{Actual}\right) \times \text{base RTT} \quad (2)$$

$$= \left(\frac{\text{CWND}}{\text{base RTT}} - \frac{\text{CWND}}{\text{RTT}}\right) \times \text{base RTT}$$

$$= \text{CWND} \times \left(\frac{\text{RTT} - \text{base RTT}}{\text{RTT}}\right) \quad (3)$$

RTT 為目前量測到的 RTT 時間，RTT 時間為封包從發送端發送出去的那一刻至收到由接收端所發送此封包的 ACK 之間的時間，其中這一段時間也包含了佇列延遲時間（封包停留在網路佇列的時間）與其它開銷的時間；base RTT 為在整個傳輸過程中最小的 RTT 時間，該值通常是建立連線後第一個被發送的封包的 RTT 時間；CWND 為壅塞視窗大小，即為我們發送的封包數；Expected 為期望的傳輸率（$\frac{\text{CWND}}{\text{base RTT}}$）；Actual 為實際的傳輸率（$\frac{\text{CWND}}{\text{RTT}}$）；Diff 為實際傳輸率和預期傳輸率兩者的差值，同時 Diff 值亦可做為可用頻寬的依據；Δ表

示目前停留在網路佇列中的封包數量，同時此公式亦可化簡至公式（3）。

TCP Vegas 的壅塞控制機制運作的方式是搭配緩啟動階段、壅塞避免階段與快速重傳這三個機制來實現壅塞控制，但是 TCP Vegas 在這三個機制中對壅塞視窗大小的增減與控制方式並不同先前的 TCP Tahoe 與 TCP Reno。因此在接下來的段落中將會說明關於 TCP Vegas 的壅塞控制機制。

緩啟動階段：TCP Vegas 在緩啟動階段中，除了要能迅速的掌握可用的網路頻寬之外，同時也必須要預防發送封包的速度過快導致封包遺失的現象發生，TCP Vegas 採用減緩壅塞視窗的成長速度，壅塞視窗會經過兩個 RTT 時間增加一倍。同時，在緩啟動階段中，TCP Vegas 會透過一個參數來決定是否要離開緩啟動階段，此參數為 γ。γ 代表可容許停留在網路佇列中的封包數量，TCP Vegas 則藉比較 γ 與Δ的大小，若Δ的值大於 γ（預設值為 1）則表式網路佇列中的封包數量已經超過 γ 的上限，即結束緩啟動階段，並將壅塞視窗減小（$\frac{1}{8}$），之後就進入壅塞避免階段。若Δ的值小於 γ 則持續留在緩啟動階段。判斷緩啟動階段是否結束規則如公式（4）所示：

$$\begin{cases} \Delta = CWND \times \left(\frac{RTT - base\ RTT}{RTT}\right) > \gamma, \text{離開緩啟動階段} \\ \Delta = CWND \times \left(\frac{RTT - base\ RTT}{RTT}\right) < \gamma, \text{留在緩啟動階段} \end{cases} \quad (4)$$

壅塞避免階段：在壅塞避免階段中一共用到兩個門檻值，α 與 β（預設值為 1、3），在每一次的 RTT 時間中將依照Δ落在由 α 與 β 兩個門檻值形成的三個控制區間中的位置，來調整壅塞視窗。壅塞視窗調整的規則如公式（5）所示：

$$\text{new CWND} = \begin{cases} \text{CWND} - 1, \Delta > \beta \\ \text{CWND} + 1, \Delta < \alpha \\ \text{CWND}, \alpha \leq \Delta \leq \beta \end{cases} \quad (5)$$

若Δ小於 α，表示網路現在並不壅塞且仍然有頻寬可使用，可以增加壅塞視窗來加快發送的封包速率。若Δ大於 β 時，表示網路現在處於壅塞且沒有足夠的頻寬可使用，因此必須減少壅塞視窗的大小來減緩封包發送的速度，以免造成網路壅塞。若Δ是在 α 和 β 之間則不做任何變動。關於Δ落在 α 和 β 之間的狀況，以另一方面來看待，這是 TCP Vegas 的目標，將吞吐量控制在 α 和 β 之間，保持穩定的吞吐量。

TCP Vegas 的快速重傳機制，TCP Vegas 的發送端會去記錄封包發送出去的時間及計算預定接收到該封包 ACK 的時間（Timeout 時間）。當收到 Duplicate ACK 時，TCP Vegas 會去檢查目前時間與封包的發送時間的時間間隔是否大於 Timeout 時間。如果大於 Timeout 時間，則會立即重傳此封包。當上一個遺失的封包經過快速重傳機制重傳後，TCP Vegas 會留意重傳後的第一個或第二個所回傳的 ACK，確認已發

送的封包是否發生 Timeout，並重傳遺失的封包。

從圖五可以看到 TCP Vegas 的運作情形，從連線一開始在緩啟動階段就不斷增加壅塞視窗大小，接著結束緩啟動階段並進入壅塞避免階段。從圖五中可以看出 TCP Vegas 的機制並沒有發生像 TCP Reno 的周期性網路壅塞的現象。並且 TCP Vegas 採用測量每一回的 RTT 的方式調整壅塞視窗大小，比起 TCP Reno 使用偵測封包遺失作為調整依據更能快速反應實際網路的狀況。

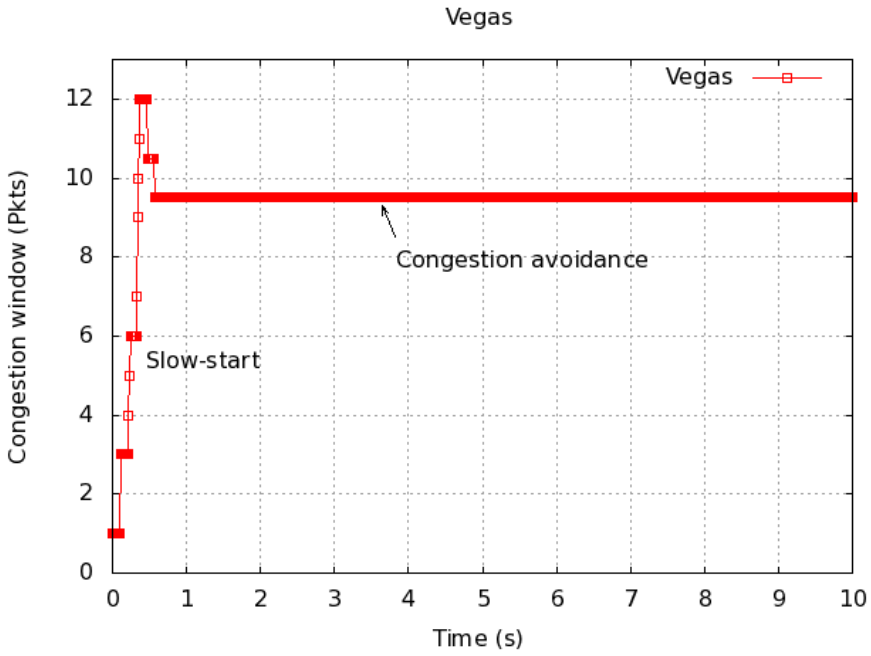


圖 五：TCP Vegas 的運作

雖然 TCP Vegas 可以避免發生 TCP Reno 與 TCP Tahoe 的缺點，但將 TCP Vegas 應用在高頻寬延遲乘積網路上仍會碰到兩個問題：

- 由於提早結束緩啟動階段，造成必須花費大量時間在壅塞避免階段才能達到對網路頻寬的完全利用。
- 壅塞避免階段的設計，並無法快速的適應不斷變化的網路環境。

2.3. 現有的解決機制

現有不少相關論文也在探討TCP在高頻寬延遲乘積網路上的問題，這裡將介紹一些基於高頻寬延遲乘積網路所發展的TCP版本與一些學者所提出的改良機制。目前所發展的改良機制可區分成以TCP Reno為基礎和以TCP Vegas為基礎的改良版，但是以TCP Reno為基礎的改良版有存在著壅塞窗大小、RTT、瓶頸點佇列長度與吞吐量等有大幅振盪的問題，這些缺點並不利於網路傳輸效能【21】。因此本文著眼在以TCP Vegas為基礎的改良版。

2.3.1. 頻寬估計來計算新的γ

在文獻【17】中提出一種以TCP Vegas為基礎的改良版本，此版本修改TCP Vegas的壅塞控制機制裡緩啟動階段中判斷是否結束緩啟動階段的參數γ，透過頻寬的估計與穩定時期壅塞視窗的大小來計算

新的γ值，並取代原本的γ值，來達到延緩結束緩啟動階段的條件。其計算γ值的公式如公式（6）所示：

$$\gamma = \frac{\mathrm{CWND}^{2}}{4\mu d + \mathrm{CWND}} \qquad (6)$$

在公式（6）中CWND為壅塞視窗大小；μ為頻寬；d為最小的來回時間，即是base RTT。

但是上述的改進機制在文獻【8】中指出，由於計算出來的γ值的後續選擇過程複雜，且頻寬的估計也存在一定難度，因為實際的網路環境中背景流量與其他許多因素的影響使得網路可用頻寬的經常變動，使得新的γ值未必可以符合目前的網路環境，有時可能造成更嚴重的壅塞，或是仍然有提前結束緩啟動階段的問題。

2.3.2. FAST TCP

FAST TCP【10】是一個以TCP Vegas為基礎衍生的TCP版本，FAST TCP是採用調整壅塞視窗來進行壅塞控制，FAST TCP衡量壅塞的狀況是透過佇列的延遲時間與封包遺失率。FAST TCP會利用平均往返時間RTT與周期性的計算與更改壅塞視窗的大小，其壅塞視窗的計算如公式（7）所示，另外FAST TCP的計算壅塞視窗的公式，是利用量測到的RTT和base RTT來計算出壅塞視窗要改變的比例，接著並與兩倍的壅塞視窗取最小值，此機制亦可以防止壅塞視窗增加速度過快。

$$\mathrm{CWND} \leftarrow \min\left\{2\mathrm{CWND}, (1 - \gamma)\mathrm{CWND} + \gamma\left(\frac{\text{base RTT}}{\mathrm{RTT}}\mathrm{CWND} + \alpha\right)\right\} \quad (7)$$

在公式（7）中，CWND代表壅塞視窗的大小；α代表停留在佇列中的封包個數，通常α值預設為100，若堆積在網路瓶頸點上的封包數目遠小於α時，Fast TCP會以倍數的方式增加壅塞視窗的大小，當堆積在網路瓶頸點上的封包數目很接近α時，Fast TCP則改以線性的方式調整壅塞視窗的大小。

2.3.3. Quick Vegas

Quick Vegas【6】【23】【24】是一種針對高頻寬延遲乘積網路環境的TCP Vegas改良版本。Quick Vegas是基於TCP Vegas的壅塞控制機制的基礎上並修改緩啟動階段與壅塞避免階段，以下將介紹Quick Vegas所修改的部分：

緩啟動階段：有別於原始的TCP Vegas的壅塞視窗成長方式，Quick Vegas改成每一個RTT時間都會增加壅塞視窗的大小，且每一次的增加量均為上一回的壅塞視窗大小的一半。Quick Vegas有效的解決在高頻寬延遲乘積網路環境中，因高往返時間所導致壅塞視窗調節速度緩慢的問題，同時也避免了封包發送的速度過快的問題。

壅塞避免階段：如圖六所示，Quick Vegas在原始TCP Vegas中的α與β之間的控制區間再區分出兩個新控制區間，這兩個新控制區間分別是α到$(\frac{\alpha+\beta}{2})$與β到$(\frac{\alpha+\beta}{2})$，所以Quick Vegas一共有四個控制區間，

每一個控制區間對於壅塞視窗的調整的策略都不一樣。同時Quick Vegas也新增了一個新的參數succ，此參數是用來增加壅塞視窗的成長幅度。

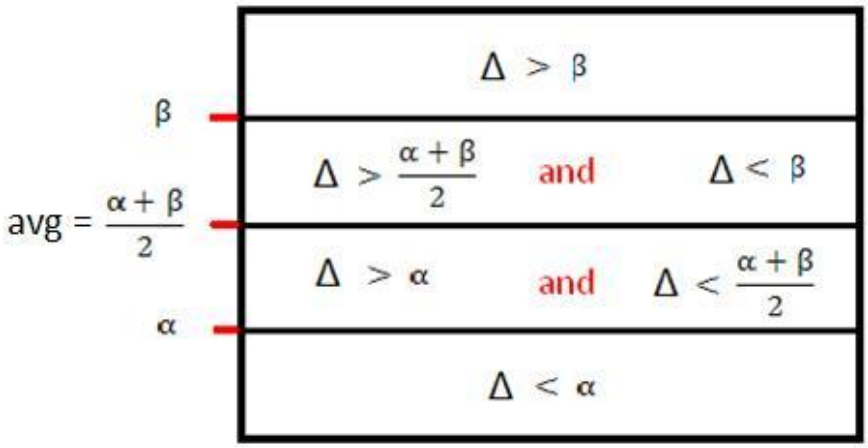


圖 六：Quick Vegas 壅塞避免階段示意圖

從上圖六所示，當Δ大於β時，表示網路壅塞狀況嚴重，故將壅塞視窗減去（$\frac{(\Delta-(\frac{\alpha+\beta}{2}))}{2}$）。當Δ落在β到（$\frac{\alpha+\beta}{2}$）之間時，表示網路已經有壅塞的狀況，並將壅塞視窗大小減一。若Δ落在α到（$\frac{\alpha+\beta}{2}$）之間時，壅塞視窗的成長則是增加當下壅塞視窗大小的（$\frac{1}{\mathrm{CWND}}$）倍。若Δ小於α時，表示網路狀況良好，接下來Quick Vegas則會先將succ參數加1，之後會檢查（$(\beta-\Delta)\times \mathrm{succ}$）是否有大於當下的CWND值，若比當下的CWND值大，則將壅塞視窗加一，反之若小於當下的CWND值則將CWND加上（$\frac{(\beta-\Delta)}{\mathrm{CWND}}$）的大小。最後若Δ值等於（$\frac{\alpha+\beta}{2}$），則表示壅塞視窗的大小適當，故不對壅塞視窗大小做調整。

Quick Vegas的改良將有助於壅塞避免階段中壅塞視窗的調整速度與成長幅度。

2.4. 基礎機制的選擇

綜觀上述所介紹的 TCP 版本，在探討 TCP Tahoe、TCP Reno、TCP Vegas 以及 2.3.1 小節的改良機制、2.3.2 小節的 FAST TCP 與 2.3.3 小節的 Quick Vegas 後，發現 TCP Tahoe 與 TCP Reno 因為本身判定壅塞現象的方式與壅塞控制機制的設計會讓壅塞視窗呈現周期性震盪的現象，此現象會造成應用在高頻寬延遲乘積網路上有較低的頻寬利用率。然而 TCP Vegas 的機制並沒有壅塞視窗有周期性震盪的現象，但是應用在高頻寬延遲乘積網路環境下仍有頻寬利用率不高的問題。所以有了 2.3.1 小節、Fast TCP 與 Quick Vegas 的改良機制，但是 2.3.1 小節的改良機制仍有過於複雜容易受外來因素影響【8】，Fast TCP 有 α 值的選擇問題【1】【25】，Quick Vegas 對於緩啟動階段壅塞視窗的成長幅度仍不理想【7】，雖有缺點但是他們的基礎機制均選擇 TCP Vegas，原因是用 RTT 的改變來預測可用頻寬與評估網路壅塞狀況的方式，比起 TCP Tahoe、TCP Reno 或是以 TCP Reno 為基礎的衍生版本，使用偵測封包遺失的方式精確許多，並且 TCP Vegas 的公平性與效率也較好【9】【11】【13】。所以本文決定以使用測量 RTT 來調整壅塞視

窗大小與評估網路壅塞狀況的 TCP Vegas 為基礎機制。

第三章　TCP Vegas 在高頻寬延遲乘積網路環境中運作的問題分析

3.1. 緩啟動階段

在 2.2.3 小節中，我們提到 TCP Vegas 的緩啟動階段的運作流程。在理想的情況下，緩啟動階段結束時壅塞視窗的大小必須能讓一個連線能掌握可用頻寬，但實際上 TCP Vegas 應用在高頻寬延遲乘積網路中的情形，則是會發生如本文中 2.2.3 小節所提到 TCP Vegas 的第一個問題：緩啟動階段提前結束與壅塞視窗的大小無法有效掌握網路頻寬的問題。造成發生上述問題的原因，即是因為 Δ 在短時間之內迅速增大並且大於 γ 值，接著就離開緩啟動階段並進入壅塞避免階段，同時也因為離開緩啟動階段時當下的壅塞視窗的大小過小，所以並無法有效的掌握網路可用頻寬。

在 2.2.3 小節中我們也提到 TCP Vegas 在緩啟動階段中，是經過兩個 RTT 時間才將壅塞視窗成長一倍。在這兩個 RTT 時間，其中一個 RTT 時間稱成長期，在這段期間內若發送端收到上一回發送的封包群中其第一個封包的 ACK 後，發送端會立即發送出兩個封包，直到收到上一回發送的封包群中的最後一個封包的 ACK 後為止，TCP Vegas 用此方式讓壅塞視窗成長。另一個 RTT 時間則稱觀察期，不會改變壅塞

視窗大小只是純粹觀察 RTT 變化。其中，當觀察期結束後，TCP Vegas 會利用量測到的 RTT 計算出 Δ 值來決定緩啟動結束與否。現在我們則更進一步指出，提前離開緩啟動階段的原因就是觀察期結束後，量測到的 RTT 變大，導致 Δ 增大。

造成 RTT 變大的原因可以分成兩個方面，即所處的高頻寬延遲乘積網路環境與 TCP Vegas 對於成長期的壅塞視窗成長行為兩方面。

在所處的網路環境方面，因為在高頻寬延遲乘積網路環境中，網路環境本身就具有較高的 RTT 特性，所以也影響了觀察期所量測到的 RTT。

接著在壅塞視窗的成長行為方面，在緩啟動階段裡，封包在短時間內（在一個 RTT 時間）密集的被發送的行為，將導致網路上的佇列會發生堆積大量的封包的現象，此現象將伴隨著壅塞視窗的大小將會越來越明顯，並且此現象也使得每一次觀察期所量測到的 RTT 中，關於封包停留在網路佇列的時間大增。

最後基於上述的兩方面原因，故使得壅塞視窗在較小的狀況下，Δ 值就超過 γ，並導致提早結束緩啟動階段並進入壅塞避免階段，如圖七所示。

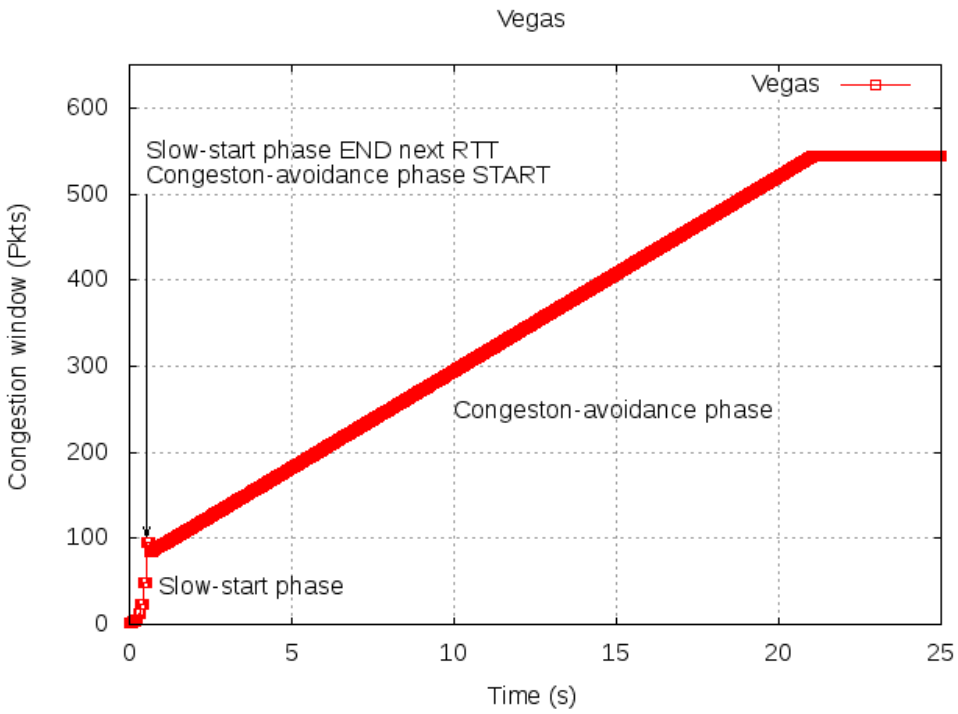


圖 七：提早結束緩啟動階段時壅塞視窗成長示意圖（BDP 為 1000）

提早結束緩啟動階段會使得壅塞視窗在壅塞避免階段中的一段時間內呈線性增長，此現象將造成壅塞避免階段將消耗大量的時間來掌握網路頻寬，這將影響 TCP 的傳輸效率。

3.2. 壅塞避免階段

在 2.2.3 小節中，我們提到原始的 TCP Vegas 在壅塞避免階段的壅塞視窗調整是透過Δ值落在由 α 與 β 所形成的三個控制區間中的位置來決定壅塞視窗的的大小。

但是原始的壅塞避免階段若用在高頻寬延遲乘積網路環境中則會產生如圖八的結果。

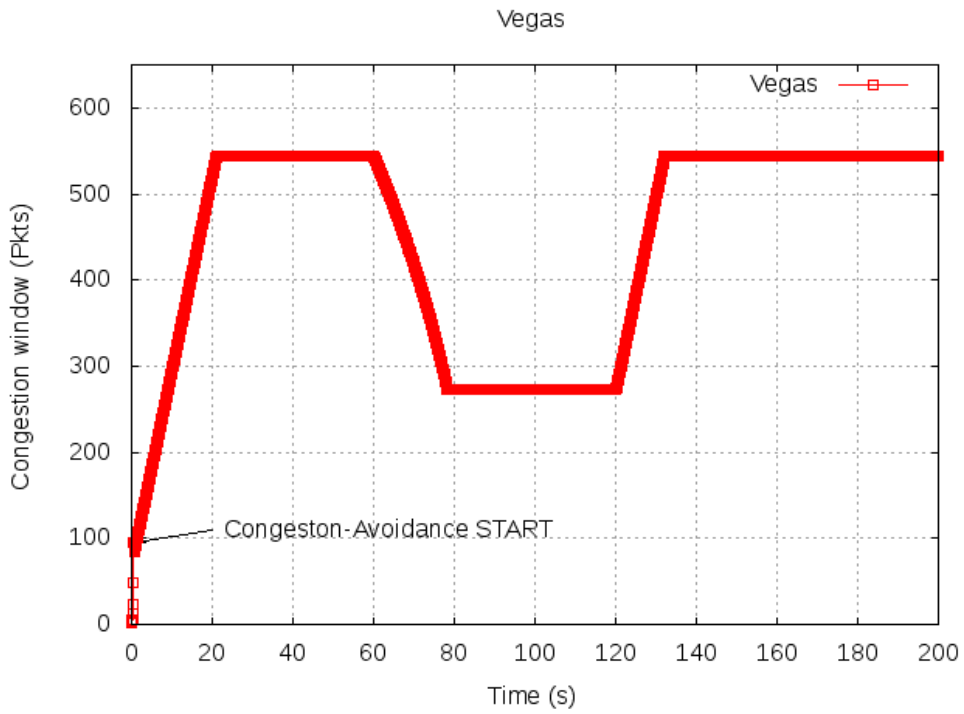


圖 八：壅塞避免階段的壅塞視窗成長示意圖（BDP 為 1000）

從圖八的結果我們可以觀察原始的 TCP Vegas 在第 60 秒以及第 120 秒時，遇到網路中背景流量變化時所反映出的壅塞視窗的變化。我們可以明顯的發現 TCP Vegas 對於壅塞視窗改變的速度十分緩慢。關於圖八的結果，則對映了在 2.2.3 小節中所提到 TCP Vegas 無法快速適應不斷變化的網路環境之問題。造成此問題的原因，同樣也可以從網路環境與壅塞視窗的控制方式兩方面來得知。

在網路環境方面，由於在高頻寬延遲乘積網路環境中，具有較高的 RTT 特性，造成藉由量測 RTT 來調整壅塞視窗的 TCP Vegas，在固定時間內對壅塞視窗作調整的次數相對變少，導致 TCP Vegas 無法快速的適應網路環境的變化。

在壅塞視窗的控制方式方面，原始的 TCP Vegas 所預設的 α 與 β 值，並不適用於高頻寬延遲乘積網路環境中壅塞視窗的控制。並且原始的壅塞避免階段中，雖然有由 α 與 β 值所形成的三個控制區間，但是原始的壅塞避免階段中的控制區間數量太少，導致壅塞視窗的調整能力較弱。同時原始的壅塞避免階段對於壅塞視窗的調整幅度太小，將導致一條連線在運作過程中若有另一條連線建立時，原始的壅塞避免階段將無法迅速的釋放頻寬給另一條連線，同樣的當另一條連線停止運作時，也將無法迅速的掌握此條連線所釋放的網路頻寬。

第四章　改進機制介紹

在這章節中將介紹基於高頻寬延遲乘積網路環境的 TCP Vegas 改良版本，High Bandwidth-Delay Product Vegas（HBDP Vegas）。首先，將會說明 HBDP Vegas 所新增的一種衡量網路壅塞程度的方式，接下來則會介紹 HBDP Vegas 的壅塞控制機制中，緩啟動階段與壅塞避免階段所修改的部分與其演算法。

4.1. 衡量網路壅塞程度的方式

TCP Vegas 衡量網路壅塞的程度是利用公式（2）算出停留在網路佇列中的封包數量Δ，並利用Δ來衡量網路壅塞程度，但是Δ的大小要多大才算是發生壅塞現象，或是Δ的大小要小於多少才算是沒有發生壅塞現象，均無法直接定義。雖然 TCP Vegas 在緩啟動階段與壅塞避免階段，均有再搭配γ以及α與β值來做為網路是否發生壅塞的依據，但是若 α 與 β 以及 γ 的值若選擇不當，反而可能會造成網路壅塞或是降低傳輸效率等問題。因此我們需要一種新的衡量網路壅塞程度的方法，其衡量的結果值是可以直接表達網路壅塞的程度，並且使用此方法來補強原本的 TCP Vegas 在衡量網路壅塞程度機制的不足。

在重新檢視 TCP Vegas 的運作過程中，我們觀察了 TCP Vegas 每一次所量測的 RTT。關於每一次所量測的 RTT 中，在 2.2.3 小節曾提到所量測到的 RTT 之中包含了佇列延遲時間（封包停留在佇列的時間）與其他開銷的時間，但是若從另一方面來看，若排除 RTT 之中所有的延遲時間，剩下的時間則是，是在沒有任何延遲之下的傳輸時間，即 base RTT。因此在 HBDP Vegas 衡量網路壅塞程度的部分，我們利用上述對每一次所量測的 RTT 的觀察結果，我們提出透過 base RTT 與每一次所量測的 RTT 的比例值來衡量目前的網路壅塞程度，如公式（8）所示，K 值代表 base RTT 與 RTT 的比例值。

$$0 < \mathrm{K} = \frac{\text{base RTT}}{\text{RTT}} < 1 \qquad (8)$$

在公式（8）中，若只看公式的表面上的意涵，常數 K 代表了 base RTT 在每一次所量測的 RTT 中所占的比例值，但若再深入探討，我們可以從 base RTT 在 RTT 中所占的比例來觀察網路的壅塞程度，若比例值越接近 1，則表示目前所量測的 RTT 近似 base RTT，同時也表示網路狀況良好。相反的，比例值越接近 0，除了表示目前所量測的 RTT 大於 base RTT，同時也表示網路狀況已處於壅塞。

4.2. 緩啟動階段

緩啟動階段的目標是在緩啟動階段結束時，能使一條連線的壅塞視窗大小足以能掌握網路頻寬。

4.2.1. 緩啟動階段的門檻值 K'

HBDP Vegas 的緩啟動階段中關於緩啟動的結束與否方面，我們會利用 4.1 節所介紹的 K 值之外，並且會再另外使用一個預設的比例值，門檻值 K'。 透過門檻值 K'與 k 值的比較來決定緩啟動階段的結束與否。其中，若 K 值大於或等於 K'則表示留在緩啟動階段。反之，若 K 小於 K '則表示結束緩啟動階段。上述關於緩啟動階段之規則如公式（9）所示：

$$\begin{cases} K \geq K', \text{留在緩啟動階段} \\ K < K', \text{離開緩啟動階段} \end{cases} \quad (9)$$

4.2.2. 緩啟動階段門檻值 K'的選擇

在這一小節中，我們將說明門檻值 K' 的選擇。從上一小節中我們得知門檻值 K' 決定了緩啟動階段結束與否。但若再深入探討，門檻值 K' 它可以決定一條連線在緩啟動階段結束時掌握網路頻寬的程度。關於門檻值 K' 的選擇，則可以從表格一與表格二中的實際的數據來得知。

表格 一：緩啟動階段結束時當下的壅塞視窗大小（1）

緩啟動階段結束時當下的壅塞視窗大小（1） 單位：packets									
門檻值 K' / BDP	0.9	0.85	0.8	0.75	0.7	0.65	0.6	0.55	0.5
100	69.375	76.75	90.3125	106.25	112.875	112.875	112.875	119.875	127.312
200	149.438	149.812	176.062	207	219.938	219.938	219.938	233.625	248.188
400	310.312	292	343.438	404	429.25	429.25	429.25	456.062	484.562
800	605.188	570.062	670.688	838.312	838.312	838.312	890.688	890.688	1047.88
1000	757.062	712.562	838.312	1047.88	1047.88	1047.88	1113.31	1113.31	1309.75
2000	1478.38	1391.56	1637	2046.06	2046.06	2046.06	2173.94	2557.44	2557.44
3000	2309.94	2173.94	2557.44	3008.75	3196.75	3196.75	3196.75	3396.5	3608.75

表格 二：緩啟動階段結束時當下的壅塞視窗大小（2）

緩啟動階段結束時當下的壅塞視窗大小（2） 單位：packets								
門檻值 K' / BDP	0.45	0.4	0.35	0.3	0.25	0.2	0.15	0.1
100	135.25	159.125	179.625	202.688	243	309.625	419.062	627.812
200	292	310.25	350.188	420	503.75	604.125	801.75	1225.31
400	570.065	605.688	683.688	819.938	983.375	1253.12	1664.56	2394.62
800	570.062	605.688	1418.69	1601.5	1920.81	2401	3251	4969.5
1000	1391.56	1478.5	1773.31	2001.88	2401	3001.25	4063.75	6211.69
2000	2717.25	3067.5	3462.88	4153.5	4981.88	6227.12	7935.94	12130.8
3000	4245.44	4510.75	5092.12	6107.75	7325.94	9336.25	12399.4	17839

從表格一與表格二中所記錄的數據，均為使用不同的門檻值K'，於不同的頻寬延遲乘積（BDP）網路環境中，在緩啟動階段結束時當下的壅塞視窗大小。在表格一與表格二中可以得知，門檻值K' 的選擇並不建議選擇太接近1的值，因為門檻值K' 太接近1會影響連線掌握頻寬的能力，若門檻值K' 太接近0則反而可能會使得網路發生壅塞

現象，因此我們建議的門檻值K' 為0.7。因為在不同的BDP網路環境中，門檻值K' 都在0.7的情況下，在緩啟動階段結束時，此刻的壅塞視窗大小也已經足以能夠讓一個連線能夠掌握大部分的可用頻寬，並且在後續的實驗中，門檻值K' 在0.7的情況下將有較佳的表現。最後，若因所處在的網路環境需求，需要加強連線對網路頻寬的掌握能力，則可以設定門檻值K' 為一個比0.7小且大於0.45的值。

4.2.3. 緩啟動階段壅塞視窗的成長方式

在壅塞視窗的成長方式方面，HBDP Vegas 的壅塞視窗成長方式有別於原始的 TCP Vegas，由原本每兩個 RTT 增加一倍壅塞視窗的方式，改為每一個 RTT 時間增加當前壅塞視窗的大小，如此一來將可以解決在高頻寬延遲乘積網路環境中，因高往返時間導致壅塞視窗調整速度過於緩慢的問題。

在壅塞視窗的增加量的方面，我們提出將原本的 TCP Vegas 緩啟動階段再分成兩個階段，每一個階段中壅塞視窗的成長的倍數則透過是否有封包堆積在網路佇列上與當下 base RTT 與 RTT 的比例值 K 來決定增加當前壅塞視窗的大小的 0.25 倍或是 0.0625 倍。實際的壅塞視窗的增加量規畫如公式（10）所示，其中 CWND 為當下的壅塞視窗大小。

$$\text{newCWND}\begin{cases} \text{CWND} + (\text{CWND} \times 0.25), & \text{第一階段 } \Delta \leq 1 \\ \text{CWND} + (\text{CWND} \times 0.25), & \text{第二階段 } \Delta > 1 \text{ 且 } K \geq \left(\frac{1-K'}{2}\right) + K' \\ \text{CWND} + (\text{CWND} \times 0.0625), & \Delta > 1 \text{ 且 } K < \left(\frac{1-K'}{2}\right) + K' \\ & \text{且 } K \geq K' \end{cases} \quad (10)$$

如公式（10）所示在第一階段中，連線一開始就進入第一階段，當（$\Delta \leqq 1$）時，表示當下的網路並沒有發生壅塞且佇列中封包堆積狀況並不嚴重，所以我們希望在此階段可以藉由快速的增加壅塞視窗大小來達到完全利用頻寬的目的。因此，階段一為每一個RTT時間壅塞視窗的增加量，為當前壅塞視窗的大小的0.25倍，直到佇列中封包堆積量變大時（$\Delta > 1$），才會離開第一階段並進入第二階段。

如公式（10）所示在第二階段中，由於佇列上已經有明顯的封包堆積的現象，所以每一次所量測的RTT已經受到佇列延遲影響，因此在此階段開始在每一個RTT時間去計算RTT與base RTT的比例值，即K值。同時在第二階段中則又在細分成兩個小區間。

第一個區間為大於（$\left(\frac{1-K'}{2}\right) + K'$），在此區間中壅塞視窗的成長倍數為0.25倍($\text{CWND} \times 0.25$)。第二個區間為小於（$\left(\frac{1-K'}{2}\right) + K'$），且大於 K'，在此區間中壅塞視窗的成長倍數為 0.0625 倍（$\text{CWND} \times 0.0625$）。

4.2.4. HBDP Vegas 的緩啟動階段與原本的 TCP Vegas 比較

上述修改過的緩啟動階段的和原本的 TCP Vegas 的緩啟動階段比較之下，除了能在緩啟動階段結束時能使壅塞視窗大小足以能掌握網路頻寬之外，並且也可以有效緩和在短時間內密集地發送封包，導致網路佇列堆積大量封包的狀況以及因高往返時間導致壅塞視窗調整速度過於緩慢的問題。

最後則是附上 HBDP Vegas 的緩啟動階段虛擬碼：

```
00  計算 K 值
01  if(當下的網路並沒有發生壅塞且佇列中封包堆積狀況並不嚴重時（Δ ≦ 1）){
02     壅塞視窗的增加量，為當前壅塞視窗的大小的 0.25 倍
03  }
04  else if(佇列中封包堆積量變大時（Δ > 1）){
05      if(佇列上已經有封包堆積的現象同時 RTT 與 base RTT 的比例值（K 值）小於 1 時){
06          if(RTT 與 base RTT 的比例值（K 值）小於門檻值 K '時){
07              結束緩啟動階段
08          }
09          else{
10              if(RTT 與 base RTT 的比例值（K 值）大於或是等於 ( ( (1 - K ') / 2 ) + K ' ) ){
11                  壅塞視窗的增加量，為當前壅塞視窗的大小的 0.25 倍
12              }
13              else if ((RTT 與 base RTT 的比例值（K 值）落在( ( (1 - K ') / 2 ) + K ') 與
14                門檻值 K '之間)){
15                  壅塞視窗的增加量，為當前壅塞視窗的大小的 0.0625 倍
16              }
17          }
18      }
19  }
```

4.3. 壅塞避免階段的修改

壅塞避免階段的目的是讓連線能夠適應當下不斷變化的網路環境。其中，能具體反映出當下網路環境的變化即是壅塞視窗，因此在壅塞避免階段中壅塞視窗大小的控制即是一個重要的課題。因此我們將在4.3節介紹HBDP Vegas的壅塞避免階段所做的改進。

4.3.1. α與β值的選擇

由於原始TCP Vegas所預設的α與β值並不適用於高頻寬延遲乘積的網路環境，應此會發生緩啟動階段結束後，因為較大的Δ值導致壅塞視窗大幅下降的問題。因此在α與β值的選擇方面，HBDP Vegas的壅塞避免階段會在連結第一次進入壅塞避免階段時，利用當下所量測到的Δ值來設定α與β值。往後就會使用此組α與β值來調整往後的壅塞視窗大小，同時也會以此組α與β值為基礎，在不同的網路狀況下再次針對α與β值作調整。

4.3.2. 壅塞避免階段中壅塞視窗大小的控制

修改過的壅塞避免階段將分成四個控制區間來調整壅塞視窗，如圖九所示，新的壅塞避免階段比原始壅塞避免階段多了兩個控制區間。這兩個新控制區間是透過在α與β之間新增一個新的參數avg所產生的，

參數avg是（$\frac{(\alpha+\beta)}{2}$）。新的壅塞避免階段可以改進原本的TCP Vegas在壅塞避免階段具較弱的壅塞視窗調整能力導致無法有效的掌握網路頻寬的問題。

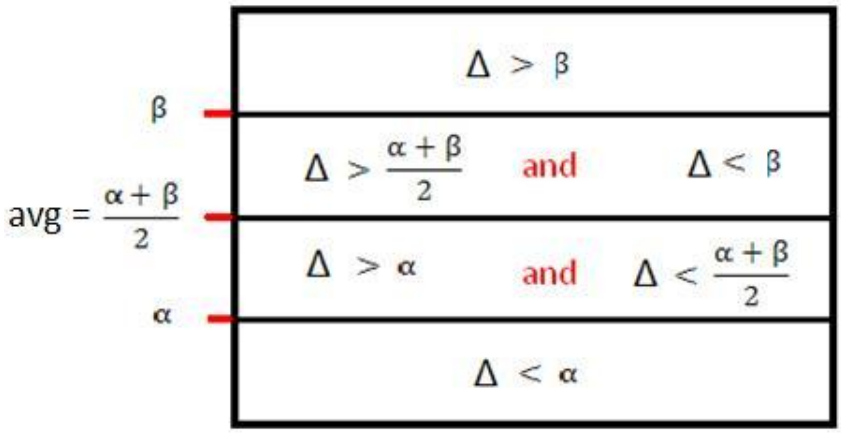


圖 九：壅塞避免階段區段配置示意圖

第一控制區間：如圖九所示，Δ大於β的情況下，表示網路已處於壅塞，且佇列上已堆積許多封包，因此第一控制區間的目標即是要避免壅塞現象的發生，因此必須減少封包的發送量。故在此控制區間會利用4.1節所提到的K來值評估網路的壅塞狀況並決定壅塞視窗的減小幅度。其中減少幅度則從最大的（Δ − avg）、$(\frac{(\Delta-\text{avg})}{2})$到最小的$\frac{(\Delta-\text{avg})}{4}$。最後，在第一控制區間中，若壅塞視窗的大小遞減量使得值評估網路壅塞狀況的K值達到0.5時，在本文的機制中，則是被認定已達到一個穩定狀態，故不再減小壅塞視窗，同時則開始對α與β值做修改。

第二控制區間：如圖九所示，Δ小於β且大於avg的情況下，表示網路狀況穩定，沒有嚴重的壅塞。但在此控制區間仍會去微調壅塞視窗大小與α、β值。同時，在此控制區間中也會紀錄第一次進入此控制區間的壅塞視窗大小，以幫助往後在第二控制區間、第三控制區間與第四控制區間的壅塞視窗調整。

第三控制區間：如圖九所示，Δ小於avg且大於α的情況下，表示網路狀況良好，沒有明顯的壅塞，壅塞視窗可以增長。在此控制區間的壅塞視窗的調整幅度，最大的是（0.5 × CWND），其次是（0.25× CWND）與（0.125× CWND）最小的為（0.0625× CWND），其中成長幅度的控制，則將利用4.1節提到的K值，所評估的網路壅塞狀況來決定。同時，在調整壅塞視窗的過程中，也會增加α與β值以增加停留在第三控制區間的機會。最後，在壅塞視窗的調整過程中，也會參考第一次進入第二控制區間時的壅塞視窗大小與檢查k值是否達到0.4以下。其原因是為了避免發生壅塞視窗成長幅度過快又再次發生壅塞現象。

第四控制區間：如圖九所示，Δ小於 α 的情況下，表示網路狀況良好，沒有壅塞，可以大幅度的增加壅塞視窗，使得連結能快速掌握頻寬。在此控制區間中壅塞視窗增加幅度可分成三種，分別為（0.25 × CWND）與（0.125× CWND），關於壅塞視窗的調整幅度選擇，則利用 4.1 節所提到的 K 值來決定，同時也會增加 α 與 β 值以增加停留在

第四控制區間的機會。其中，在壅塞視窗的調整過程中也會參考第一次進入第二控制區間當下的壅塞視窗大小的原因，是為了避免發生壅塞視窗成長幅度過快又再次發生壅塞現象。

最後，如圖九所示，若Δ等於 avg，表示網路狀況穩定，則不對壅塞視窗做任何調整，也不對 α 與 β 值做調整。

4.3.3. HBDP Vegas 的壅塞避免階段與原本的 TCP Vegas 比較

HBDP Vegas在壅塞避免階段上除了改進了原始TCP Vegas對於α與β值的設定問題以及控制區間的規劃，同時也搭配本文所提出的評估網路壅塞程度的方式，算出每一個RTT時間的K值，並利用K值來調整壅塞視窗。上述的改善機制將可以改進原始的TCP Vegas在壅塞避免階段中對於壅塞視窗過於保守且不足的調整能力。

HBDP Vegas 的壅塞避免階段虛擬碼如下：

```
00  計算 K 值
01  if(第一次進入壅塞避免階){
02      利用當下所量測到的Δ值來設定 α 與 β 值以及 avg 值。
03  }
04  if(網路已處於壅塞（Δ > beta）){ //第一控制區間
05      利用 K 值評估網路的壅塞狀況來決定壅塞視窗的減小幅度。
06      若 K 值達到 0.5 時，表示已達到穩定狀態，故不再減小壅塞視窗，同時則開
07      始對 α 與 β 值以及 avg 值做修改。
08  }
09  if(網路狀況穩定（(Δ <=beta_) && (Δ >avg)）){ //第二控制區間
10      紀錄第一次進入第二控制區間的壅塞視窗大小。
11      視狀況調整壅塞視窗大小、α 與 β 值以及 avg 值。
12  }
```

```
13  if(網路狀況良好，沒有明顯的壅塞(Δ >=alpha) && (Δ <( avg))) { //第三控制區間
14      利用 K 值與此連線第一次進入第二控制區間時的壅塞視窗大小來定壅塞視窗
15      成長幅度。
16      視狀況調整壅塞視窗大小、α 與 β 值以及 avg 值。
17  }
18  else if(網路狀況良好，沒有壅塞（Δ < alpha）) { //第四控制區間
19      利用 K 值與此連線第一次進入第二控制區間時的壅塞視窗大小來定壅塞視窗
20      成長幅度。
21      視狀況調整壅塞視窗大小、α 與 β 值以及 avg 值。
22  }
23  else if(Δ等於 avg){
24      不對壅塞視窗做調整，也不對 α 與 β 值以及 avg 值做調整。
25  }
```

第五章　實驗結果

本文將透過網路模擬軟體 NS-2（Network Simulator，version 2）【15】來探討 FAST TCP、Quick Vegas、TCP Vegas 以及在第四章中所提出的 HBDP Vegas 的效能。

在本文中所使用的網路模擬軟體 NS-2，是由 UC Berkeley 所開發的一款物件導向的網路模擬器。NS-2 可以模擬真實網路上的路由器與網路上結點的運作以及模擬傳輸的延遲與封包的遺失等現象。因此我們將使用 NS-2 來驗證我們所提出的改進機制。

5.1. 實驗目標與網路拓樸

在本節中我們將說明我們所設計的四組實驗之目標以及說明實驗過程中所使用的網路拓樸以及各版本 TCP 協定參數設定。

在實驗目標的部分，我們將從下列四個目標來探討 HBDP Vegas：

1. 緩啟動階段的比較：比較不同的 TCP 協定在緩啟動階段對頻寬的掌握程度。
2. 收斂實驗：比較不同的 TCP 協定從連線啟動後到完全掌握頻寬所需的時間以及在傳輸過程中適應網路環境變化所需的時間。

3. 面對其連線競爭時之狀況：在此實驗中我們將再從三方面來探討不同的 TCP 協定運作之情形：

- 面對相同的壅塞控制機制同時競爭網路頻寬時的收斂情形。
- 面對與不同流量競爭時壅塞視窗的變化與吞吐量的變化。
- 面對多個新加入的流量時，個別流量的吞吐量。

4. 公平性：在多條連線同時運作之下，觀察不同的TCP協定共用網路頻寬的狀況。

最後關於上述實驗的過程與結果將會在5.2節以後的章節再作詳細的說明。

在網路拓樸部分，我們設計了一個網路拓樸，如圖十所示。同時在後續的實驗所使用的網路拓樸均以圖十基礎。

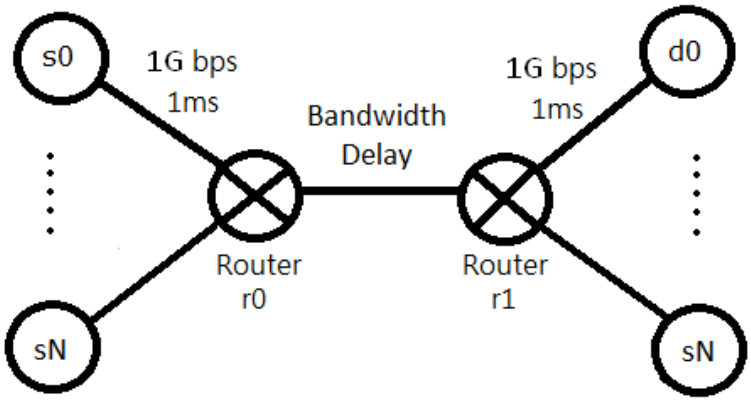


圖 十：網路拓撲

如圖十的網路拓樸中，發送端與接收端的數量會依照往後的實驗需求而有所不同。具有相同編號的發送端與接收端（s0 – d0），則會透過兩個路由器（r0、r1）連接起來。每個發送端（s0 - sN）與接收端（d0 - dN）與路由器之間的網路頻寬為 1G bps，傳輸延遲時間為 1ms，接下來在兩個路由器（r0、r1）之間的網路頻寬與傳輸延遲時間，則是依照往後的實驗需求再作設定，如此設計是為了藉由改變兩個路由器（r0、r1）相互連接線段之間的網路頻寬與傳輸延遲，來調整實際的頻寬延遲乘積（Bandwidth-Delay Product，BDP）大小與造成一個容易產生壅塞現象的環境。路由器的緩衝區管理機制為 DropTail，緩衝區大小則是需要有足夠的大小為原則。封包大小為 1 KB。

在不同版本的 TCP 協定中的預設參數方面，TCP Vegas 與 Quick Vegas 使用的參數值為（α＝2、β＝4、γ＝2）。FAST TCP 的 α 值則是依照往後的實驗所需的 BDP 而調整。HBDP Vegas 因為不像 TCP Vegas 與 Quick Vegas 是使用 γ 來判斷是否離開緩啟動階段，而是改採用門檻值 K'，同時因為自身壅塞避免階段的設計，並不用對 α 與 β 值設定預設值，最後關於門檻值 K'的設定，則是依照往後的實驗需求而調整。

在後續的實驗中，我們將會利用由圖十所衍生的網路拓樸，來觀察與記錄在不同的網路狀況下，不同版本的 TCP 連線運作中的參數變化。在後續的實驗中，我們記錄的參數均以壅塞視窗大小為主，因為

壅塞視窗大小除了表示發送的封包數量之外，也能直接的反映出不同版本的 TCP 傳輸效能。

5.2. 緩啟動階段的比較

在理想的情況下，緩啟動階段結束時壅塞視窗的大小必須能讓一個連線能掌握大部分的可用頻寬，若緩啟動階段結束時壅塞視窗大小越大，則表示對網路頻寬的掌握程度越高。故透過圖十一的網路拓樸，來比較 HBDP Vegas、Quick Vegas 與 TCP Vegas 在緩啟動階段結束後，進入壅塞避免階段前的壅塞視窗大小，以及建立連結開始在 20 秒以內的吞吐量與頻寬利用率。

本次實驗的網路拓樸則如圖十一所示，在此網路拓樸中只有一組發送端與接收端，關於此網路拓樸的詳細的實驗參數則如表格三所示。

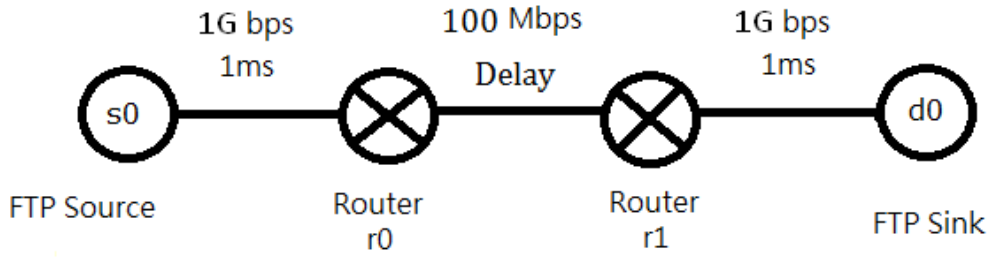


圖 十一：網路拓樸

表格 三：模擬環境參數表

模擬環境參數表	
網路模擬器	NS-2
節點（s0 與 d0）	● 使用檔案傳輸控制協定（FTP）來作資料傳輸 ● FTP Source（s0）經由 r0 連上網路，並將資料發送給 FTP Sink（d0），並持續 20 秒。
兩個路由器（r0、r1）相互連接線段之間網路頻寬	100Mbps
兩個路由器（r0、r1）相互連接線段之間傳輸延遲時間	視 BDP 而調整
TCP Vegas α、β、γ	α＝2、β＝4、γ＝2
Quick Vegas α、β、γ	α＝2、β＝4、γ＝2
HBDP Vegas 門檻值 K'	0.7

接下來，關於實驗過程中所得到的數據將會分別呈現在 5.2.1 小節與 5.2.2 小節中，最後會在 5.2.3 小節說明緩啟動階段的比較結果。

5.2.1. 緩啟動階段結束時壅塞視窗大小比較

在緩啟動階段結束時壅塞視窗大小部分，我們可以從下面的表格四與 5.2.1.1 小節至 5.2.1.7 小節中，針對在不同的 BDP 以及不同版本 TCP 的壅塞視窗變化的描述，我們可以發現當 BDP 愈大時 HBDP Vegas 效果愈好。

表格 四：緩啟動階段結束時壅塞視窗大小比較

緩啟動階段結束時壅塞視窗大小 單位：packets			
BDP	**HBDP Vegas**	**Quick Vegas**	**TCP Vegas**
3000	3196.75	274.5	192
2000	2046.06	183	192
1000	1047.88	122	96
800	838.312	122	96
400	429.25	81.5	48
200	219.983	54.5	48
100	112.875	36.5	48

5.2.1.1. BDP 為 100 時，各版本 TCP 的壅塞視窗變化

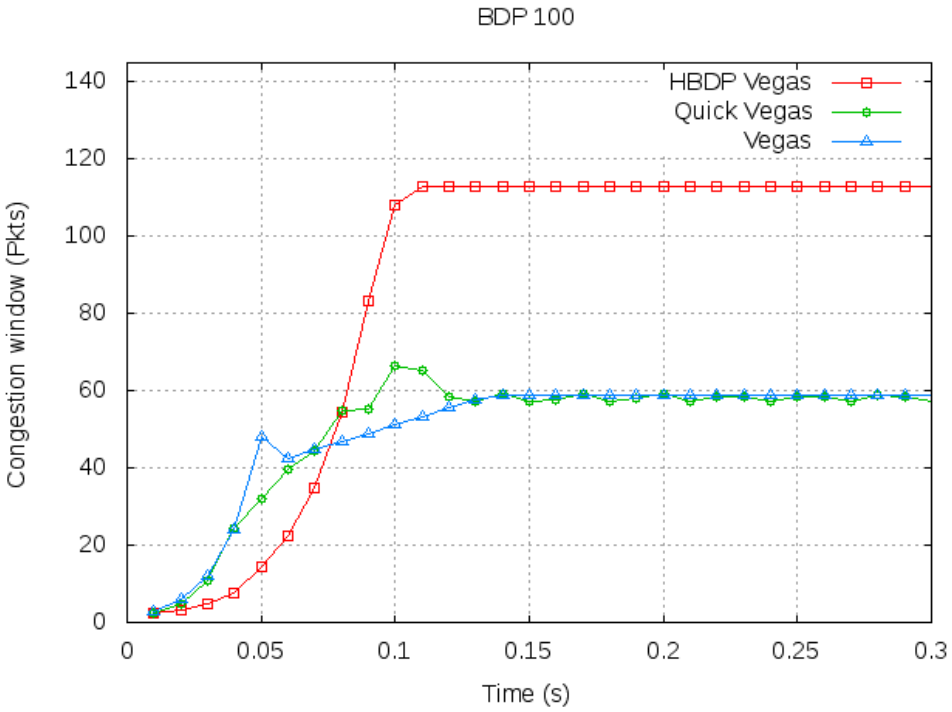


圖 十二：BDP 為 100 時，各版本 TCP 的壅塞視窗變化

在圖十二中，網路的 BDP 為 100 時，HBDP Vegas 在 0.1099 秒離開緩啟動階段，同時此時的壅塞視窗大小已經足以能夠讓一個連線能夠掌握大部分的可用頻寬。 TCP Vegas 與 Quick Vegas 兩者分別在 0.05 秒與 0.04 秒離開緩啟動階段，但此時兩者的壅塞視窗大小均無法讓連線能有效的掌握網路頻寬，因此 TCP Vegas 與 Quick Vegas 兩者都必須在壅塞避免階段中再花費一段時間來掌握網路頻寬。

5.2.1.2. BDP 為 200 時，各版本 TCP 的壅塞視窗變化

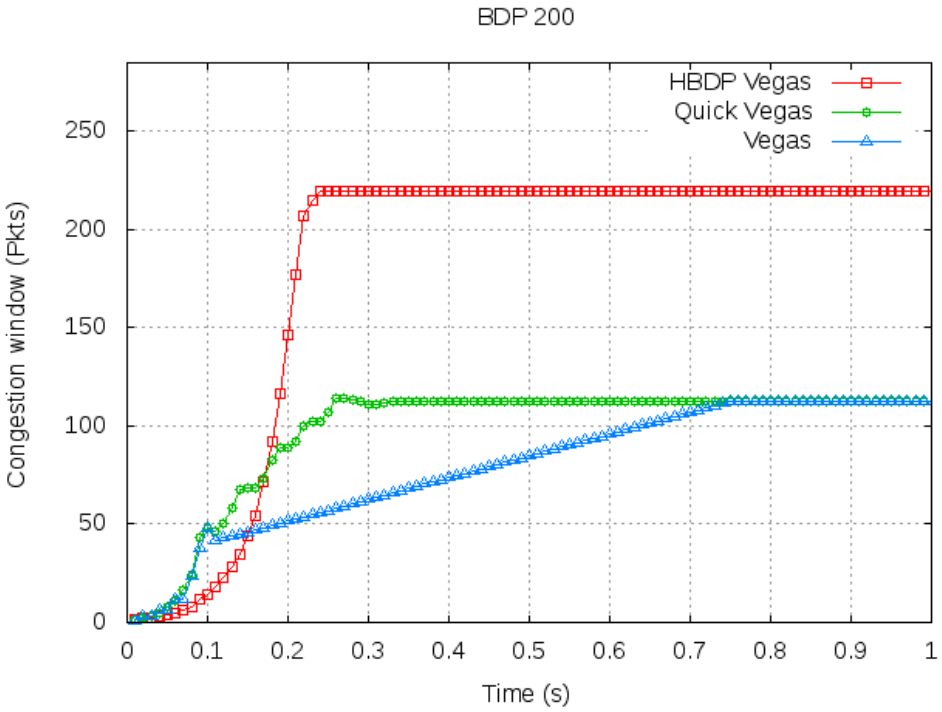


圖 十三：BDP 為 200 時，各版本 TCP 的壅塞視窗變化

在圖十三中，網路的 BDP 為 200 時，HBDP Vegas 在 0.24 秒離開緩啟動階段，同時此時的壅塞視窗大小已經足以能夠讓一個連線能夠掌握大部分的可用頻寬。TCP Vegas 與 Quick Vegas 兩者都在 0.09 秒結束了緩啟動階段，但此時兩者的壅塞視窗大小均無法讓連線能有效的掌握網路頻寬，因此 TCP Vegas 與 Quick Vegas 兩者都必須在壅塞避免階段中再花費一段時間來掌握網路頻寬。

5.2.1.3. BDP 為 400 時，各版本 TCP 的壅塞視窗變化

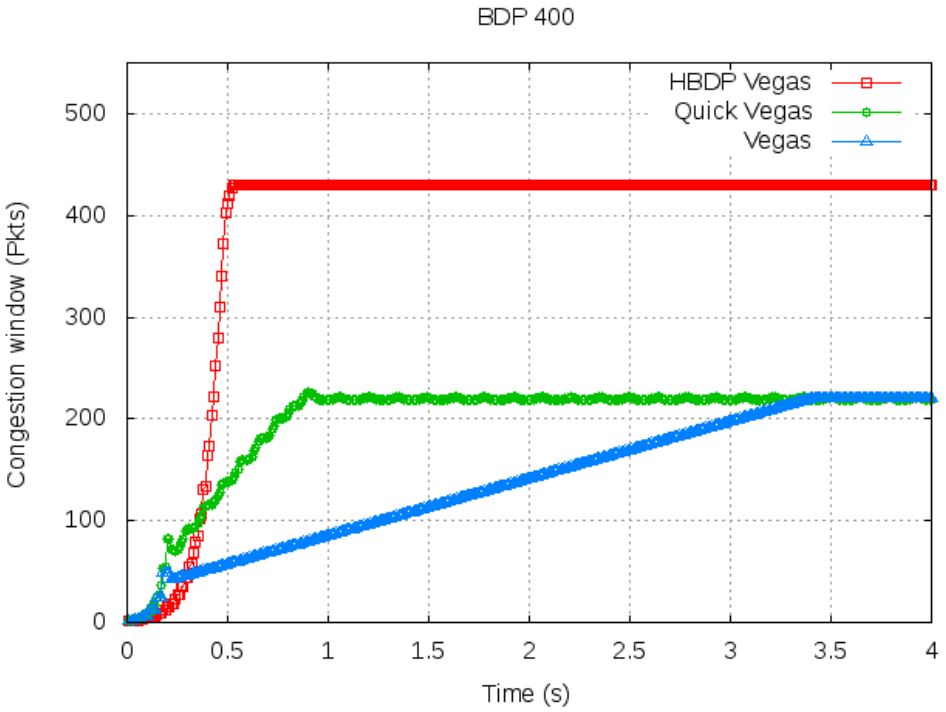


圖 十四：BDP 為 400 時，各版本 TCP 的壅塞視窗變化

在圖十四中，網路的 BDP 為 400 時，HBDP Vegas 在 0.53 秒離開緩啟動階段，同時此時的壅塞視窗大小已經足以能夠讓一個連線能夠掌握大部分的可用頻寬。TCP Vegas 與 Quick Vegas 兩者大約分別在 0.18 秒與 0.21 秒結束了緩啟動階段，但此時兩者的壅塞視窗大小均無法讓連線能有效的掌握網路頻寬，因此 TCP Vegas 與 Quick Vegas 兩者都必須在壅塞避免階段中再花費一段時間來掌握網路頻寬。

5.2.1.4. BDP 為 800 時，各版本 TCP 的壅塞視窗變化

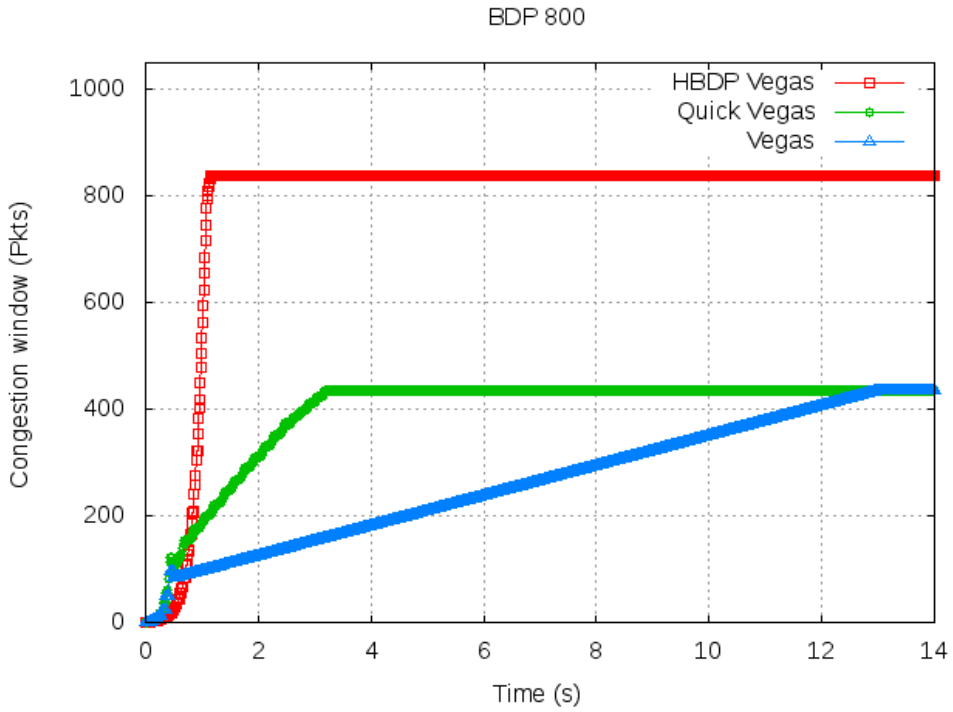


圖 十五 BDP 為 800 時，各版本 TCP 的壅塞視窗變化

在圖十五中，網路的 BDP 為 800 時，HBDP Vegas 在 1.15 秒離開緩啟動階段，同時此時的壅塞視窗大小已經足以能夠讓一個連線能夠掌握大部分的可用頻寬。TCP Vegas 與 Quick Vegas 兩者分別在 0.43 秒與 0.46 秒離開緩啟動階段，但此時兩者的壅塞視窗大小均無法讓連線能有效的掌握網路頻寬，因此 TCP Vegas 與 Quick Vegas 兩者都必須在壅塞避免階段中再花費一段時間來掌握網路頻寬。

5.2.1.5. BDP 為 1000 時，各版本 TCP 的壅塞視窗變化

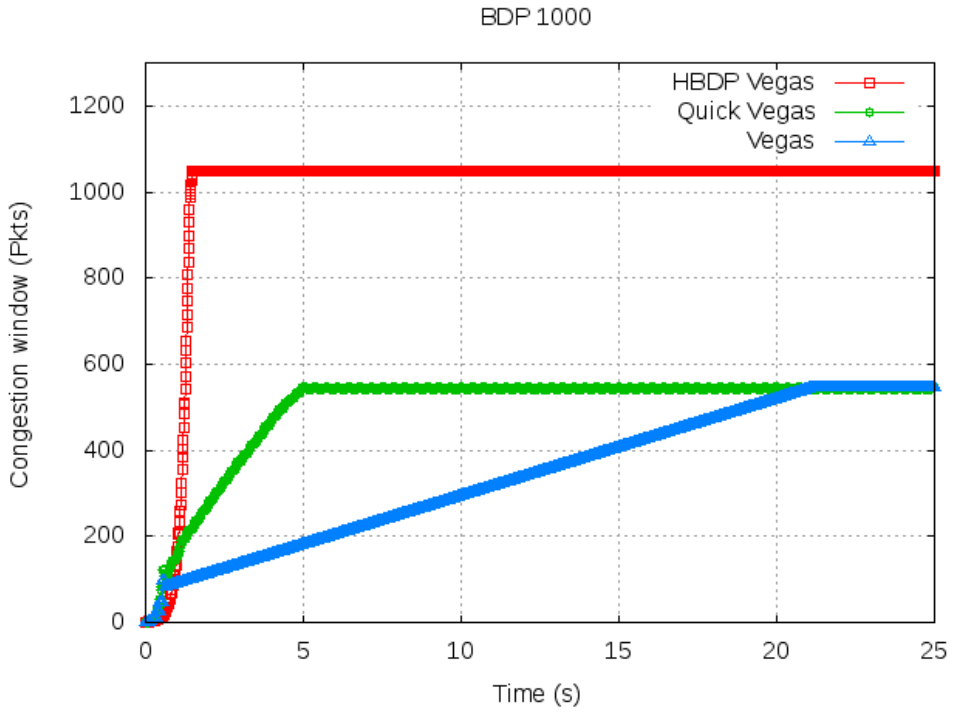


圖 十六：BDP 為 1000 時，各版本 TCP 的壅塞視窗變化

在圖十六中，網路的 BDP 為 1000 時，HBDP Vegas 在 1.48 秒離開緩啟動階段，同時此時的壅塞視窗大小已經足以能夠讓一個連線能夠掌握大部分的可用頻寬。TCP Vegas 與 Quick Vegas 兩者分別在 0.54 秒與 0.57 秒離開緩啟動階段，但此時兩者的壅塞視窗大小均無法讓連線能有效的掌握網路頻寬，因此 TCP Vegas 與 Quick Vegas 兩者都必須在壅塞避免階段中再花費一段時間來掌握網路頻寬。

5.2.1.6. BDP 為 2000 時，各版本 TCP 的壅塞視窗變化

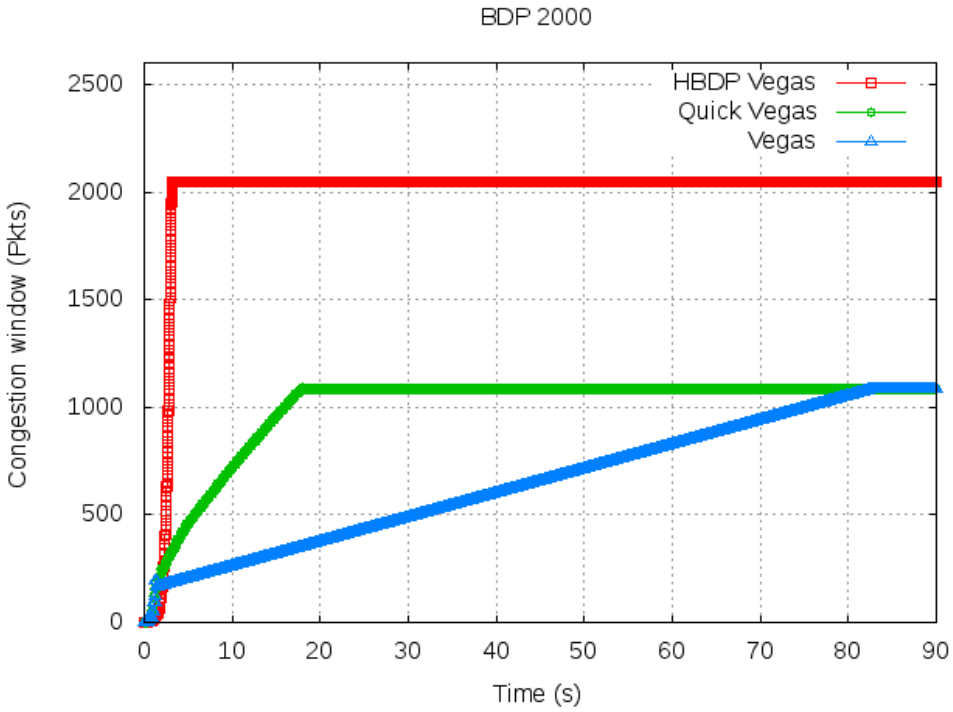


圖 十七：BDP 為 2000 時，各版本 TCP 的壅塞視窗變化

在圖十七中，網路的 BDP 為 2000 時，HBDP Vegas 在 3.2199 秒離開緩啟動階段，同時此時的壅塞視窗大小已經足以能夠讓一個連線能夠掌握大部分的可用頻寬。TCP Vegas 與 Quick Vegas 兩者分別在 1.25 秒與 1.23 秒離開緩啟動階段，但此時兩者的壅塞視窗大小均無法讓連線能有效的掌握網路頻寬，因此 TCP Vegas 與 Quick Vegas 兩者都必須在壅塞避免階段中再花費一段時間來掌握網路頻寬。

5.2.1.7. BDP 為 3000 時，各版本 TCP 的壅塞視窗變化

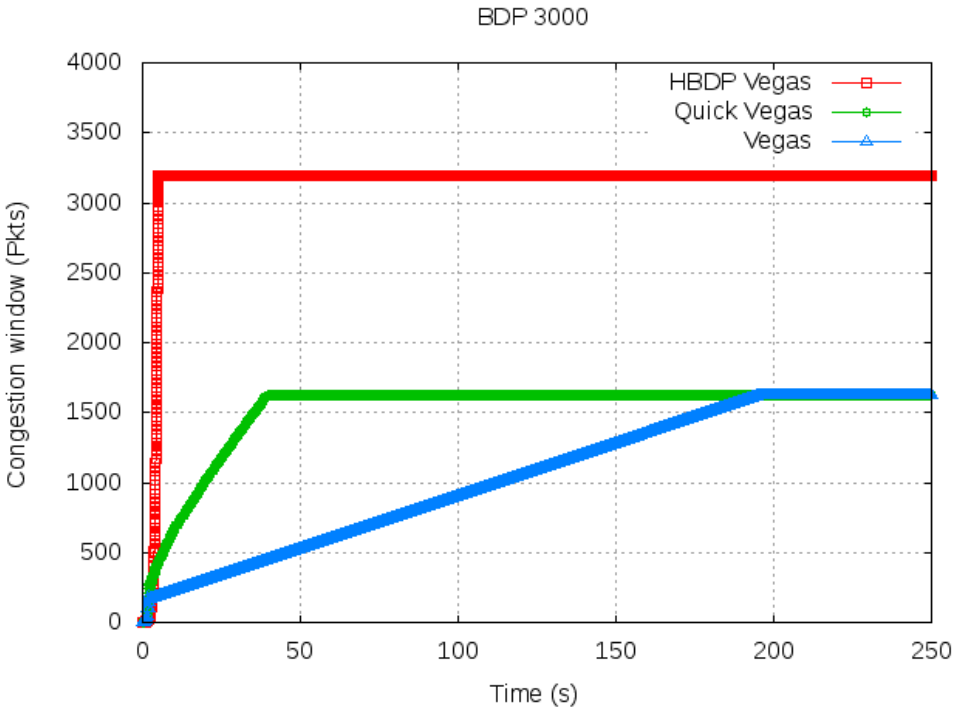


圖 十八：BDP 為 3000 時，各版本 TCP 的壅塞視窗變化

在圖十八中，網路的 BDP 為 3000 時，HBDP Vegas 在 5.1099 秒離開緩啟動階段，同時此時的壅塞視窗大小已經足以能夠讓一個連線能夠掌握大部分的可用頻寬。TCP Vegas 與 Quick Vegas 兩者分別在 1.87 秒到 1.98 秒結束了緩啟動階段，但此時兩者的壅塞視窗大小均無法讓連線能有效的掌握網路頻寬，因此 TCP Vegas 與 Quick Vegas 兩者都必須在壅塞避免階段中再花費一段時間來掌握網路頻寬。

5.2.2. 20 秒以內的吞吐量與頻寬利用率

表格五為 HBDP Vegas、TCP Vegas 與 Quick Vegas 從建立連結開始在 20 秒以內的吞吐量與頻寬利用率。可以發現 HBDP Vegas 在吞吐量與頻寬利用率均高於 TCP Vegas 與 Quick Vegas 兩者。

表格 五：各版本 TCP 從建立連結開始在 20 秒以內的吞吐量與頻寬利用率

各版本 TCP 從建立連結開始在 20 秒以內的吞吐量與頻寬利用率						
	HBDP Vegas		TCP Vegas		Quick Vegas	
BDP	吞吐量	頻寬利用率	吞吐量	頻寬利用率	吞吐量	頻寬利用率
100	99.68Mbps	99.7	99.74 Mbps	99.7	99.75Mbps	99.8
200	99.24Mbps	99.2	98.55Mbps	98.6	99.32Mbps	99.3
400	98.21Mbps	98.2	92.59Mbps	92.6	97.90Mbps	97.9
800	95.87Mbps	95.9	72.49Mbps	72.5	93.05Mbps	93.1
1000	94.62Mbps	94.6	54.10Mbps	54.1	89.19Mbps	89.2
2000	87.87Mbps	87.9	23.59Mbps	23.6	62.94Mbps	62.9
3000	80.61Mbps	80.6	13.13Mbps	13.1	37.60Mbps	37.6

5.2.3. 緩啟動階段的比較結果

綜觀圖十二至圖十八與表格五，我們發現 HBDP Vegas 可以隨著網路 BDP 的提高，能在短時間提高壅塞視窗，這表示 HBDP Vegas 可以在短時間之內迅速的掌握頻寬，此結果可以從表格五中看得出來。反觀 TCP Vegas 與 Quick Vegas 兩者則明顯的無法在短時間內利用緩啟動階段提高自己壅塞視窗，雖然在 BDP 為 800 以後 Quick Vegas 的壅

塞視窗大小有高過 TCP Vegas，但其壅塞視窗大小明顯仍然不夠，此現象表示 TCP Vegas 與 Quick Vegas 兩者無法在短時間之內掌握頻寬。

5.3. 收斂時間的比較

所謂的收斂時間是指一條 TCP 連線從一個穩定的狀態到達另一個穩定的狀態所需花費之時間。在本實驗中我們將會針對 HBDP Vegas、FAST TCP、Quick Vegas 與 TCP Vegas 在搭配一個具有固定傳送速率的 UDP 連線的網路拓樸下，在 BDP 為 1000 的網路環境下，觀察其整體運作過程中三個部份的壅塞視窗變化狀況與收斂時間，這三個部份分別是：

1. 從連線建立之時至穩定
2. 遇到背景流量時至穩定
3. 背景流量消失時至穩定

本實驗所使用的網路拓樸如圖十九所示，網路拓樸中共有兩組發送端與接收端，關於此網路拓樸的詳細的實驗參數則如表格六所示。

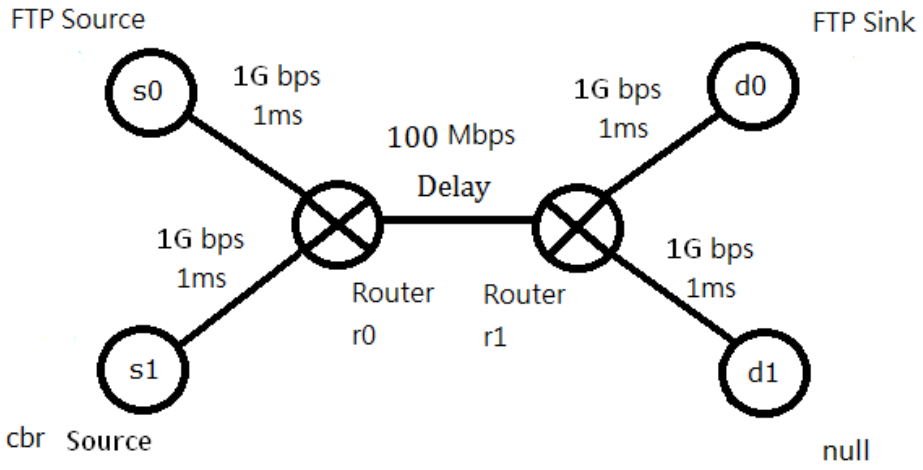


圖 十九：網路拓樸

表格 六：模擬環境參數表

模擬環境參數表	
網路模擬器	NS-2
節點（s0 與 d0）	● 使用檔案傳輸控制協定（FTP）來作資料傳輸。 ● FTP Source（s0）經由 r0 連上網路，並將資料發送給 FTP Sink（d0），並持續一段時間。
節點（s1 與 d1）	● 採用固定傳送速率的 UDP 連結。 ● cbr Source（s1）在等待 FTP Sink（s0）開始傳輸資料一段時間後，接著也啟動傳輸並經由 r0 連上網路，並產生一個具有固定傳送速率的連結把資料發送給 null（d1）並持續一段時間。
兩個路由器（r0、r1）相互連接線段之間網路頻寬	100Mbps
兩個路由器（r0、r1）相互連接線段之間傳輸延遲時間	20.05792ms
TCP Vegas **α、β、γ**	α＝2、β＝4、γ＝2
Quick Vegas **α、β、γ**	α＝2、β＝4、γ＝2

FAST TCP **α**	529
HBDP Vegas **門檻值 K'**	0.7

接下來，關於實驗過程中觀察 HBDP Vegas、FAST TCP、Quick Vegas 與 TCP Vegas 其整體運作過程中三個部份的壅塞視窗變化狀況與收斂時間將會分別呈現在 5.3.1 小節、5.3.2 小節與 5.3.3 小節中，最後會在 5.3.4 小節說明收斂時間的比較結果。

5.3.1. 連線建立之時至穩定的收斂時間

如圖二十，從連線建立之時至穩定，在這一階段中 HBDP Vegas 與 FAST TCP 分別花費 1.48 秒與 1.03 秒，Quick Vegas 與 TCP Vegas 分別花費 5.13 秒與 21.05 秒。

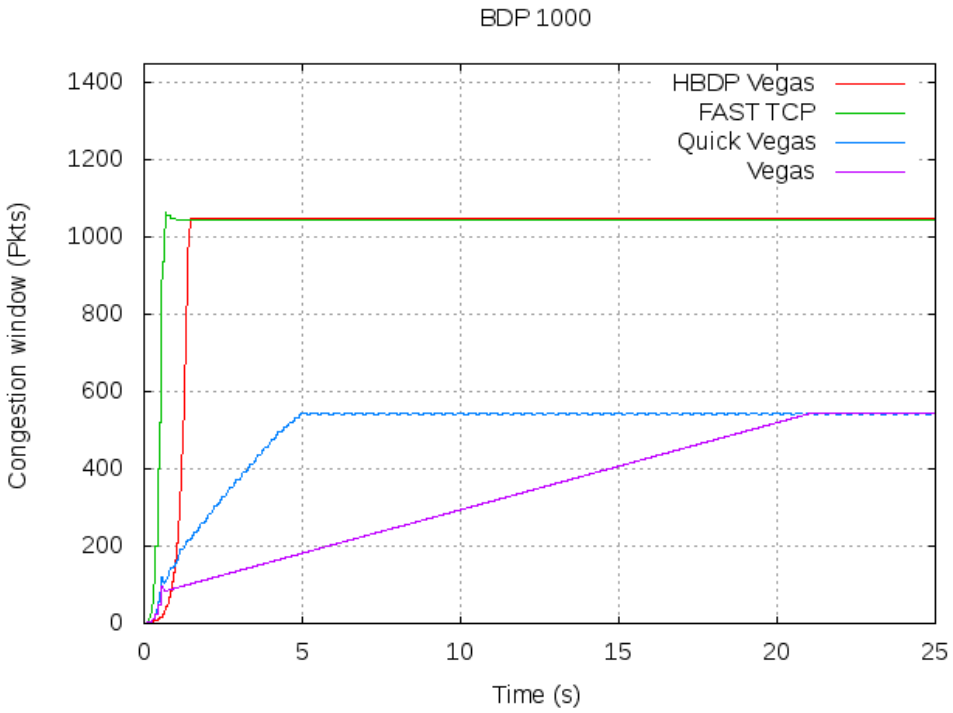


圖 二十：從連線建立之時至穩定的壅塞視窗變化狀況

5.3.2. 遇到背景流量時至穩定時的收斂時間

如圖二十一與圖二十二，背景流量於第 30 秒開始運作至穩定。在圖二十一中此階段的 HBDP Vegas 與 FAST TCP 分別花費 0.76 秒與 1.49 秒。在圖二十二中此階段的 Quick Vegas 與 TCP Vegas 分別花費 1.50 秒與 18.21 秒。

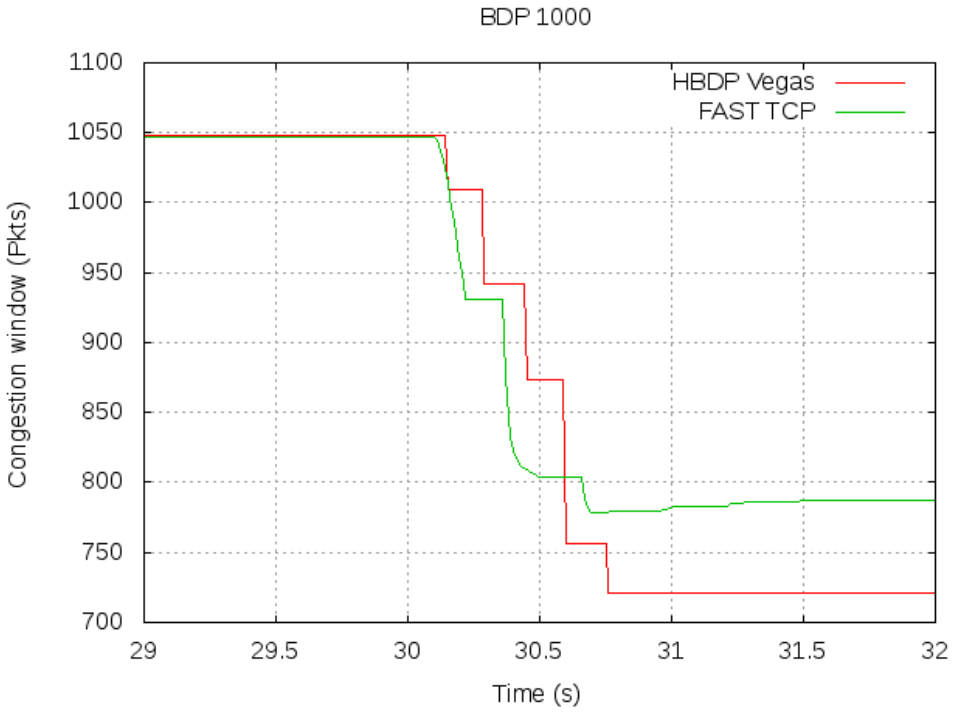


圖 二十一：遇到背景流量時至穩定時壅塞視窗變化狀況 1

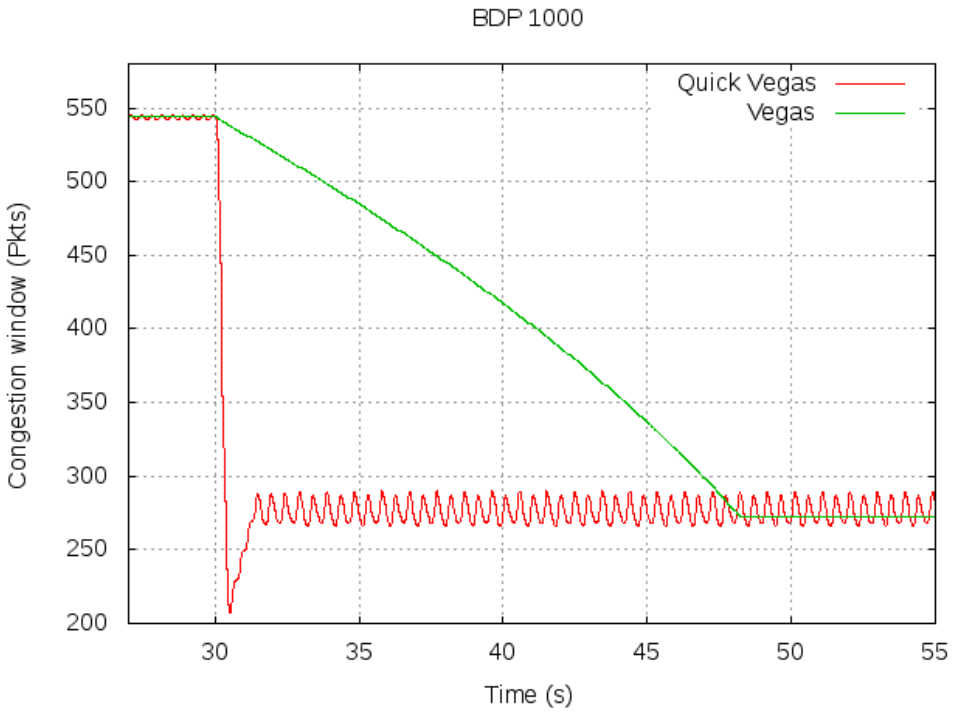


圖 二十二：遇到背景流量時至穩定時壅塞視窗變化狀況 2

5.3.3. 背景流量消失時至穩定的收斂時間

背景流量於第 80 秒停止運作至穩定，分別呈現在圖二十三與圖二十四中，在圖二十三中此階段的 HBDP Vegas 與 FAST TCP 花費時間為 0.36 秒與 0.6 秒，在圖二十四中此階段的 Quick Vegas 與 TCP Vegas 花費時間為 1.21 秒與 12.13 秒。

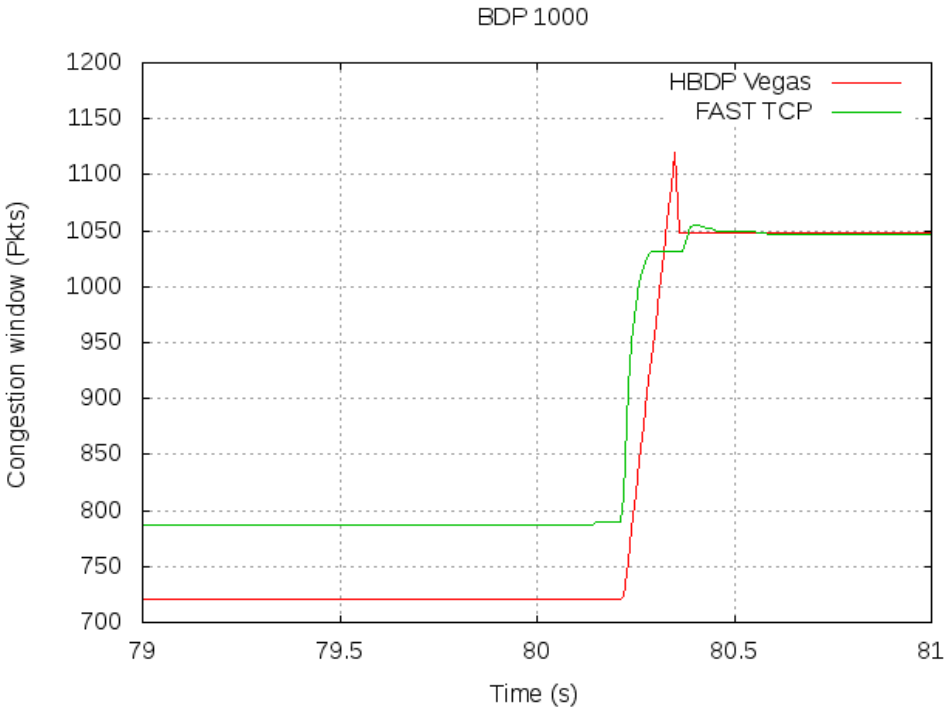


圖 二十三：背景流量消失時至穩定的壅塞視窗變化狀況 1

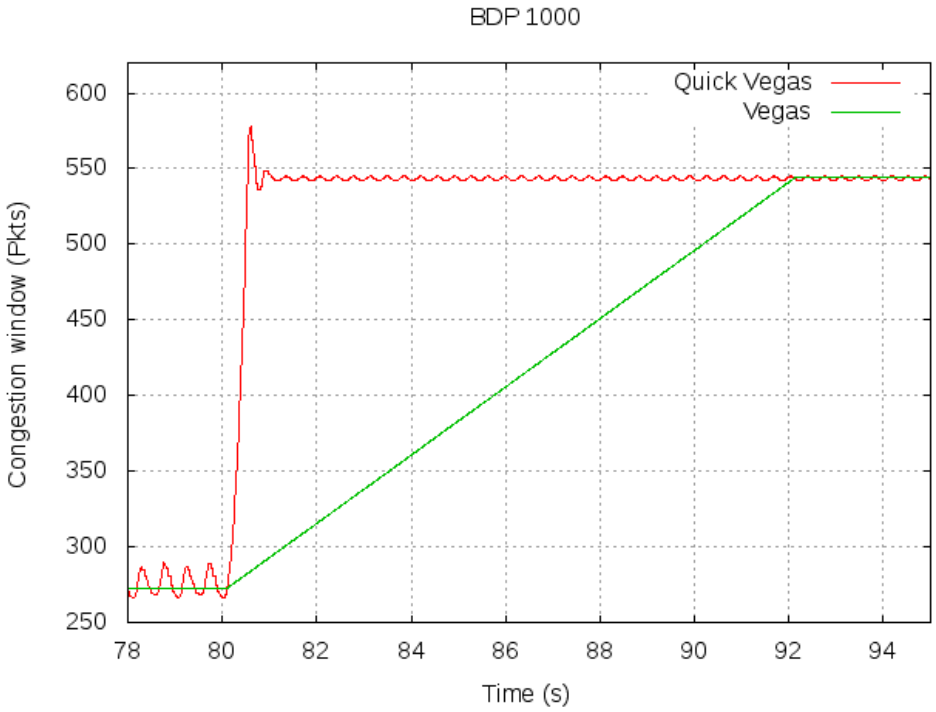


圖 二十四：背景流量消失時至穩定的壅塞視窗變化狀況 2

5.3.4. 收斂時間的比較結果

綜觀圖二十至圖二十四我們可以發現，在圖二十中，Quick Vegas 與 TCP Vegas 的收斂時間均大於 HBDP Vegas 與 FAST TCP，其原因是 Quick Vegas 與 TCP Vegas 太早離開緩啟動階段，導致必須花費大量時間在壅塞避免階段才能達到對網路頻寬的完全利用，此狀況 TCP Vegas 更為明顯。

在圖二十一至圖二十四中，Quick Vegas 與 TCP Vegas 的收斂時間均大於 FAST TCP 與 HBDP Vegas，故以上的現象則是表示 Quick Vegas 與 TCP Vegas 並無法迅速的應付背景流量的影響，這一點在其中 TCP Vegas 更為明顯。另外 FAST TCP 是單純的利用自身的壅塞視窗調整公

式來快速改變壅塞視窗。HBDP Vegas 是利用動態調整 α 與 β 和經修改過的壅塞視窗成長幅度與成長策略來達成快速改變壅塞視窗的目標，雖然 Quick Vegas 可以利用參數 succ 累加的次數改變成長幅度來達到快速改變壅塞視窗的目標，但是其 α 與 β 是固定的值，故無法更進一步對壅塞視窗大小做調整。TCP Vegas 的調整幅度則是過於保守。

最後，同樣的再次利用圖十九的網路拓樸，透過修改路由器（r0）與路由器（r1）之間的傳輸延遲時間，形成不同的 BDP 的網路環境之下三個部分的收斂時間。其結果列於下方表格七，其中從表格七中的結果我們可以發現其數據結果和上述的現象十分接近。

表格 七：在不同的頻寬延遲乘積的網路環境之下三個部分的收斂時間

不同的頻寬延遲乘積的網路環境之下三個部分的收斂時間　　單位：秒												
	HBDP Vegas			FAST TCP			Quick Vegas			TCP Vegas		
BDP	第一部分	第二部分	第三部分	第一部分	第二部分	第三部分	第一部分	第二部分	第三部分	第一部分	第二部分	第三部分
100	0.10	0.08	0.05	0.16	0.21	0.15	0.12	2.24	0.10	0.13	0.21	0.14
200	0.24	0.15	0.08	0.29	0.41	0.30	0.33	0.47	0.20	0.75	0.79	0.52
400	0.53	0.3	0.17	0.46	0.71	0.42	0.97	0.77	1.84	3.40	3.0	1.99
800	1.15	0.52	0.29	0.83	1.23	0.57	3.22	1.63	0.79	13.02	11.75	7.80
1000	1.48	0.76	0.36	1.03	1.49	0.60	5.13	1.50	1.12	21.05	18.21	12.13
2000	3.21	1.42	0.65	1.97	2.77	1.14	18.31	5.47	2.77	82.50	72.11	48.0
3000	5.10	1.83	1.06	3.08	4.55	1.61	39.65	15.02	4.38	195.16	161.98	107.86

5.4. 面對其它連線競爭時之狀況

5.4.1 面對相同版本 TCP 協定之競爭情形

在本次實驗中我們透過兩個相同的壅塞控制機制在 BDP 為 1000 的網路環境下的競爭來觀察 HBDP Vegas、FAST TCP、Quick Vegas 與 TCP Vegas，收斂的狀況。

本實驗所使用的網路拓樸則如圖二十五所示，網路拓樸中共有兩組發送端與接收端，關於此網路拓樸的詳細的實驗參數則如表格八所示。

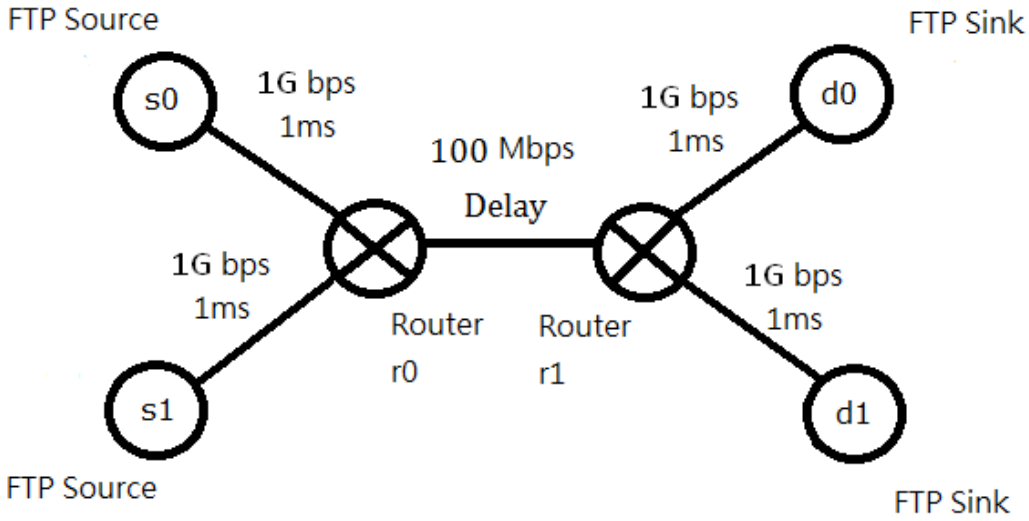


圖 二十五：網路拓樸

表格 八：模擬環境參數表

模擬環境參數表	
網路模擬器	NS-2
節點（s0 與 d0）	● 使用檔案傳輸控制協定（FTP）來作資料傳輸。 ● FTP Source（s0）經由 r0 連上網路，並將資料發送給 FTP Sink（d0），並持續 160 秒。
節點（s1 與 d1）	● 使用檔案傳輸控制協定（FTP）來作資料傳輸。 ● FTP Source（s1）在第 80 秒啟動傳輸並經由 r0 連上網路，並將並將資料發送給 FTP Sink（d1）並持續到第 160 秒。
兩個路由器（r0、r1）相互連接線段之間網路頻寬	100Mbps
兩個路由器（r0、r1）相互連接線段之間傳輸延遲時間	20.05792ms
TCP Vegas **α、β、γ**	α＝2、β＝4、γ＝2
Quick Vegas **α、β、γ**	α＝2、β＝4、γ＝2
FAST TCP **α**	529
HBDP Vegas **門檻值 K'**	0.7

在實驗結果部分，HBDP Vegas、FAST TCP、Quick Vegas 與 TCP Vegas 運作的結果將呈現在圖二十六至圖二十九，同時將在此小節的最後說明實驗的結果。

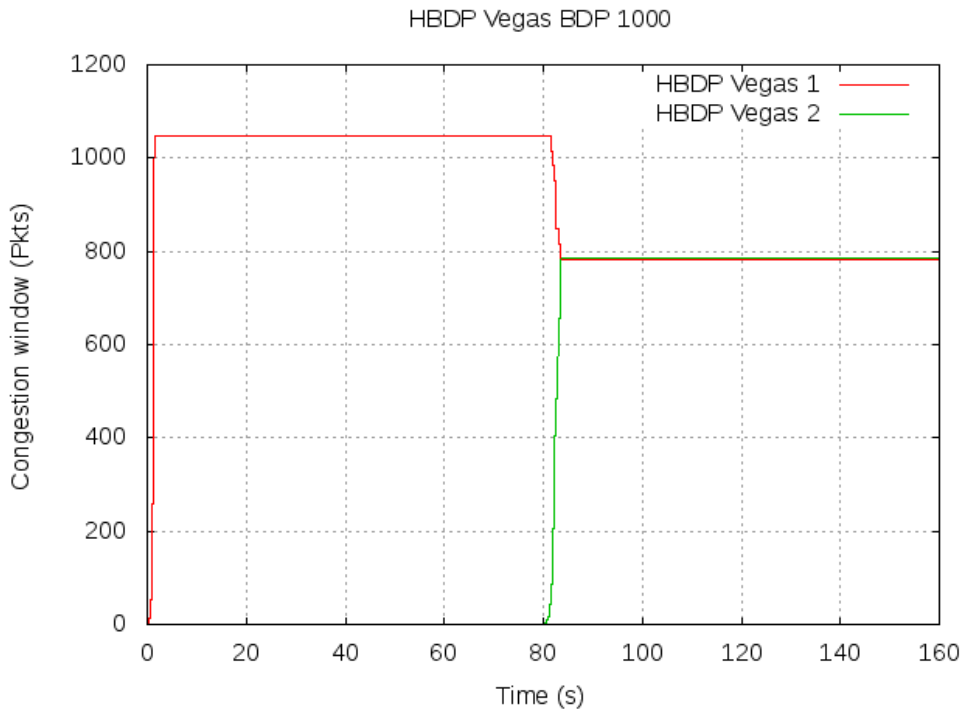


圖 二十六：兩個 HBDP Vegas 鏈結相互競爭過程與壅塞視窗變化

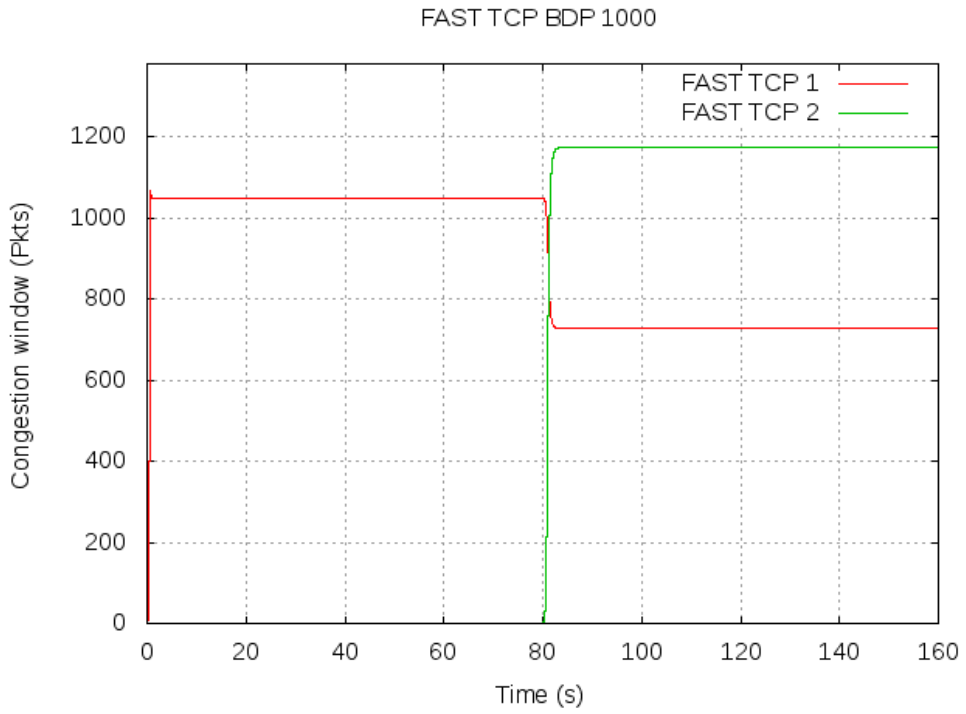


圖 二十七：兩個 FAST TCP 鏈結相互競爭過程與壅塞視窗變化

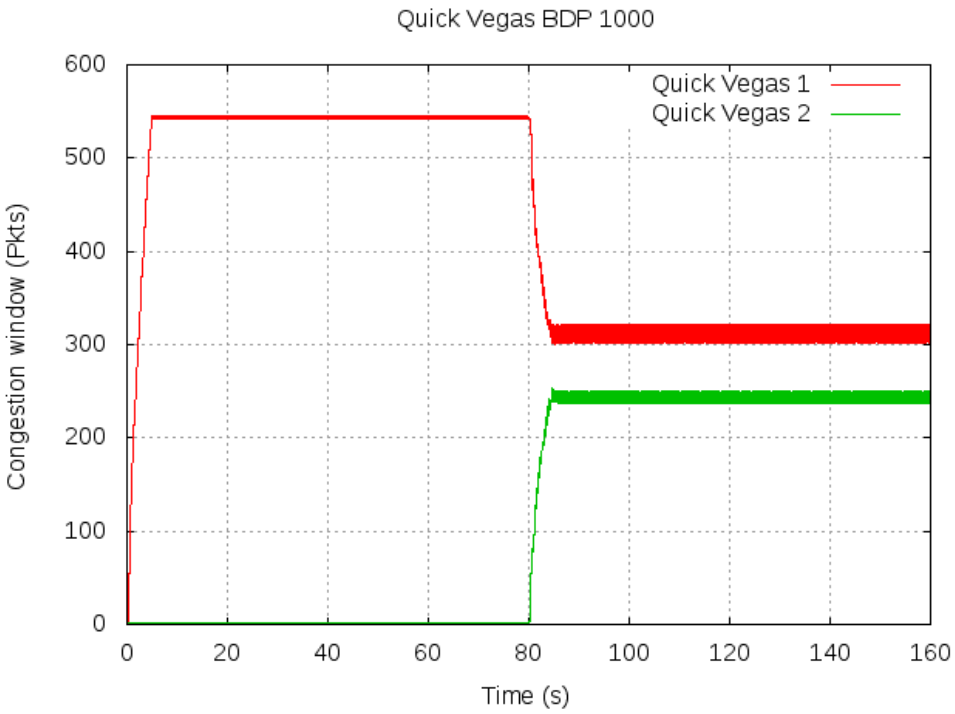


圖 二十八：兩個 Quick Vegas 鏈結相互競爭過程與壅塞視窗變化

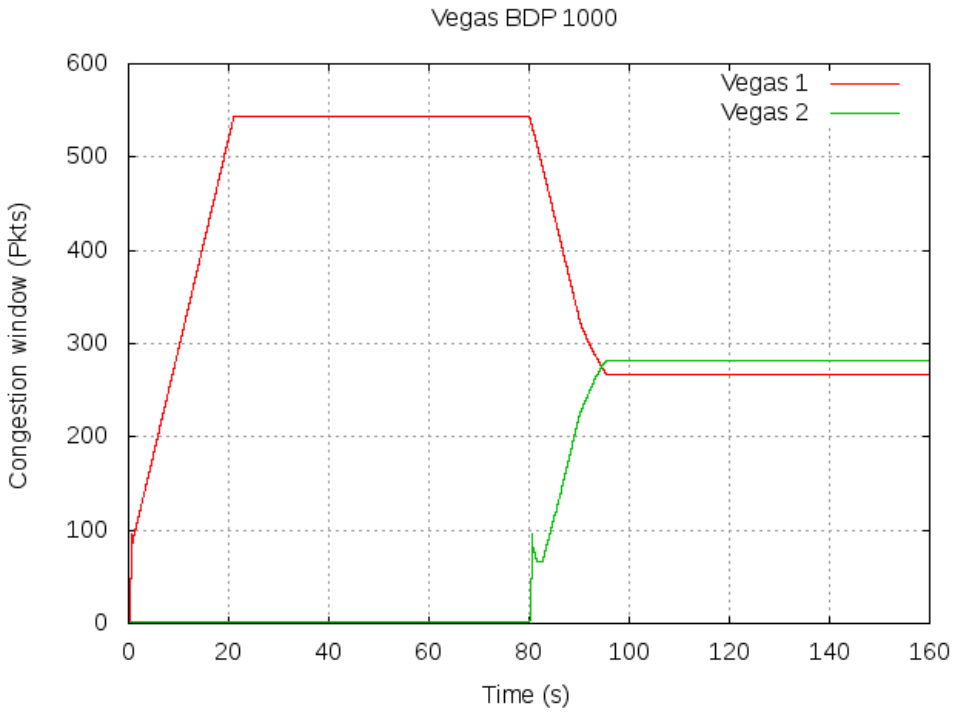


圖 二十九：兩個 TCP Vegas 鏈結相互競爭過程與壅塞視窗變化

綜觀圖二十六至圖二十九，可以發現在第 80 秒時一條新連線加入網路時，HBDP Vegas 於 83 秒兩條連線的壅塞視窗調達到穩定，接著 FAST TCP 也在 83 秒兩條連線的壅塞視窗調達到穩定，接下來是 Quick Vegas 於 84 秒兩條連線的壅塞視窗調達到穩定並在一固定範圍內穩定的持續震盪，最後是 TCP Vegas 於 95 秒兩條連線的壅塞視窗調達到穩定。

但是從圖二十六至圖二十九中可以發現，Quick Vegas、TCP Vegas 與 HBDP Vegas 在另一條新連結加入網路時，最後都可以與新連線平分網路頻寬，但是 FAST TCP 的兩條連線卻出現了明顯的差距，此問題與 FAST TCP 的壅塞視窗控制機制有關，因為 FAST TCP 是透過來回時間 RTT 的平均值並利用公式（7）來調整壅塞視窗大小，若當下網路環境中已有其他連線存在時，較晚建立的連線其所量測的來回時間 RTT 亦會受到影響並連帶使得自身的壅塞視窗大小控制機制受影響。

5.4.2. 與不同流量競爭時

在本次實驗中，我們將探討在 BDP 為 1000 的網路環境下，分別透過三種不同固定傳輸速率（25Mb、50Mb、75Mb）之連線，並搭配 HBDP Vegas、FAST TCP、Quick Vegas 以及 TCP Vegas 與之競爭觀察其運作狀況與吞吐量變化。

本實驗所使用的網路拓樸如圖三十所示，在此網路拓樸中一共使用兩組發送端與接收端，關於此網路拓樸的詳細的實驗參數則如表格九所示。

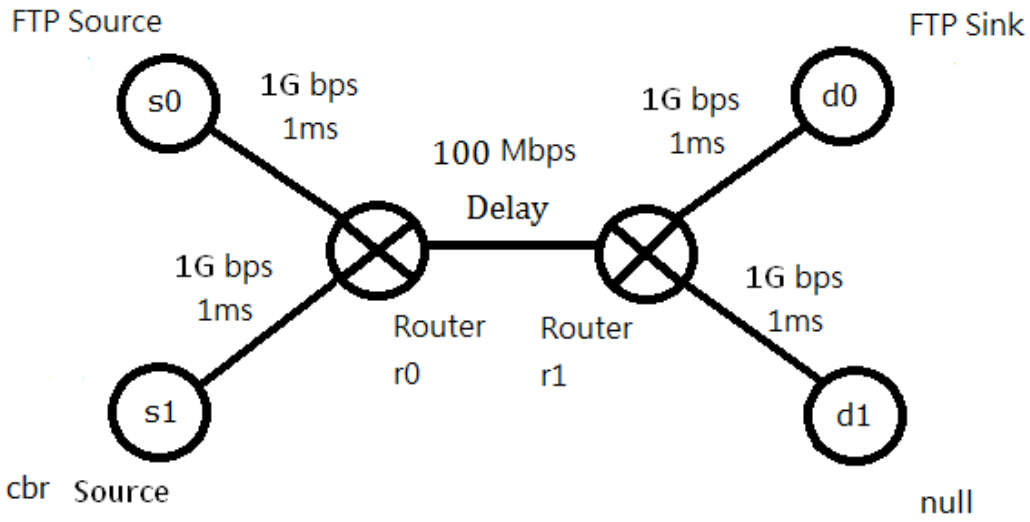


圖 三十：實驗環境拓樸

表格 九：模擬環境參數表

模擬環境參數表	
網路模擬器	NS-2
節點（s0 與 d0）	● 使用檔案傳輸控制協定（FTP）來作資料傳輸。 ● FTP Source（s0）經由 r0 連上網路，並將資料發送給 FTP Sink（d0），並持續 200 秒。
節點（s1 與 d1）	● 採用固定傳送速率的 UDP 連結。 ● cbr Source（s1）在第 60 秒啟動傳輸並經由 r0 連上網路，並將產生一個具有固定傳送速率的連結把資料發送給 null（d1）並持續到第 120 秒。
兩個路由器（r0、r1）相互連接線段之間網路頻寬	100Mbps
兩個路由器（r0、r1）相互連接線段之間傳輸延遲時間	20.05792ms
TCP Vegas **α、β、γ**	α＝2、β＝4、γ＝2
Quick Vegas **α、β、γ**	α＝2、β＝4、γ＝2
FAST TCP **α**	529
HBDP Vegas **門檻值 K'**	0.7

接下來，關於實驗過程中觀察 HBDP Vegas、FAST TCP、Quick Vegas 與 TCP Vegas 其整體運作過程中的壅塞視窗與吞吐量變化將會分別呈現在 5.4.2.1 小節、5.4.2.2 小節與 5.4.2.3 小節中，最後會在 5.4.2.4 小節說明實驗結果。

5.4.2.1. 在 25Mb/s 之固定傳輸速率下壅塞視窗與吞吐量變化狀況

觀察圖三十一與圖三十二，在壅塞視窗變化方面，可以發現 FAST TCP、HBDP Vegas、Quick Vegas，遇到背景流量運作時，均可快速的控制壅塞視窗的大小，TCP Vegas 則是明顯的緩慢許多。再遇到背景流量運作時調整後的壅塞視窗變化，FAST TCP、HBDP Vegas 在調整壅塞視窗後可以達到穩定，反觀 Quick Vegas 在調整壅塞視窗後，壅塞視窗則在一個固定範圍內震盪。

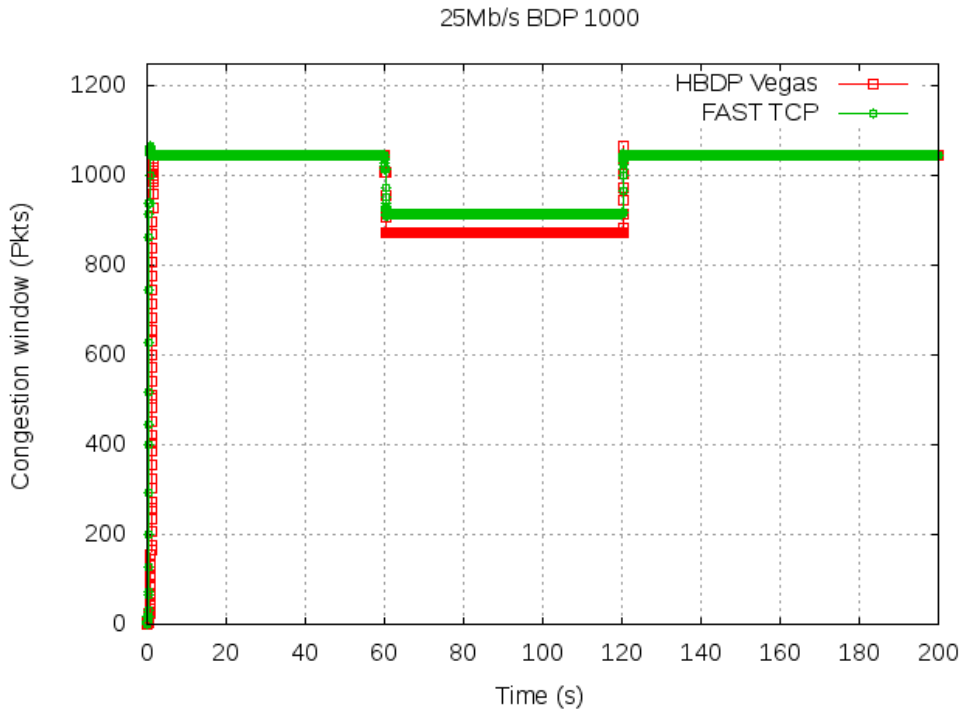


圖 三十一：在 25Mb/s 之固定傳輸速率下 FAST TCP 與 HBDP Vegas 的壅塞視窗變化

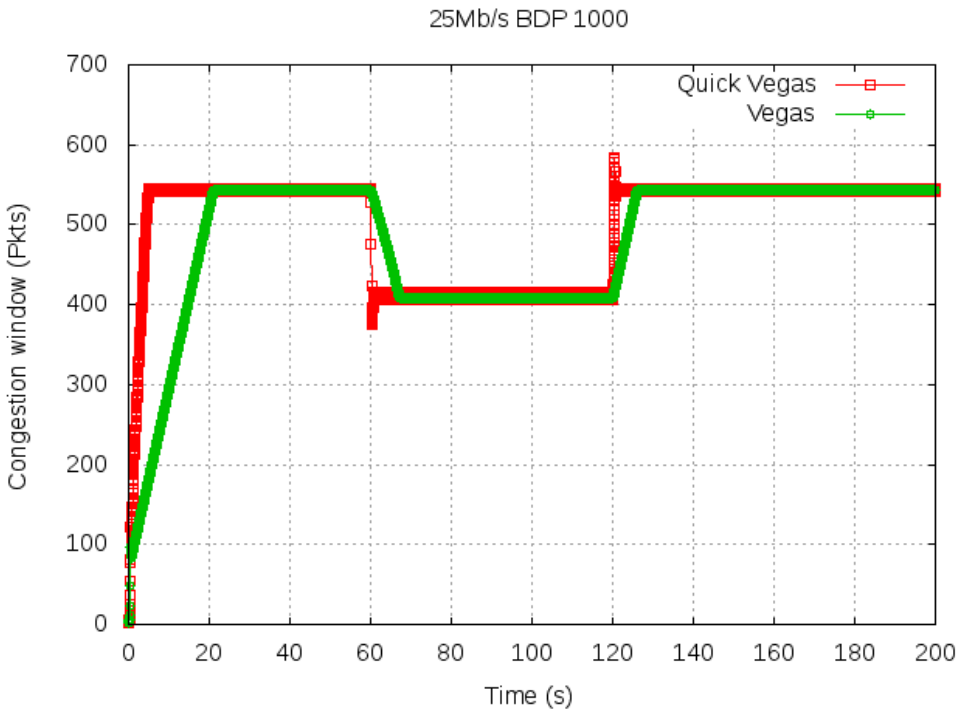


圖 三十二：在 25Mb/s 之固定傳輸速率下 Quick Vegas 與 TCP Vegas 的壅塞視窗變化

觀察圖三十三，在吞吐量的變化方面，FAST TCP 與 HBDP Vegas 在緩啟動階段與背景流量消失後其吞吐量均可在短時間內恢復至最大與維持穩定，Quick Vegas 與 TCP Vegas 在緩啟動階段與背景流量消失後其吞吐量恢復速度較慢，其中以 TCP Vegas 較明顯，其原因是在緩啟動階段提前結束，與其壅塞避免階段視窗調整幅度過小所致。

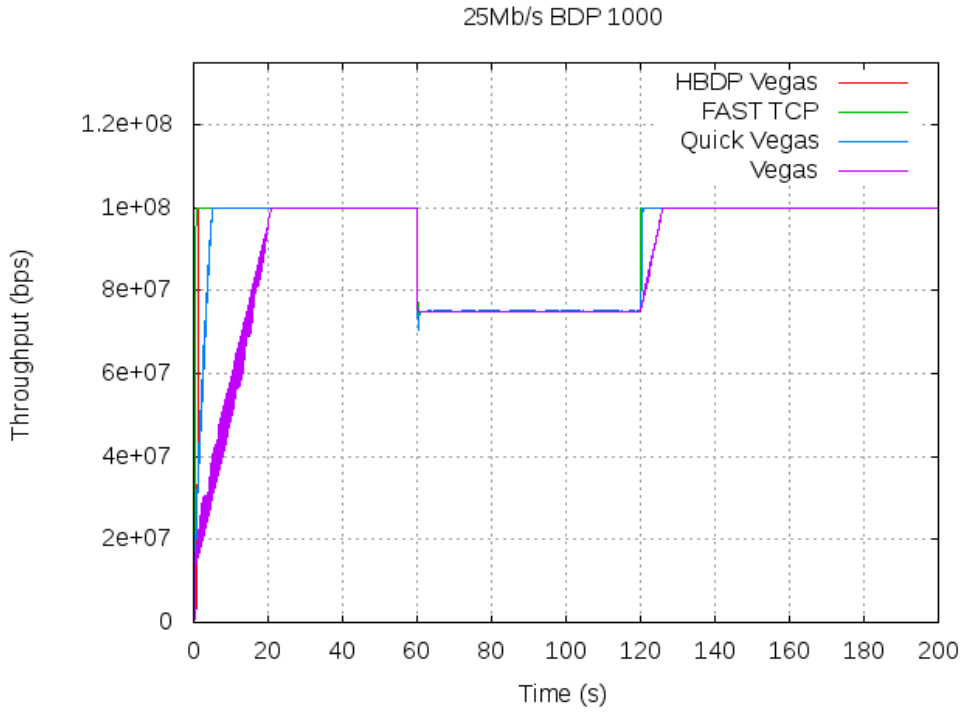


圖 三十三： 在 25Mb/s 之固定傳輸速率下 FAST TCP、HBDP Vegas、Quick Vegas 與 TCP Vegas 的吞吐量變化

5.4.2.2. 在 50Mb/s 之固定傳輸速率下壅塞視窗與吞吐量變化狀況

觀察圖三十四與三十五，在壅塞視窗變化方面，可以發現 FAST TCP、HBDP Vegas、Quick Vegas，遇到背景流量運作時，均可快速的控制壅塞視窗的大小，TCP Vegas 則是明顯的緩慢許多。再遇到背景流量運作時調整後的壅塞視窗變化，FAST TCP、HBDP Vegas 在調整壅塞視窗後可以達到穩定，反觀 Quick Vegas 在調整壅塞視窗後，壅塞視窗則在一個固定範圍內震盪。

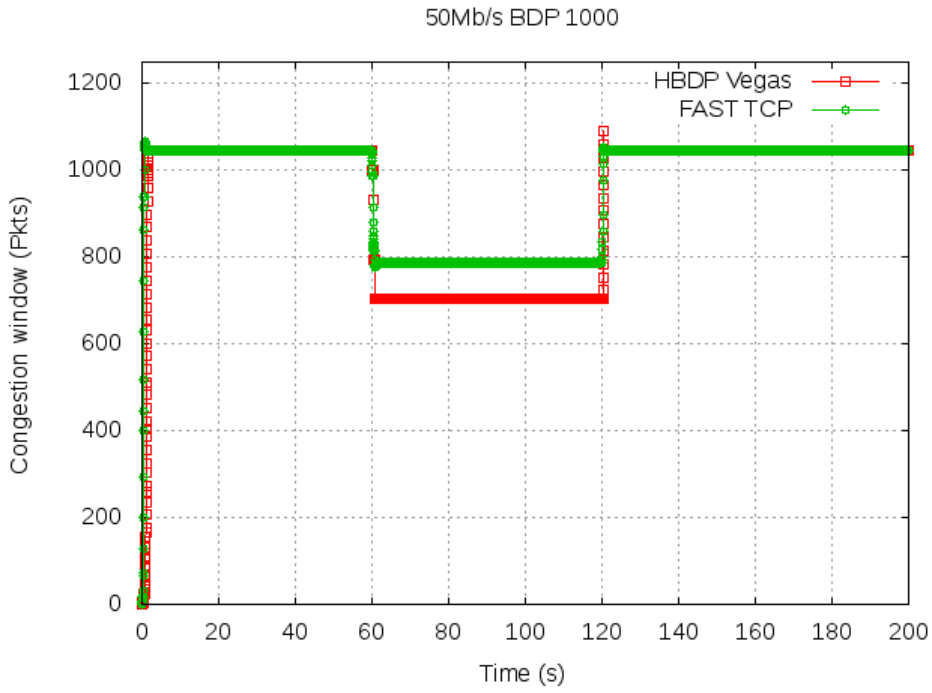


圖 三十四：在 50Mb/s 之固定傳輸速率下 FAST TCP 與 HBDP Vegas 的壅塞視窗變化

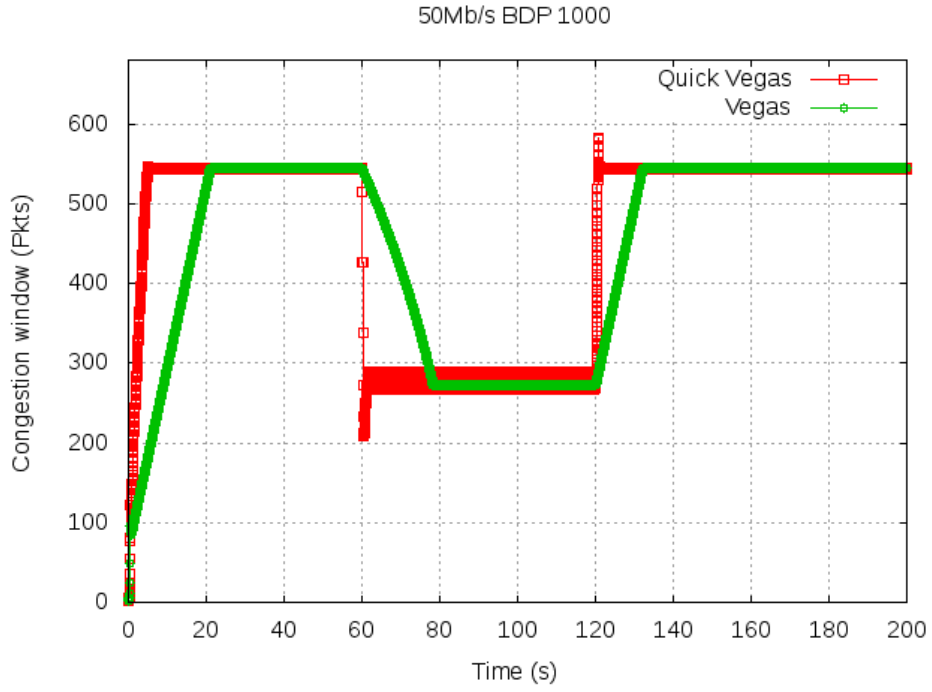


圖 三十五：在 50Mb/s 之固定傳輸速率下 Quick Vegas 與 TCP Vegas 的壅塞視窗變化

觀察圖三十六，在吞吐量的變化方面，FAST TCP 與 HBDP Vegas 在緩啟動階段與背景流量消失後其吞吐量均可在短時間內恢復至最大與維持穩定，Quick Vegas 與 TCP Vegas 在緩啟動階段與背景流量消失後其吞吐量恢復速度較慢，其中 Quick Vegas 在第 60 秒後因為自身壅塞視窗調整幅度過大導致吞吐量一度明顯降低，TCP Vegas 則是塞視窗調整能力明顯不理想，其原因是在緩啟動階段提前結束，與其壅塞避免階段視窗調整幅度過小所致。

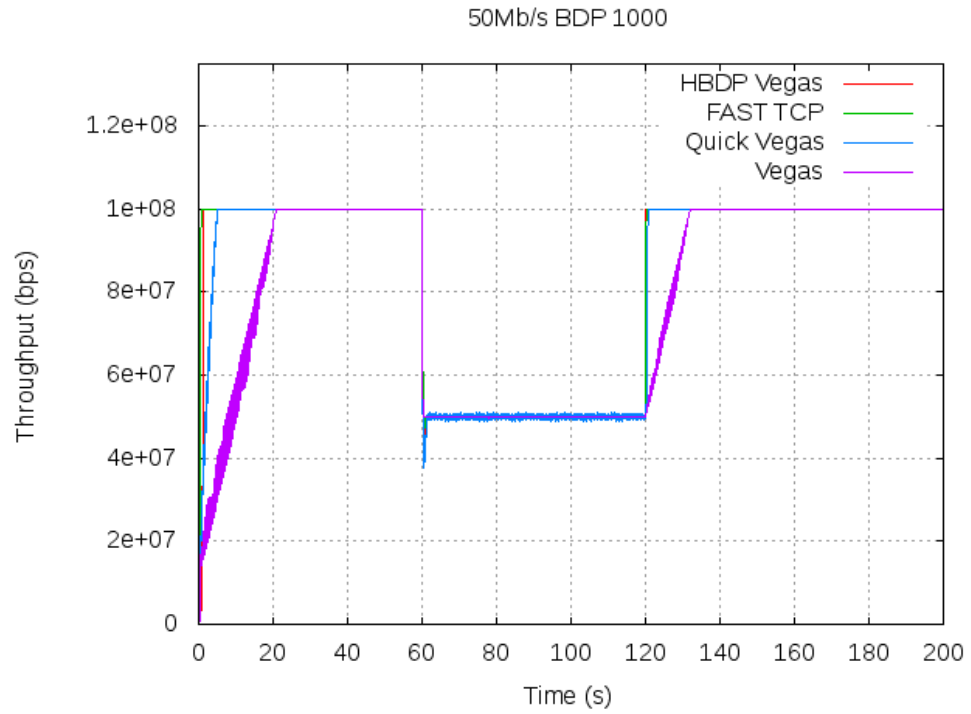


圖 三十六： 在 50Mb/s 之固定傳輸速率下 FAST TCP、HBDP Vegas、Quick Vegas 與 TCP Vegas 的吞吐量變化

5.4.2.3. 在 75Mb/s 之固定傳輸速率下壅塞視窗與吞吐量變化狀況

觀察圖三十七、三十八與三十九，在壅塞視窗變化方面，可以發現 FAST TCP、HBDP Vegas、Quick Vegas，遇到背景流量運作時，均可快速的控制壅塞視窗的大小，TCP Vegas 則是明顯的非常緩慢。再遇到背景流量運作時調整後的壅塞視窗變化，FAST TCP、HBDP Vegas 在調整壅塞視窗後可以達到穩定，反觀 Quick Vegas 在調整壅塞視窗後，壅塞視窗則在一個固定範圍內震盪。

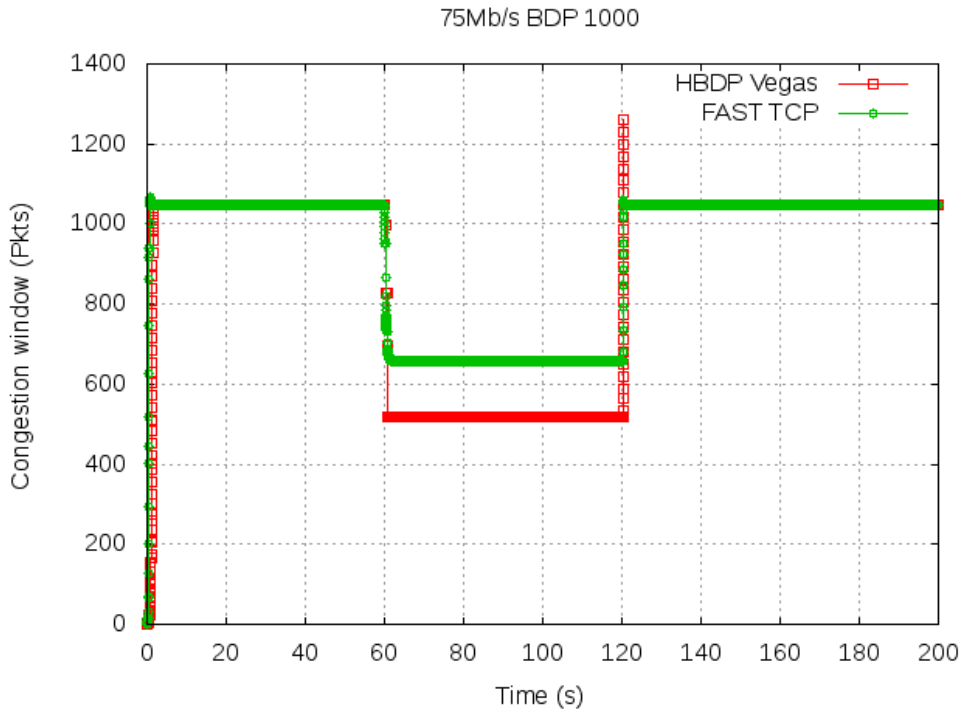


圖 三十七：在 75Mb/s 之固定傳輸速率下 FAST TCP 與 HBDP Vegas 的壅塞視窗變化

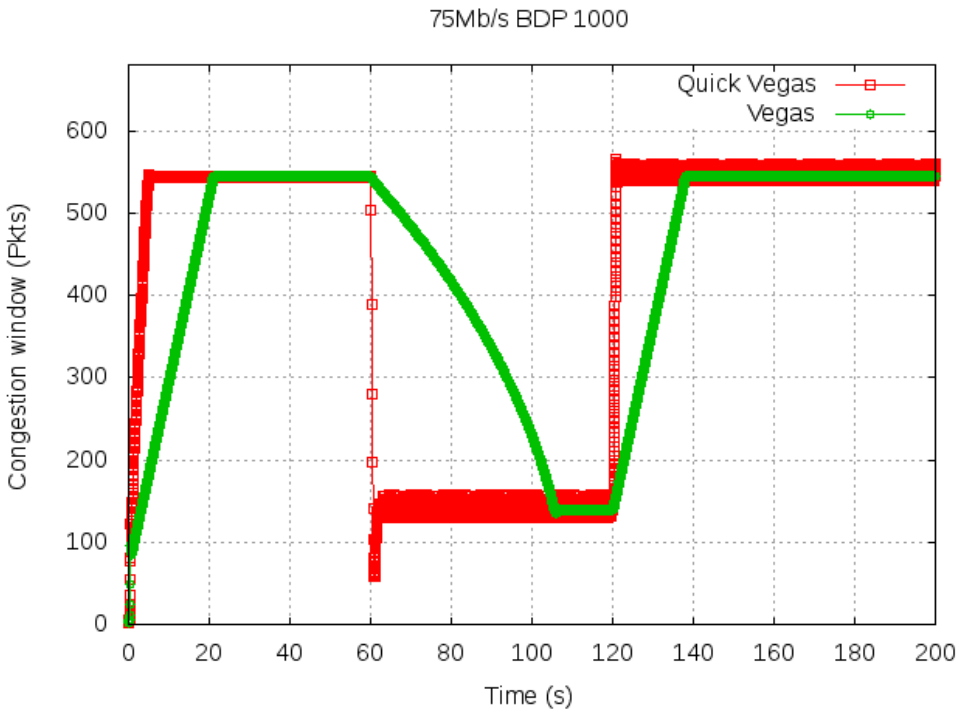


圖 三十八：在 75Mb/s 之固定傳輸速率下 Quick Vegas 與 TCP Vegas 的壅塞視窗變化

在吞吐量的變化方面，FAST TCP 與 HBDP Vegas 在緩啟動階段與背景流量消失後其吞吐量均可在短時間內恢復至最大與維持穩定，Quick Vegas 與 TCP Vegas 在緩啟動階段與背景流量消失後其吞吐量恢復速度較慢，其中 Quick Vegas 在第 60 秒後因為自身壅塞視窗調整幅度過大導致吞吐量一度明顯降低，TCP Vegas 則是塞視窗調整能力明顯不足，其原因是在緩啟動階段提前結束，與其壅塞避免階段視窗調整幅度過小所致。

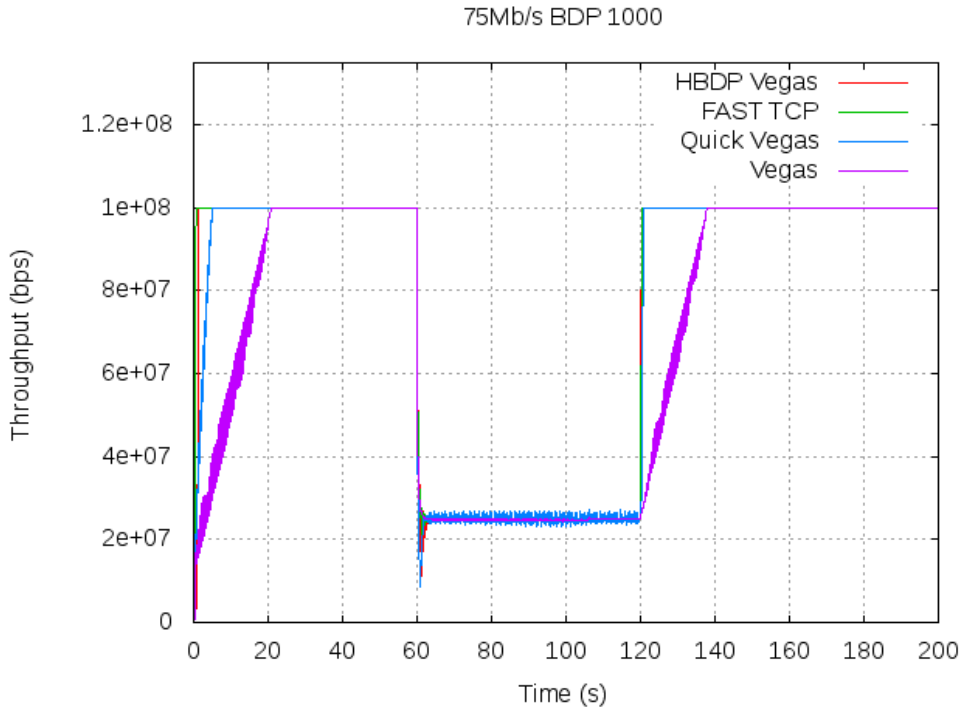
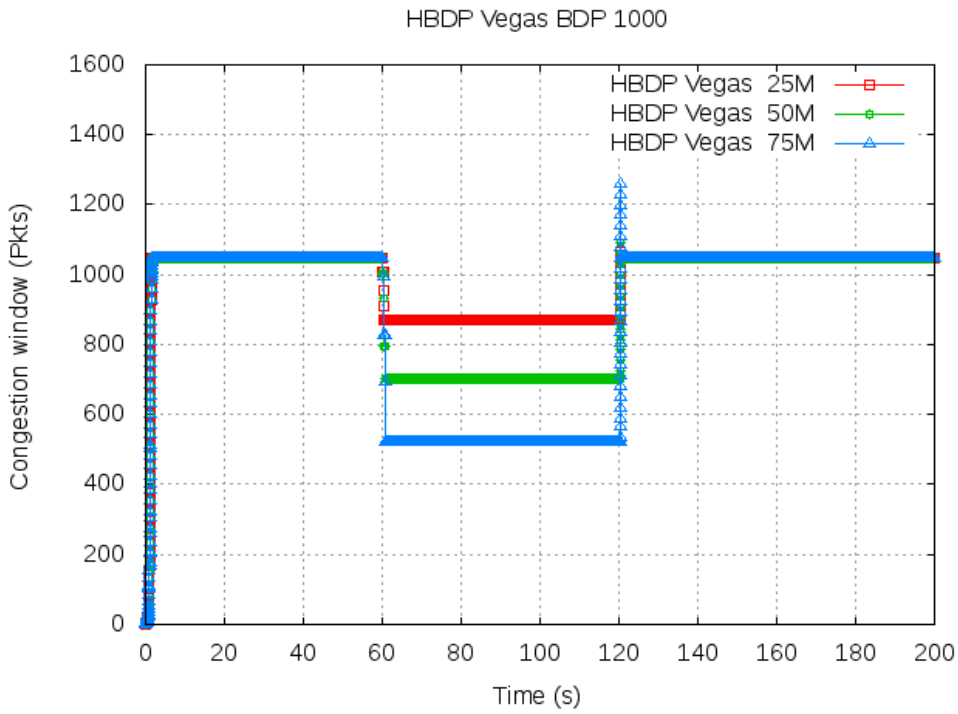


圖 三十九： 在 75Mb/s 之固定傳輸速率下 FAST TCP、HBDP Vegas、Quick Vegas 與 TCP Vegas 的吞吐量變化

5.4.2.4. 與不同流量競爭之結果

綜觀 5.4.2.1 小節、5.4.2.2 小節與 5.4.2.3 小節的實驗數據與圖表，可以發現 HBDP Vegas 與 FAST TCP 在與不同的流量競爭時，如圖四十至圖四十三所示，其壅塞視窗大小與吞吐量的變化，均表示兩者可以有效的掌握網路頻寬資源與提供穩定的吞吐量。



圖四十：HBDP Vegas 與不同的流量競爭時，壅塞視窗大小的變化

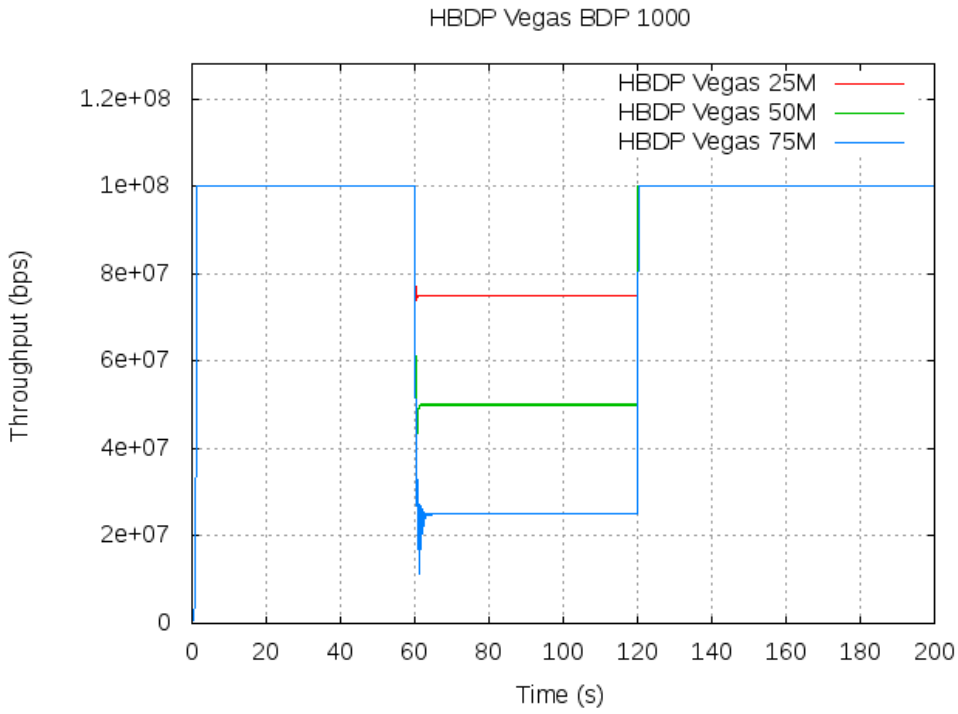


圖 四十一：HBDP Vegas 與不同的流量競爭時，吞吐量的變化

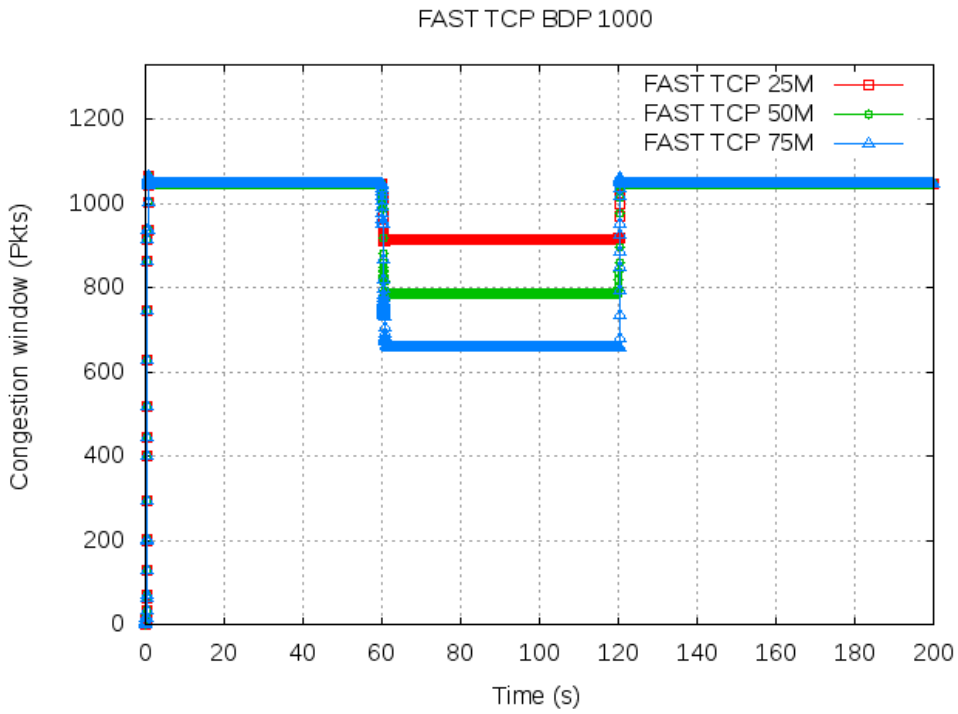


圖 四十二：FAST TCP 與不同的流量競爭時，壅塞視窗大小的變化

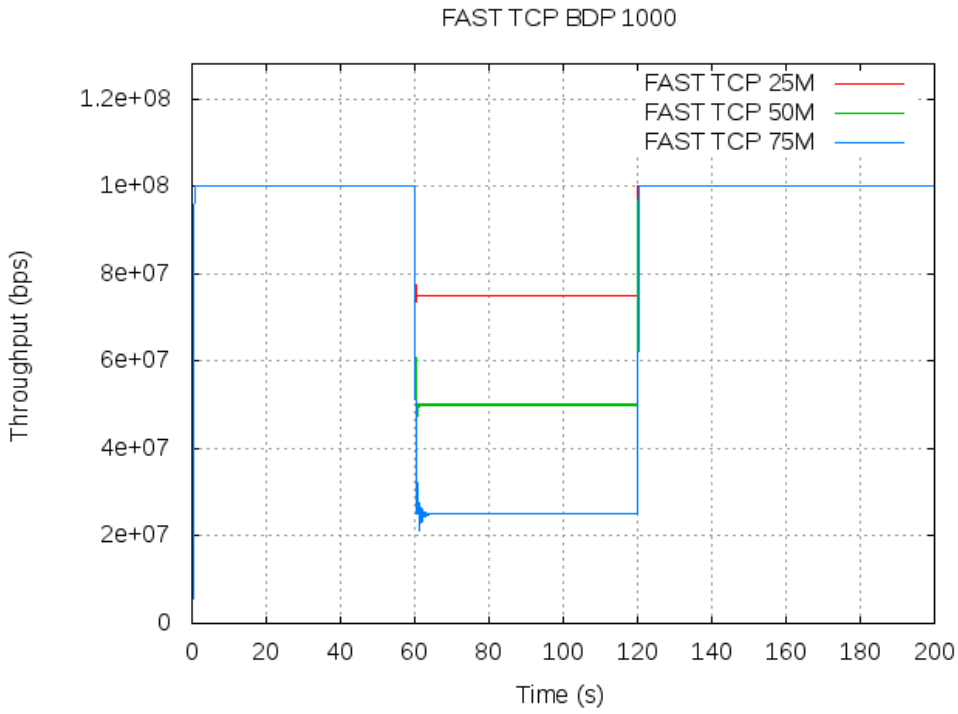
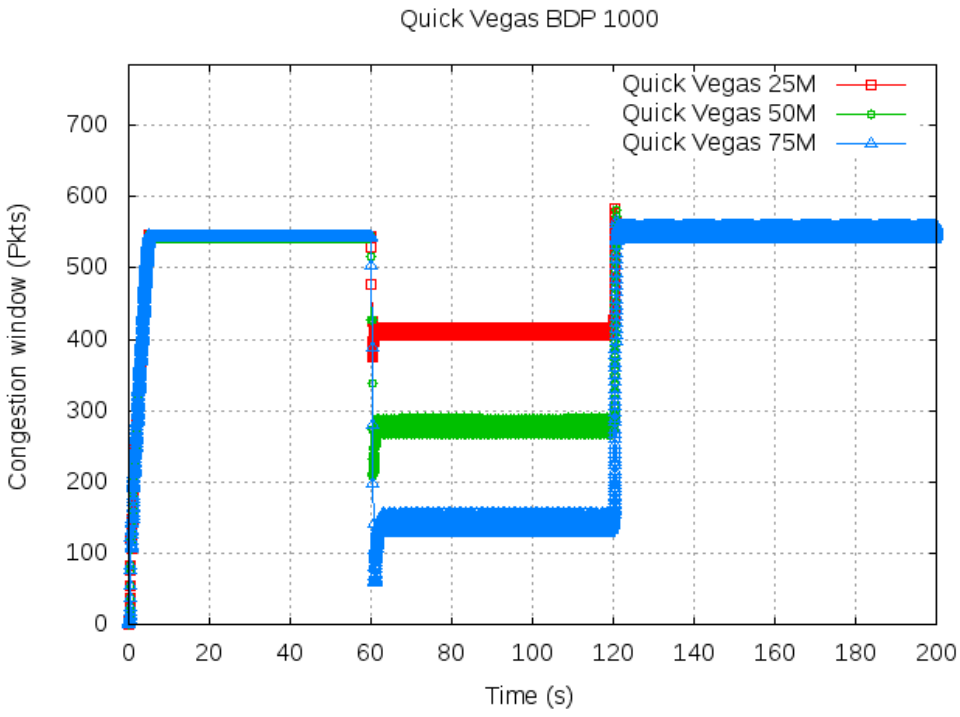


圖 四十三：FAST TCP 與不同的流量競爭時，吞吐量的變化

反觀 Quick Vegas 與 TCP Vegas 與不同的流量競爭時，在壅塞視窗大小與吞吐量的變化：

- Quick Vegas：雖然在與不同的流量競爭時，其壅塞視窗大小的變化，表示 Quick Vegas 可以有效的掌握網路頻寬資源，如圖四十四。但是再從圖四十四中可以發現 Quick Vegas 在調整壅塞視窗後，壅塞視窗會有發生震盪的現象。壅塞視窗發生振盪將會影響吞吐量，從圖四十五中的 Quick Vegas 吞吐量變化則可以看得出來。



圖四十四：Quick Vegas 與不同的流量競爭時，壅塞視窗大小的變化

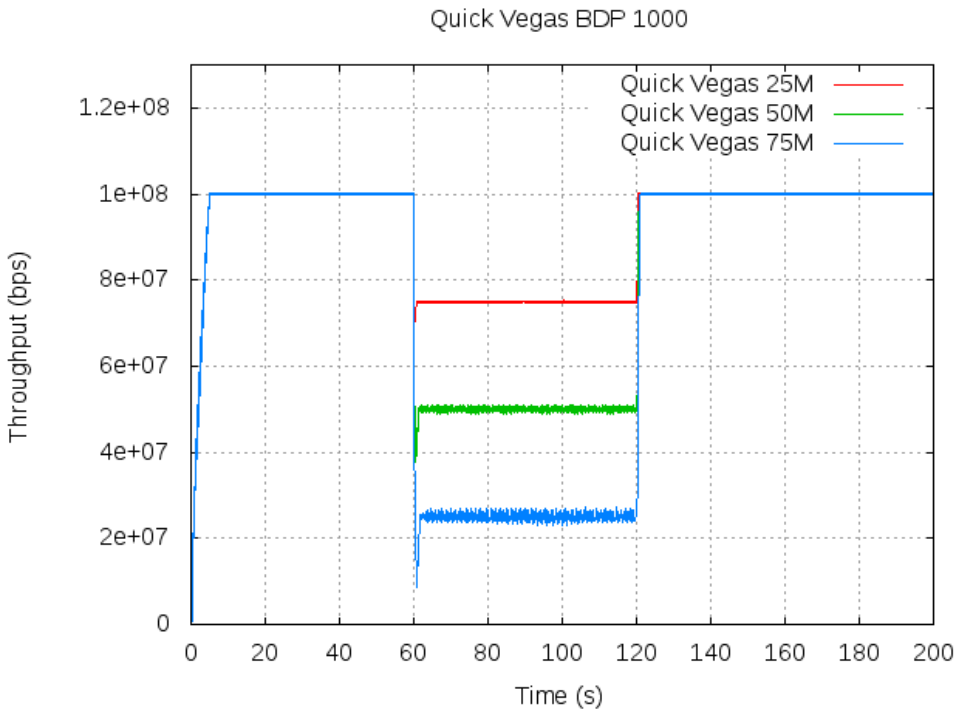


圖 四十五：Quick Vegas 與不同的流量競爭時，吞吐量的變化

- TCP Vegas：從 5.4.2.1 小節、5.4.2.2 小節與 5.4.2.3 小節的數據中指出，TCP Vegas 明顯有無法有效的掌握網路頻寬資源的問題，此問題會隨著流量變大而更加明顯，如圖四十六，同時此問題也反映在吞吐量的變化，如圖四十七。

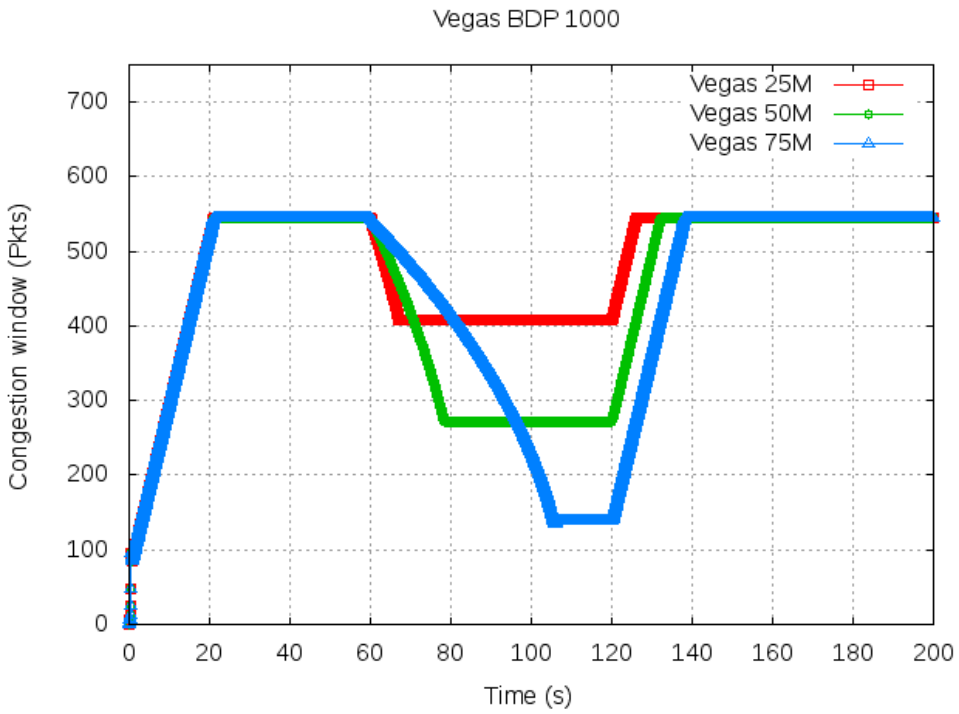


圖 四十六：TCP Vegas 與不同的流量競爭時，壅塞視窗大小的變化

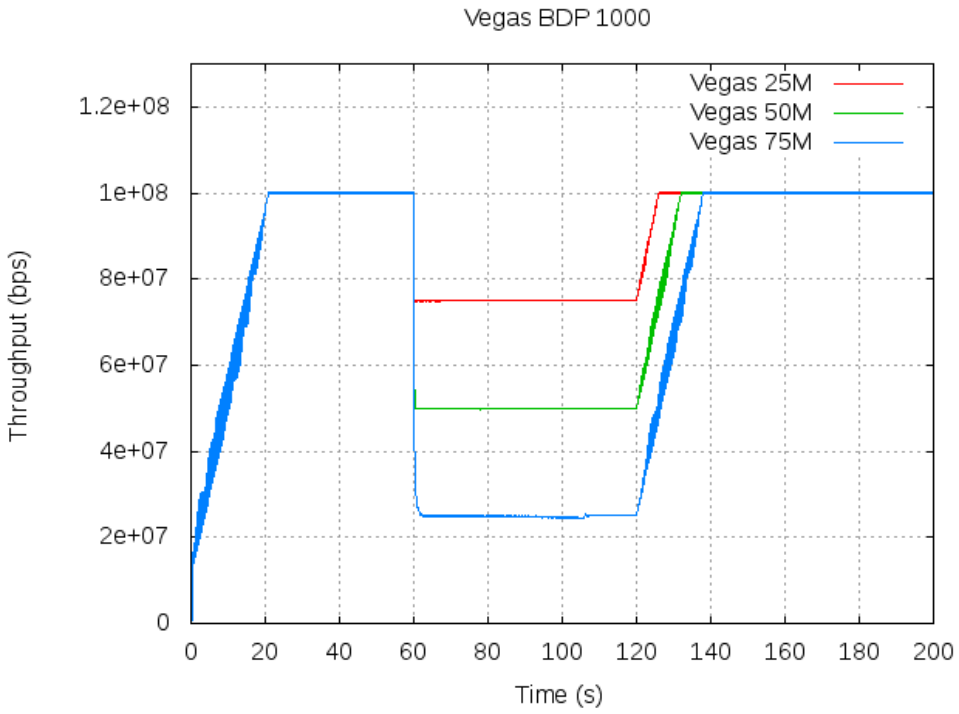


圖 四十七：TCP Vegas 與不同的流量競爭時，吞吐量的變化

5.4.3. 與多個流量競爭時

在本次實驗中，在 BDP 為 2000 的網路環境中，分別使用 HBDP Vegas、FAST TCP、Quick Vegas 與 TCP Vegas，並搭配多個固定傳輸速率（25Mb）之連線，觀察其壅塞視窗變化狀況與吞吐量變化。

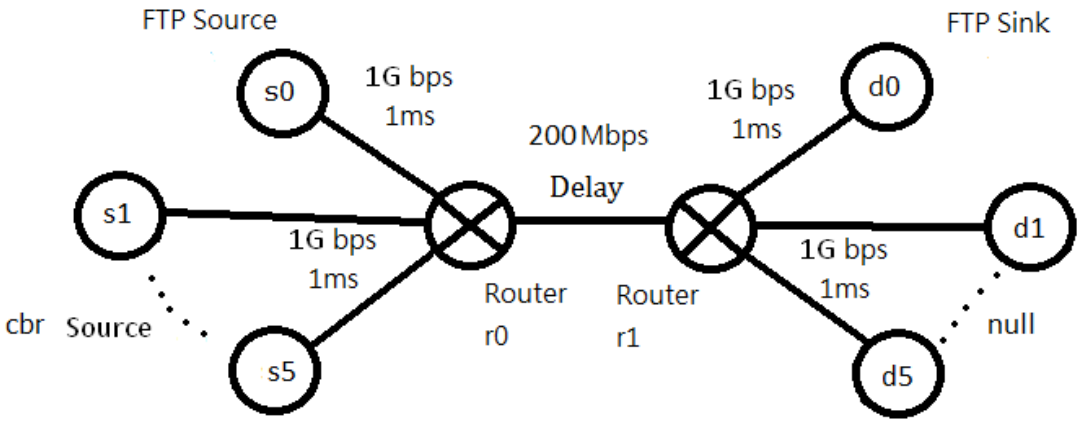


圖 四十八：實驗環境拓樸

本實驗所使用的網路拓樸如圖四十八所示，在此網路拓樸中一共使用六組發送端與接收端，關於此網路拓樸的詳細的實驗參數則如表格十所示。

表格 十：模擬環境參數表

模擬環境參數表	
網路模擬器	NS-2
節點（s0 與 d0）	● 使用檔案傳輸控制協定（FTP）來作資料傳輸。 ● FTP Source（s0）經由 r0 連上網路，並將資料發送給 FTP Sink（d0），並持續 160 秒。
節點（s1 – s5 與 d1 - d5）	● 採用固定傳送速率的 UDP 連結。 ● cbr Source（s1 – s5）分別在第 60 秒、第 80 秒、第 100 秒、第 120 秒與第 140 秒各啟動一條 UDP 連結。 ● 傳輸路徑會經由 r0 連上網路，並將產生一個具有固定傳送速率的連結把資料發送給 null（d1 - d5）。
兩個路由器（r0、r1）相互連接線段之間網路頻寬	200Mbps
兩個路由器（r0、r1）相互連接線段之間傳輸延遲時間	20.05792 ms
TCP Vegas α、β、γ	α＝2、β＝4、γ＝2
Quick Vegas α、β、γ	α＝2、β＝4、γ＝2
FAST TCP α	1010
HBDP Vegas 門檻值 K'	0.7

在實驗結果部分，HBDP Vegas、FAST TCP、Quick Vegas 與 TCP Vegas 運作的結果將呈現在表格十一與圖四十九到圖五十二，同時將在此小節的最後將說明實驗的結果。

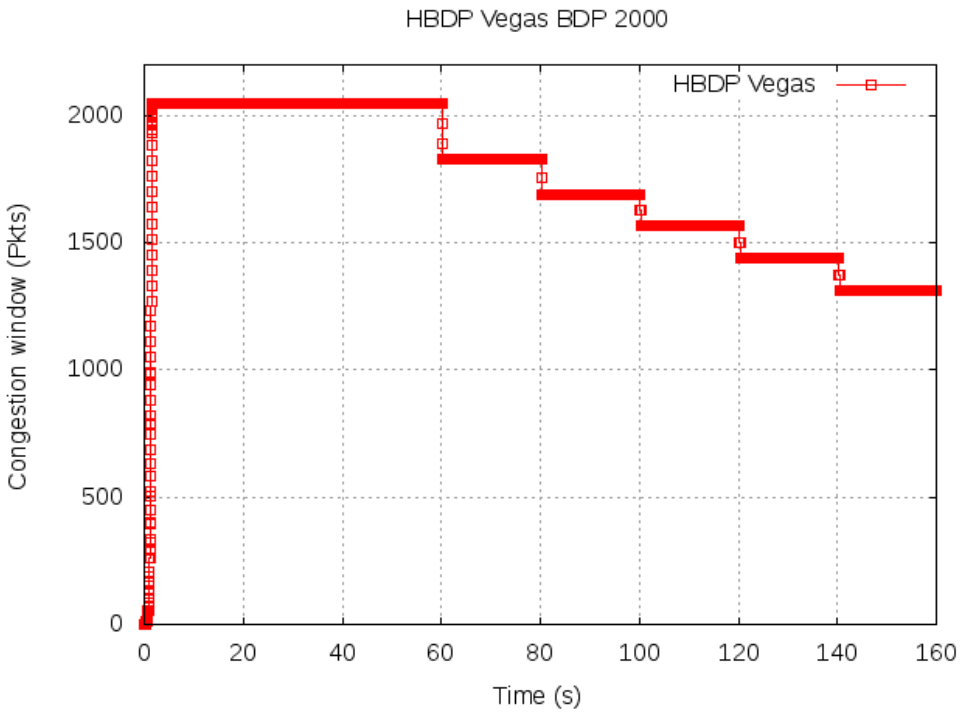


圖 四十九：HBDP Vegas 的壅塞視窗變化

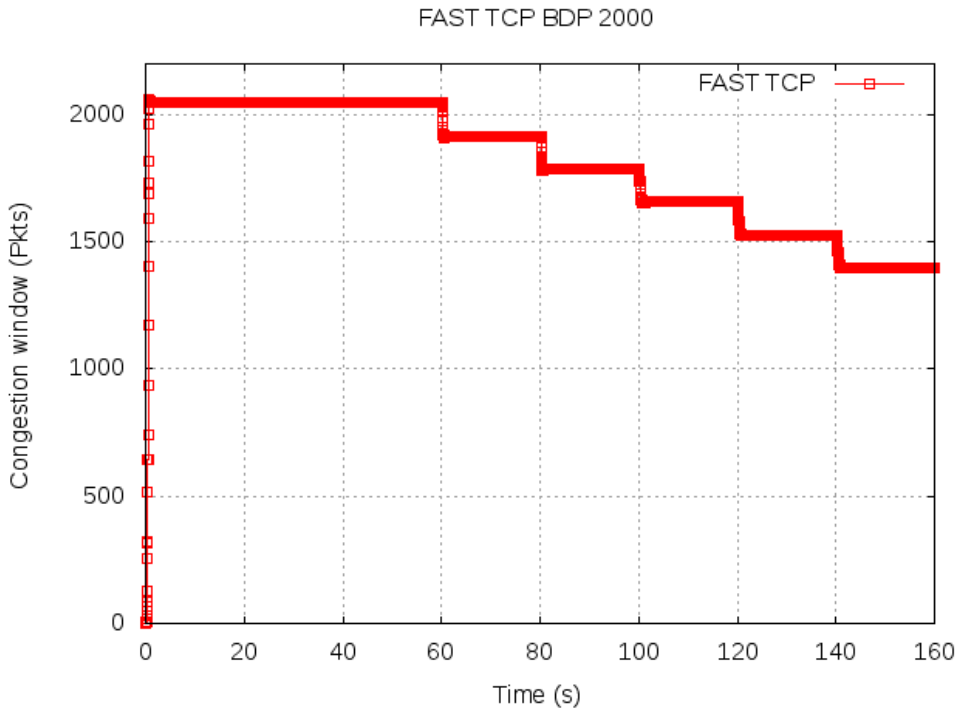


圖 五十：FAST TCP 的壅塞視窗變化

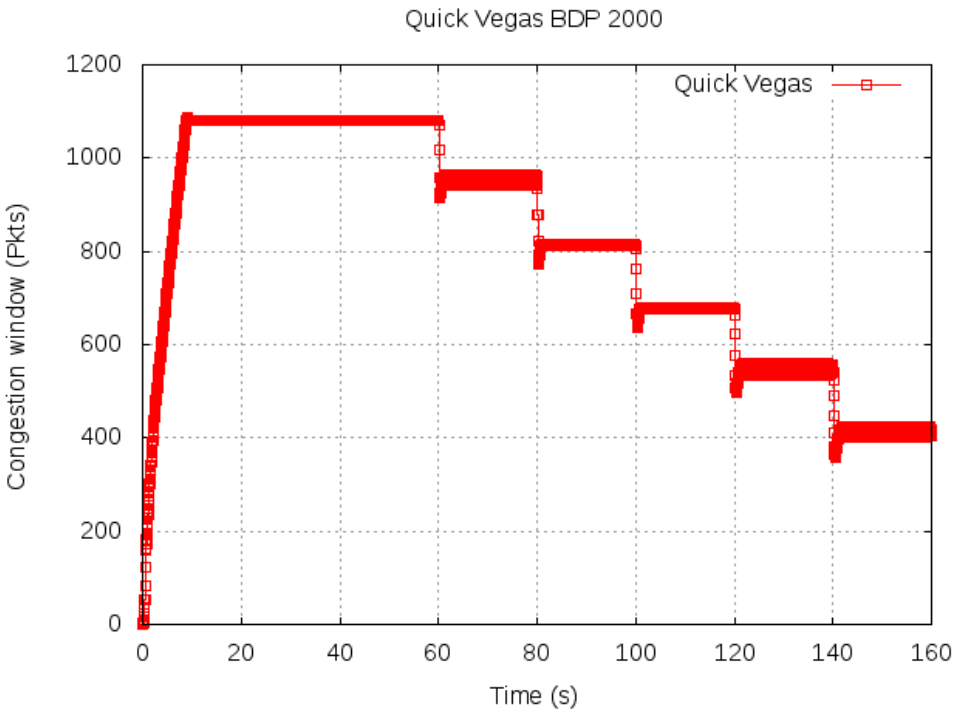


圖 五十一：Quick Vegas 的壅塞視窗變化

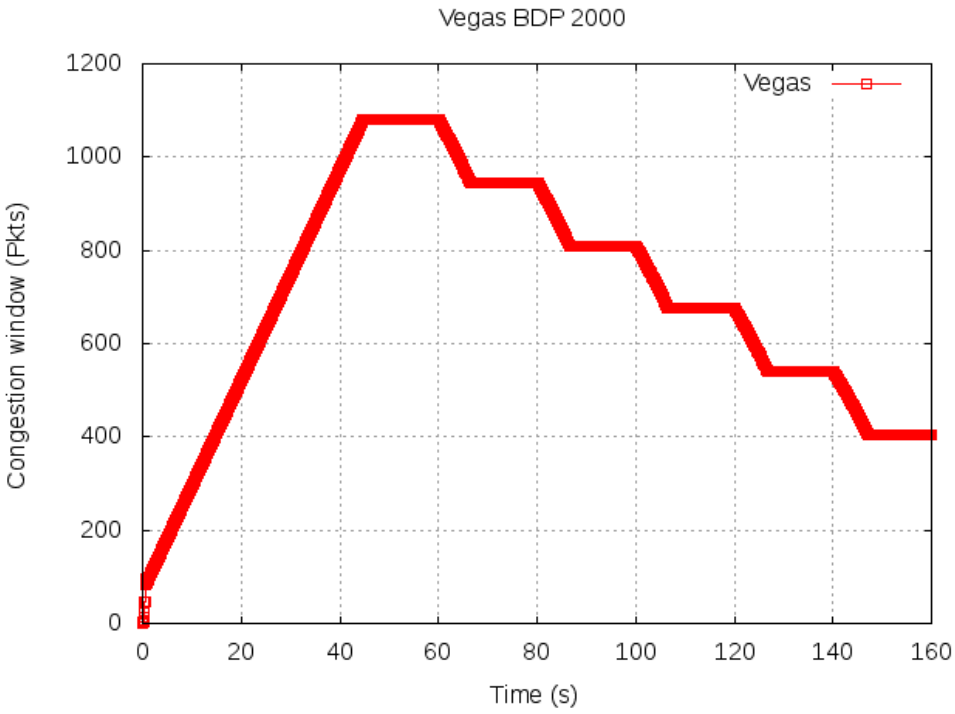


圖 五十二： TCP Vegas 的壅塞視窗變化

表格 十一：多條 CBR 背景流量（25Mb）平均吞吐量比較表

多條 CBR 背景流量（25Mb）平均吞吐量比較表				
流量名稱	HBDP Vegas	FAST TCP	Quick Vegas	TCP Vegas
CBR1	24.98Mbps	24.98Mbps	24.99Mbps	25.00Mbps
CBR2	24.98Mbps	24.97Mbps	25.00Mbps	25.00Mbps
CBR3	24.97Mbps	24.97Mbps	24.99Mbps	25.00Mbps
CBR4	24.97Mbps	24.97Mbps	24.99Mbps	25.00Mbps
CBR5	24.96Mbps	24.96Mbps	25.00Mbps	25.00Mbps

觀察圖四十九到圖五十二,在壅塞視窗變化方面,可以發現 HBDP Vegas、FAST TCP、Quick Vegas 與 TCP Vegas，遇到 CBR 背景流量運作時,在控制壅塞視窗的大小以接納新的背景流量所花費的平均時間分別是 0.404、1.298、1.51 與 6.676，其中 HBDP Vegas 花費時間最少，TCP Vegas 則是花費時間明顯最多。

在各條 CBR 背景流量所獲得的吞吐量方面,從表格十一中可以發現，HBDP Vegas 則可以和 FAST TCP、Quick Vegas、TCP Vegas 一樣，在面對新的背景流量加入時,可以依照連接之流量而分享此連線所須的頻寬。

最後,綜合上面兩段所述,HBDP Vegas 可以迅速的調整壅塞視窗，這表示新加入的流量可以在短時間內取得最大的吞吐量。同時,HBDP Vegas 也會分享其所得到的頻寬，給新加入的流量，而不會佔據整個

網路頻寬。

5.4. 公平性

在本實驗中，我們將觀察在多條連線同時運作時，每一條連線的吞吐量，來驗證 HBDP Vegas 是否具有公平性。在實驗的過程中我們將會驗證在 BDP 為 1600 的網路環境下 HBDP Vegas、FAST TCP 與 TCP Vegas 的公平性。其中，關於公平性的驗證則是利用由文獻【16】所提到的公平性指數公式來驗證，如公式（11）所示。

$$0 < \text{Fairness} = \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n\sum_{i=1}^{n} {x_i}^2} \leq 1 \qquad (11)$$

在公式（11）中，n表示目前所有的連線數，X_i則是代表第i條連線的吞吐量，經由公平性指數公式計算出來的數值將落在0到1之間。若公平性指數值越接近1，則表示公平性越好，同時也表示網路中每一條連線的吞吐量是相等的，反之若公平性指數值越接近0則越不具公平性。

圖五十三是本實驗所使用的網路拓樸，在此網路拓樸中共有 100 組發送端與接收端，關於此網路拓樸的詳細的實驗參數則如表格十二所示。

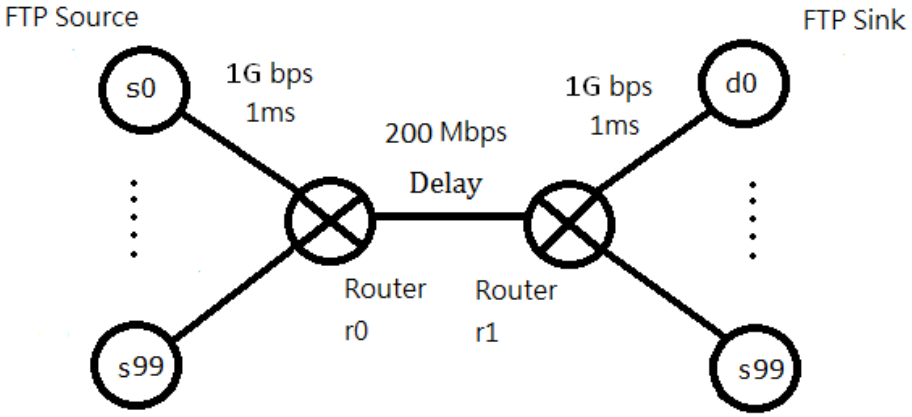


圖 五十三：實驗環境拓樸

表格 十二：模擬環境參數表

模擬環境參數表	
網路模擬器	NS-2
節點（s0 – s99 與 d0 – d99）	● 使用檔案傳輸控制協定（FTP）來作資料傳輸。 ● FTP Source（s0 – s99）經由 r0 連上網路，並將資料發送給 FTP Sink（d0 – d99），並持續 100 秒。
兩個路由器（r0、r1）相互連接線段之間網路頻寬	200Mbps
兩個路由器（r0、r1）相互連接線段之間傳輸延遲時間	15.64192ms
TCP Vegas **α、β、γ**	DropTail
Quick Vegas **α、β、γ**	1KB
FAST TCP **α**	852

HBDP Vegas 門檻值 K'	0.52

本實驗將驗證 FAST TCP、HBDP Vegas 與 TCP Vegas 在 2、4、6 … 100 條連結同時運作之下的公平性，其結果如表格十三、表格十四與表格十五所示。

表格 十三：HBDP Vegas 公平性指數

HBDP Vegas 公平性指數	
連線數量	**公平性指數（%）**
2	99.3
4	99.6
6	98.8
8	99.6
10	99.5
12	99.0
14	99.3
16	99.4
18	99.1
20	99.4
30	99.1
40	99.7
50	99.5
60	99.3
70	99.1
80	99.6
90	99.3
100	99.6

表格 十四：FAST TCP 公平性指數

FAST TCP 公平性指數	
連線數量	公平性指數（%）
2	99.9
4	99.9
6	98.9
8	99.9
10	98.4
12	92.4
14	94.2
16	93.7
18	92.1
20	91.8
30	98.8
40	99.0
50	99.2
60	99.3
70	99.4
80	99.2
90	98.9
100	98.8

表格 十五：TCP Vegas 公平性指數

TCP Vegas 公平性指數	
連線數量	公平性指數（%）
2	99.9
4	99.0
6	98.2
8	97.4
10	95.7
12	94.2

14	93.1
16	92.0
18	91.3
20	90.1
30	87.5
40	91.0
50	89.4
60	90.8
70	91.0
80	95.7
90	94.7
100	94.4

綜觀表格十三、十四與十五，我們可以發現 HBDP Vegas 的公平性指數在大多數的狀況下都可以保持在 99%，表示每一個連線均可分享到網路頻寬。其原因則和連線一開始建立之初到達到穩定的過程中壅塞視窗成長行為有關，因為在實際的實驗過程中，雖然在拓樸中所有的連線是同時一起運作，但是實際上每一條連線運作的時間仍有一點差距【3】，連線之間開始運作時間的差距將影響到每一條連線所量測的 RTT 以及對網路壅塞程度的評估結果，因此 HBDP Vegas 針對於連線一開始建立之初到達到穩定，這一段時間內的壅塞視窗的成長行為，則參考 Quick Vegas 對壅塞視窗成長的控制方式來提出藉由評估網路壅塞程度來決定增加當前壅塞視窗的大小的 0.25 倍或是 0.0625 倍的作法。此作法將可以防止連線一開始建立之初壅塞視窗成長速度過於快速，並避免短時間之內網路佇列被大量使用，同時也盡

可能的使得每一個連線所量測的 RTT 能一致，讓每一個連結能有機會爭取到頻寬。其成效將可從共有 40 條連線運作的圖五十四與圖五十五看出。

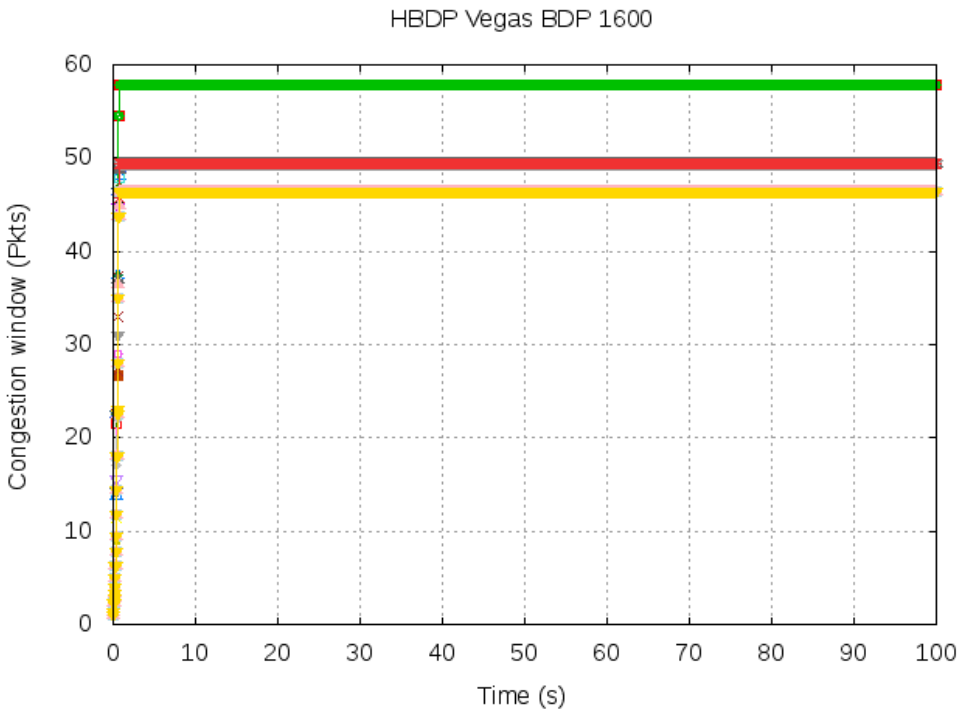


圖 五十四：HBDP Vegas：40 條連線同時運作時壅塞視窗的變化 1

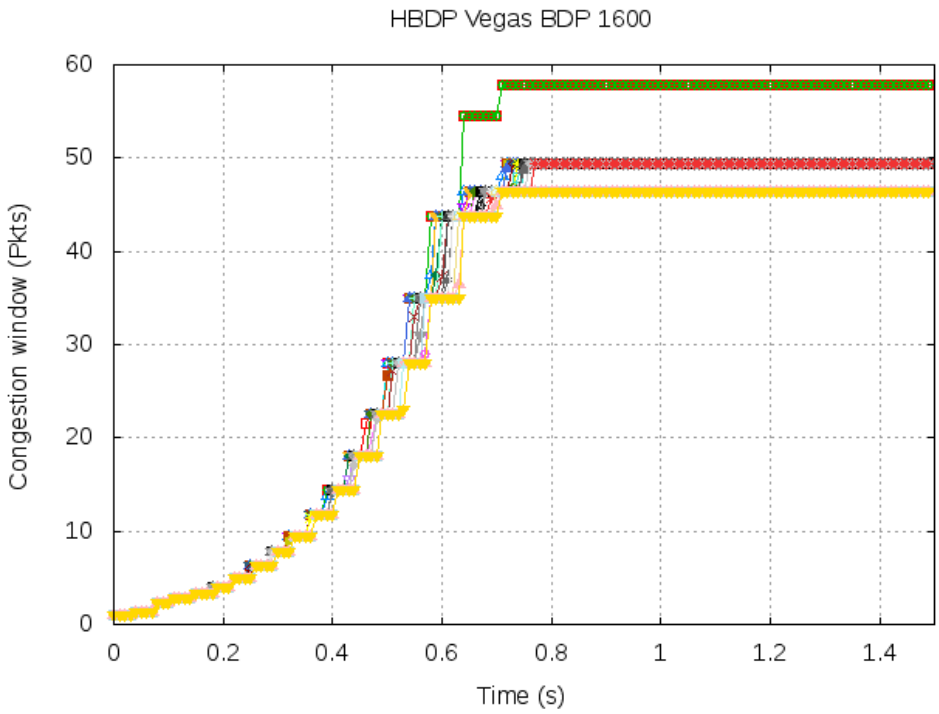


圖 五十五：HBDP Vegas：40 條連線同時運作時在 1.5 秒內壅塞視窗的變化 2

FAST TCP 的公平性指數在連線數為 8 條以前均維持在 99%，但在 10 條連線之後則會發生部分連線的壅塞視窗開始出現固定範圍的振盪，例如圖五十六，同時公平性指數開始下降。

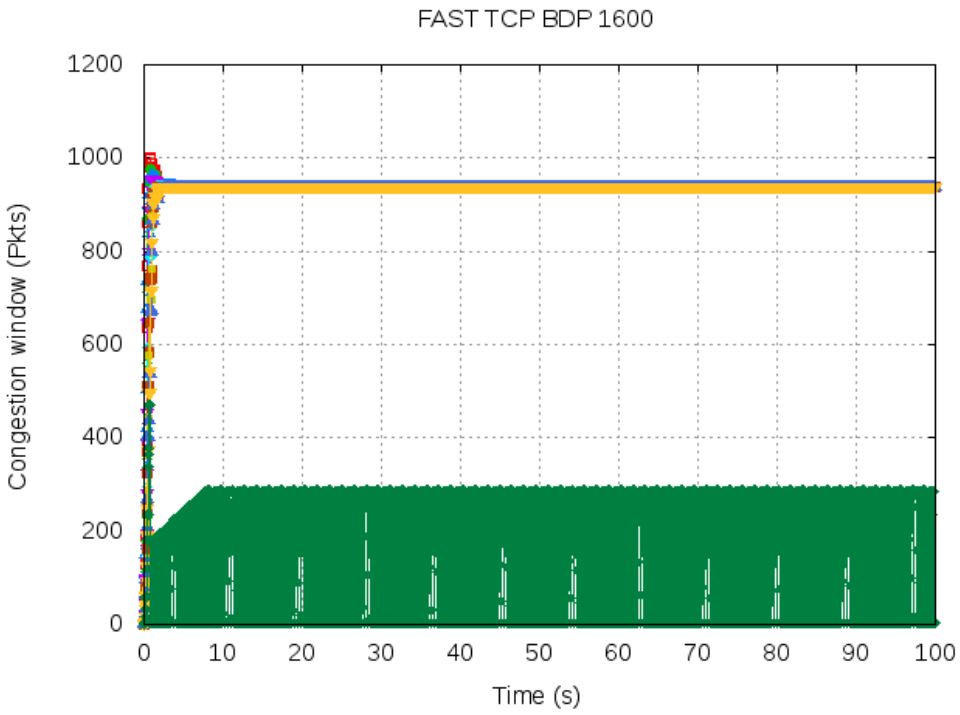


圖 五十六：FAST TCP：10 條連線同時運作時壅塞視窗的變化

最後當連線數為 30 條連線以上時，所有的連線的壅塞視窗則開始出現固定範圍的振盪，例如圖五十七，此時的公平性指數則大致上停留在 98%。

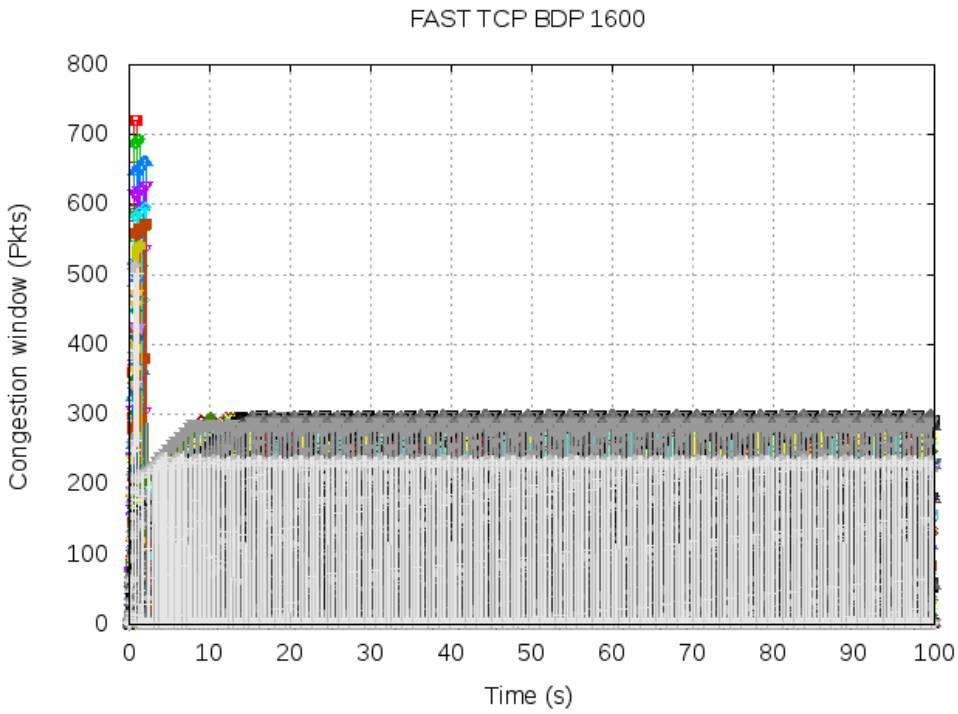


圖 五十七：FAST TCP：30 條連線同時運作時壅塞視窗的變化

出現如圖五十六與圖五十七的現象之原因是和 FAST TCP 在連線一開始建立之初到達到穩定的過程中壅塞視窗的成長行為有關。在大部份的連線開始運作時，所量測到的 RTT 與網路佇列封包堆積量均較小，若以 FAST TCP 對於壅塞視窗的成長控制方式，則會採取倍數的成長。但是此種成長行為再加上連線之間開始運作時間的差距，將影響到每一條連線所量測的 RTT 以及造成短時間內網路佇列被大量使用，最後則會發生部分連線無法分享到網路頻寬或是影響到所有的連線對頻寬的掌握能力。

TCP Vegas 的公平性指數同樣也會隨著連線數變多而下降，造成上述現象的原因則是連線之間開始運作時間的差距，使得稍晚啟動傳輸的連線所量測到的 RTT 有所偏差，同時也因為 TCP Vegas 壅塞避免階段的設計，造成連結在運作過程中所計算出來的Δ值，若是落在 α 和 β 之間，將造成壅塞視窗不會有任何變動，如此一來將造成部分的連線的壅塞視窗將會出現長期停留在一個高處或是一個低處的現象，即如圖五十八與圖五十九，此現象會影響每一條連線對頻寬的掌握。

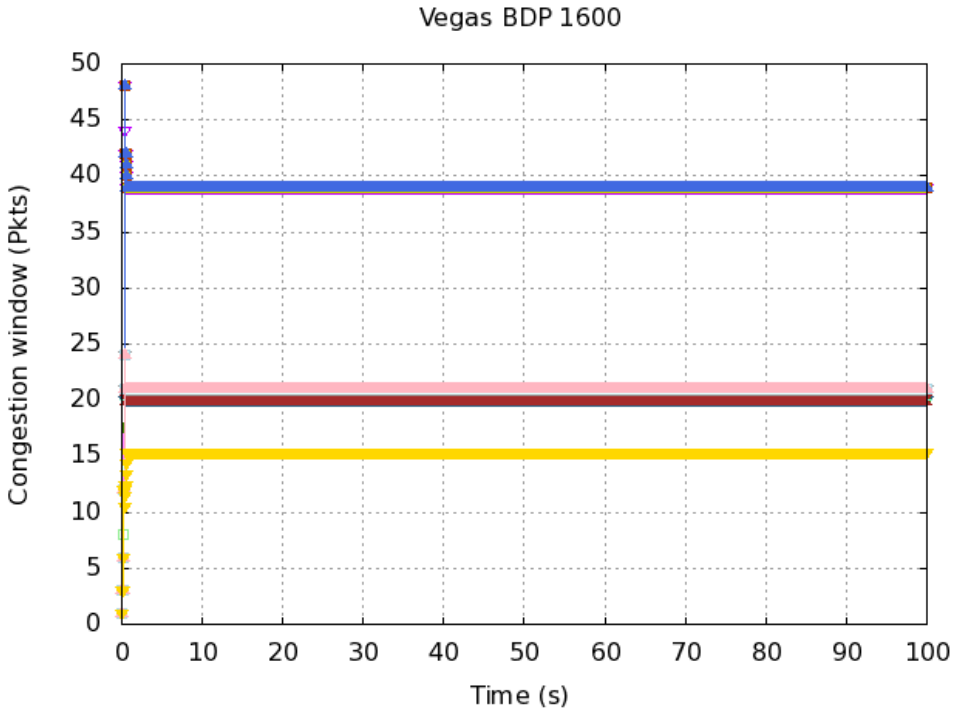


圖 五十八：TCP Vegas：40 條連線同時運作時壅塞視窗的變化 1

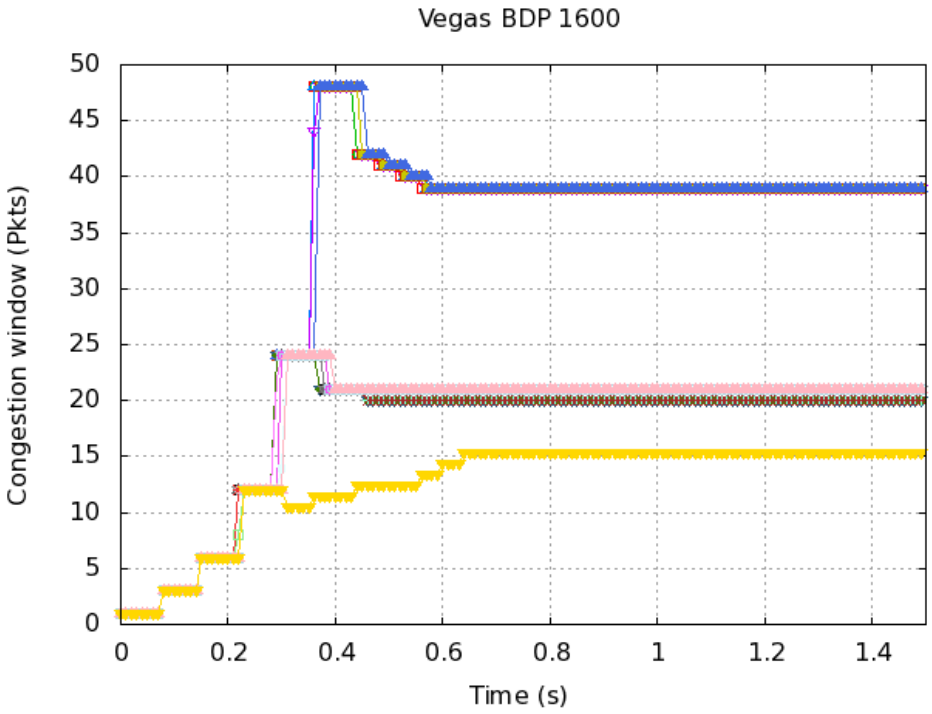


圖 五十九：TCP Vegas：40 條連線同時運作時在 1.5 秒內壅塞視窗的變化 2

第六章　結論

本文提出了一個可以應用在高頻寬延遲乘積網路環境的 TCP Vegas 改良版本，HBDP Vegas。HBDP Vegas 可以解決 TCP Vegas 在高頻寬延遲乘積網路環境中無法有效掌握網路頻寬與反應速度過於緩慢的問題。

HBDP Vegas 在評估網路壅塞程度的方式以及壅塞控制機制中的緩啟動階段與壅塞避免階段部分，所提出的改進機制，分述如下 ：

- 在評估網路壅塞程度的方式部分，則是提出一種藉由計算 base RTT 與每一次所量測的 RTT 的比例值來衡量網路壅塞的程度的方法，同時此方法也實際應用在 HBDP Vegas 的壅塞控制機制中。
- 在緩啟動階段部分，藉由修改了壅塞視窗的成長方式並搭配新的評估網路壅塞程度的方法來做為是否離開緩啟動階段的依據與壅塞視窗的成長量。改良過的緩啟動階段可以讓連線得以快速的掌握網路頻寬，同時亦可讓壅塞視窗的成長不會過快。
- 在壅塞避免階段部分，修改了原始的壅塞避免階段中的壅塞視窗的大小控制規劃與成長倍數，同時也搭配新的評估網路壅塞程度的方法，來達到可以快速的調整壅塞視窗與快速達到穩定來維持吞吐量。

在 HBDP Vegas 的驗證，我們則是透過了四組實驗來做驗證。關於驗證的結果，則如下所述：

- 緩啟動階段的比較：HBDP Vegas 被證實可以在緩啟動結束時，就掌握了大部份的可用頻寬。
- 收斂實驗：HBDP Vegas 被證實可以在短時間之內適應不斷變化的網路環境於壅塞避免階段。
- 面對其它連線競爭時之狀況：HBDP Vegas 被證實在面對其它連線競爭時或是爭取某條連線所釋放的頻寬時，均可以快速的釋放與掌握網路頻寬，並且 HBDP Vegas 也可以公平的與其它連線分享網路頻寬，並不會發生佔據整個網路頻寬的問題。
- 公平性：HBDP Vegas 被證實即使在多條連線同時競爭的網路環境下，仍然能保持其公平性。

同時，HBDP Vegas 的實驗結果與 FAST TCP、Quick Vegas、TCP Vegas 的實驗結果，相較起來均有所超越與改進。

本文未來的改進目標是改善瓶頸點的佇列使用狀況，在實驗過程中發現 HBDP Vegas 部分特性類似 FAST TCP。但是在其它學者的實驗中則有發現 FAST TCP 在整體的運作過程中對於在瓶頸點上佇列需求量較大【5】，所以 HBDP Vegas 也有類似的問題，因此希望能在未來

的研究中來解決在瓶頸點上佇列需求量較大的問題。

參考文獻

1. 洪盟峰、鍾博文，基於 FAST TCP 可支援雲間長距離快速資料移動之視窗管理法，TANET2011 研討會，國立宜蘭大學。
2. 張均瑋，2004，在不同壅塞控制的方法下之 TCP 效能分析與公平性改進，國立交通大學資訊工程系碩士論文。
3. 張凱翔，2005，改善 TCP Vegas 緩啟動機制之研究，國立彰化師範大學資訊工程學系碩士論文。
4. 閆二輝、朱敏、丁青、李運濤、溫韜，2011 年一月，一種基於比例因子的 TCP Vegas 緩啟動策略，計算機應用研究，第 28 卷，第 1 期： 253-255 頁。
5. 詹益禎、王孝恩、徐志榮、黃柏霖、許嘉樺、黃瀚璋、黃琦閔，2007，高頻寬延遲乘積網路上之 TCP Vegas 效能改進研究，2007 資訊科技國際研討會，朝陽科技大學。
6. 詹益禎，2004，TCP 壅塞控制技術之研究與設計，國立交通大學資訊工程學系博士論文。
7. 蘇倍豪，2008，可支援長距離網路上高速傳輸之動態 TCP Vegas 設計，樹德科技大學資訊工程研究所碩士班碩士論文。
8. 顧明、張軍、蘇東林，2007 年四月，大頻寬時延積網路 TCP Vegas 自我調整緩啟動演算法，電訊技術，第 47 卷，第 2 期：27-30 頁。

9. L. S. Brakmo and L. L. Peterson, “TCP Vegas: end to end congestion avoidance on a global Internet”, pp. 1465–1480, IEEE Journal on Selected Areas in Communications 13 (8) 1995.
10. C. Jin, D. Wei and S. Low, “Fast TCP: Motivation, architecture, algorithm, performance,” in Proc. IEEE INFORCOM 2004, vol. 4, pp. 2490-2501, Mar. 2004.
11. G. Hasegawa, M. Murata, and H. Miyahara, “Fairness and stability of congestion control mechanism of TCP,” Telecommunication Systems Journal, pp. 167-184, Nov. 2000.
12. J. Postel , “Transmission Control Protocol,” RFC 793, Sep. 1981
13. Lawrence S. Brakmo , Sean W. O’Malley, Larry L. Peterson ,”TCP Vegas: New Techniques for Congestion Detection and Avoidance,” Proc. Of ACM SIGCOMM, Aug. 1994, pp. 24-35.
14. M. Allman , V. Paxson and W. Stevens, RFC 2581: TCP Congestion Control, April 1999.
15. NS2, Network Simulator version 2, http://www.isi.edu/nsnam/
16. R. Jain, The art of computer systems performance analysis: Techniques for experimental design, measurement, simulation and modeling., Wiley, New York, 1991
17. S. Vanichpun and W. Feng, “On the transient behavior of TCP Vegas,” in Proc. IEEE ICCCN’02 Oct. 2002, pp: 504-508.
18. Stevens, W. , "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms," RFC 2001, Jan. 1997.
19. V. Jacobson ,”Congestion Avoidance and Control,” Proc. of ACM SIGCOMM, Aug. 1988, pp. 314-329.
20. V. Jacobson ,”Modified TCP Congestion Avoidance Algorithm,” mailing list , end2end-interest, Apr. 1990.
21. W. Feng and P. Tinnakornsrisuphap, “The failure of TCP in high-performance computational grids,” in Proc. SC 2000: High-performance Networking and Computing Conf., Nov. 2000.
22. W. Stevens, “TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms, “ IETF RFC 2001, 1997.
23. Yi-Cheng Chan、Chia-Liang Lin and Cheng-Yuan Ho, ”Quick Vegas: Improving Performance of TCP Vegas for High Bandwidth-Delay Product Networks,” IEICE TRANS. COMMUN. APRIL 2008 , vol.E91–B , NO.4

24. Y. C. Chan, C. L. Lin, C. T. Chan and C. Y. Ho , “Improving Performance of TCP Vegas for High Bandwidth-Delay Product Networks,” Proceedings of IEEE Advanced Communication Technology , 2006, vol. 1, pp.987-997.

25. ZHANG Mu，ZHANG Shun-yi , “Comparison and research on TCP Reno、TCP Vegas、Fast TCP, “ Computer Engineering and Applications , 2007 , 43(1)：19-23

附錄

附錄內容將會呈現 HBDP Vegas 在緩啟動階段與壅塞避免階段之程式碼，同時將對程式碼中數個重要變數作說明。

重要變數：

- v_k_：即常數 K，代表了 base RTT 在每一次所量測的 RTT 中所占的比例值，用來表示網路的壅塞程度。
- v_k_p_s_：即門檻值 K'，將會與 k 值進行比較來決定緩啟動階段的結束與否。
- v_finsec：用來紀錄第一次進入第二控制區間的壅塞視窗大小。

網路模擬軟體：NS-2（Network Simulator，version 2）。

網路模擬軟體版本：2.33 版。

程式碼：

```
00  if(cwnd_ < ssthresh_) {   // slow-start
01      v_k_ = v_baseRTT_/rtt;
02      if(delta < 1){
03          v_incr_ = 0.25;
04      }
05      else if(delta >= 1){
06          if((delta >= 1) && (v_k_ < 1)){
07              if(v_k_ < v_k_p_s_){
08                  ssthresh_ = 2;
09                  if(cwnd_<2){
```

```
10                          cwnd_ = 2.;
11                      }
12                      v_incr_ = 0;
13                      v_before_delta = delta;
14                      v_max = cwnd_;
15                  )
16                  else if(v_k_ >= v_k_p_s_){
17                      double temp;
18                      temp = (1-v_k_p_s_)/2;
19                      if(v_k_ >= (temp+v_k_p_s_)){
20                          v_incr_ = 0.25;
21                      }
22                      else if((v_k_ < (temp+v_k_p_s_)) && (v_k_ >= v_k_p_s_)){
23                          v_incr_ = 0.0625;
24                      }
25                  }
26              }
27          }
28      }
29      else {    // congestion avoidance
30          v_k_ = v_baseRTT_/rtt;
31          double temp_delta;
32          if(v_in_ca_ == 0){ //第一次進入壅塞避免階段時
33              temp_delta = delta;
34              if(temp_delta <= 1){
35                  v_beta_ = 10;
36                  v_alpha_ = v_alpha2_ = 2;
37                  v_avg = (v_beta_ + v_alpha_)/2;
38              }
39              else{
40                  v_beta_ = temp_delta + 5;
41                  v_alpha_ = v_alpha2_ = temp_delta/2;
42                  v_avg = (v_beta_ + v_alpha_)/2;
43              }
44              v_in_ca_ = 1;
45          }
46          if(delta>v_beta_){ //第一控制區間
```

```
47              if(v_k_ < 0.2){
48                  if( ((delta-v_avg)) >= 0 ){
49                      cwnd_ = cwnd_ - ((delta-v_avg));
50                  }
51                  if(cwnd_<2) {
52                      cwnd_ = 2;
53                  }
54                  v_incr_ = 0;
55              }
56              else if((v_k_ < 0.3) && (v_k_ >= 0.2)){
57                  if( ((delta-v_avg)/2) >= 0 ){
58                      cwnd_ = cwnd_ - ((delta-v_avg)/2);
59                  }
60                  if(cwnd_<2) {
61                      cwnd_ = 2;
62                  }
63                  v_incr_ = 0;
64              }
65              else if((v_k_ < 0.4) && (v_k_ >= 0.3)){
66                  if( ((delta-v_avg)/4) >= 0 ){
67                      cwnd_ = cwnd_ - ((delta-v_avg)/4);
68                  }
69                  if(cwnd_<2) {
70                      cwnd_ = 2;
71                  }
72                  v_incr_ = 0;
73              }
74              else if((v_k_ < 0.5) && (v_k_ >= 0.4)){
75                  if( ((delta-v_avg)/4) >= 0 ){
76                      cwnd_ = cwnd_ - ((delta-v_avg)/4);
77                  }
78                  if(cwnd_<2) {
79                      cwnd_ = 2;
80                  }
81                  v_incr_ = 0;
82              }
83              else if(v_k_ >= 0.5){
```

```
84                  v_beta_ = v_beta_ + 10;
85                  v_alpha_ = v_alpha_ + 10;
86                  v_avg = (v_beta_ + v_alpha_)/2;
87                  v_incr_ = 0;
88              }
89          }
90          else if((delta<=v_beta_) && (delta>v_avg)){ //第二控制區間
91              if (v_finsec_bit == 0){
92                  v_finsec = cwnd_;
93                  v_finsec_bit += 1;
94              }
95              if(cwnd_ > v_finsec){
96                  cwnd_ = v_finsec;
97                  v_incr_ = 0;
98              }
99              if( v_alpha_ > v_alpha2_ ){
100                 int v_temp_beta_;
101               int v_temp_avg;
102               int v_temp_alpha_;
103                 v_temp_beta_ = v_beta_ - 1;
104               v_temp_alpha_ = v_alpha_ - 1;
105                 if( ((delta<=v_temp_beta_) && (delta>v_temp_avg)) ) {
106                     v_beta_ = v_beta_ - 1;
107                     v_alpha_ = v_alpha_ - 1;
108                     v_avg = (v_beta_ + v_alpha_)/2;
109                 }
110             }
111         }
112         else if((delta>=v_alpha_) && (delta<(v_avg))) {   //第三控制區間
113             if(v_finsec == 0) {
114                 v_finsec = v_max;
115                 v_finsec_bit += 1;
116             }
117             if(cwnd_ < v_finsec) {
118                 if(v_k_ <= 0.4) {
119                     v_beta_ = v_beta_ - 10;
120                     v_alpha_ = v_alpha_ - 10;
```

```
121                 v_avg = (v_beta_ + v_alpha_)/2;
122             }
123             else if((v_k_ > 0.4) && (v_k_ <= 0.5)) {
124                 v_incr_ = 0.0625;
125                 v_beta_ = v_beta_ + 20;
126                 v_alpha_ = v_alpha_ + 10;
127                 v_avg = (v_beta_ + v_alpha_)/2;
128             }
129             else if((v_k_ > 0.5) && (v_k_ <= 0.6)) {
130                 v_incr_ = 0.125;
131                 v_beta_ = v_beta_ + 30;
132                 v_alpha_ = v_alpha_ + 20;
133                 v_avg = (v_beta_ + v_alpha_)/2;
134             }
135             else if((v_k_ > 0.6) && (v_k_ <= 0.7)) {
136                 v_incr_ = 0.25;
137                 v_beta_ = v_beta_ + 40;
138                 v_alpha_ = v_alpha_ + 30;
139                 v_avg = (v_beta_ + v_alpha_)/2;
140             }
141             else if((v_k_ > 0.7) && (v_k_ <= 0.8)) {
142                 v_incr_ = 0.5;
143                 v_beta_ = v_beta_ + 50;
144                 v_alpha_ = v_alpha_ + 40;
145 
146                 v_avg = (v_beta_ + v_alpha_)/2;
147             }
148         }
149         else if(cwnd_ >= v_finsec) {
150             cwnd_ = v_finsec;
151             v_incr_ = 0;
152             if(v_alpha_ > 1){
153                 int v_temp_beta3_;
154                 int v_temp_avg3;
155                 int v_temp_alpha3_;
156                 v_temp_beta3_ = v_beta_ - 1;
157                 v_temp_alpha3_ = v_alpha_ - 1;
```

```
158                 v_temp_avg3 = (v_temp_beta3_ + v_temp_alpha3_)/2;
159                 if( (v_temp_beta3_ > 0) && (v_temp_alpha3_ > 0) ) {
160                     v_beta_ = v_beta_ - 1;
161                     v_alpha_ = v_alpha_ - 1;
162                     v_avg = (v_beta_ + v_alpha_)/2;
163                 }
164             }
165         }
166     }
167     else if(delta<v_alpha_){   //第四控制區間
168         if(v_finsec == 0) {
169             v_finsec = v_max;
170             v_finsec_bit += 1;
171         }
172         if(cwnd_ < v_finsec) {
173             if((v_k_ > 0.6) && (v_k_ <= 0.7)) {
174                 v_incr_ = 0.125;
175                 v_beta_ = v_beta_ + 25;
176                 v_alpha_ = v_alpha_ + 25;
177                 v_avg = (v_beta_ + v_alpha_)/2;
178             }
179             else if((v_k_ > 0.7) && (v_k_ <= 0.8)) {
180                 v_incr_ = 0.25;
181                 v_beta_ = v_beta_ + 45;
182                 v_alpha_ = v_alpha_ + 45;
183                 v_avg = (v_beta_ + v_alpha_)/2;
184             }
185             else if(v_k_ > 0.8) {
186                 v_incr_ = 0.25;
187                 v_beta_ = v_beta_ + 75;
188                 v_alpha_ = v_alpha_ + 75;
189                 v_avg = (v_beta_ + v_alpha_)/2;
190             }
191         }
192         else if(cwnd_ >= v_finsec) {
193             cwnd_ = v_finsec;
194             v_incr_ = 0;
```

```
195                 if(v_alpha_ > 1) {
196                     int v_temp_beta4_;
197                     int v_temp_avg4;
198                     int v_temp_alpha4_;
199                     v_temp_beta4_ = v_beta_ - 5;
200                     v_temp_alpha4_ = v_alpha_ - 5;
201                     v_temp_avg4 = (v_temp_beta4_ + v_temp_alpha4_)/2;
202                     if( (v_temp_beta4_ > 0) && (v_temp_alpha4_ > 0) ) {
203                         v_beta_ = v_beta_ - 5;
204                         v_alpha_ = v_alpha_ - 5;
205                         v_avg = (v_beta_ + v_alpha_)/2;
206                     }
207                 }
208             }
209         }
210         else{ //Δ＝avg
211             if (v_finsec_bit == 0) {
212                 v_finsec = cwnd_;
213                 v_finsec_bit += 1;
214             }
215             v_incr_ = 0;
216         }
217     }
```