基於高頻寬延遲乘積網路中 TCP Vegas 慢啟動階段效能改善

作者[1]羅勝暉　　作者[2]王鑫一　　作者[3] 朱延平
[1]東海大學資訊工程學系 g99350053@thu.edu.tw
[2]中興大學資訊科學與工程學系 d100056009@mail.nchu.edu.tw
[3]東海大學資訊工程學系 ypchu@thu.edu.tw

摘要

在高頻寬延遲乘積(BDP)網路中，適用於低頻寬的TCP協定，已經無法勝任當下的網路環境，故現在關於TCP在BDP網路的傳輸效能，已經成為一個研究重點。在本文中則是探討TCP Vegas的慢啟動階段的改進，希望能夠解決TCP Vegas提早離開慢啟動階段的問題。

提早離開慢啟動階段會導致在慢啟動階段結束時壅塞視窗仍不夠接近BDP的大小，此現象將嚴重降低TCP Vegas的傳輸效率與吞吐量。因此本文經過多次的實驗與觀察，提出一個修改TCP Vegas的慢啟動階段的構想，最後經實驗證實，確實可以改善上述問題。

關鍵詞：慢啟動、TCP Vegas、高頻寬延遲乘積、壅塞視窗、往返時間

1. 研究動機

近年來隨著網路技術的演進，高頻寬延遲乘積(bandwidth-delay product，BDP)網路，漸漸已成為當下十分普遍的網路環境，並且也是眾多學者所關注的研究的主題之一。目前在高BDP網路中，目前最普遍的TCP/IP傳輸層協定中的TCP Reno已經有研究證實其傳輸性能不佳，因為TCP Reno是通過觀測是否有封包遺失來估計目前網路擁塞狀況及可用頻寬，來調整擁塞視窗（Congestion window，cwnd），因此，近年來發展起來的TCP Vegas已經有人開始研究在高BDP網路下的傳輸性能。TCP Vegas是依據連線的往返時間(Round Trip Time，RTT)來調整擁塞視窗的大小，比上述TCP Reno調整擁塞視窗的方式精確多了，並且也有研究指出TCP Vegas比TCP Reno有更好的頻寬利用率，且吞吐量也更大更穩定。

但是 TCP Vegas 在高 BDP 網路環境下的運作仍然有一些問題，此問題為提前離開慢啟動階段的現象，此現象使得在慢啟動階段結束時擁塞視窗大小距離 BDP 的大小仍有一段距離，導致在壅塞避免階段中的一段時間內壅塞視窗呈線性成長，即必須花費大量時間才能達到對網路頻寬的完全利用。上述因提前離開慢啟動階段後，接連出現的狀況會嚴重降低 TCP Vegas 的傳輸效率與吞吐量。因此，本文將對以上問題提出了一些改良機制，希望能解決以上的問題。

接下來的章節中，本文的第二章將介紹TCP的壅塞控制技術與運作原理並介紹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BDP網路環境下慢啟動階段的問題。第四章將介紹一種基於TCP Vegas的TCP Vegas_Q，此章節將說明TCP Vegas_Q的慢啟動演算法，如何達到較理想的慢啟動效果，以提高Vegas的傳輸性能。最後，因為修改了原本的慢啟動機制，故在壅塞避免階段也必須做一些修正，使得在慢啟動階段中修改的機制的效果得以延續。接下來，第五章將會利用第四章中所提出的改良機制利用網路模擬軟體NS-2進行模擬與驗證，最後在第六章將得到結論。

2. 文獻探討

在本章節將介紹TCP的壅塞控制技術，接下來將針對以偵測是否有封包遺失為主的TCP Tahoe、TCP Reno與依據RTT的TCP Vegas這兩類的壅塞控制技術做介紹與比較。最後會再介紹以TCP Vegas為基礎在高BDP網路上相關的壅塞控制改良版本，並解釋為什麼會以TCP Vegas作為本次研究中的基礎機制。

2.1 TCP壅塞控制

TCP是目前主流的傳輸層通訊協定，它提供了可靠性的傳輸服務、流量控制與壅塞控制機制。但是早期TCP的功能在RFC793中卻只定義六個功能：基本的資料傳輸、可靠度、流量控制、多工、連結、安全性。顯然RFC793並沒有考慮到網路壅塞現象的問題。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，導致傳輸失敗的現象，稱壅塞現象。因此，後來的RFC2001與RFC2581已經定義了壅塞控制機制。壅塞控

制機制的目的是控制資料的傳輸量與傳輸速度以避免壅塞現象的發生。

2.1.1 TCP Tahoe

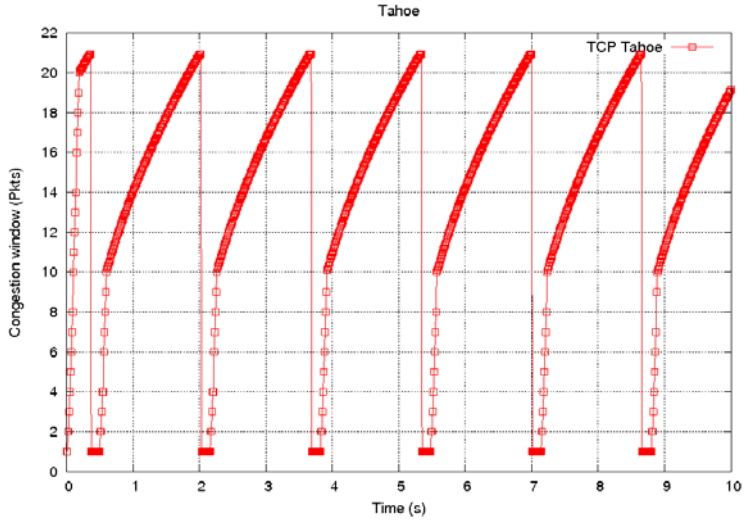
TCP Tahoe 主要分成三個階段來控制壅塞現象，分別為慢啟動階段、壅塞避免階段和快速重傳機制。

慢啟動階段：當一個連結被建立起來之後，就進入慢啟動階段。首先發送方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 timeout 以前到達發送端，則壅塞視窗增加為目前的兩倍大小，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 ssthresh（slow-start threshold）值。以上的過程被稱為慢啟動階段或是以壅塞視窗成長的型態稱指數成長階段。

壅塞避免階段：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$\frac{1}{cwnd}$，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，再次進入慢啟動階段。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：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 會利用 Duplicate ACK 當作重傳封包的依據，當有封包遺失發生時，發送端會收到 Duplicate ACK。當 TCP 收到三次的 Duplicate ACK，傳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，並且將門檻值設為偵測到有封包遺失時，當下視窗大小的一半，同時將壅塞視窗值重新設置為 1 之後，重新進入慢啟動階段。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。

接下來我們藉由圖一來觀察 TCP Tahoe 的運作。從連線一被建立起來時，先進入慢啟動階段，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，並從新設定 ssthresh 值，同時將壅塞視窗降至為 1，接著又進入慢啟動階段。



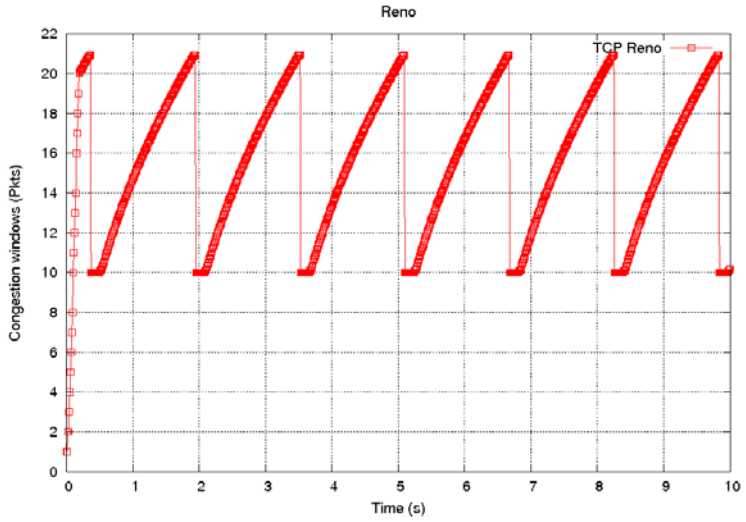
圖一：TCP Tahoe 的運作

2.1.2 TCP Reno

基本上 TCP Reno 在慢啟動階段、壅塞避免階段和快速重傳機制這三階段的運作方式和 TCP Tahoe 的運作方式是一樣的。但是 TCP Reno 增加了一個新的機制為快速復原。

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，直到接收端收到遺失封包的 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。

同樣藉由圖二來觀察 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從新設定 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，從圖二中我們可以發現，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。



圖二：TCP Reno 的運作

我們比較圖一與圖二的差別，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 BDP 網路時，會有兩個問題。第一個問題是因為高延遲往返時間引起調節速度緩慢導致網路效能下降。原因是進入壅塞避免階段後，壅塞視窗成長速度緩慢，導致無法有效利用頻寬。第二個問題是，若發生封包遺失的現象，從減半的壅塞視窗回復到最高傳輸頻寬所需時間過久，此問題可藉由觀察圖二得知，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。

2.1.3 TCP Vegas

TCP Vegas 是一種依據 RTT 來調整壅塞視窗的大小與偵測網路壅塞狀況的 TCP 版本。TCP Vegas 藉由觀察每一次量測的 RTT 時間，藉由比較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），接著再搭配慢啟動階段、壅塞避免階段與快速重傳這三個機制來調整壅塞視窗、提高吞吐量與減少封包的遺失。TCP Vegas 的壅塞控制機制如下：

$$\begin{aligned} &\text{Diff}=(\text{Expected}-\text{Actual})\times\text{base RTT} \\ &=\left(\frac{\text{CWND}}{\text{base RTT}}-\frac{\text{CWND}}{\text{RTT}}\right)\times\text{base RTT} \\ &=\text{CWND}\times\left(\frac{\text{RTT}-\text{base RTT}}{\text{RTT}}\right) \end{aligned} \qquad (1)$$

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；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與收到接收端發送此封包的 ACK 之間的時間，其中也包含了佇列延遲時間與其他開銷的時間；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；Actual 為實際的傳輸率；Diff 為實際傳輸率和預期傳輸率兩者的差值，該值也表示目前網路佇列的長度與可用頻寬的依據。

在 TCP Vegas 慢啟動階段中，為了避免發送封包的速度過快導致封包遺失的現象發生，TCP Vegas 採用減緩擁塞視窗的成長速度，擁塞視窗會經過兩個 RTT 時間增加一倍。並且 TCP Vegas 會判斷 Diff 值是否大於 γ，若 Diff 的值大於 γ（預設值為 1）則表式網路佇列長度已經超過 γ 的上限，即結束慢啟動階段，並將壅塞視窗減小$\frac{1}{8}$，之後就進入壅塞避免階段。若 Diff 的值小於 γ 則持續留在慢啟動階段。判斷慢啟動階段是否結束規則如公式（2）所示：

$$\begin{cases} \text{Diff}=\text{CWND}\times\left(\frac{\text{RTT}-\text{base RTT}}{\text{RTT}}\right)>\gamma, \text{離開慢啟動階段} \\ \text{Diff}=\text{CWND}\times\left(\frac{\text{RTT}-\text{base RTT}}{\text{RTT}}\right)<\gamma, \text{留在慢啟動階段} \end{cases} \qquad (2)$$

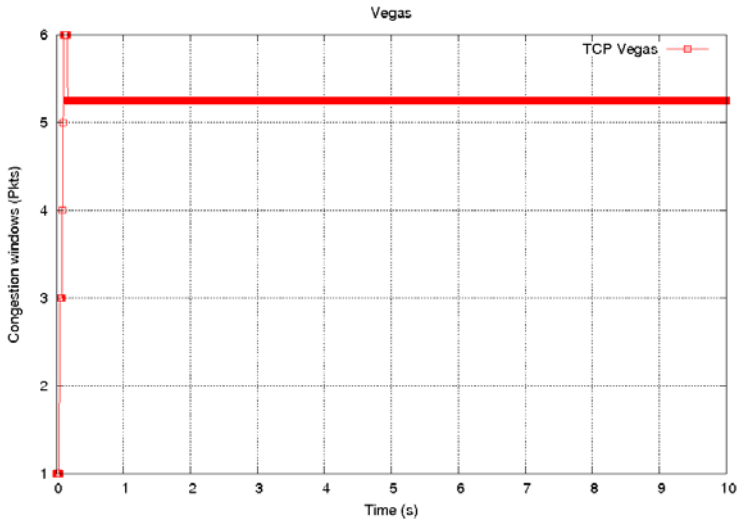
在壅塞避免階段中一共用到兩個門檻值 α 與 β（預設值為 1、3），在每一次的 RTT 時間中依照 Diff 落在由兩個門檻值形成的三個區間中的位置，來調整壅塞視窗。壅塞視窗調整的規則如公式（3）所示：

$$\text{newCWND} = \begin{cases} \text{CWND} - 1, \text{Diff} > \beta \\ \text{CWND} + 1, \text{Diff} < \alpha \\ \text{CWND}, \alpha \leq \text{Diff} \leq \beta \end{cases} \quad (3)$$

若 Diff 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 Diff 大於 β 時，表示網路現在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 Diff 是在 α 和 β 之間則不做任何變動。關於 Diff 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 的傳送端會去記錄發送出去封包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，會確認已發送的封包是否發生 Timeout，並重傳遺失的封包。

從圖三可以看到 TCP Vegas 的運作情形，從連線一開始在慢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。



圖三：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 BDP 網路上仍會碰到一些問題： 第一，因為 RTT 值過大而造成壅塞視窗更新速度過慢導致無法利用到整個網路頻寬。第二，因為 RTT 值過大導致提早結束慢啟動階段，使得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。第三，壅塞視窗成長速度仍不足夠應付高 BDP 網路。

2.2 現有的解決機制

現有不少相關論文也在探討TCP在高BDP網路上的問題，這裡將介紹一些基於高BDP網路所發展的協定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Reno為基礎和以TCP Vegas為基礎的改良版，但是以Reno為基礎的改良版有存在著壅塞窗口大小、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（S. Vanichpun and W. Feng，2002）。因此本文著眼在以TCP Vegas為基礎的改良版。

2.2.1 頻寬估計來計算新的γ

在文獻【9】中提出一種以TCP Vegas為基礎的改良版本，此版本修改TCP Vegas慢啟動階段中判斷是否結束慢啟動階段的參數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，並取代原本的γ，來達到延緩結束慢啟動階段的條件。其計算γ的公式如下：

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

其中W為壅塞視窗大小；μ為頻寬；d為最小的來回時間，base RTT。

但是在文獻【4】中指出，由於計算出來的γ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慢啟動的問題。

2.2.2 FAST TCP

Fast TCP是一個以Vegas為基礎衍生的TCP版本（C. Jin,D. Wei and S. Low，2004），Fast TCP在壅塞避免階段，採用倍數的方式調整壅塞窗口，由此達到快速適應網路環境的效果。Fast TCP會去記錄每次的RTT和設定base RTT，利用二者的比例去更改壅塞窗口的大小，以下為計算壅塞窗口的方式如下：

$$W \leftarrow \min\left\{2W, (1-\gamma)W + \gamma\left(\frac{\text{base RTT}}{\text{RTT}}W + \alpha\right)\right\} \qquad (5)$$

其中W代表壅塞視窗的大小，Fast TCP利用每一次量測的RTT和base RTT來計算出壅塞視窗要改變的比例，在與兩倍的壅塞視窗取最小值，此機制可以防止壅塞窗口增加速度過快。α代表停留在佇列中的封包個數，通常預設為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。

2.3 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 2.2.1 小節的改良機制與 2.2.2 小節的 FAST TCP 後，發現 TCP Tahoe 與 TCP Reno 因為本身機制的設計會讓壅塞視窗呈現周期性震盪的現象，此現象會造成在高 BDP 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 BDP 網路環境下仍有頻寬利用率不高的問題。所以有了 2.2.1 小節與 Fast TCP 的改

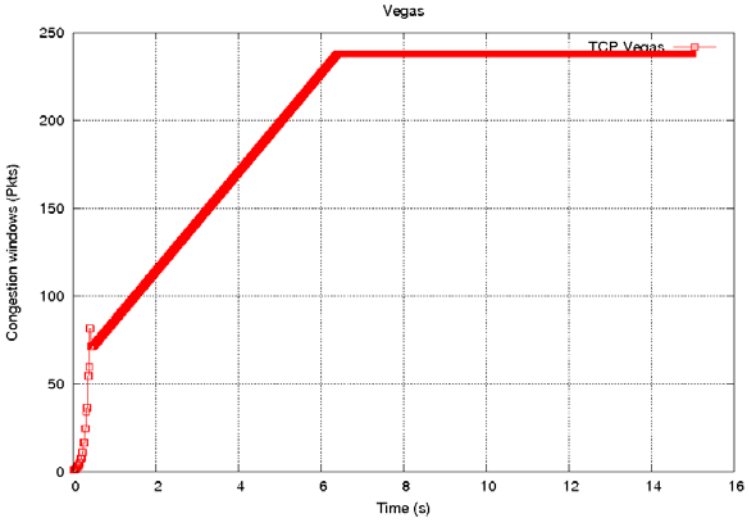
良機制，但是 2.2.1 小節的改良機制仍有過於複雜容易受外來因素影響，Fast TCP 有 α 值的選擇問題，雖有缺點但是他們的基礎機制均選擇 TCP Vegas，原因是用 RTT 的改變來預測可用頻寬的方式，比 TCP Reno 或以此為基礎的版本用偵測封包遺失的方式精確許多，並且 TCP Vegas 的公平性與效率也較好。所以本文決定以使用測量 RTT 來調整壅塞視窗大小的 TCP Vegas 機制為基礎機制。

3. TCP Vegas 慢啟動階段問題分析

在2.1.3小節中，我們提到TCP Vegas的慢啟動階段的運作流程。在理想的情況下，慢啟動階段會在壅塞視窗的大小接近所處在的網路的可用頻寬時結束（顧明、張軍、蘇東林，2007）但實際上，經觀察TCP Vegas應用在高BDP網路環境下會發生壅塞視窗成長的大小並無法做到有效利用頻寬，且距離網路的BDP值有一段距離，即表示發生Diff提前增大並大於γ，接著就離開慢啟動階段並進入壅塞避免階段，最後進入壅塞避免階段後就會出現在本文中2.1.3小節所提到TCP Vegas的第二個問題。

在 2.1.3 小節中我們提到 TCP Vegas 在慢啟動階段中經過兩個 RTT 時間才將擁塞視窗成長一倍。在這兩個 RTT 時間，其中一個 RTT 時間稱成長期，在這段期間若發送端收到上一回發送的封包中的第一個封包的 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中的最後一個封包的 ACK 後為止（張凱翔，2005），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。TCP Vegas 的慢啟動階段就是一直重覆增加視窗與觀測兩個動作。其中當觀察期結束後，TCP Vegas 會利用量測到的 RTT 並計算 Diff 來決定慢啟動結束與否。現在我們更進一步指出提前離開慢啟動階段的原因就是觀察期結束後，量測到的 RTT 變大，導致 Diff 增大。

造成 RTT 變大的原因與成長期的壅塞視窗成長方式有關，在慢啟動階段後期，會由於壅塞視窗的增大再加上慢啟動階段中，封包在短時間（在一個 RTT 時間）內密集的被發送的行為，導致網路上的佇列堆積大量的封包，使得每一次所量測到的 RTT 會多算到封包停留在佇列的時間，因此使得 Diff 值大大增大並超過 γ，最後導致提早結束慢啟動階段並進入擁塞避免階段，如圖四所示。提早結束慢啟動階段會使得擁塞視窗在擁塞避免階段中的一段時間呈線性增長，此現象將造成擁塞避免階段將消耗大量的時間來達到平衡，這將影響 TCP 的傳輸效率。



圖四：壅塞視窗成長示意圖（網路的 BDP 為 800）

4. 改進機制介紹

在這章節中將介紹基於高 BDP 網路環境的 TCP Vegas 改良版本，TCP Vegas_Q。首先，針對慢啟動階段會說明修改的部分與演算法，接下來的部分將介紹擁塞避免階段修改的部分。

4.1 慢啟動階段的修改

修改過的慢啟動機制的目標是在高的 BDP 網路環境中，使得慢啟動階段結束時能使擁塞視窗大小接近 BDP 大小。我們所提出的改良機制是透過一個常數 k 來衡量目前的網路狀況，常數 k 代表了每一次所量測的 RTT 與 base RTT 的比值，並且也透過 base RTT 在每一次所量測的 RTT 中佔的比例來決定慢啟動階段的結束與否以及壅塞視窗的成長倍數。

$$k = \frac{\text{base RTT}}{\text{RTT}} \tag{6}$$

在擁塞視窗的增加的方式部份，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，其中成長的倍數則透過是否有封包堆積在佇列與 RTT 與 base RTT 的比值來決定增加當前壅塞視窗的大小的 0.5 倍、0.25 倍與 0.0625 倍。

$$\text{newCWND} = \begin{cases} \text{oldCWND} + (\text{oldCWND} \times 0.5), & \text{Diff} \le 2 \\ \text{oldCWND} + (\text{oldCWND} \times 0.25), & \text{Diff} > 2 \text{ 且 } k \ge \left(\frac{1-\text{預設的比例值}}{2}\right) + \text{預設的比例值} \\ \text{oldCWND} + (\text{oldCWND} \times 0.0625), & \text{Diff} > 2 \text{ 且 } k < \left(\frac{1-\text{預設的比例值}}{2}\right) + \text{預設的比例值} \\ & \text{且 } k \ge \text{預設的比例值} \end{cases} \tag{7}$$

此成長方式和原本的 Vegas 比較之下，在同一時間內可以有效緩和壅塞窗口在短時間內密集地發送封包導致發生暴衝現象，也舒緩網路佇列的堆積狀況。接下來改良機制更進一步將原本的慢啟動階段分成兩個階段：

在第一階段中，連線一開始就進入第一階段，當Diff≦2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時間增加當前壅塞視窗的壅塞視窗大小的0.5倍，直到Diff>2為止並進入階段二。

在第二階段中，由於佇列上已經有封包堆積的現象，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的比值，即常數k。當常數k小於一個預設的比例值時表示每一次所量測的RTT中的佇列延遲已經變大了，故此時就離開慢啟動階段。同時在此階段中在常數k小於一個預設的比例值之前壅塞視窗的成長會依據常數k是否大於$\left(\frac{1-\text{預設的比例值}}{2}\right)+\text{預設的比例值}$或是小於$\left(\frac{1-\text{預設的比例值}}{2}\right)+\text{預設的比例值}$且大於預設的比例值，來決定壅塞視窗的成長倍數為0.25倍或是0.0625倍。

修改過的慢啟動階段程式碼如下：

```
01  if(Diff <= 2){
02      v_incr_ = 0.5;
03  }
04  else if(Diff > 2){
05          if((Diff > 2) && (k < 1)){
06                  k    =    base RTT / rtt;
07                  if(k <  預設的比例值){
08                          v_incr_ = 0;
09                          slow-start end
10                  }
11                  else{
12                          if(k >= ( ( (1 -預設的比例值) / 2 ) +  預設的比例值  ) ){
13                                  v_incr_ = 0.25;
14                          }
15                          else if ((k < ( ( (1 -預設的比例值) / 2 ) +  預設的比例值) && (k >=預
      設的比例值)){
16                                  v_incr_ = 0.0625;
17                          }
18                  }
19          }
```

```
20  }
```

4.2 壅塞避免階段的修改

修改壅塞避免階段的目的是為了修改α與β，讓連線能夠適應當下不斷劇烈變化的環境。

修改過的壅塞避免機制會在第一次進入壅塞避免階段時利用慢啟動階段結束時的Diff值與第一次進入壅塞避免階段的Diff的值來修改α與β值，並用此組可以反映當時的網路狀況的α與β值來調整往後的壅塞視窗大小，目的是防止慢啟動階段結束後因為較大的Diff值導致壅塞視窗大幅下降的問題。接下來壅塞避免階段仍分成三個階段來進行調整壅塞視窗。

在第一階段，Diff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，故在此階段需要減小壅塞視窗的大小，因此在第一階段壅塞視窗的大小都會減少（Diff-β）/ 2。在此階段中並不對α與β做修改。

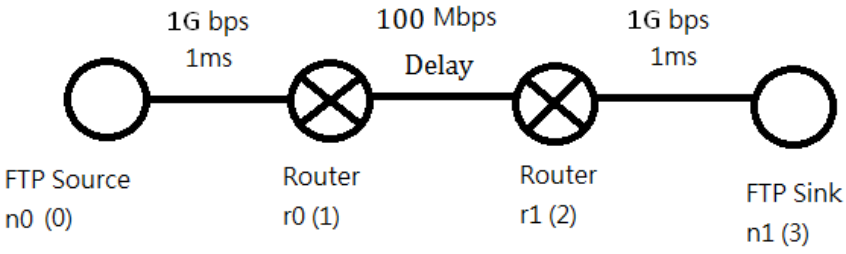
第二階段，Diff小於α的情況下，表示網路狀況良好，沒有壅塞，壅塞視窗可以增長，故在第二階段中，壅塞視窗的大小都會增加（β-Diff）。在此階段中也會修改α與β值，修改過的α與β值會給下一回壅塞避免階段使用，目的是讓壅塞視窗仍有成長的機會，不要提前進入（α<diff<β）階段。

第三階段，即α<diff<β，則不對α、β與壅塞視窗做任何調整。以下是改過的壅塞避免階段程式碼：

```
01  if(第一次進入壅塞避免階){
02          temp_Diff = (Diff + before_Diff) / 2;
03          beta = temp_Diff;
04          alpha = temp_Diff / 2;
05  }
06  if(Diff > beta){
07          cwnd = cwnd - ( (Diff - beta) / 2 );
08  }
09  else if(Diff < alpha){
10          beta = beta + 10;
11          alpha = alpha + 5;
12          cwnd = cwnd + (beta - Diff);
13  }
14  else
15          v_incr_ = 0;
```

5. 實驗結果

為了驗證在第四章中由本文所提出的 TCP Vegas_Q，我們使用網路模擬軟體 NS-2 來驗證我們的改良機制。模擬的拓樸如下圖所示。



圖五：實驗環境拓樸

上圖是一個由兩個路由器再加上兩個節點所組成的網路環境。r0 和 r1 為路由器。FTP Source（n0）經由 r0 連上網路，並將資料發送給 FTP Sink（n1）。

路由器的 buffer 管理機制為 DropTail，路由器相互連接線段頻寬為 100Mbps，傳輸延遲時間視狀況調整。發送端和接收端與路由器連接的線段的頻寬為 1Gbps，傳輸延遲時間為 1ms，如此設計是為了藉由改變路由器相互連接線段（瓶頸點）之間的延遲，來調整實際的 BDP 大小與造成一個容易產生壅塞現象的環境。最後預設封包大小為 1KB，來回時間預設為 100ms，緩衝區容量為目前網路的 BDP。

由於壅塞視窗即為發送的封包量，也反映了 TCP 的傳輸效能，故我們以壅塞視窗為觀察期傳輸效能的參數。

表格一：模擬環境參數表

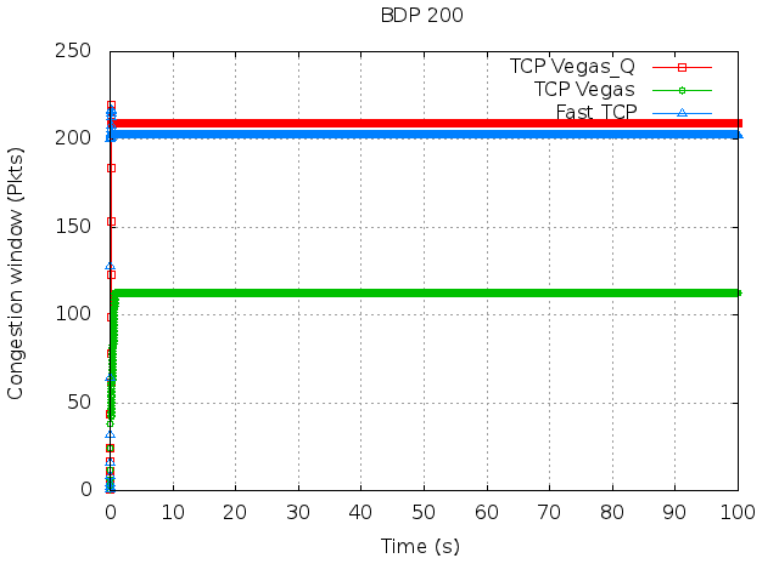
模擬環境參數表	
網路模擬器	NS-2
瓶頸點頻寬	100Mbps
瓶頸點傳輸延遲時間	視 BDP 而調整
瓶頸點以外頻寬	1Gbps
瓶頸點以外傳輸延遲時間	1ms
路由器的 buffer 管理機制	DropTail
封包大小	1KB
來回時間	100ms
緩衝區容量	BDP

在模擬的過程中， TCP Vegas 使用的參數值為($\alpha=2$、$\beta=4$、$\gamma=2$)，TCP Vegas_Q 因為不是使用 γ 來判斷是否離開慢啟動階段且 α 與 β 會因當下網路狀況改變故沒有預設值，取而代之的是 base RTT 與 RTT 的預設的比例值，在此次實驗中預設的比例值預設為 0.7，Fest TCP 則使用預設值，接著將透過改變瓶頸點之間的延遲，於數種不同大小的 BDP 值下，除了比較三者壅塞視窗成長狀況，並且也針對在慢啟動階段結束後 TCP

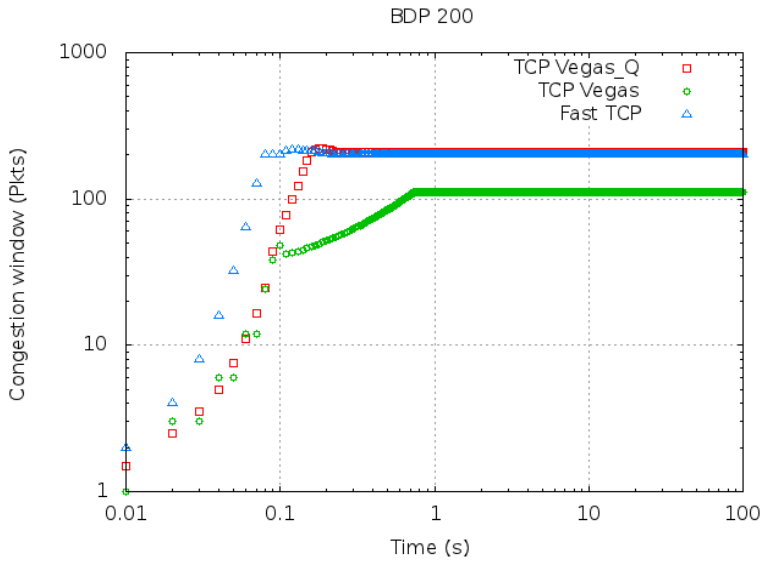
Vegas 與 TCP Vegas_Q 的壅塞視窗大小作比較。從下面的表格二與圖片（圖六－圖十七）中我們可以發現當 BDP 值愈大時 TCP Vegas_Q 效果愈好。

表格二：慢啟動階段結束時壅塞視窗大小比較

慢啟動階段結束時壅塞視窗大小		
BDP	TCP Vegas	TCP Vegas_Q
3000	192	3389.88
2000	192	2259.62
1000	96	963.375
800	96	770.812
400	48	410.875
200	48	219.938

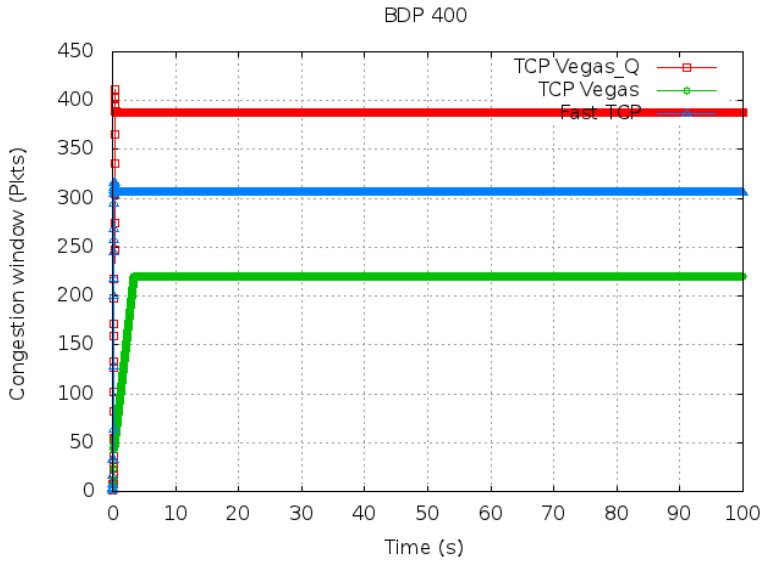


圖六：BDP 為 200 時， 各版本 TCP 的壅塞視窗變化 1

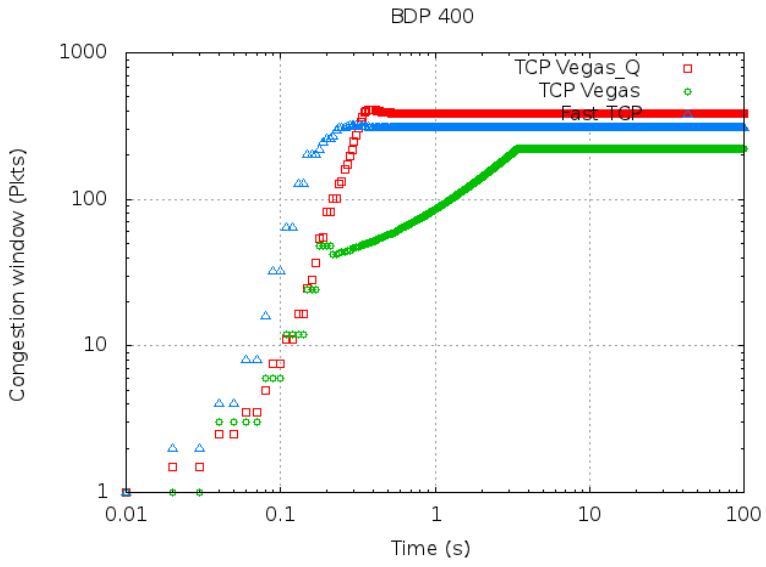


圖七：BDP 為 200 時， 各版本 TCP 的壅塞視窗變化 2

在圖六與圖七中，網路的 BDP 為 200 時，Fast TCP 與 TCP Vegas_Q 兩者在大約分別在 0.12 到 0.18 秒之間兩者的壅塞視窗大小就有很明顯的提升，TCP Vegas_Q 的壅塞視窗大小接近 Fast TCP 的壅塞視窗大小。同時 TCP Vegas 卻早在 0.09 秒時已經離開慢啟動階段。

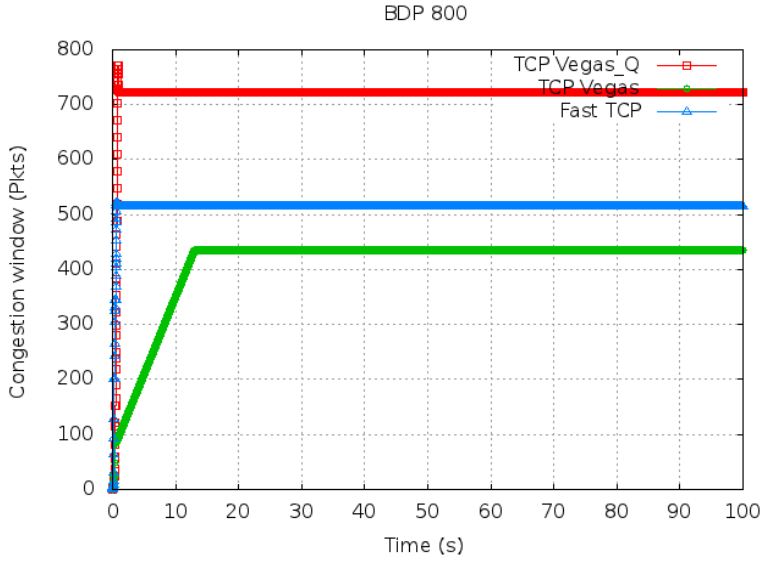


圖八：BDP 為 400 時，各版本 TCP 的壅塞視窗變化 1

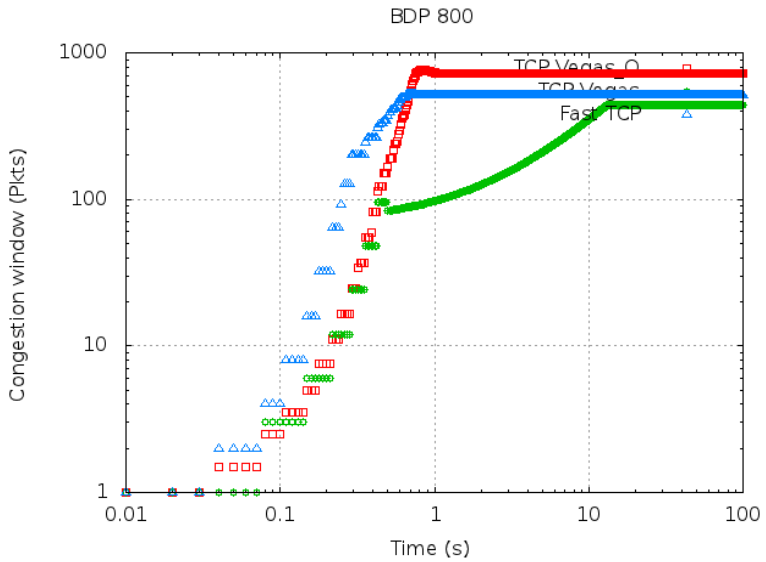


圖九：BDP 為 400 時， 各版本 TCP 的壅塞視窗變化 2

在圖八與圖九中，網路的 BDP 為 400 時，Fast TCP 在 0.28 秒時壅塞視窗以達到最高點，TCP Vegas_Q 則在 0.38 秒到達最高點，雖然在圖九中兩者到達最高點的時間並不一樣但是壅塞視窗的大小已有差別，但 TCP Vegas 卻早在 0.18 秒時已經離開慢啟動階段。

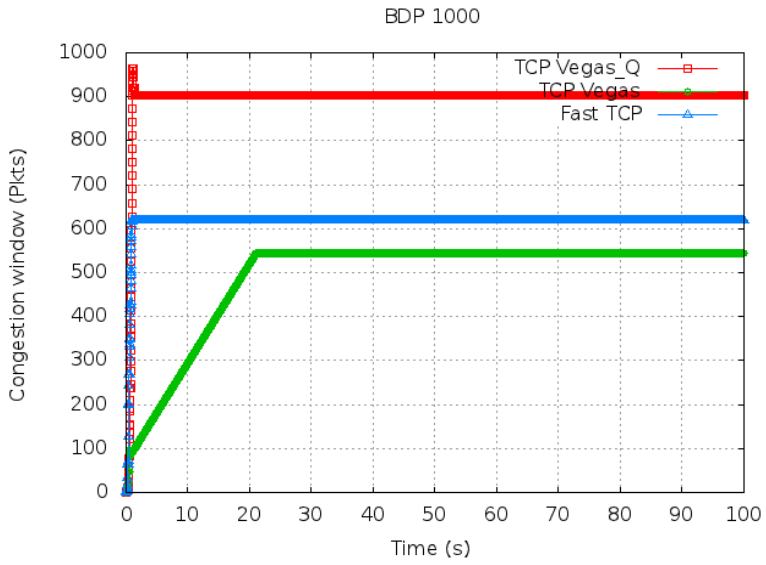


圖十：BDP 為 800 時，各版本 TCP 的壅塞視窗變化 1

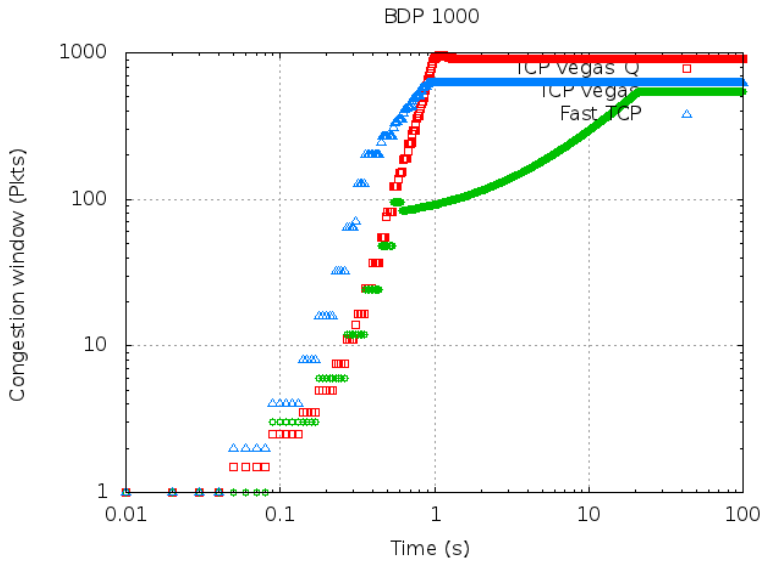


圖十一：BDP 為 800 時，各版本 TCP 的壅塞視窗變化 2

在圖十與圖十一中，網路的 BDP 為 800 時，Fast TCP 在 0.67 秒時壅塞視窗以達到最高點，TCP Vegas_Q 則在 0.82 秒到達最高點，雖然在圖十一中兩者到達最高點的時間並不一樣但是壅塞視窗的大小已有明顯的差別，但 TCP Vegas 卻早在 0.43 秒時已經離開慢啟動階段。

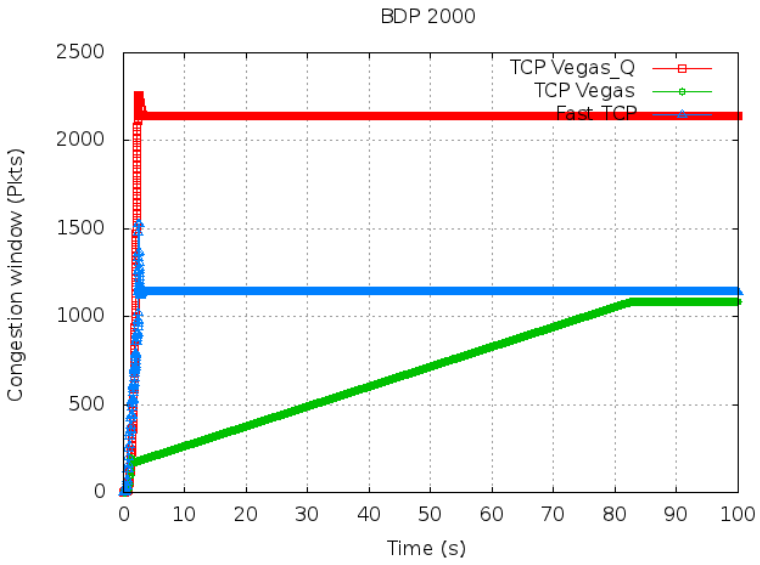


圖十二：BDP 為 1000 時，各版本 TCP 的壅塞視窗變化 1

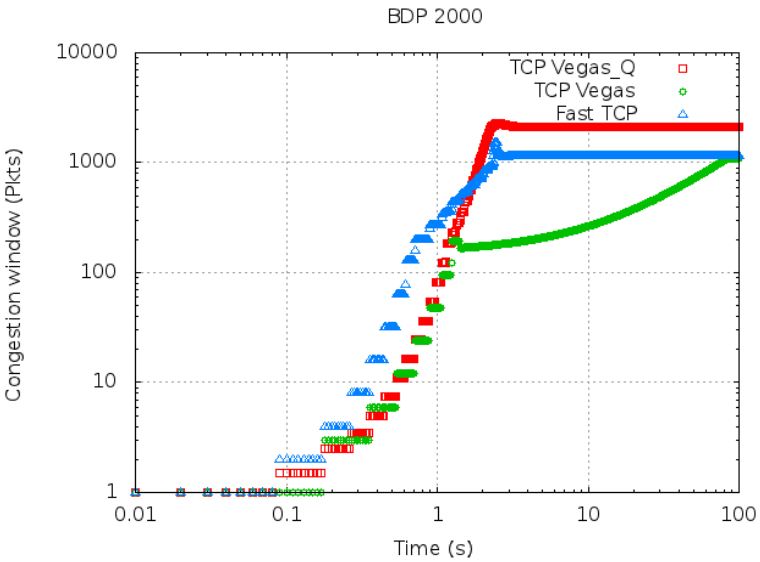


圖十三：BDP 為 1000 時，各版本 TCP 的壅塞視窗變化 2

在圖十二與圖十三中，網路的 BDP 為 1000 時，Fast TCP 在 1.01 秒時壅塞視窗以達到最高點，TCP Vegas_Q 則在 1.07 秒到達最高點，雖然在圖十三中兩者到達最高點的時間並不一樣但是壅塞視窗的大小已有明顯的差別，但 TCP Vegas 卻早在 0.54 秒時已經離開慢啟動階段。

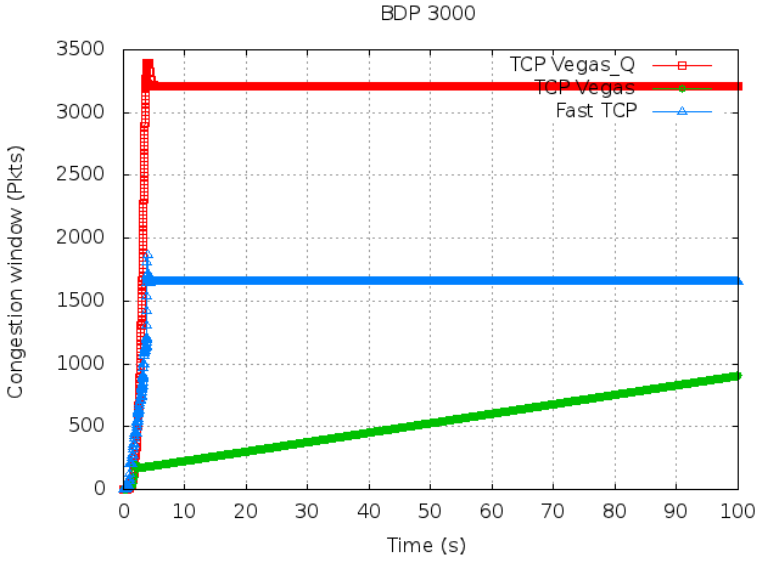


圖十四：BDP 為 2000 時，各版本 TCP 的壅塞視窗變化 1

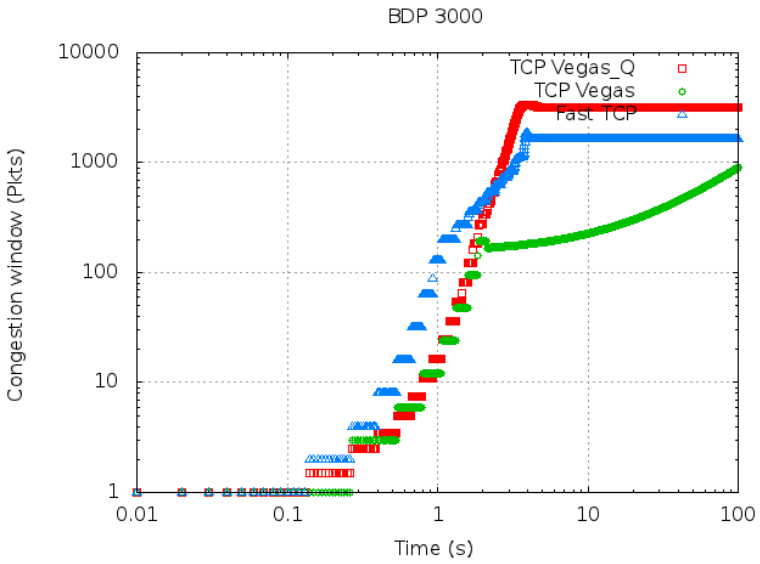


圖十五：BDP 為 2000 時，各版本 TCP 的壅塞視窗變化 2

在圖十四與圖十五中，網路的 BDP 為 2000 時，Fast TCP 在 2.40 秒時壅塞視窗以達到最高點，TCP Vegas_Q 則在 2.45 秒到達最高點，雖然在圖十五中兩者到達最高點的時間並不一樣但是壅塞視窗的大小已有明顯的差別，但 TCP Vegas 卻早在 1.25 秒時已經離開慢啟動階段。



圖十六：BDP 為 3000 時，各版本 TCP 的壅塞視窗變化 1



圖十七：BDP 為 3000 時，各版本 TCP 的壅塞視窗變化 2

在圖十六與圖十七中，網路的 BDP 為 3000 時，Fast TCP 在 4.98 秒時壅塞視窗以達到最高點，TCP Vegas_Q 則在 3.82 秒到達最高點，雖然在圖十七中兩者到達最高點的時間並不一樣但是壅塞視窗的大小已有明顯的差別，但 TCP Vegas 卻早在 1.87 秒時已經離開慢啟動階段。

綜觀圖六到圖十七，我們發現 TCP Vegas_Q 與 Fast TCP 隨著網路 BDP 的提高，兩者均能在短時間提高壅塞視窗，反觀 TCP Vegas 則明顯的無法在短時間內利用慢啟動階段提高自己壅塞視窗，TCP Vegas 必須利用在壅塞避免階段慢慢的提高自己的壅塞視窗。在相同時間內壅塞視窗提高的幅度上 TCP Vegas_Q 比 Fast TCP 高，TCP Vegas 最低，這表示 TCP Vegas_Q 可以比 Fast TCP 更有效的利用頻寬，TCP Vegas 無法有效利用頻寬。

6. 結論與未來展望

本文提出了一個針對高 BDP 網路環境的 TCP Vegas 改良版本，在此改良版本的慢啟動階段部分，修改了壅塞視窗的成長方式，使壅塞視窗的成長不會過快，並利用每一次所量測的 RTT 與 base RTT 的比值來衡量網路狀況做為是否離開慢啟動階段的依據，改良過的慢啟動階段可以避免發生慢啟動階段提前結束的現象，且此方式在實際使用上計算較為簡單也比較容易實施，也不容易受網路上其他因素干擾，且根據實驗後得到的數據與結果發現在愈高的 BDP 網路上效果愈明顯。最後也在壅塞避免階段針對 α 與 β 的修改，達到快速的讓壅塞視窗達到穩定來維持壅塞視窗的大小與提高吞吐量。

本文未來的努力目標是改善瓶頸點的佇列使用狀況及慢啟動階段結束後達到穩定的速度，根據實驗發現，在整體的運作過程中，堆積在瓶頸點上的封包仍然嫌多與達到穩定的速度與 Fast TCP 仍然嫌稍慢，希望能在未來解決此項問題。

參考文獻

1. 張凱翔，2005，改善TCP Vegas慢啟動機制之研究，國立彰化師範大學資訊工程學系碩士論文。
2. 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 慢啟動策略，計算機應用研究，第 28 卷，第 1 期：253-255 頁。
3. 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。
4. 顧明、張軍、蘇東林，2007年四月，大頻寬時延積網路TCP Vegas自我調整慢啟動演算法，電訊技術，第47卷，第2期：27-30頁。
5. Brakmo L S and Peterson L L ” TCP Vegas：End—to—endCongestion Avoidance on a Global Interact[J” IEEEJournal on Selected Areas in Communication 1995,pp:1 -1480
6. 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.
7. J. Postel, “Transmission Conter Protocol”, RFC 793, Sep. 1981.
8. M. Allman、V. Paxson and W. Stevens, RFC 2581:TCP Congestion Control, April 1999.
9. S. Vanichpun and W. Feng, “On the transient behavior of TCP Vegas,” in Proc. IEEE ICCCN’02 Oct. 2002, pp: 504-508.
10. Stevens, W., "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms", RFC 2001, Jan. 1997.

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

Author[1] Shen-Huei Lo Author[2] Hsin-Yi Wang Author[3] Yen-Ping Chu

[1]Tunghai University Computer Science Department g99350053@thu.edu.tw
[2] National Chung-Hsing University Computer Science and Engineering Department d100056009@mail.nchu.edu.tw
[3] Tunghai University Computer Science Department ypchu@thu.edu.tw

Abstract

Low bandwidth TCP protocol has been unable to qualify current network environment in high-bandwidth delay product (BDP) network. However, how to enhance TCP performance in BDP transmission network has become a research focus now. In this paper the enhancement of slow-start mechanism of TCP Vegas is investigated and the researcher tries to solve TCP Vegas to leave slow-start phase early.

Leaving slow-start phase early will result in the size of congestion window closed and insufficient to BDP. This phenomenon will reduce the transmission efficiency and TCP Vegas throughput severely. In this paper the researcher proposed a new concept, the modified slow-start phase in TCP Vegas. Simulation results showed that can solve the problem of leaving slow-start phase early.

Keywords: slow-start、TCP Vegas、BDP、congestion window、RTT