高頻寬延遲乘積網路上之 TCP Vegas 效能改進研究

詹益禎/王孝恩*/徐志榮/黃柏霖/許嘉樺/黃瀚璋/黃琦閔
國立彰化師範大學資訊工程學系
Email: *s92610012@mail.ncue.edu.tw

摘要

Reno和Vegas是現今最被廣泛討論的兩個TCP壅塞控制技術，其中Reno已在網際網路上被普遍採用，然而隨著網路鏈結頻寬的增加以及封包傳輸距離的加長，許多研究文獻已經證明，Reno無法在高頻寬延遲乘積網路(high bandwidth-delay product networks)上有效地利用網路資源。另一方面，Vegas藉由分析封包來回的延遲時間靈活的調整其壅塞窗口大小，許多實驗已經證明在整體的網路使用效率(utilization)、穩定性(stability)、公平性(fairness)、以及封包的遺失情形(packet loss)上相較於Reno，Vegas都有較佳的表現。於是我們以Vegas做為改良的基礎，提出了我們的建議『Vegas+』。Vegas+改進了Vegas的緩啟動(Slow Start)以及壅塞避免(Congestion Avoidance)機制，經由模擬程式以及網路實驗證明，Vegas+確實可讓TCP在高頻寬延遲乘積(bandwidth-delay product, BDP)網路中，有更快速且穩定的傳輸速率，顯著提升了網路頻寬的使用效能。

關鍵詞：高頻寬延遲乘積網路、傳輸層協定、壅塞控制、TCP Vegas。

Abstract

Reno and Vegas are the two most famous TCP variants, especially for Reno, it has been widely deployed on the current Internet. However, due to the great progress of network technologies and the continuous extension of network boundaries, the per-flow product of available bandwidth and delay increases. Many papers have shown that as the per-flow product of bandwidth and latency increases, Reno becomes inefficient and prone to instability. In the other hand, by investigating the round-trip delay of packets, Vegas may intelligently adjust its congestion window size. Studies have demonstrated that Vegas outperforms Reno in the aspects of overall network utilization, stability, fairness and packet loss. In this paper, we propose a new mechanism, Vegas+, that improves the slow-start and congestion avoidance techniques of original Vegas. Through the various simulation and experiment results we can find that Vegas+ significantly improves the performance of connections when the bandwidth-delay product increases.

Keywords: high bandwidth-delay product networks, transport protocol, congestion control, TCP Vegas.

1. 簡介

TCP傳輸協定是目前網際網路上最主要的傳輸層協定，它提供網際網路端對端(end-to-end)間一個可以自我調適速度的可靠性傳輸，避免網路資源因過度被使用而造成的崩潰。現今網路應用層如HTTP、FTP、SMTP、TELNET…等，皆以TCP作為傳輸層協定，因此TCP在網路上扮演著極重要的角色。不過在許多的理論與實驗中顯示出TCP在高頻寬延遲乘積網路中會變的非常沒有效率[1]，這個問題隨著網際網路頻寬的提高、傳輸延遲的增加而變得日益嚴重。

TCP Reno [2]是現今網際網路上最為被廣泛使用的 TCP 版本，它以 AIMD (Additive

Increase and Multiplicative Decrease)演算法調整其壅塞窗口(congestion window, CWND)，亦即它會不斷的增大其壅塞窗口以加快連線(connection)的傳送速率，直到發現封包遺失之後才將壅塞窗口減半，而後再繼續增大其壅塞窗口。TCP Reno 會主動造成封包週期性的遺失，使連線的壅塞窗口、封包的往返時間(round-trip time, RTT)以及網路瓶頸點(Bottleneck)上的佇列(buffer)長度處於震盪的情形之中，很多研究報告指出這種震盪的網路現象不適合許多新的網際網路應用而且也無法善用網路資源[3]。

相較於遺失式(loss-based)的壅塞控制，TCP Vegas [4,5]使用延遲式(delay-based)的壅塞控制技術分析封包來回的延遲時間，它能夠偵測出初期的網路壅塞並加以有效的控制，使得網路的整體效率[5,6,7]、穩定性[3,8]、連線間的公平性[3,8]以及封包遺失的情形[3,5,6,7]等皆有大幅的改善，更重要的是延遲式壅塞控制免除週期性封包遺失的現象，不會主動造成網路的震盪，徹底解決了遺失式壅塞控制根深蒂固的包袱[3,5,6,7]。但是在高頻寬延遲乘積網路下，TCP Vegas會有過早離開緩啟動進入壅塞避免階段的現象，使得Vegas停止以指數成長的方式增加壅塞窗口而開始以線性方式緩慢的調整壅塞窗口，直到它的壅塞窗口的大小足以充分使用頻寬[9]。由於Vegas會產生這種現象，導致一條新啟動的連線必須要經過一段很長的時間才可充分使用頻寬。除此之外，因為網路上可用的資源和使用者的數目是並非以線性方式在改變[10]，而TCP Vegas在壅塞避免階段卻採用線性方式調整其壅塞窗口，導致無法快速的調整到理想的大小，使得TCP Vegas在高頻寬延遲乘積網路的環境下反應過於緩慢。

在這篇論文中，我們針對 TCP Vegas 在高頻寬延遲乘積網路上的不足提出一個新的改進版本名為 Vegas+。在緩啟動階段，我們增快調整壅塞窗口的頻率，減少每次調整的幅度，藉此不僅可以增加壅塞窗口增加的速度也可以緩和暴衝(bursty)現象所帶來的問題。在壅塞避免階段，根據連續成功增加壅塞窗口的次數[11]和實際上囤積在瓶頸點的封包數量，更改壅塞避免階段的演算法使壅塞窗口調整的方式更靈活，經過實驗證明，修改後的 TCP 連線在高頻寬延遲乘積網路中能夠更快速的調整壅塞窗口而且也增進了整體的效能。

接下來的章節安排如下：第二章中我們對相關研究進行回顧。第三章詳細說明我們所提出來的方法『Vegas+』。第四章呈現並分析在 ns-2 模擬以及 Linux 實驗時所產生的數據。最後在第五章做出結論。

2. 相關研究回顧

由於TCP在高頻寬延遲乘積網路上的問題日益嚴重，近幾年已有許多針對性的研究，其中以 Reno 為基礎的有 BIC TCP [12]、CUBIC-TCP [13]、STCP [14]、LTCP [15]、HTCP [16]、HSTCP [17]、TCP Westwood [18]和TCP Africa [19]；另外以TCP Vegas為基礎的改良版本有AdaVegas [20]以及Fast TCP [21]。

BIC和CUBIC這兩種機制具有類似的行為，都是藉由不斷的計算目標壅塞窗口值來調整壅塞窗口改變的速度；LTCP則是採用分層的概念，在不同層級有不同的調整速度；STCP、HTCP和HSTCP，他們藉著調整AIMD的比例加快壅塞窗口改變的速度；TCP Westwood則是嘗試著去偵測出可用頻寬的多寡，做為其調整壅塞窗口的依據；TCP Africa 使用 MIMD (Multiplicative Increase and Multiplicative Decrease)的方式去調整壅塞窗口。以上這些版本皆是出自於改進TCP Reno的方案，因此無可避免的，壅塞窗口大小、來回時間延遲(round-trip delay)、瓶頸點佇列長度等皆會有大

幅振盪的現象，這些缺點不利於許多新的網際網路應用[3]。

AdaVegas [20]是 TCP Vegas 的一衍生版本，在壅塞避免階段為了使壅塞窗口調整的更快速，於是他去記錄每次成功傳送封包的次數，當達到一定的值後就會改變壅塞窗口增加的幅度，當傳送失敗時也會根據壅塞窗口之前增加量的多寡，去減少一定量的壅塞窗口，增加的愈多減少的也愈多。

Fast TCP 則是另一以 Vegas 為基礎衍生的版本[21]，Fast TCP 更改了 Vegas 的壅塞避免階段中壅塞窗口調整的方式。在壅塞避免階段，Fast TCP 為了能夠快速的調整壅塞窗口的大小，他捨棄了 TCP Vegas 線性調整壅塞窗口的方式，改採用倍數的方式調整壅塞窗口，藉此達到快速適應網路環境的效果。Fast 會去記錄每次的 RTT 和設定 BaseRTT，藉著二者的比例去更改壅塞窗口的大小，計算壅塞窗口的方式如下：

$$w \leftarrow \min\left\{2w, (1-\gamma)w + \gamma\left(\frac{\text{BaseRTT}}{\text{RTT}} \times w + \alpha\right)\right\}. \quad (1)$$

w 代表壅塞窗口的大小，運用 BaseRTT 除以 RTT 算出壅塞窗口要改變的比例，在與兩倍的壅塞窗口做比較，如此就不會發生壅塞窗口提升速度過快。此時的α通常被設為 100，當囤積瓶頸點上的封包數目Δ遠小於α時，Fast 會以倍數的方式增加壅塞窗口的大小，當Δ很接近α時 Fast 則以線性的方式調整，Fast 試著將囤積在瓶頸點上的封包數目維持在α附近，若延遲時間突然增加則 Fast 也會快速的減少傳送速率避免壅塞發生。根據理論和實驗發現 Fast 在瓶頸點的使用效率、收斂速度、連線間的公平性以及穩定性，都有較好的表現[21]。

3. Vegas 與 Veags+

TCP Vegas 是一種延遲式的壅塞控制演算法，主要利用封包來回的延遲時間去調整壅塞窗口的大小，相較於 Reno，Vegas 在下面的三個演算法都做出了改進：(1) 壅塞避免 (2) 緩啟動 (3) 快速重送和恢復(fast retransmission and recovery)，由於 Vegas+修改了 Vegas 的緩啟動和壅塞避免這兩階段的演算法，所以我們在這裡對於 Vegas 的緩啟動以及壅塞避免演算法詳加說明。

3.1 Vegas

首先我們介紹壅塞避免階段，在此階段下 TCP Vegas不會盲目的持續增加壅塞窗口，取而代之的是運用預期的吞吐量 (Expected Throughput) 和實際的吞吐量 (Actual Throughput)做比較，推算出有多少封包被囤積在瓶頸點，我們稱這個值為Δ，且根據Δ調整壅塞窗口的大小。Vegas會去記錄每次的RTT和設定最小的來回時間(BaseRTT)，利用以下的公式是推算出有多少封包被囤積在網路中：

$$\Delta = (Expected - Actual) \times BaseRTT . \quad (2)$$

預期的吞吐量是使用現在的壅塞窗口去除以 BaseRTT 得到的，而實際的吞吐量則是使用現在的壅塞窗口去除以這次的 RTT。Δ若介於門檻值α和β之間則壅塞窗口保持不變(在此篇論文中若無特別說明則α=2、β=4)；如果Δ大於β代表網路可能快發生壅塞，因此將壅塞窗口大小減少，另外一方面若Δ小於α，代表此條連線可能未完全利用到全部的可用頻寬，因此壅塞窗口會增加。Vegas 在每經過一個 RTT 後會依照下列的方法調整壅塞窗口的大小：

$$CWND = \begin{cases} CWND + 1, & \text{if } \Delta < \alpha \\ CWND - 1, & \text{if } \Delta > \beta \\ CWND, & \text{if } \alpha \le \Delta \le \beta \end{cases} . \quad (3)$$

在緩啟動階段，Vegas 希望連線(connection)能夠快速的使用可用頻寬，因此與 Reno 相同都以指數的方式增加壅塞窗口，然而為了避免和

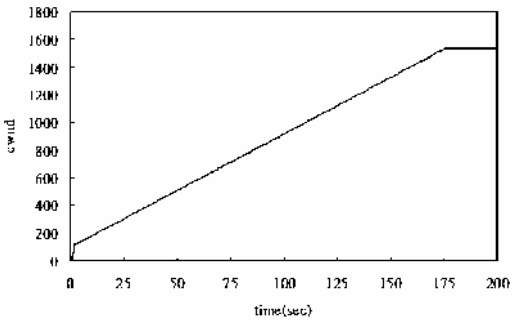

圖 **1**：在一獨占環境下 **Vegas** 連線壅塞窗口變化情形。

有效偵測壅塞的發生，於是 Vegas 改採每兩個 RTT 增加一倍壅塞窗口的方式去調整壅塞窗口，同時也比較預期吞吐量和真實吞吐量，推算出Δ值，運用Δ值去決定是否要進入壅塞避免階段，如果Δ大於γ，Vegas 會減少 1/8 的壅塞窗口然後離開緩啟動階段進入壅塞避免階段。

3.2 Vegas+

TCP Vegas 主要有兩個問題，第一 TCP 在緩啟動階段為了處理大量增加的封包，於是 TCP 會以非常急促的方式傳送封包，這種現象導致 TCP Vegas 在高頻寬延遲乘積網路中過早從緩啟動階段切換至壅塞避免階段，我們稱之為暴衝現象(Bursty)；第二 TCP Vegas 在頻寬改變後無法快速的調整壅塞窗口。在我們提出的 Vegas+中，我們試著緩和暴衝現象所帶來的問題，在壅塞避免階段我們試著以更積極快速的方式調整壅塞窗口的大小，使得 TCP 在高頻寬延遲乘積網路中能夠以更靈活準確的方式調整壅塞窗口。

3.2.1 緩啟動階段

在緩啟動階段，Vegas 根據公式(1)計算出囤積在瓶頸點的封包數目(Δ)且每經過一個 RTT 會將壅塞窗口變為兩倍，當Δ大於γ(通常設為 1)，Vegas 會離開緩啟動階段，問題發生在壅塞窗口變為兩倍時，因為當 Vegas 的傳送端收到一個 ACK 後會連續送出兩個封包，這導致

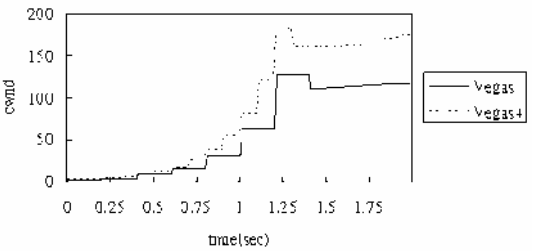

圖 **2**： **Vegas** 和 **Vegas+**在緩啟動階段下壅塞窗口變化的差異。

在短時間內就會有兩倍的壅塞窗口被傳送出去，這種暴衝現象會創造出短時間內囤積在瓶頸點上的封包數目遽增，使得 Vegas 在頻寬未充分使用的情況下離開緩啟動階段。我們利用 ns-2 [22]模擬這個現象，圖 1 為在 50Mb BDP (頻寬為 100Mb/s 延遲時間為 50ms)的環境下所模擬 Vegas 連線在傳送封包時的壅塞窗口變化圖，由圖 1 可以發現 Vegas 在 1.6 秒壅塞窗口為 128 個封包時離開緩啟動階段進入壅塞避免階段，因為過早的離開緩啟動階段導致需經過一段很長的時間後壅塞窗口才在 175.1 秒時達到平衡。

在 Vegas+中我們將壅塞窗口改變的方式由原本每兩個 RTT 增加 1 倍壅塞窗口的方式，改為每個 RTT 增加 0.5 倍，因為在同樣的時間間隔中 Vegas 增加了 1 倍的壅塞窗口而 Vegas+只增加 0.5 倍，有效緩和壅塞窗口在短時間內的暴衝現象，因而使壅塞窗口增加的速度更為流暢，在切換進入壅塞避免階段時的壅塞窗口大小也更加適宜。如圖 2 所示，在相同的γ值設定中，Vegas+能夠在擁有比 Vegas 更高的壅塞窗口值下離開緩啟動階段，代表經過我們的修改以後 Vegas+的確有比 Vegas 有更好的效能。

3.2.2 壅塞避免階段

修改壅塞避免階段目的是改善壅塞窗口調整的速度，讓連線能夠快速的適應不斷劇烈變化的環境。TCP Vegas+一樣會利用預期的吞吐量和實際吞吐量去推算有多少封包被囤積

在瓶頸點上(Δ)。我們將壅塞避免分成兩個部份做說明：

第一部分，當Δ小於α，Vegas+會根據連續成功增加封包的次數去改變壅塞窗口(CWND)值，我們利用下面的公式：

$$CWND = CWND + count, \quad (4)$$

其中 count 是用來記錄連續成功增加封包的次數，即連續經過 count 個 RTT(round-trip time)皆成功增加封包(亦即Δ皆小於α)，此時壅塞窗口就會在一個 RTT 內增加 count 個封包，假設在第 1 個 RTT 下成功增加壅塞窗口，CWND 增加 1，在第 2 個 RTT 下也成功增加壅塞窗口，則 CWND 增加 2，以此類推。取代原本線性增加的方式，如此使得 Vegas+在增加 CWND 的速度上更有效率。

第二部份，當Δ大於β時，Vegas+利用Δ和$(\alpha+\beta)/2$ 的差異去更改 CWND 值，更改的規則如下列公式：

$$CWND = CWND - \left(\frac{\Delta - \left(\frac{\alpha+\beta}{2}\right)}{2}\right) \quad (5)$$

我們利用Δ去推算出囤積在瓶頸點的封包數超過理想值($(\alpha+\beta)/2$)多少，藉此取得適當的壅塞窗口減少量；此外當我們在第 i 個 RTT 調整壅塞窗口後，必須等到第(i+2)個 RTT 才能偵測出改變的效果，因此為了避免減少的太多，所以一次只減少一半應減的量。

為了要成功提升連線間的公平性，我們希望每條連線囤積在瓶頸點上的封包數目都相同(等於$(\alpha+\beta)/2$)，所以當Δ算出來介於α和β之間，Vegas+採取線性的方式調整壅塞窗口讓壅塞窗口逐漸靠近($(\alpha+\beta)/2$)。在壅塞避免階段壅塞窗口調整的的演算可整理成如下的虛擬碼(pseudo code)：

```
if (Δ > β)
    CWND = CWND – ( (Δ - (α+β)/2) / 2 )
    incr = 0; count = 0;
else if(Δ < α)
    count = count + 1
    incr = 1/CWND × count
else if (Δ > (α+β)/2)
    CWND = CWND-1;incr = 0;count = 0
else if (Δ < (α+β)/2)
    incr = 1/CWND; count=0
else /* Δ == (α+β)/2 */
    incr = 0; count = 0;
```

其中 *incr* 代表每次收到一個 ACK 所調整壅塞窗口的量，以避免一次增加多個封包所帶來的負面影響。

4. 效能評估

在這部分我們將分成兩個方面來進行效能的評估：第一部分是 ns-2 [22]模擬，第二部份是藉由修改 Linux kernel 在實際網路上的實驗。

4.1 ns-2

我們運用 ns-2 網路模擬軟體來測試 Vegas、Fast、以及 Vegas+的效能。首先在 Vegas 和 Vegas+中我們使用相同的參數設定(γ=1, α=2, β=4)；Fast 則使用與[21]相同的預設參數值α=100，每個封包的大小為 1Kbytes。我們分成三方面去測試：(1)比較 Vegas 和 Vegas+在緩啟動階段壅塞窗口增加的速度以及比較何者可在較適當的 CWND 值進入壅塞避免階段。(2)比較 Vegas、Fast、和 Vegas+從連線啟動後直到完全使用可用頻寬所需的時間、在壅塞避免階段連線適應網路變化所需的時間以及觀察瓶頸點佇列的使用情形。(3)比較 Vegas、Fast、和 Vegas+在相同 RTT 和不同 RTT 時連線間的公平性。

圖 **3**：測試緩啟動機制所用的網路環境。

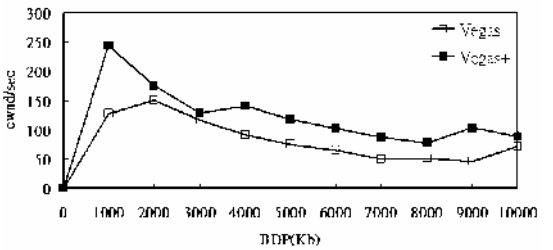

圖 **4**：**Vegas** 和 **Vegas+**在不同頻寬延遲乘積網路中每秒增加 **CWND** 值的速度比較。

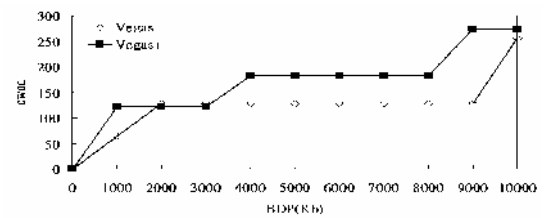

圖 **5**：**Vegas** 和 **Vegas+**在不同頻寬延遲乘積網路的情況下進入壅塞避免階段的 **CWND** 值比較。

表一：**Vegas** 和 **Vegas+**在緩啟動階段前六個 **RTT** 壅塞窗口的變化情形

RTT	0	1	2	3	4	5	6
Vegas	2	2	4	4	8	8	16
Vegas+	2	3	4.5	6.75	10.13	15.9	22.78

4.1.1 Vegas 和 Vegas+在緩啟動階段的比較。

在這部分我們運用如圖 3 的網路環境測量傳送端 S0 傳送資料至接收端 D0 時壅塞窗口變化的情形。圖 4 代表 Vegas 和 Vegas+在不同 BDP 下壅塞窗口增加的速度，圖中可以發現 Vegas+在不同 BDP 的環境下 CWND 增加的速度都比 Vegas 來的快，我們利用表一表示 Vegas 和 Vegas+的壅塞窗口變化的情形，Vegas 使用每 2 個 RTT 增加 1 倍壅塞窗口，Vegas+採用每個 RTT 增加 0.5 倍的壅塞窗口，由表一得知 Vegas+在第 2 個 RTT 後壅塞窗口就比 Vegas 大。圖 5 中 Vegas+離開緩啟動的 CWND 值也都比 Vegas 來的高，因為在同樣的時間間隔中

圖 **6**：測試收斂時間時所用的網路環境。

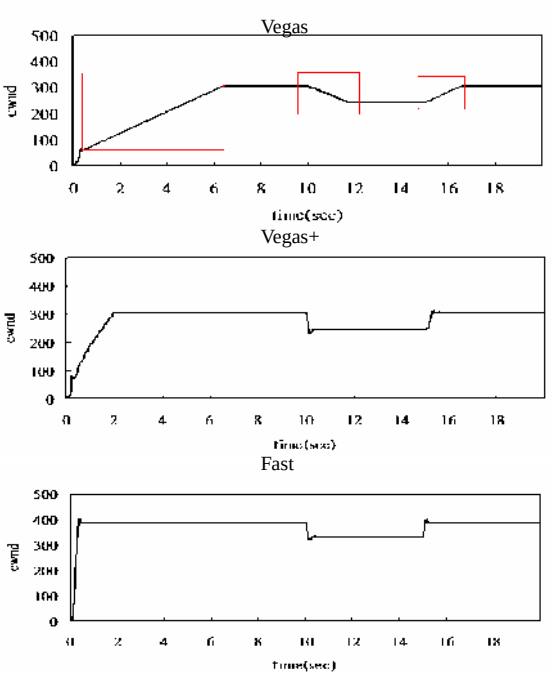

圖 **7**：**Vegas**、**Vegas+**和 **Fast** 的在圖 **6** 的測試環境下壅塞窗口的變化情形。

Vegas 增加了 1 倍的壅塞窗口而 Vegas+只增加 0.5 倍，有效減緩壅塞窗口在短時間內的倍增所帶來的暴衝效應。除了在 BDP 為 2000 及 3000 時，這是因為 Vegas+每個 RTT 會增加 0.5 倍的 CWND，在此環境下恰巧跨越了進入壅塞避免階段的門檻。

4.1.2 收斂時間的比較

接下來我們運用圖 6 的網路環境進行收斂時間的測試，所謂的收斂時間指的是 TCP 連線從一穩定狀態到達另一穩定狀態所需的時間。首先由 0 秒啟動 S0~D0 連線直到壅塞窗口充份使用可用頻寬後，再啟動 S1~D1 連線，S1~D1 連線是擁有一定速率的 UDP 連線，藉此將頻寬減少，以測得各 TCP 版本連線在頻寬減少後壅塞窗口的收斂時間；接著結束 S1~D1

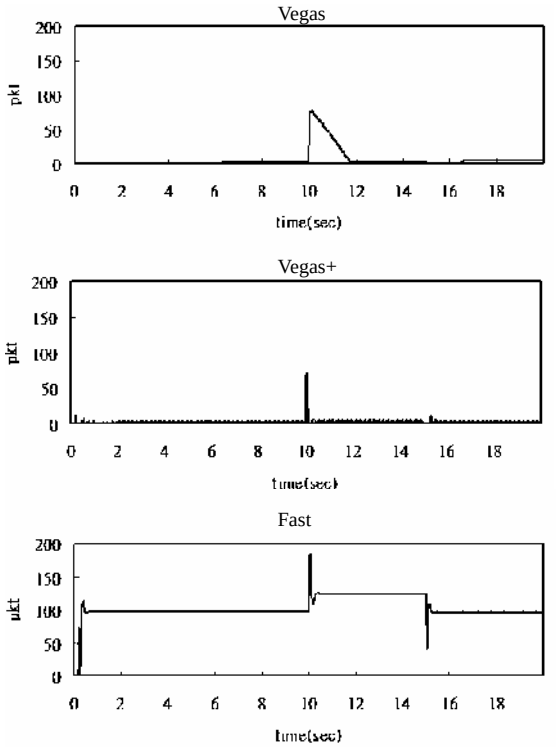

圖 **8**：**Vegas**、**Fast** 和 **Vegas+**在圖 **6** 網路環境下，瓶頸點佇列的使用情形。

連線，使頻寬恢復，再次去測量因頻寬增加所需的收斂時間。

觀察圖 7 我們可分成三個階段討論，第一階段，一條新的連線由緩啟動開始直到平衡所花的時間，在此階段 Vegas+和 Fast 花的時間都比 Vegas 來的少，這是因為 Vegas 過早的離開緩啟動階段，而進入線性成長的壅塞避免階段，所以花費許多時間才達到飽和，而 Vegas+離開緩啟動時的壅塞窗口比 Vegas 來的大，且進入壅塞避免階段後根據連續成功增加的次數調整其壅塞窗口，使 Vegas+能較快的使用可用頻寬；Fast 則是以公式(1)快速的調整壅塞窗口，讓連線能在最短時間內使用到可用頻寬；當 UDP 資料流加入網路可用頻寬減少，我們可以發現 Vegas+和 Fast 都可迅速降低其壅塞窗口，但是觀察圖 8，Fast 囤積在瓶頸點的封包數目都至少在 100 以上，而且當 UDP 資料流加入以後 Fast 無法有效的減少封包囤積的數量，反而使封包囤積的數量變的更多，對於瓶

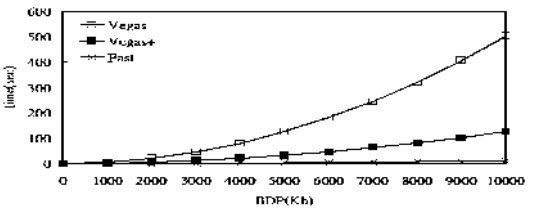

圖 **9**：**Vegas**、**Vegas+**和 **Fast** 在不同頻寬延遲乘積網路下，由緩啟動開始直到平衡所需要的時間。

頸點佇列的使用度非常高；當 UDP 資料流離開網路可用頻寬增加，同樣的 Vegas+和 Fast 都可迅速增大其壅塞窗口迅速的達到飽和。

由圖 8 可得知，Vegas 在 0~10 秒時只有極少量的封包囤積在瓶頸點，10 秒後因為 UDP 資料流的加入導致囤積在瓶頸點上的封包總數提高，由於 Vegas 調整壅塞窗口的速度較慢所以直到 12 秒以後囤積封包的數量才能減少到正常的位置；反觀 Vegas+因為它擁有比 Vegas 更能適應網路環境的改變，所以它只花 0.14 秒的時間即可將囤積封包的數量恢復正常，另外 Vegas+最大的囤積封包數量(73 個封包)也較 Vegas 的最大囤積數量(77 個封包)為少，較少的最大封包囤積數量意味著封包遺失的機率也隨之降低。

接下來藉由更改 R0 和 R1 之間鏈結的延遲以產生不同的 RTT，來測試並觀察在不同 BDP 網路中各 TCP 版本的收斂行為。

圖 9 顯示一條新的連線由啟動到達飽和所需的時間，我們可以發現 Vegas+和 Fast 所需時間皆較 Vegas 為短，但由於緩啟動階段的暴衝現象未完全解決，使得 Vegas+離開緩啟動階段時的壅塞窗口值仍然偏低，而且在壅塞避免階段也無法大量累積連續成功增加壅塞窗口的次數，所以在此狀況下 Vegas+的改善幅度沒有 Fast 來的顯著，但需要一提的是在此實驗中我們將佇列的長度設定為一個很大的值，因此不需考慮封包遺失的可能，在實際的網路中 Fast

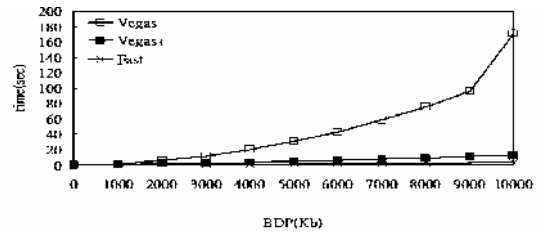

圖 **10**：**Vegas**、**Vegas+**和 **Fast** 在不同頻寬延遲乘積網路下，當頻寬減少後達到平衡所需要的時間。

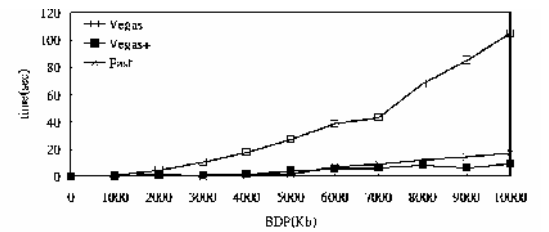

圖 **11**：**Vegas**、**Vegas+**和 **Fast** 在不同頻寬延遲乘積網路下，當頻寬增加後達到平衡所需要的時間。

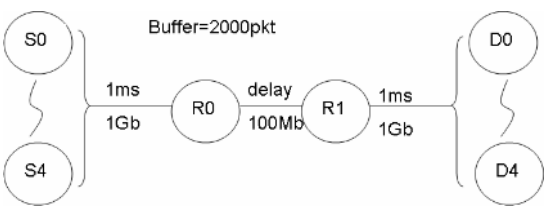

圖 **12**:連線在具有相同 **RTT** 時的網路環境。

可能會因為對於佇列的過度使用，造成封包的大量遺失而導致效能的降低，關於這點我們在後面的實驗中有進一步的映證。

圖 10 和圖 11 顯示出當可用頻寬減少和增加時，Vegas、Vegas+和 Fast 三個 TCP 版本收斂的時間，可以看出 Vegas+和 Fast 都能比 Vegas 更快適應網路環境的改變。

4.1.3 在相同 RTT 的環境下公平性的比較

接下來比較 Vegas、Fast 和 Vegas+在相同 RTT 環境下連線間公平性的問題，首先利用圖 12 的網路環境，更改 R0~R1 間的延遲時間以造成不同的頻寬延遲乘積網路環境；由 S0~S4 利用相同的 TCP 協定傳輸資料給 D0~D4，去記錄每條連線所佔的頻寬，用這種方式來比較

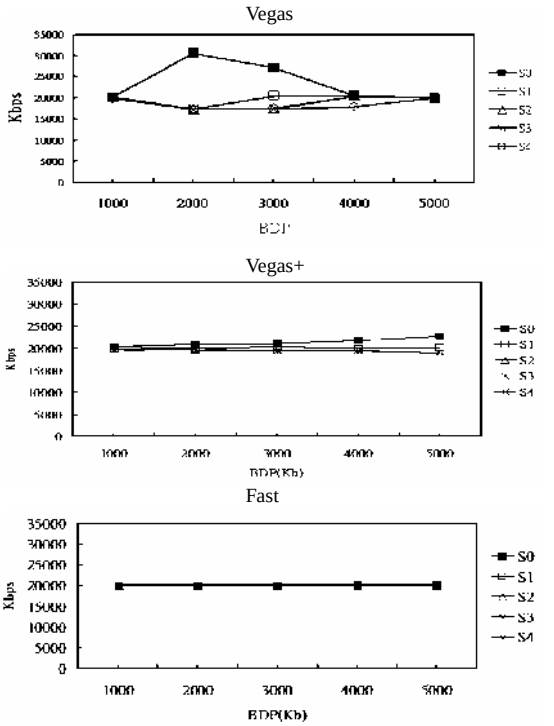

圖 **13**：**Vegas**、**Vegas+**和 **Fast** 在具有相同 **RTT** 環境下的公平性比較。

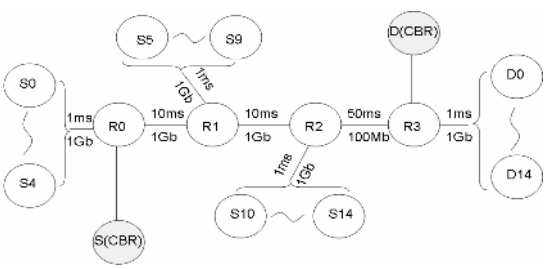

圖 **14**:連線在具有不同 **RTT** 時的網路環境。

他們之間的公平性。

圖 13 分別代表 Vegas、Fast、Vegas 的公平，我們觀察每條連線的緊密程度，愈緊密代表愈公平。Vegas+和 Fast 的公平性都比 Vegas 好。Vegas 連線因為每條連線囤積在瓶頸點的封包數都不一樣導致效能有所不同，而 Vegas+每條連線囤積的封包數目皆等於 $(\alpha+\beta)/2$ 所以能夠有較公平的表現；Fast 也相同他試圖將每條連線囤積在瓶頸點上的封包數都維持在 100，所以它也能比 Vegas 有更公平的表現。

4.1.4 在相異 RTT 的環境下公平性的比較

接著我們在圖 14 中將 TCP 連線分為三群

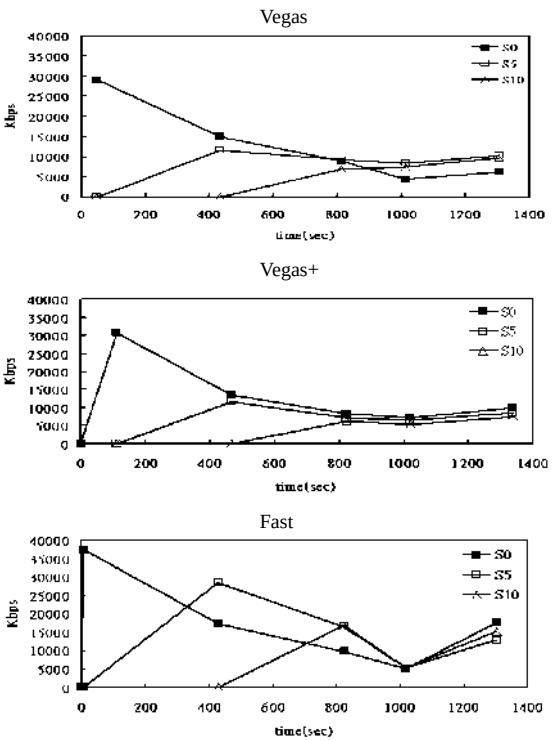

圖 **15**：：**Vegas**、**Vegas+**和 **Fast** 在具有不同 **RTT** 環境下的公平性比較。

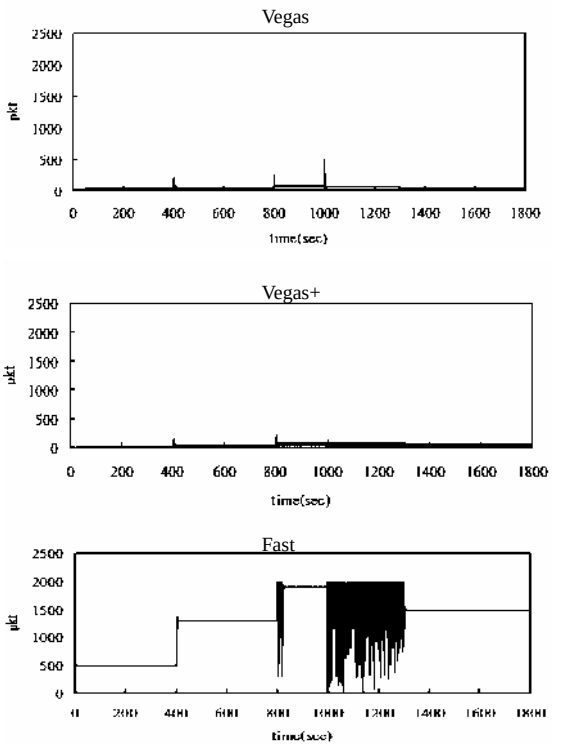

圖 **16**：**Vegas**、**Vegas+**和 **FAST** 在不同 **RTT** 的環境瓶頸點佇列的使用情形。

分批啟動，群與群之間具有不同的 RTT，0 秒先啟動 S0~S4，400 時啟動 S5~S9，800 秒時起動 S10~S14，在 1000 秒時啟動一條 UDP 資料流佔去 25Mb 的可用頻寬，1200 秒後關掉 UDP，做此實驗的目的在於測試連線間具有不同的 RTT 是否影響公平性，此外頻寬減少與增加是否也影響既有連線間的公平性。

根據圖 15 顯示在不同 RTT 的環境下 Fast 公平性較差，Vegas+的公平較佳，值得注意的是在此實驗中瓶頸點的佇列長度為 2000 個封包，由圖 16 觀察可得知，在 1000 秒到 1200 秒之間由於 UDP 資料流的加入，網路可用頻寬減少，瓶頸點的佇列長度已無法滿足 Fast 連線所需，因此造成大量的封包遺失，此時 Fast 的公平性雖佳但其平均效能卻最低，反觀 Vegas 和 Vegas+平均囤積在瓶頸點的封包個數很低，因此沒有發生封包遺失現象。

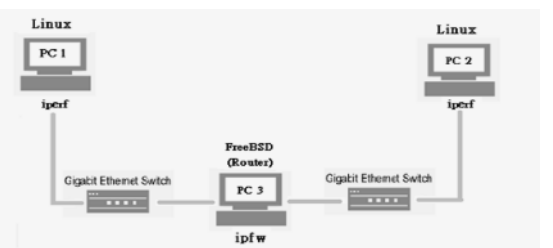

圖 **17**：實際網路傳輸實驗環境。

4.2 Linux

在修改 Linux kernel(版本:2.6.18.1)以進行在實際網路上的實驗方面，我們使用三台電腦來進行實際的網路傳輸，如圖 17 所表示，透過此實驗我們要觀察兩件事：(1)比較 Vegas 和 Vegas+的緩啟動和壅塞避免兩階段壅塞窗口增加的速度。(2)比較 Fast、Vegas 和 Vegas+在瓶頸點上佇列大小不同時，網路傳輸的快慢。經由實際的網路實驗驗證 Vegas+的優點。

(1) 緩啟動和壅塞避免階段的比較

首先我們利用 FreeBSD 模擬路由器的行

表二：**Vegas** 在四種不同的環境下所取得的

相關資訊，佇列長度為 **100Kbytes**。

Bandwidth(Mbits/s)	Router delay(ms)	進入 Congestion Avoidance 時間點(sec)	進入 Congestion Avoidance 時 cwnd 值(pkts)	Slow Start 時爬升速度 (pkts/sec)	達到平衡時間點(sec)	達到平衡時 cwnd 值(pkts)	Congestion Avoidance 時爬升速度 (pkts/sec)
100	25	1.31	116	88.54	13.79	190	7.09
100	50	2.75	166	60.36	23.00	205	2.95
200	25	1.41	121	85.81	22.02	200	4.56
200	50	2.96	126	42.56	52.20	220	2.22

表三：**Vegas+**在四種不同的環境下所取得的

相關資訊，佇列長度為 **100Kbytes**。

Bandwidth(Mbits/s)	Router delay(ms)	進入 Congestion Avoidance 時間點(sec)	進入 Congestion Avoidance 時 cwnd 值(pkts)	Slow Start 時爬升速度 (pkts/sec)	達到平衡時間點(sec)	達到平衡時 cwnd 值(pkts)	Congestion Avoidance 時爬升速度 (pkts/sec)
100	25	2.04	128	62.74	29.49	199	3.16
100	50	3.98	128	32.16	69.64	220	1.64
200	25	2.15	128	59.53	51.90	199	1.74
200	50	4.18	128	30.62	74.84	215	1.45

表四：傳輸固定大小檔案所需的時間**(sec)** **(**路由器設定：延遲時間 **50ms**，頻寬大小 **100Mb**，佇列長度 **100Kbytes)** 。

	1Mbytes	5Mbytes	10Mbytes	50Mbytes	100Mbytes	500Mbytes
Vegas	3.5	8.4	14.1	45.4	86.1	404.4
Vegas+	2.5	6.1	10.7	44.8	84.4	404.0
Fast	3.7	11.5	18.3	103.2	207.5	902.7

表五：傳輸固定大小檔案所需的時間**(sec)** **(**路由器設定：延遲時間 **50ms**，頻寬大小 **100Mb**，佇列長度 **200Kbytes)** 。

	1Mbytes	5Mbytes	10Mbytes	50Mbytes	100Mbytes	500Mbytes
Vegas	3.3	8.6	14.3	46.4	86.1	404.3
Vegas+	2.5	5.7	9.7	45.4	84.5	402.0
Fast	2.9	6.5	10.7	40.9	80.5	400.7

為藉由設定不同的頻寬與延遲時間產生四種不同的環境，分析 Vegas 和 Vegas+的 CWND 變化圖，從中取得相關比較的資訊，製作成表格，藉此來比較兩版本緩啟動和壅塞避免階段的優劣。

表二和表三的比較，我們發現 Vegas+在進入壅塞避免階段時的 CWND 值並不是每次都比 Vegas 高，主要是因為實作的環境並不像 ns-2 模擬實驗環境一樣的單純，有一些因素(如：實作環境牽扯到比 TCP 更上層的傳輸或者是硬體配備…等因素)導致每次實驗的結果並不一定如預期，但如果比較單位時間內的爬升速度以及到達平衡的時間，仍然可以明顯地看出 Vegas+效能上的改進。在緩啟動階段爬升速度比較，四種環境下(Vegas：62.74、32.16、59.53、30.62　Vegas+：88.54、60.36、85.81、42.56)Vegas+分別提升了 41%、88%、44%、39%的效能。在到達平衡的時間方面，四種環境下(Vegas：29.49、69.64、51.90、74.84；Vegas+：13.79、23.00、22.02、52.20)Vegas+分別比 Vegas 快了 2.1 倍、3.0 倍、2.4 倍、1.4 倍。在壅塞避免階段爬升速度，四種環境下(Vegas：3.16、1.64、1.74、1.45　Vegas+：7.09、2.95、4.56、2.22)Vegas+分別提升了 124%、80%、162%、53%的效能。由此可得知，無論在緩啟動或壅塞避免的階段，Vegas+都大幅度的提升了原本 Vegas 的效能。

(2) 在不同佇列大小的情況，網路傳輸快慢的比較

在這個部份，主要在兩種不同的環境中，傳輸固定大小檔案來比較三個版本傳輸速度的快慢，另外也藉此驗證 Fast 對瓶頸點佇列的高依賴性。

由表五得知，在瓶頸點佇列足夠的情況下 Fast 和 Vegas+相較於 Vegas 都有較傑出的效能表現，因此所需的傳輸時間較短，反觀表四，Fast 在佇列不夠的情況下，無論跟 Vegas 或

Vegas+比，傳輸效能都明顯地較差，而 Vegas+則有最佳的效能表現。值得注意的是由於 Vegas 和 Vegas+在達到平衡後所傳輸的速度是一樣的，此時網路環境沒有任何變化，因此在傳輸較大檔案時所需時間上並不會有太大的差異。

5. 結論與未來展望

在這篇論文中，我們提出一個以 TCP Vegas 為基礎的改良版本名為 Vegas+，它改進了原本 Vegas 的緩啟動與壅塞避免機制，使得 Vegas+在高頻寬延遲乘績網路中有較優越的表現。相較於 Fast TCP，Vegas+囤積在瓶頸點上的封包數較少，對於瓶頸點的佇列大小較不依賴，因此降低當連線數增加時封包遺失的風險。

這篇論文未來的努力目標有兩個方向，第一，我們希望提供一個完整的分析模型以數學證明 Vegas+的有效性；第二，根據實驗的結果顯示在緩啟動機制中暴衝問題仍然存在，因此 Vegas+的緩啟動機制還有相當的改善空間。

參考文獻

[1] D. Katabi, M. Handley, and C. Rohrs, “Congestion control for high bandwidth-delay product networks,” in Proc. ACM SIGCOMM'02, vol. 31, Issue 4, Aug. 2002.

[2] B. Feng; D. Ghosal,; N. Kannappan ,”Impact of ATM ABR control on the performance of TCP-Tahoe and TCP-Reno” ,Global Telecommunications Conference, 1997. GLOBECOM '97., IEEE.

[3] 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.

[4] J. Mo, R. J. La, V. Anantharam, and J. Walrand, “Analysis and comparison of TCP Reno and Vegas,” INFOCOM '99. Vol. 3, pp.1556 – 1563, March 1999.

[5] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, “TCP Vegas: New techniques for congestion detection and avoidance,” in Proc. ACM SIGCOMM'94, no. 4, pp. 24-35, Aug. 1994.

[6] L. S. Brakmo and L. L. Peterson, “TCP Vegas: End to end congestion avoidance on a global Internet,” IEEE J. Select. Areas Commun., vol. 13, pp. 1465-1480, Oct. 1995.

[7] J. S. Ahn, P. B. Danzig, Z. Liu, and L. Yan, “Evaluation of TCP Vegas: Emulation and experiment,” in Proc. ACM SIGCOMM'95, vol. 25, pp. 185-195, Aug. 1995.

[8] G. Hasegawa, M. Murata, and H. Miyahara, “Fairness and stability of congestion control mechanism of TCP,” Telecommunication Systems Journal, pp. 167-184, Nov. 2000.

[9] S. Vanichpun and W. Feng, “On the transient behavior of TCP Vegas,” in Proc. IEEE ICCCN'02, pp. 504-508, Oct. 2002.

[10] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson, “On the self-similar Nature of Ethernet Traffic,” IEEE/ACM Trans. Networking, vol. 2, no. 1, Feb. 1994.

[11] Y. C. Chen, C. T. Chan, C. Y. Ho, and C. L. Lin, “Improving Performance of TCP Vegas for High Bandwidth-Delay Product Networks,” in Proc. ICACT’06, February 2006, pp. 459 - 464.

[12] L. Xu, K. Harfoush., I. Rhee, “Binary increase congestion control (BIC) for FAST long-distance networks”, INFOCOM 2004. Vol. 4, pp. 2514 - 2524, March 2004.

[13] I. Rhee, L.Xu, “CUBIC: A New TCP-Friendly High-Speed TCP Variant”, Prof. of PFLDnet 2005 ,Lyon, France, Feb 2005.

[14] T. Kelly, ”Scalable TCP: Improving Performance in Highspeed Wide Area Networks”, ACM SIGCOMM Computer Communications, April 2003.

[15] S. Bhandarkar, S. Jain. and A. L. Narasimha.” LTCP: Improving the Performance of TCP in Highspeed Networks” ACM SIGCOMM Computer Communication. Vol. 36, Issue 1, January 2006.

[16] D. J. Leith., R. N. Shorten., Y. Li, ”H-TCP: A framework for congestion control in high-speed and long-distance networks” HI Technical Report, http://www.hamilton.ie/net/htcp/ ,August 2005.

[17] S. Floyd “HighSpeed TCP for Large Congestion Windows”. RFC 3649.

December 2003
[18] R. Wang., K.Yamada, M. Y. Sanadidi, M. Gerla, M.,"TCP with sender-side intelligence to handle dynamic, large, leaky pipes" Selected Areas in Communications, IEEE Journal on Selected Areas in Communication, Vo.23, pp. 235 – 248 Issue 2, Feb 2005.
[19] R. King, R. Riedi, R. Baraniuk,"TCP-Africa: An Adaptive and Fair Rapid Increase Rule for Scalable TCP ," in Proc. INFOCOM 2005.
[20] A. Maor and Y. Mansour, "AdaVages: Adaptive control for TCP Vegas," in Proc. IEEE GLOBECOM'03, vol. 7, p.3647-3651, Dec. 2003.
[21] 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.
[22] UCB/LBNL/VINT Network Simulator – ns(version 2), http://www.isi.edu/nsnam/ns/.