國立清華大學資訊工程系
碩士論文

動態平行存取映射站台的 Web 資源之技術研究

Dynamic Parallel Access to Mirrored Web esources

所　　別： 資訊工程系

研 究 生： 陳寬達（Kuan-Ta Chen）

指導教授： 黃能富（Nen-Fu Huang）

中華民國八十九年六月

摘要

為了減少從網路下載檔案的等待時間，許許多多的檔案被置放於映射站台，讓使用者自由選擇站台來進行瀏覽、下載，不但節省下載時間，也分散伺服器的負載。不過，選擇傳輸速度最佳的站台並不是件簡單的事，若選擇錯誤，很可能得到的極低的傳輸效率。平行存取技術由此而生：同時與多個映射站台連線，分別從各站台下載檔案的不同部分，再組合為原來的檔案，不但可大幅加快下載速度，還可分散網路及伺服器的負載，以及避免困難的站台選擇工作。

平行存取技術的關鍵為，將檔案切分為許多區塊，分別從不同站台下載各個區塊的資料。不同的區塊尺寸可決定整體的下載效率，大致來說，區塊尺寸越大，下載效能越高。不過當區塊尺寸過大，由於各站台區塊傳輸結束時間的不一致，反而可能使下載時間增長。區塊尺寸過小或過大都可能降低效能，要如何選擇最佳的區塊尺寸是使平行存取技術達到高效能的重要研究。

在本篇論文中，我們研究檔案大小、區塊尺寸對於平行存取技術傳輸效能的影響，並建立區塊尺寸－下載時間模型，找出計算最佳的區塊尺寸的方法，並藉此發展出動態平行存取技術。

動態平行存取技術可動態地根據根據各站台的連線及下載情形以時間預估演算法來計算最佳的區塊尺寸，企圖在各種情況下皆能得到合理的下載效率。根據我們的評估，動態平行存取技術的表現雖然不若直接指定最佳區塊尺寸的平行存取技術，不過在各種情況下能夠獲得較穩定的表現。在我們的實驗中，動態平行存取技術通常能獲得理想下載速率 85% 以上的傳輸效能。

目錄

圖表目錄

第一章

簡介

網際網路不斷以驚人的速度增長著，在通訊速率、線材價格、連線費用及頻寬需求上皆是如此。1997 年後網際網路上的資料流量以每六個月倍增的趨勢持續上攀 [Lawr 00]，遠遠地將當年半導體產業那令人咋舌的摩爾定律拋在後頭。

頻寬需求不斷地升高，但世界各地的網路交換設備、通訊線纜等等硬體設施並沒有辦法以同樣的速度增建或升級，於是頻寬不足成了全球網際網路參與者亟欲解決的共同課題。

除了新興的即時視訊、音訊或其它資訊的推播應用外，從誕生至今，Web 一直是網路頻寬的最大食客，無人能與爭鋒。雖然有了各種代理伺服器（proxy server）及快取（cache）、快取共享（cache sharing）技術的研發及落實，但不論是何種代理伺服器或快取技術，只要快取空間不足，或者用戶請求的資料未存在於快取區域時，一樣要面對緩慢呆滯的網際網路回應及傳輸速率。又由於 Web 的成長更為驚人，早已不可能存在所謂的「足夠大的快取空間」來滿足廣大的 Web 空間及各式各樣的使用者需求。

西元 2000 年六月中旬，估計 WWW 上已有約 20.7 億頁的網頁及 4.6 億張影像檔，包含了 38.8 Tera Bytes（即 38800 GB）的文字及 7.7 Tera Bytes 的影像。而光是過去的二十四小時內，網際網路上就增加了四百二十萬頁網頁及九十四萬張影像檔。更驚人的是，整個 Web 的大小每年約增長一倍 [Cens]。

這樣的暴增速度，遑論世界各個區域的代理伺服器，就連許多專業的搜尋引擎站台也都負荷不了 WWW 每日的成長量，跟不上網頁及資訊的更新 [Cens]。

頻寬不足是主因，Web 的資訊量使得各種快取技術功能帶來的優勢越不明顯，Web 文件／資源的下載速度日趨緩慢。一個十分直覺、有效的解決方案是，在全球各地廣佈同樣的檔案或文件，使用者可以選擇最靠近自己的檔案位置來下載。雖然許許多多的檔案搜尋站台或 archie 服務都能夠洋洋灑灑地列出數十個包含同一個檔案的映射站台供人點選，但它們只能做到這樣，沒有人能夠告訴你，要從哪一個站台下載檔案才是最迅速、最節省時間的選擇。

即使深諳網際網路協定及技術，就算知道所有站台的主機名稱及網路位址，正確地選擇下載速度最快的站台也絕不是件容易的事。而且網路連線狀況隨時變動，可能上一秒鐘十分順暢，這一秒鐘就開始塞車；可能這一秒鐘還十分平穩，下一秒鐘就開始大量地遺落封包，只能採用預測法來評估接下來短時間內表現「可能」最好的站台，沒有絕對的標準答案。

有不少研究著重在上面提到的問題：如何在眾多映射站台之間做出明智的選擇 [Zong 98] [Crov 97] [Mehm 98]。目前為止，此問題尚未得到良好的解答，最惱人的是，一旦判斷錯誤，誤選下載速率較慢的站台，對檔案傳輸花費時間的影響十分嚴重。因此有另外一種形式的作法出現：既然難以選擇最佳的下載站台，那麼就同時與多個站台進行連線，分別由各個站台取得同一個檔案的不同部分，下載到客戶端後，再將它們組合回來檔案的原始面貌 [Pabl 00]。

同時從多個映射站台協力下載一個檔案的手法，稱為平行存取技術（Parallel Access Technique）。在理想情況下，客戶端（即下載檔案者）所能得到資料下載速率為各個映射站台資料傳輸速率的總和。

平行存取技術的關鍵處在於，必須將檔案分為許多區塊，同時向不同的映射站台下載不同的區塊，最後再組合起來。面對各種長度的檔案、不同數目的映射站台、各別映射站台的不同連線速率、以及詭譎多變的網路狀況，區塊尺寸的決定，以及如何將它們分配給多個連線速度及路徑不同的映射站台是影響平行存取技術效率的關鍵。區塊尺寸越小，下載請求次數增多，會影響效率；區塊尺寸越大，

越難以分派區塊，達成各個站台全力運作，同時結束的理想目標。

在此篇論文裡，第二章首先介紹平行存取相關的技術及研究；第三章研究平行存取技術在現實網路架構落實的可行性。

第四章介紹平行存取技術的運作流程，並評估下載效能及改善空間。第五章討論檔案大小與下載效率的關係，並說明為什麼平行存取技術能夠改善檔案大小對於下載效率的影響。

接著，在第六章中，我們建立平行存取技術的區塊尺寸－下載時間模型，分析檔案大小、區塊尺寸、各站台可用頻寬、封包來回傳遞時間等等因素與平行存取下載檔案傳輸效率的關係。憑藉區塊尺寸－下載時間模型的正確性，我們發展動態平行存取技術，在平行存取技術下載檔案的過程中，動態地依據下載情形計算最佳的區塊尺寸，讓區塊尺寸由演算法自行決定，而且能夠在各種情況下皆能達到與理想下載速率十分接近的平均傳輸速率。

最後，我們在第七章做總結，並簡述未來的發展方向。

第二章

相關技術及研究

進入平行存取技術的討論前，我們首先探討各種與平行存取技術相關的議題。包括如何選擇最佳的映射站台、平行存取多個站台的各種技術以、取得映射站台資訊的方法以及以群播技術提供檔案等等討論。

2.1 選擇最佳的映射站台

選擇最佳映射站台已是行之有年的研究，包括各種形式的技術，如：

- □使用群播將伺服器資訊告知欲下載檔案的主機 [Bern 95]
- □客戶端動態偵測與各站台連線路徑的可用頻寬 [Crov 97]
- □結合伺服器廣播資訊以及客戶端動態偵測可用頻寬 [Zong 98]
- □使用先前的映射站台速率統計資料 [Sesh 97]

由於整個網際網路暢通、阻塞的情況隨時都可能發生劇烈的變動，只要人類沒有辦法掌握時間的流動，就沒有辦法在下載之前取得絕對正確的最佳映射站台解答。所以最佳的解決方式就是，即使某個站台很明顯地看起來最有贏面，仍然不將所有籌碼賭在那個站台上，而是分散下注，同時與多個映射站台進行連線，以平行存取的方式下載檔案。

2.2 平行存取多個站台

Byers 等人 [Byer 99] 提出可以配合 erasure codes 同時從多個站台下載資料的概念 [Luig 97] [Luby 97]。使用 erasure codes，必須事先將原始檔案切割為 k 個資料區塊，以演算法產生 h 個同位資訊區塊，將資料區塊與同位資訊區塊置於每個

映射站台上頭。此後，需要此檔案的客戶端只要取得此 $(k + h)$ 個區塊中的任意 k 個區塊，不論這 k 個區塊是資料區塊或是同位資訊區塊，或由哪個站台取得，都足以供客戶端程式進行處理，重建回原始檔案。

若將 erasure codes 用於平行下載檔案用途，編碼／解碼速度都不能太慢，因此要設計特別的 erasure codes 演算法，例如 Tornado Codes [Luby 97] 等等。使用 Tornado Codes，不必煩惱區塊的分派順序及重覆問題，只要任意地從眾多映射站台取得足夠數目的資料或同位資訊區塊，就可以進行解碼、組合動作，在客戶端重現原始檔案。

不過，使用 erasure codes 技術的重大缺陷是，所有的映射站台必須備有編碼程式，事先將所有檔案編碼存放（需要額外的儲存空間），或是在輸出檔案時即時編碼（需要足夠的運算能力）；另外客戶端軟體也必須修改，加入解碼程式，才能將被編碼的區塊資料還原回原始檔案。

Pablo 等人 [Pabl 00] 提出能夠在現行網際網路協定／架構下，不需對現有的伺服器軟體進行任何修改即能派上用場的平行存取技術。

此技術完全使用 HTTP 協定所規範的功能，將目的檔案切分為多個大小相同的區塊，動態地交由不同映射站台下載。進行下載時，與每個站台建立 HTTP 連線，分別向每個站台要求一個區塊。當某站台完成請求，順利傳回客戶端要求的區塊時，就再向該站台要求一個未下載的區塊，如此反覆進行，直到整個檔案的所有區塊下載完成為止。

2.3 取得映射站台資訊

雖然有許許多多的檔案已被全世界各地的管理者或程式手動或自動地散佈放置於不同的站台上，不過當使用者欲下載某特定檔案時，目前仍然沒有完善的技術可讓我們快速、準確地得知，將要進行下載的檔案，同時也存在於網際網路上的其它哪些站台。

傳統的 archie 服務服務範圍只限於 FTP 通訊協定的存取空間，而且 archie 伺服器的映射站台列表是由伺服器管理者手動指定，再由程式周期性地至管理者指定的映射站台蒐集資訊，而非動態地隨外界的變異自動更新，因此經常不能獲得足夠且正確的映射站台資訊。

有些搜尋引擎除了提供資訊搜尋，同時也提供檔案搜尋的功能，可依檔案名稱尋找所有擁有指定檔案的映射站台及網址列表 [Lyco]；有些還可按照封包遺失率及封包往返傳遞時間來進行站台排序列表，供使用者選擇網路狀況最佳的站台 [CNet]。

有些組織或公司修改它們的 DNS 伺服器，當客戶端查詢所屬站台的 IP 位址時，回傳地理位置距離使用者最近的映射站台 IP 位址（根據封包的來源位址）。也就是說，對於同一個主機名稱，不同所在地的查詢者可能得到不同的回答。有些研究則將 Anycast 的觀念引入，讓客戶端可由地域距離、回應時間、網路連線距離等等不同規則，很方便地取得最合適的映射網站位址 [Bhat 97]。

另外，有些研究提出建議，將熱門檔案的映射站台資訊納入 DNS 的服務範圍 [Kang 99]，或是建立集中式的資料庫來維護映射站台列表 [Gadd 97]。

現行的一些快取共享協定中 [Rous 98] [Fan 98]，本地端程式會主動維護檔案是否在鄰近的快取區，當欲存取的檔案不在本地端快取區時，便會自動向最近的鄰近快取區取得。

2.4 以群播技術提供檔案

根據 [Amma 97] 的實驗，通常 95% 的 HTTP 請求要求的是前 5% 的熱門檔案；而根據 [Calf 94] 的統計資料，NCSA Mosaic 伺服器所收到的 59% HTTP 請求集中於下載前 25 名熱門檔案（約佔檔案數目的 0.27%）。

由於 Web 伺服器的存取動作具有目標集中的特性，對於每分鐘必須處理上百至

上千個 HTTP 請求的忙碌伺服器來說，熱門檔案可能在短短數秒鐘內，就必須分別傳送多次給不同的客戶端主機。這對於伺服器本身及伺服器對外的網路頻寬來說，都是很沈重的負擔。

因此另一種很奇特的下載方式為，當支援群播功能的客戶端發出檔案下載請求前，先加入某個群播位址（使用網址－群播位址轉換公式而得），若伺服器認為此檔案為熱門檔案，且幾乎同時也有其它客戶端要求下載同一個檔案，就建立可信賴的群播連線，將檔案以群播方式傳遞給要求檔案的各個客戶端。

由於群播位址的加入、可信賴群播連線的建立都帶來極大的 overhead，因此此方法只適用於檔案長度夠大，且短時間內要求同一個檔案的請求夠多，才能帶來正面效益，降低伺服器的負載及回應時間，以及減少頻寬的浪費。

第三章

平行存取技術的可行性

平行存取技術的優勢之一，就是可以在不修改現行網際網路架構及通訊協定前提下，在 HTTP [Frys 96] [Fiel 97] 及 FTP [Post 85] 協定上運作。為了不使討論範圍過廣，本文只討論 HTTP 協定上的技術及應用，FTP 協定上的平行存取技術留待未來的研究。

Web 資源的平行存取技術需要兩種 HTTP 1.1 特性的支援才能實行，分別是檔案範圍的指定（byte-range support），以及持續連線的維持（persistent connection support）。

3.1 範圍指定

平行存取技術需要將檔案切割為許多區塊分別下載，每次只下載特定區塊，所以伺服端必須支援範圍指定特性，才能達到單一檔案、多個區塊、同時下載、各個擊破的手法。

下載文件的範圍以 Range 標頭指定，例如要取得首頁目錄 foo.bar 檔案的前 500 個位元組，送出如下請求即可：

```
GET /foo.bar HTTP/1.1
Range: bytes=0-499
Host: skynet.cs.nthu.edu.tw
```

Range 標頭是 HTTP 1.1 規格所制定，所以有些過時的伺服器並不支援此特性；根據規格，支援 HTTP 1.1 的伺服器不一定得支援指定範圍存取的功能，就算支援，伺服器管理員也可視情況將此功能關閉。

雖然範圍指定功能制定於 HTTP 1.1，但只要伺服器支援，在 HTTP 1.0 請求中加

上 Range 標頭同樣可以生效。

3.2 持續連線

以平行存取技術下載檔案時，各個伺服器必須不停地接收、處理來自客戶端的區塊下載請求，在傳輸過程中，每個伺服器可能得傳送數十次甚至數百次的下載請求。

假始每個請求需要不同的 TCP 連線來進行，那麼，這些為數眾多的 TCP 連線，光是連線建立及終止動作就會造成額外的流量及時間浪費，在正常情況下，TCP 建立連線需要三個封包，關閉連線需要四個封包才能達成。

最無法接受的影響是，由於 TCP 的 slow start 機制（請見圖 3-1），每個 TCP 連線必須經過約 $R\log_2 W$ 時間（R 為封包來回傳遞時間，W 為資料接收端的 TCP 接收視窗大小）才能達成全速傳送 [Jaco 88]。若每個區塊都以不同的 TCP 連線來傳送，那麼所有區塊在開始傳送後，都會因為 slow start 機制，以緩慢的步調運作，慢慢地才到達全速，而此時該區塊可能已經送出一半或快要送完了。

如果能夠與映射站台維持 TCP 連線不要斷線，在檔案下載過程中重覆使用此連線，則那麼不但可以節省 TCP 連線建立／中止的時間及頻寬浪費，也只需經歷一次 slow start 機制的影響，對於區塊資料的傳輸效率可有大幅度的改善，所以持續連線的支援也是平行存取技術運作的必要條件之一。我們將在第五章詳細討論 slow start 機制對於檔案下載效率的影響。

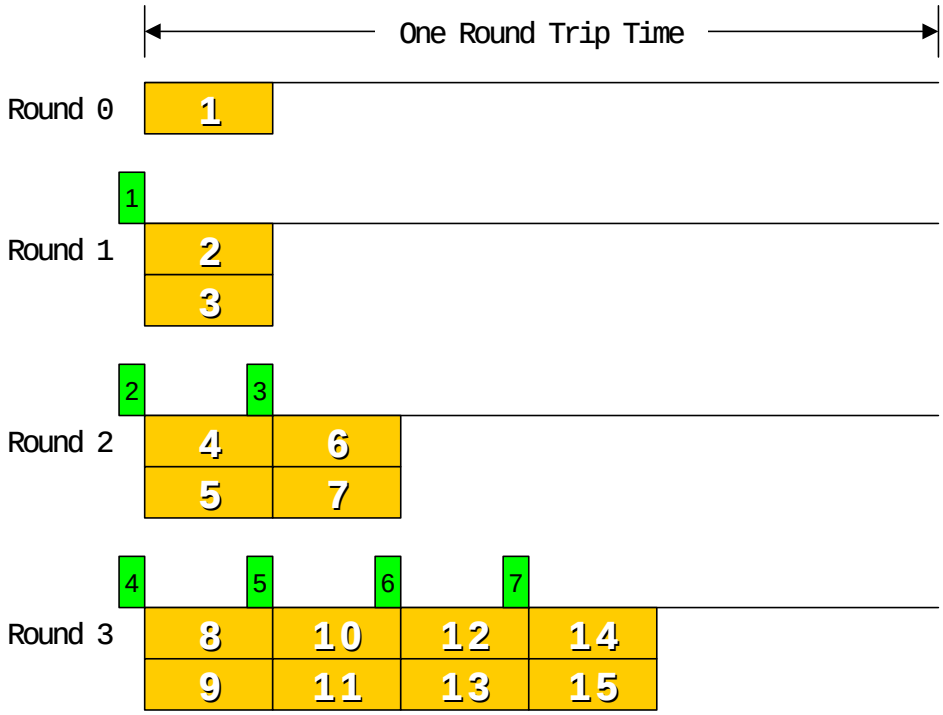

圖 3-1 / Slow start 機制的運作程序

HTTP 1.0 協定的行為模式為標準的請求－回應模式，HTTP 連線建立後，由客戶端發出請求，當伺服端處理請求，並傳送出回應資料後，就會主動結束連線。這種行為模式造成太多的 TCP 連線數目，除了嚴重影響資料的傳輸效能外 [Heid 96]，還會使伺服端產生大量處於 TIME_WAIT 狀態的 TCP 連線，浪費伺服端的資源，也讓伺服器的服務效能降低 [Padm 96] [Mogu 95]。

HTTP 1.1 規格修正此行為，預設情況下，若以 HTTP 1.1 協定進行請求，伺服器不會在回應完成後主動切斷連線，此連線會維持到客戶端主動要求結束連線或某些因素導致伺服器強制中斷連線為止。

雖然範圍指定功能制定於 HTTP 1.1，但只要伺服器支援持續連線特性，Connection 標頭也可同樣使用於 HTTP 1.0 動作。若不支援此能力或某種理由取消此支援的伺服器，必須以 Connection 標頭，指定連線動作為 "Close " 來告知客戶端。

另外，即使伺服器支援持續連線功能，當客戶端在同一個連線中連續發出超過一定數目的請求，或伺服器本身資源不足、連線數目過多等等因素，伺服器都可以

隨時要求中止連線。例如 Apache web server [Apac] 的預設行為是，對於同一個客戶端，在一段時間內，每當請求次數抵達一百次時，就中止連線，防止被特定客戶端獨佔資源的情形。

綜合以上所述，並沒有簡單的規則來判斷某些伺服器是否支援範圍指定及持續連線兩項特性，必須實際對現實世界的諸多伺服器進行實測，才能得知這兩項功能的支援程度。

我們自行撰寫程式來動態搜尋網站，從清華資訊工程系網頁 [NTHU] 出發，取得網頁後，搜尋網頁內的 URL 以獲取新的網站位址，不斷向外擴展，再向新取得的網站取得網頁。以遞迴尋訪的方式取得約兩萬個站台的列表，對它們進行範圍指定及持續連線兩項功能的支援度測試，結果列於表 3-1。

表 3-1 / 隨機取得的兩萬個網站範圍指定及持續連線特性支援度

網域	持續連線支援	範圍存取支援	同時支援	測試站台數目
.COM	3647	5215	2800	8070
.TW	3266	3250	3110	4285
.EDU	1989	2173	1842	3005
.NET	498	579	464	760
.ORG	1139	1247	966	1730
.GOV	334	381	280	500
.UK	358	325	271	450
其它	1797	1861	1499	2440
總計	13028	15031	11232	21240
百分比	61.34%	70.77%	52.88%	100%

由統計的結果，大約有百分之五十的網站同時支援範圍指定及持續連線兩項特性；換句話說，約有一半的網站可適用於平行存取技術。而且隨著舊版本軟體的更新、新伺服器的加入，支援這兩項能力的伺服器比率將會持續增加，這表示我們可以輕易地將平行存取技術落實於今日的網際網路中。

第四章

平行存取技術

平行存取技術的關鍵是，將檔案切分為許多相同或不同尺寸的區塊，同時由不同的映射站台下載這些區塊，全部取得後，再組合回原來檔案。

本章詳細介紹平行存取技術的運作程序，並分析平行存取技術的下載效能表現以及改善空間。

4-1 平行存取技術運作程序

假設欲使用平行存取技術由 M 個映射站台下載檔案，檔案大小為 S。首先，將欲下載的檔案分為 B 個區塊，且 B 大於 M。

開始下載時，主機對 M 個映射站台建立連線，並分別送出區塊 0 ~ M - 1 下載請求，然後開始接收從 M 個連線而來的區塊資料。當任何一個站台將它所負責的區塊成功傳送完成後，主機立即再向它要求下一個尚未要求下載的區塊，在送出所有區塊的下載請求前，不斷重覆此循環。

在所有區塊皆已發出傳送要求，但未下載全部完成前，若某站台 X 成功傳送完它所負責的區塊時，進行如下處理：

- 將所有正在傳送區塊的映射站台依估計剩餘時間進行排序。
- 尋找估計剩餘時間最久，剩餘未傳送的資料量大於某一定資料量（我們設定此值為 32 KB），平均傳送速率慢於站台 X，且正在傳送同樣區塊的站台不超過一定數目（我們設定此值為 2）的另一個站台。
- 若成功找到符合以上要求的站台 Y，則讓映射站台 X 進行與站台 Y 相同的工作，下載同一個區塊資料，同時進行。不同的是，站台 X 將由站台 Y 目前的

進度開始下載，而不必重覆傳送站台 Y 已取得的部分。

若兩個或兩個以上的站台同時傳送同一個區塊，就稱這些站台為「雙胞站台」。我們的設定為，只容許兩個站台互為雙胞站台，由於雙胞站台同時下載同一個區塊的資料，無可避免地一定會造成網路頻寬的浪費。不過雙胞站台有可能讓檔案下載提早完成，這是種賭注，所以勢必得投入一些成本。

當雙胞站台中任一個站台完成區塊的傳送後，客戶端程式會立即中止負責同一區塊其它站台的 TCP 連線，以防止頻寬的額外浪費。由於我們的設定為只容許兩個站台互為雙胞站台，所以在最壞情況下，可能會浪費掉 $(M-1)\times\frac{S}{B}$ 位元組的頻寬，不過這種情況是十分少見的，一般來說浪費的頻寬會遠少於此值。

承上，若無法找到符合條件的站台 Y，則試著尋找平均傳送速率最低，剩餘未傳送的資料量大於某一定資料量，平均傳送速率慢於站台 X 的站台 Z。將站台 Z 的工作搶過來由站台 X 接手，並結束站台 Z 的工作。

若仍沒有符合上述條件的站台，則結束映射站台 X 的工作，切斷連線，等待其它映射站台工作的結束。

4-2 效能評估

理想狀態下，平行存取技術可獲得所有映射站台傳輸速率相加起來的傳輸速率。

所謂的理想狀態，必須滿足下列條件才可能發生：

- □區塊的分派要適宜，讓所有映射站台保持忙碌傳送資料，直到最後一刻同時完成資料的傳送。
- □所有映射站台不會在檔案傳送中途主動結束連線。

雖然理想情況幾乎沒有實現的可能，不過在一般情況下，平行存取技術已能提供幾近理想傳輸速率的效能。理想傳輸速率的定義為，以平行存取技術下載檔案的過程中，所有映射站台平均資料傳輸速率的總和。從下列兩實驗結果，圖 4-1 及

圖 4-2[1]，就可看出平行存取技術的表現十分良好。

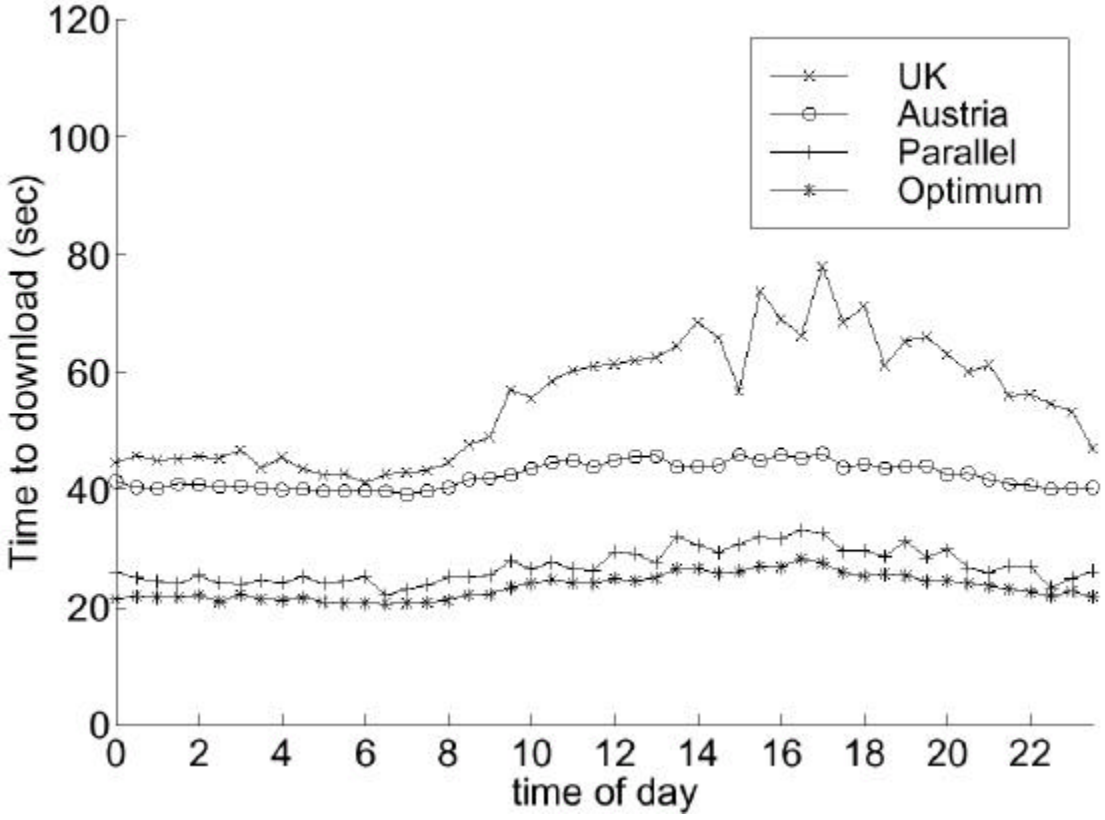

圖 4-1 / 平行存取下載時間，S = 5 MB，B = 40，M = 2

[1] 引用自 [Pabl 00]。

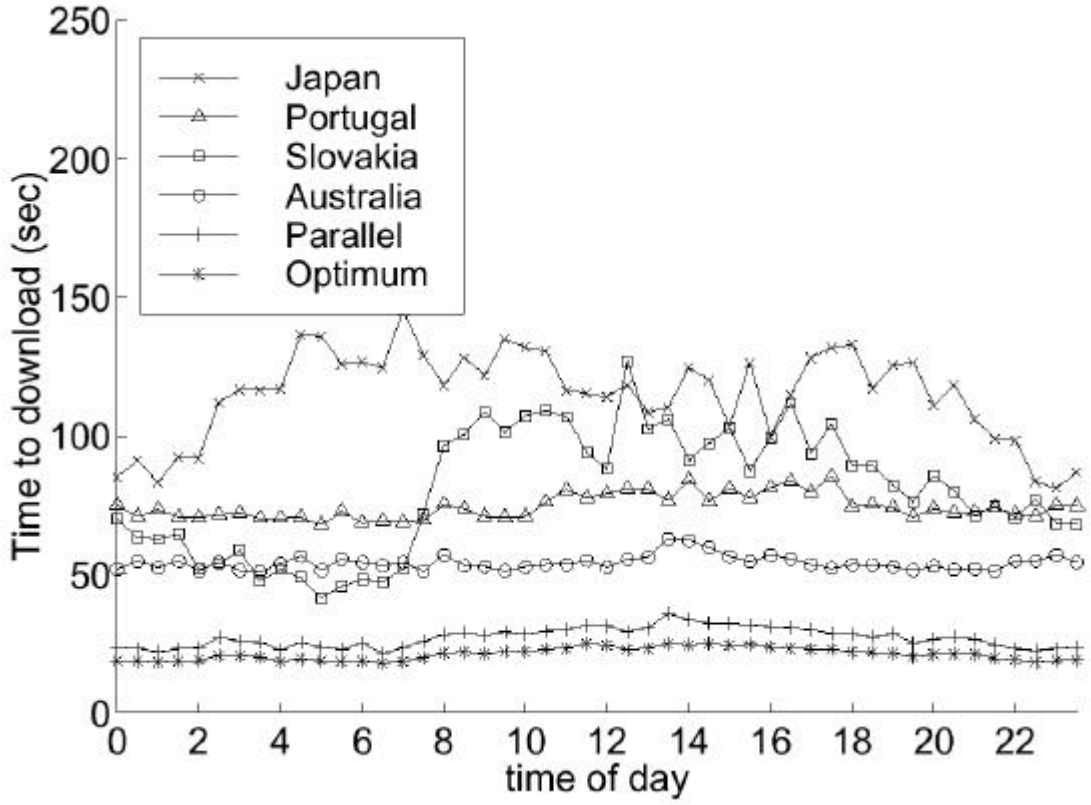

圖 4-2 / 平行存取下載時間，S = 256 KB，B = 20，M = 4

進一步評估平行存取技術的下載效能，我們進行如下的實驗量測：

□分別對 2、3 個站台以不同區塊尺寸下載 2.5MB 大小的檔案

□分別對 2、3、4、5 個站台以不同區塊尺寸下載 16MB 大小的檔案

第一組實驗提供三個站台，當進行少於三個站台的平行存取下載測試時，每次皆以亂數取出所需數目的站台；第二組實驗另外提供五個不同的站台，處理方式相同。

我們進行為期一周的實驗，不斷地進行各組、各個站台數目、各種區塊尺寸的下載測試，記錄每次下載時間及最佳下載時間。最佳下載時間的計算方式為，下載完成後，累計各個映射站台傳送資料時的傳輸速率，除檔案長度的結果。兩組實驗所得到的下載速率對理想傳輸速率比值與區塊大小、站台數目的關係列於圖 4-3 及圖 4-4。

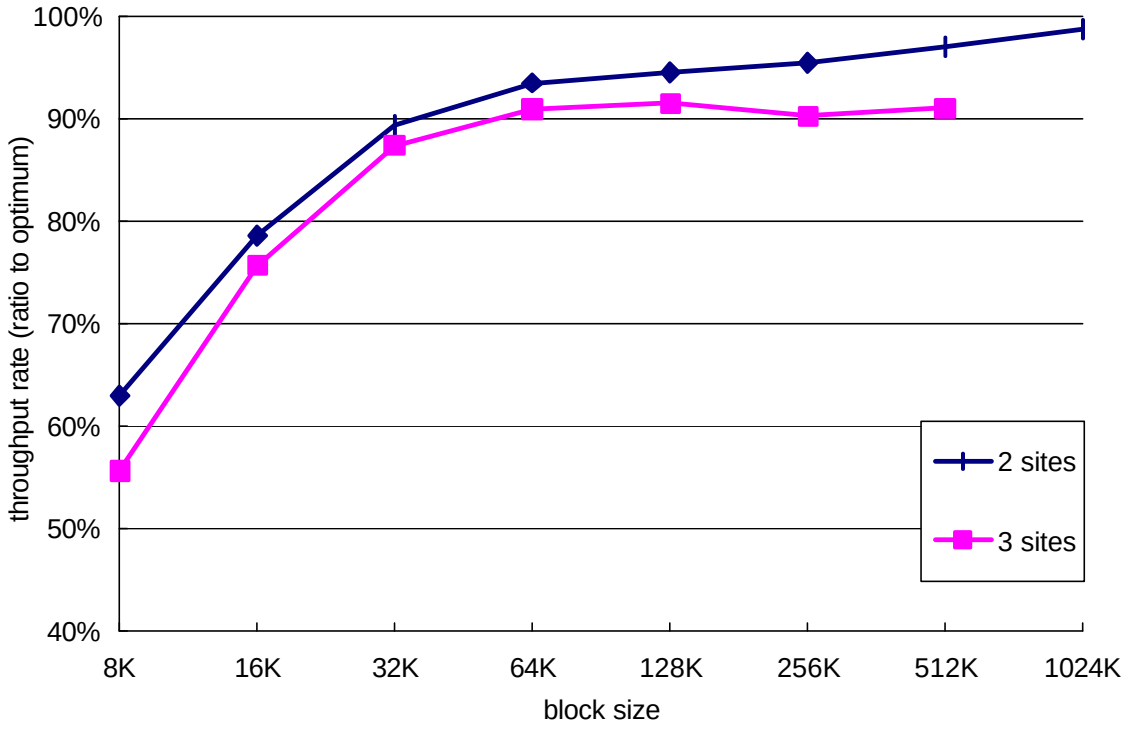

圖 4-3 / 平行存取下載效率，S = 2.5 MB，M = 2、3

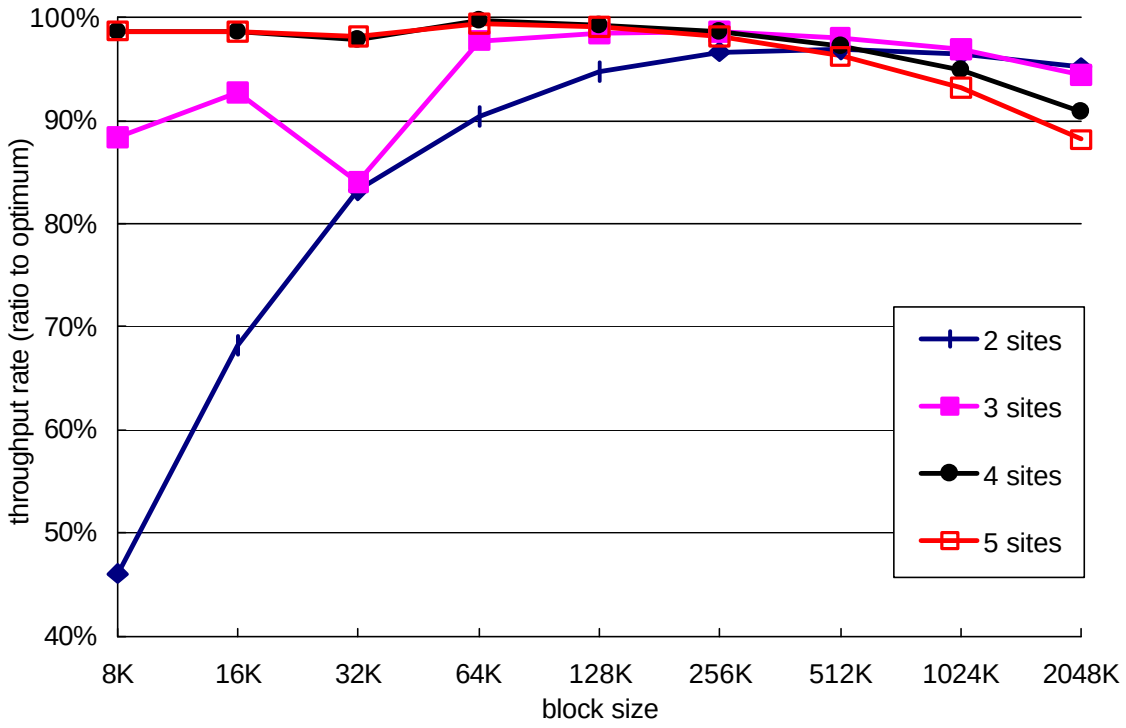

圖 4-4 / 平行存取下載效率，S = 16 MB，M = 2、3、4、5

由圖中可得，站台數目越多，區塊尺寸越多，下載效率就越高。大致上來說，只要區塊尺寸有合理的大小（如 128K、256K），就能輕易獲得理想傳輸速率九成

以上的傳輸率。甚至，在兩組實驗中，都有相當接近理想傳輸率的表現。

值得注意是的，在第二組實驗中，當區塊尺寸超過某個大小時，傳輸速率便開始下滑，這是由於區塊分布不均，讓某些站台空等其它站台傳送資料的結果，在情況在站台數目多的測試組尤其顯著，我們將會第六章詳細討探這種現象。

根據 [Pabl 00] 的實驗結論及我們進行的諸多測試，我們認為只要適當地選擇區塊尺寸，平行存取技術就能發揮十分接近理想傳輸速率的整體效能。

第五章

檔案大小與傳輸效能

TCP 是個十分成功、適應力強大的連線導向（connection-oriented）通訊協定，從慢如牛步的 2400 bps 數據機，到 Gigabit 等級的 ATM 網路，TCP 都能夠如預期般地運作良好，並發揮一定程度的效率表現。TCP 如此亮眼的表現歸功於它的諸多 flow control 及 congestion avoidance 機制 [Jaco 88]。

World Wide Web 服務所使用的 HTTP 通訊協定是基於 TCP 協定之上運作的。Web 的存取模式為次數多、資料少的請求－回應模式。當瀏覽器欲取秀出某個網頁時，必須先取得它的 HTML 網頁，分析網頁內容後，再依需求向伺服器要求網頁內嵌的影像、動畫等等檔案。一些研究指出，web 存取的平均回應長度不超過 1KB [Cunh 95]、6KB [Touc 96] 或 21KB [Arli 96]。

不幸的是，TCP 並不適合這類型頻繁的、短促的「建立連線－請求－回應－結束連線」模式，適合 TCP 特性的是大量的、長時間的資料傳輸。即使是同一條 TCP 連線，通常資料量越大，所能獲得的平均傳輸效能越高，這是由於 TCP 連線 overhead 的影響，底下會詳細討論。

撇開傳輸效率不談，為使主動中斷連線的主機能夠在最後一個 ACK 遺失時重送 ACK 封包，也為使 TCP 連線不受到使用同樣 socket pair 的前一個 TCP 連線餘留封包的干擾，TCP 連線在連線結束之後，在主動中斷連線的主機上，TCP 連線資訊不會立即消失不見，而是進入 TIME_WAIT 狀態。TIME_WAIT 狀態會維持兩個 MSL（maximum segment lifetime）[2]時間，此時 TCP 連線仍然佔用系統資源（通常為 PCB，protocol control block），直到 2MSL 時間後才能夠釋放資源。這

[2] MSL 通常定義為 30 秒、1 分鐘或 2 分鐘。

對於十分忙碌、每分鐘必須接受上千次請求的 web 伺服器來說，形成相當大的負擔。

由於傳輸效率問題及 TIME_WAIT 狀態所造成的伺服器資源不足問題，許多研究試著提出改善之道。例如改用 Transaction TCP [Brad 92][Brad 94] 做為 HTTP 的傳輸層協定，或改用 P-HTTP [Padm 96][3]，避免每次發出 HTTP 請求皆建立、中斷 TCP 連線的 overhead，並維持 congestion control 資訊，減少不必要的 congestion avoidance 機制運作所造成傳輸效能降低的影響。

另一種建議是改用以 UDP 為基礎的請求－回應模式協定，如 ARDP（Asynchronous Reliable Delivery Protocol）[Neum 92] 為 HTTP 的傳輸層協定，期能減少建立及中斷 TCP 連線的 overhead。

5-1 TCP 連線的 overhead

根據 [Padm 96] 的實驗指出，在相同的網路環境下，典型的 HTTP 回應資料通常只能達到全速傳送大量資料約 10% 的傳輸速率。分析的結果，導致這種現象的因素為 TCP 連線建立及 congestion avoidance 機制所造成的 overhead。

[3] P-HTTP 已加入 HTTP 1.1，成為 HTTP 1.1 的 persistent connection 特性。

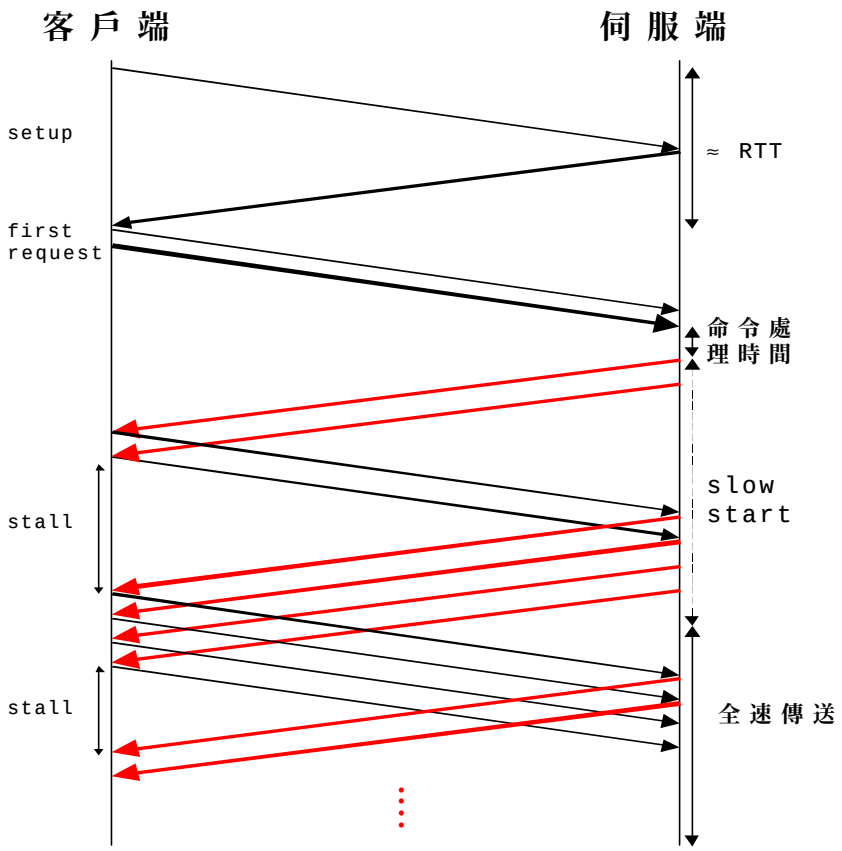

圖 5-1 / 典型的 HTTP 請求封包傳遞次序（細箭頭代表 SYN 或 ACK 封包、粗箭頭代表資料封包）

上圖是典型的 HTTP 請求及回應的封包傳遞次序。由圖中可看出客戶端欲進行某份文件或檔案的請求前，必須先送出 SYN 封包建立 TCP 連線，待收到來自伺服器的 ACK 後，才能送出 ACK 及 HTTP 請求資訊；從建立連線至送出 HTTP 請求，約需時一個封包來回傳遞時間。

HTTP 請求送出後，等待約一個封包來回傳遞時間，才能收到來自伺服端的回應。而伺服器的回應封包並不是大批蜂擁而至的，它會先送出兩個封包，待收到此二封包的 ACK 後，再發出四個封包，待收到此四封包的 ACK 後，再發出八個封包…（假設客戶端採用每個封包皆回應的原則）

簡單地說，客戶端必須負荷連線建立程序及 slow start 機制的代價後，才能收到由伺服器以全速傳送的資料封包。很顯然的，加上這些額外的代價，整個請求回應過程所得到的平均傳送速率一定會低於全速傳送時的速率。

- **建立 TCP 連線的代價**

TCP 採用 3 way handshaking 機制來建立連線。由於 HTTP 採請求－回應模式，客戶端會在連線建立後立即發出請求封包，若一切順利的話，TCP 連線建立程序約須額外花費一個封包來回傳遞時間。

- **TCP slow start 機制的代價**

為了適應網路的頻寬及其它封包的流量，TCP 連線不會在建立之後立即以全速發送資料，而以 *congestion window*（簡稱 *cwnd*）小心地控制目前可發送的封包數目，才不致造成網路阻塞，封包遺落的情形，反而減少可用頻寬。

連線建立後，*cwnd* 值設為 1。TCP slow start 機制的運作可分為兩個階段：

□在 slow start 階段，每收到一個 ACK，*cwnd* 就加 1。

□在 congestion avoidance 階段，每收到一個 ACK，*cwnd* 加上 $\frac{1}{cwnd}$。

而兩個階段的切換由 *ssthresh*（slow start threshold）以及封包遺落事件來控制。

由圖 3-1 可看出，slow start 程序事實上並不 "slow"，在網路頻寬允可及接收端的接收視窗值夠大的情況下，每經過一次 stall，可同時送出的封包數目會呈指數成長。由於在伺服器收到來自客戶端的 HTTP 請求前，已經收到一次 ACK，所以當伺服器開始發送資料封包時，*cwnd* 為 2，第一個 stall 就可連續發出兩個資料封包；接下來，每個 stall 可送出的封包數目將呈 4、8、16.指數成長。

不過上述的情形只發生在採用「回應每個資料封包」法則的 TCP 服務程式，事實上，為了減少回應封包的數目，現今大多數的 TCP 服務程式都採用延遲回應法則（delayed ACK policy）。

延遲回應法則是指，當接收端成功收到資料封包後，不會立即發出回應封包；它會等待一段時間（通常為 200 ms），看看是否有其它封包想同行，如果沒有的話，等待時間到後才送出回應封包。若在這段等待時間內，若接收端有資料封包

要送出，回應訊息會搭便車，隨著此資料封包一併送出；或者接收端收到兩個最大尺寸的封包時，回應封包就不再等待，立即送出。

因為延遲回應法則的關係，在 TCP 的 slow start 階段，每個 stall 所能送出封包個數，不再呈指數成長（如表 5-1 第三欄）；假設每次在接收端的等待時間到達前，就因收到兩個最大尺寸資料封包而提前送出回應封包，那麼每個 stall 所能送出的封包個數如表 5-1 第二欄；在最差的情況下，每次接收端等待時間內都未收到來自發送端的最大尺寸封包，那麼每個 stall 所能送出的封包個數如表 5-1 第一欄。

表 5-1 / TCP slow-start 階段，不同的確認法則下能夠遞送的封包數目（三欄分別是，延遲回應法則的最差情況、延遲回應法則的最佳情況以及每個資料封包皆回應的結果）

stall	no delayed ACKs	delayed ACKs	ACK every segment
1	2 (2)	2 (2)	2 (2)
2	3 (5)	3 (5)	4 (6)
3	3 (8)	5 (10)	8 (14)
4	6 (14)	8 (18)	16 (30)
5	9 (23)	12 (30)	32 (62)
6	12 (35)	18 (48)	64 (126)
7	18 (53)	27 (75)	128 (254)
8	27 (80)	41 (116)	256 (510)
9	42 (122)	62 (178)	512 (1022)
10	63 (185)	93 (271)	1024 (2046)

根據 [Heid 97] 所建立的模型，我們可得到 slow start overhead 在各種 bandwidth-delay product 環境下對資料傳輸時間所帶來的影響，如圖 5-2。

我們可清楚地看出，頻寬越高、封包來回傳遞時間越長，slow start 機制造成的 overhead 就越大。簡單地說，網路路徑的 bandwidth-delay product 決定了 slow start 機制的 overhead。

由圖 5-2，使用 512 位元組 TCP 資料長度的封包下載 12400 位元組的資料，頻寬 100Mbps、封包來回傳遞時間 400 ms 的情況與幾乎沒有 overhead 的假設情況相較，足足多了九倍的傳輸時間。而頻寬高、封包來回傳遞時間長的網路並不是不可能出現，如衛星連線就是個典型的例子，TCP 在這方面尚無良好的解決之道。對於今日普遍的 modem、Ethernet、ISDN 等等網路而言，bandwidth-delay product 不大，所以此影響不算嚴重；不過等到 Gigabit Ethernet、ADSL 等網路開始普遍之後，TCP overhead 導致的頻寬浪費可能又是另一項需要努力研究改善的課題了。

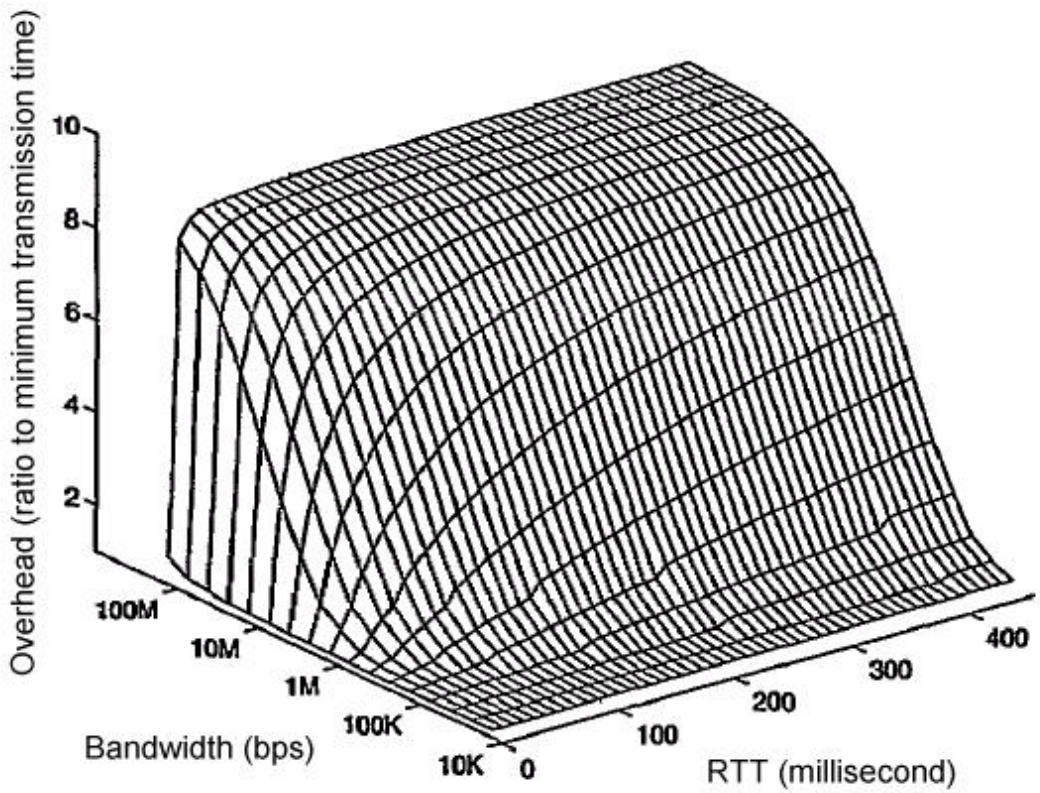

圖 5-2 / TCP slow start 機制在不同的 bandwidth-delay product 環境下對資料傳輸時間帶來的影響，使用 512 位元組 TCP 資料長度的封包下載 12400 位元組的資料[4]

[4] 此圖為 [Heid 97] Fig.2 的重製。

有了理論值的估算，實際的量測更能證明 TCP 連線 overhead 所帶來的影響。我們選擇兩部伺服器來進行實驗：一部伺服器與客戶端主機位於同一個區域網路，封包來回傳遞時間小於 1 ms，使用 Ethernet 的 TCP 最大封包尺寸上限值 1460 位元組；另一部伺服器距離遙遠，平均封包來回傳遞時間約 70 ms，分別使用 1460 位元組及 536 位元組的 TCP 封包長度進行下載。

我們分別使用常見的 TCP 接收視窗設定值－8KB 及 32KB 來進行傳輸測試。每次測試皆由客戶端程式向伺服器建立 TCP 連線、發出要求下載某個尺寸檔案的 HTTP 請求，接收完成後立即關閉連線。傳輸的檔案大小由 1KB 至 10MB 不等，每種檔案尺寸皆進行十次，再記錄其平均值。

圖 5-3 及圖 5-4 分別是遠端及近端伺服器的傳輸速率－檔案大小關係圖。

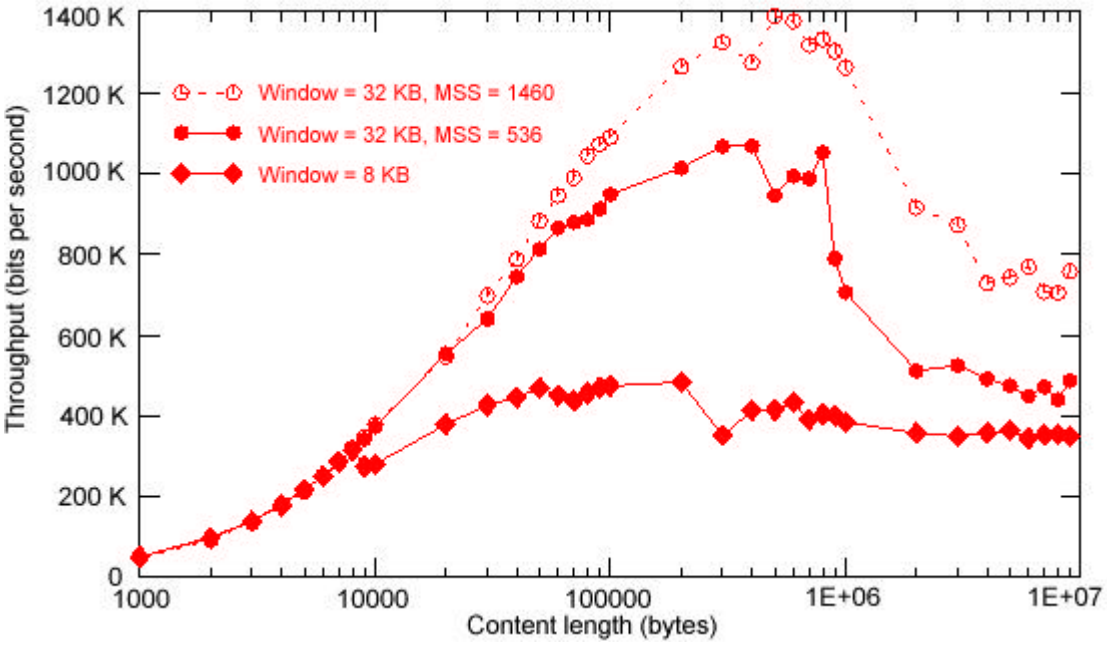

圖 5-3 / 檔案大小及傳輸效率關係，RTT » 70 ms

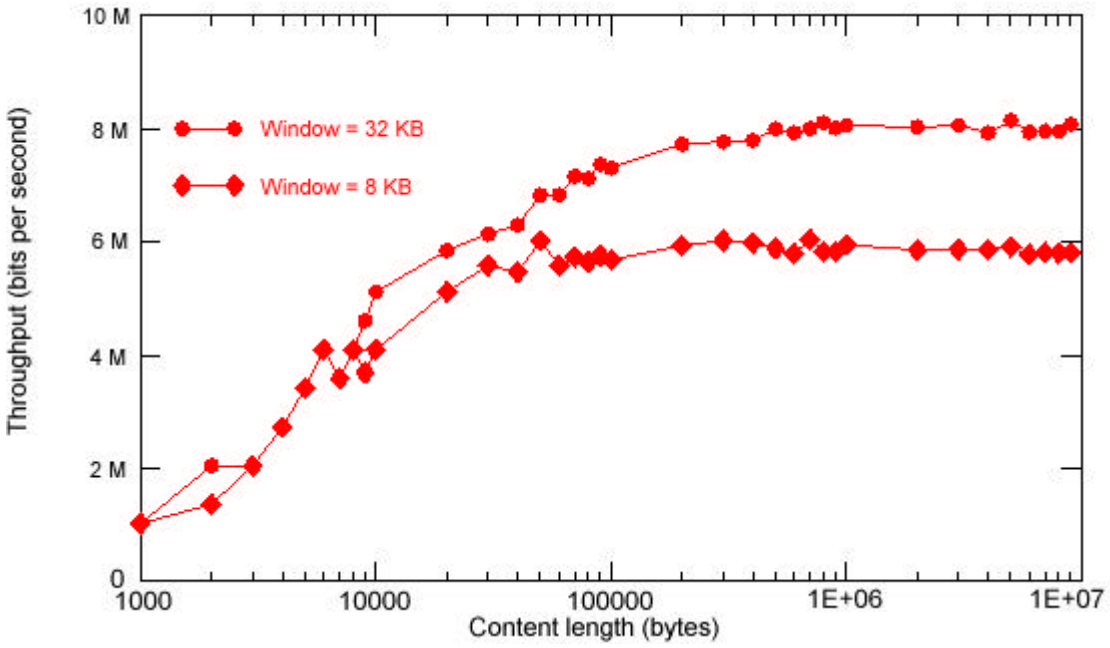

圖 5-4 / 檔案大小及傳輸效率關係，RTT ≈ 0 ms

由圖 5-3 可看出，與遠端伺服器進行下載測試，下載 2KB 的檔案只能得到最高速率的不到 10%；若使用 32KB 接收視窗尺寸，下載 20KB 的檔案只能達到最高傳輸率的一半；直到下載 500KB 大小的檔案時，才能完全免除 TCP 連線建立及 slow start overhead 的影響，達到最高傳輸效能。

我們同時也可以清楚地看出，TCP 接收視窗尺寸若設定太小，將會嚴重影響資料傳輸速率。另外，下載大小超過 500K 的檔案時，整體效能十分明顯地降低，這是因為當 *cwnd* 值越來越大，每次可送出的封包數量越來越多，再加上網路上其它封包流量，超出連線路徑上某個路由器的緩衝區負荷，造成封包掉落。

當發送端未收到其發送出去資料封包的 ACK 時，表示該資料封包掉落，視封包掉落情況是單一封包遺落還是大量封包遺落，發送端能否在特定時間間隔內再收到其它的 ACK 封包，TCP 可能有兩種不同反應：

□遺落單一或數個封包，但其它封包仍正常傳送時，就會啟動 Fast Retransmit 及 Fast Recovery 機制，修正 *cwnd* 及 *ssthresh*，在收到新的 ACK 封包後，進入 congestion avoidance 階段。

□遺落大量封包，接收端在短時間內無法收到三個 ACK。則將 *ssthresh* 設

為 window size 的一半，並將 *cwnd* 設為 1，進入 slow start 階段。

不論封包的遺落模式，或 TCP 進行的處理為何，一定會對效能帶來極大的影響，讓傳輸速率嚴重地往下掉。從圖中可看出，較大的最大封包尺寸（*mss*）能夠達到較高的最大傳輸率，對於這種超出網路負荷而引發效能降低情況也能維持較高的傳輸率。

至於與近端的伺服器傳輸資料的結果，2KB 大小的檔案可以獲得最高傳輸效能的 25% 速度；20KB 大小的檔案就能得到約 70% 的最高傳輸效能。這是因為絕大部分的 TCP 服務程式對於區域網路內部的連線，取消 slow start 機制的結果，所以很快地就可以克服各種 overhead，達到高傳輸效能。

5-2 平行存取技術對檔案大小影響的改善

TCP 連線的連線建立程序及 slow start 機制所帶來的 overhead 讓檔案尺寸因素影響檔案傳輸效率甚巨。

對於使用多條連線同時進行下載的平行存取技術，由於每條連線無可避免地同樣會有 3 way handshaking 連線建立程序以及 slow start 機制造成的 overhead，因此不論建立幾條連線，皆無法改善 overhead 的影響。

不過對於我們在圖 5-3 所看到，當路由器緩衝區不足以同時負荷主機大量傳送出的資料封包及網路上的其它流量時，就會崩潰，導致封包掉落，讓資料發送端可能啟動 slow start 機制或 Fast Retransmit、Fast Recovery 機制，而這些影響會大幅降低檔案傳輸效率。

平行存取檔案的優點在這兒顯現。由於平行存取的概念為負載分擔（load balancing），連線的映射站台分別經由不同的網路路徑將區塊資料傳送到客戶端主機，只要這些映射站台不是走相同的路徑，就可以減輕各路徑路由器的負擔，減少路由器緩衝區不足情況發生的機會，如圖 5-5。

從另一方面來看，就算與某個映射站台 M 的連線發生路由器緩衝區不足，傳送速率嚴重下滑的情況，其它站台的連線反而有機會提高傳輸率，補足站台 M 所空缺的頻寬。由於平行存取機制能夠動態地調整各映射站台所分配的區塊數目，所以不論發生狀況是哪個站台，趁機佔用頻寬的是哪個站台，都不會造成額外的分配問題，如圖 5-6。

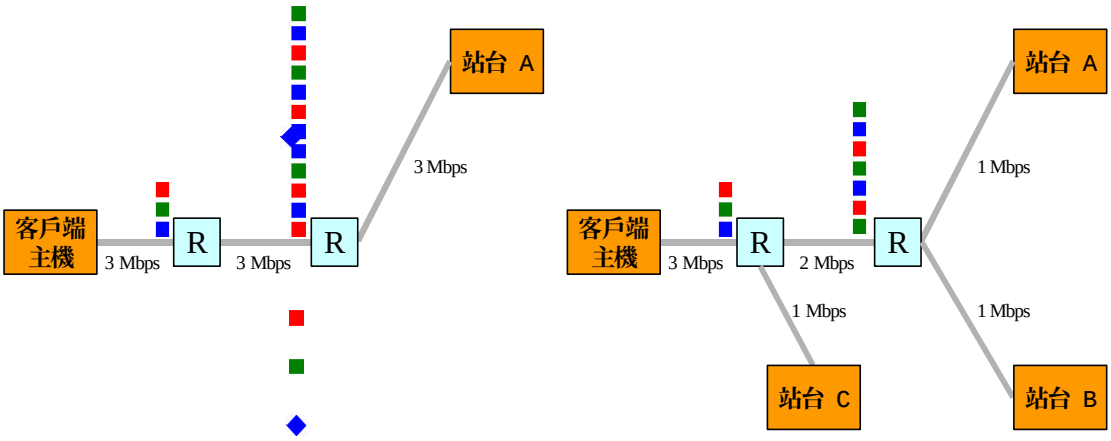

圖 5-5 / 對多個站台平行存取，可分散網路負荷

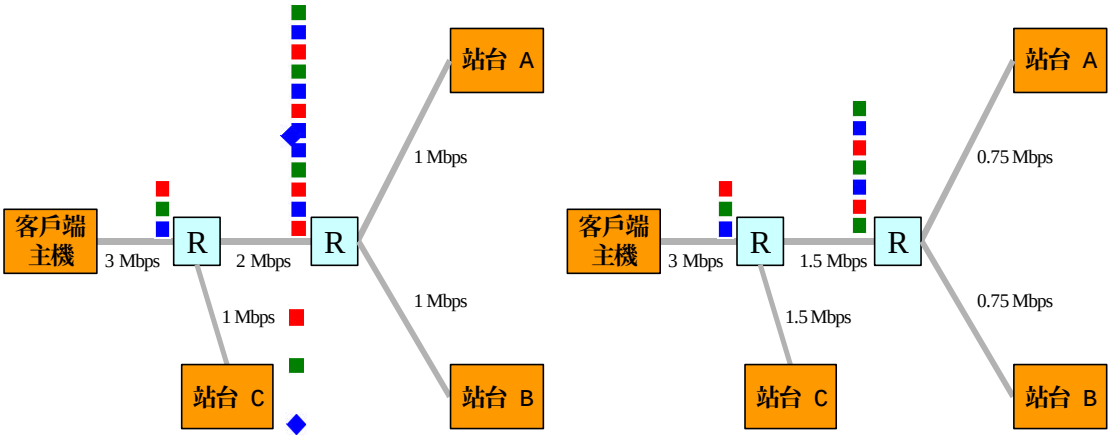

圖 5-6 / 對多個站台平行存取，當部分站台的網路超過負荷時，能夠由其它站台補充流量

對此，我們進行不同檔案大小的傳輸測試，分別對單一站台單獨下載、三個站台平行下載及五個站台平行下載，記錄平均傳輸速率。對於一個或三個站台的測試組，皆以亂數隨機由五個站台取得目的站台，最後再以平均計算，所以實驗並不會被特定站台、網路路徑或路由器所影響。

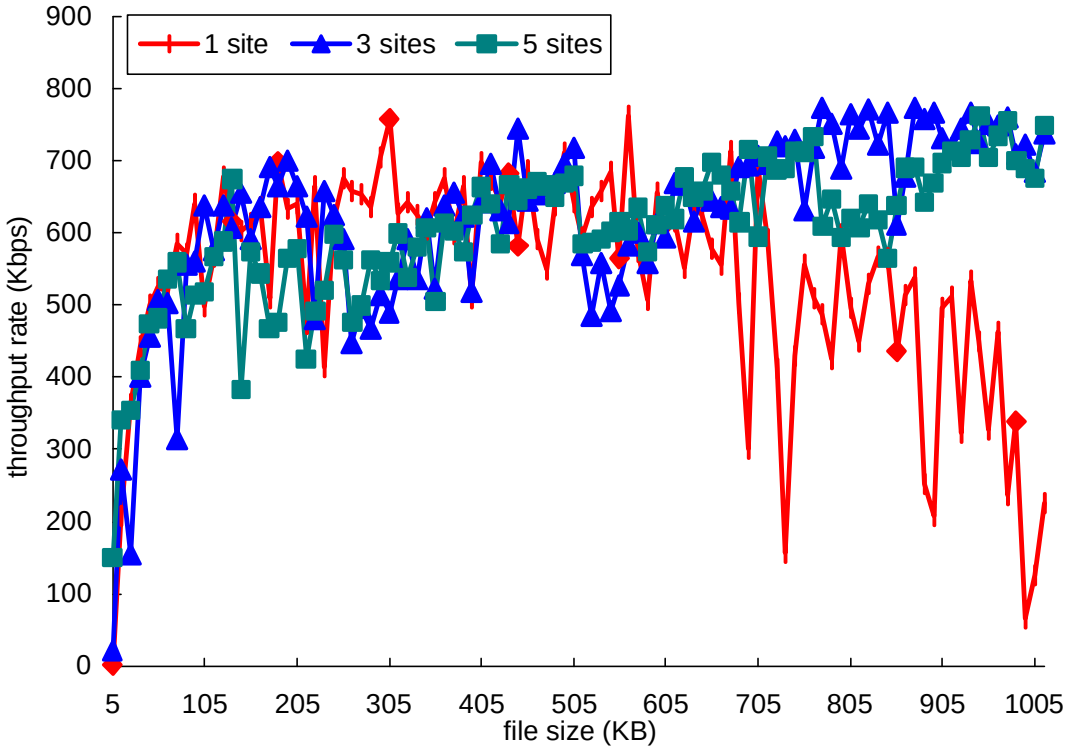

圖 5-7 / 平行存取對於檔案下載速率的改善

由圖 5-7，由於所選擇的五個站台，共享相同的頻寬瓶頸網路，所以即使是多個站台同時下載也不能讓整體傳輸速率提高多少，只能達到單一站台下載時相同的傳輸速率。但是當檔案大小超過 700KB 時，單一站台下載的測試組由於超出路由器緩衝區可接受的範圍，所以受到 congestion avoidance 機制的影響，傳輸速率大幅滑落；而分別從三個及五個站台平行下載檔案的測試組，由於將負荷分擔給不同的網路路徑，所以絲毫不受影響，傳輸速率不降反昇。

所以可知，平行存取技術可以大幅減輕網路壅塞時，封包遺落引發 congestion avoidance 機制 overhead 的影響。

第六章

區塊尺寸對平行存取下載效能的影響

由 4-2 小節的實驗結果，我們可知，區塊大小的選擇對於整體的檔案下載效率有極大的影響。本章詳細討論及分析區塊尺寸對於平行存取技術檔案下載效率所造成的影響，建立區塊尺寸－下載時間模型，計算使用各種區塊尺寸下載檔案能夠達到的傳輸率。再進一步將模型擴展為用於平行存取技術的區塊尺寸－下載時間模型，並以以此模型發展出自動計算最佳區塊尺寸的動態平行存取技術。

6-1 大小的考量

平行存取技術的目標為：「全速運行，同時結束」。將檔案切割為許多區塊，分頭進行，從各個連線站台分別取得各個區塊的資料。其中，區塊的大小是最難決定的，過大不好，過小也不行。區塊大小問題是這麼出現的：

- 區塊越小，伺服器閒置時間越多，越難達成「全速運行」目標

每回向伺服器要求的區塊成功下載後，若還有剩餘的區塊尚未發出請求，程式將立即發出新的請求，要求下載另一個區塊。但在送出前一區塊的最後一個封包後，以及收到下一道命令之前，此時伺服器將處於閒置狀態，我們將處於閒置狀態的這段時間稱為「閒置時間」，請見圖 6-1。

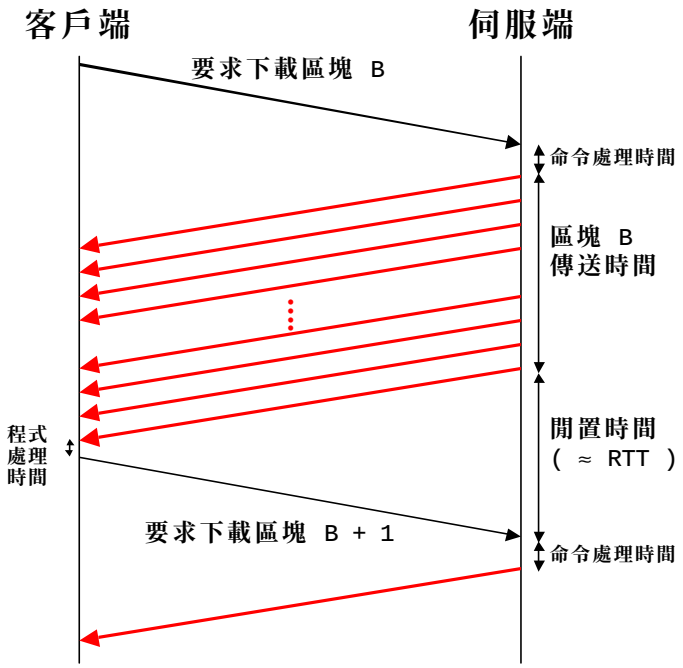

圖 6-1 / 區塊完成及請求之間的閒置時間

扣除客戶端的程式處理時間，閒置時間約等於封包來回傳遞時間；由於兩兩請求之間就會造成一次閒置狀態，所以閒置狀態發生的次數幾乎等於區塊數目。因此若區塊尺寸越小，區塊數目增加，伺服器閒置時間就越多，造成整體傳輸效率的降低；而且在頻寬大、封包來回傳遞時間長的情況下，此影響越為明顯。

「全速運行」目標指的是減少伺服器閒置時間，使平行存取下載檔案的期間，能儘量使每一個伺服器忙碌，不斷地送出資料。所以可知，區塊越小，越難達成「全速運行」目標。

- 區塊越大，越難以分派區塊，越難達成「同時結束」目標

為使伺服器閒置時間越少越好，所以區塊數目不能太多。但區塊數目越少，區塊尺寸必定增大，這麼一來，下載一個區塊的時間也許動輒數十秒上百秒，再加上網路狀況的陡變叢生，變異度極大，對同一個站台要求同樣大小的區塊，也許上一次只需要五秒，這一次就得花上三十秒。由於網路壅塞情形是無法預測的，因此要準確地預測每個區塊的下載時間幾乎是不可能的任務。

「同時結束」目標指的是，向不同伺服器要求的區塊必須盡量在同一時間結束，

才不會造成幾個伺服器已經沒有工作，但因為還有其它個伺服器的區塊資料還沒傳送完，白白浪費時間等待的情況。

若區塊尺寸越大，下載時間越長，下載花費時間的變動性也越大，無論怎麼分派區塊，各個伺服器完成請求的時間差距仍然不好控制，很容易就發生某些站台閒置下來等待其它站台結束，白白浪費時間而無法妥善利用的窘況。

6-2 影響傳輸效率的因素

無論如何，在這種難以抉擇的 trade off 下，我們仍得尋找合適的解決方案。本小節分析與區塊尺寸相關，可能影響傳輸效能的因素。

- **閒置時間**

前面已提過，閒置時間是區塊數目所帶來影響效率最著的因素，尤其對於頻寬高、延遲時間長的連線更為明顯。閒置時間指的是伺服器傳送出某個區塊資料的最後一個封包後，再接收到下一個區塊下載請求之前的這段時間。

從圖 6-1 可看出，閒置時間為單一封包來回傳遞時間加上客戶端程式處理時間，因為客戶端程式處理時間非常短，遠小於網路延遲時間，所以客戶端程式處理時間通常忽略不計，將閒置時間視為封包來回傳遞時間。

根據 [Jaco 88]，TCP 連線超過一段時間沒有傳送資料，下次再傳送資料時，因為 *cwnd* 值可能已無法反映目前的網路狀況，必須重設 *cwnd* 值，重新啟動 slow start 機制。從流傳最廣的 BSD 版本 TCP/IP 服務程式看來，上述的「一段時間」是以封包來回傳遞時間預測值來實作。

這條規則所帶來的影響是：客戶端程式完整接收某個區塊資料後，必須立即發出下一個區塊請求，伺服器送出某個區塊的最後一個封包後，伺服器至客戶端主機的封包傳送時間、客戶端程式處理時間，客戶端主機至伺服器的封包傳送時間再加上伺服器程式處理時間，必須小於伺服器 TCP/IP 服務程式所預測的封包來回

傳遞時間。否則當伺服器送出下一個區塊的第一個封包時，由於 *cwnd* 超過時間未更新，便必須再次啟動影響效率甚巨的 slow start 機制。

這就好比在交通狀況良好的道路上駕車行駛，以時速一百公里奔馳著，但卻偶爾遇上幾次紅燈，讓車子不得不緊急煞車停止，等待綠燈再重新起跑，逐漸加速回復原來的速度，雖然車子只停下幾秒鐘，不過減速再加速的動作可是會大大影響平均車行速度的。

所以，針對閒置時間這個因素，除了區塊次數不可過多的結論之外，還必須注意客戶端程式的實作：完整接收區塊資料後，必須儘快將下一個區塊請求送給伺服端，以免承受不必要的 slow start 機制影響。

- **伺服端及客戶端程式的處理時間**

即使尚未牽涉到網路上的封包傳輸動作，就算客戶端程式與伺服端程式位於同一部主機上，進行不同區塊尺寸的檔案分段傳輸，可以發現，傳輸效率仍然深受區塊尺寸的影響，如下圖。

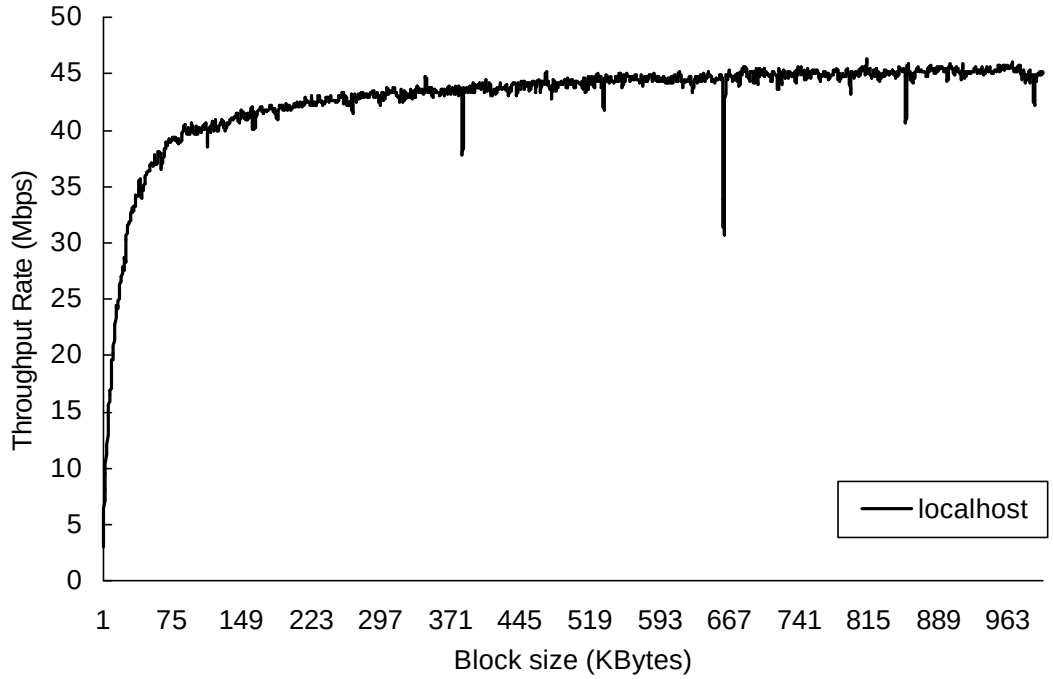

圖 6-2 / 同一部主機上的傳輸效率與區塊尺寸關係

主機自身的傳輸速率受制於處理器的運算速度、記憶體及處理器的頻寬、硬碟資

料讀取／傳送速度、記憶體及磁碟之間的頻寬、TCP/IP 服務程式的緩衝區大小、伺服器程式的處理動作以及主機同一時間的其它工作量等等，影響的因素既多且雜，不過最主要的關鍵在於，同樣大小的檔案，若請求次數越多，則效率越差。

這是因為每一次的請求，從客戶端程式產生下達請求，經由 TCP/IP 服務程式的處理，建立 TCP 連線，再經接收請求的 TCP/IP 服務程式通知伺服端應用程式，從檔案中讀取區塊尺寸的資料，再循著原來的路線回去，這些繁多且雜的動作就足已構成冗長的處理時間。

- **協定及封包標頭的代價**

請求次數越多，除了閒置時間越多外，每次的請求及回應也造成額外封包標頭的傳遞量增多。

不計資料鍊結層（data link layer）的負荷，IP 加上 TCP 標頭大小為 40 位元組；而 HTTP 標頭並沒有固定長度，客戶端發出的請求 HTTP 標頭通常較短，只佔二三十位元組，伺服端回應的 HTTP 標頭通常為兩百　三百位元組之多。

換句話說，區塊數目每加一，就會增加約三百位元組的 HTTP 標頭 overhead，也同時增加約 $\left(1+\left\lceil\frac{256+blocksize}{mss}\right\rceil\right)\times 40$ 位元組的 TCP/IP 標頭 overhead。

- **TCP 連線 overhead**

以平行存取技術下載檔案時，程式會與每部映射站台主機建立 HTTP 連線，要求使用持續連線特性，企圖在同一條 HTTP 連線上進行所有的區塊下載動作。

不過由於公平原則，通常伺服器不願意讓個別連線進行太多的請求動作，當請求動作達到某個數目，或連線超過一定時間後，它很可能就不由分說，在回應給客戶端的回應 HTTP 封包標頭指明，這次回應完成後，伺服端就主動斷線。面對這種情形，寄人籬下的客戶端程式也只好乖乖認命，重新進行連線，繼續未完成的區塊下載動作。

我們在第五章已討論 TCP 連線 overhead 所帶來的影響，而在平行存取檔案的過程中，與每部映射站台都可能遭到斷線，再重新連線的情況。每次重新連線，勢必得再接受一次 TCP 連線 overhead，這對整體傳輸率的影響，不可小覷，尤其在區塊數目多的情況下。

6-3 區塊尺寸－下載時間模型

根據前一小節的討論，我們嘗試建立一個簡單的模型，對於單一站台進行區塊式的下載動作，也就是將檔案切分為許多區塊，每次向伺服器要求一個區塊，此區塊下載完成後，再要求下一個區塊的行為，在不同的區塊尺寸條件下，所需的下載時間。

- **網路模型**

建立模型的第一步驟，首先要定義影響網路傳輸的參數，我們將它列於下表。

表 6-1 / 影響 HTTP 檔案下載效率的網路特性

rtt	封包來回傳遞時間
bw_{avail}	連線網路路徑的可用頻寬
mss	TCP segment 的最大長度（maximum segment size）
muws	最大的可用視窗尺寸（maximum useful window size）

雖然封包來回傳遞時間及可用頻寬時時變動，不過它們卻是影響傳輸率最重要的變數。套用實際數據時，以平均封包來回傳遞時間計算 *rtt*，以各連線的平均傳輸速率最高者為為可用頻寬值 bw_{avail}。

最大的可用視窗尺寸意為連線路徑的容量（bandwidth-delay product）所能容納的最大長度封包的數目，這個參數表示發送端要保持多少封包在網路上流動，才能讓網路維持「滿載」。當視窗尺寸小於 *muws* 時，表示發送端在連續送出視窗尺寸數目的封包後，必須再等待一段時間才能得到回應；若連線的視窗尺寸大於等於 *muws* 時，只要資料來源足夠，發送端就可源源不絕地送出封包。我們可以這

麼計算 *muws*：

$$muws = \left\lceil \frac{bw_{avail} \times rtt}{mss} \right\rceil$$

在這個網路模型中，我們忽略硬體或線材造成的封包傳輸錯誤率。事實上，在絕大部分的網路中，封包資料損毀所造成的封包遺失比率非常小（<<1%）[Jaco 88]，而且我們主要的目的在於分析區塊尺寸、各種 overhead 對於整體傳輸效率的關係，封包毀壞的因素已超出討論範圍，所以我們假設封包資料損壞情形從未發生。

- **流量模型**

我們針對相同長度的檔案，以不同的區塊尺寸向伺服器要求資料，每次完整的檔案傳輸階段，都採用相同的區塊尺寸。所有區塊採用循序下載，客戶端接收到完整的區塊資料後，再發出下一個區塊資料的請求至伺服端。

HTTP 標頭可以是任意長度，即使是同樣的動作，不同廠商、甚至不同版本的 HTTP 服務程式都可能產生不同長度的 HTTP 標頭。為簡化流量模型，我們假設客戶端發出的 HTTP 請求標頭，以及伺服端回應的資料封包所帶的 HTTP 標頭大小平均為 256 位元組，並假設伺服器對於每個 HTTP 請求的處理時間為零。

這種假設並不十分合理，不過由於伺服器的 HTTP 請求處理時間隨著主機硬體、作業系統、TCP/IP 服務程式、伺服器軟體及伺服器主機當時的負載可能產生的十分明顯的差異，因此我們的分析著重在網路、通訊協定對於傳輸速率的影響，忽略主機、伺服器因實作不同產生的差異性。

- **TCP slow start 機制的 overhead**

TCP 連線建立後，處於 slow start 階段，*cwnd* 設為 1，每收到一個回應封包就加 1，直到抵達門檻值 *ssthresh*，才結束 slow start 階段，進入 congestion avoidance 階段。

門檻值 *ssthresh* 的初值為 64KB。遇到網路阻塞引起封包掉落及延遲時，如果發送端能在一定時間內收到三個回應同一個資料封包序號的 ACK 封包，就可馬上啟動 Fast Retransmit 及 Fast Recovery 演算法，修正 *cwnd* 及 *ssthresh*，進入 congestion avoidance 階段，而不必受 slow start 機制的 overhead。

在檔案下載過程中，除非區塊資料已經送完，否則伺服器會源源不絕地送出資料封包，客戶端也會不斷地送出回應封包。假設資料封包 D 或該封包的回應封包遺落，當伺服器很快地收到資料封包 $(D+4)$ 或資料封包 $(D+5)$ 的回應時，就認定資料封包 *D* 已經遺失，所以會啟動 Fast Retransmit 及 Fast Recovery 機制，重送資料封包 *D*，並更新 *cwnd* 及 *ssthresh*：

$$ssthresh = \frac{\min(cwnd, receive\_window\_size)}{2}$$

$$cwnd = ssthresh + 3 \times segment_{size}(D)$$

當伺服器再度收到下一個回應新資料封包（不包含資料封包 *D*）的回應封包時，將 *cwnd* 設為 *ssthresh*，回到 congestion avoidance 階段繼續運作。

在 congestion avoidance 階段，發送端每收到一個 ACK，*cwnd* 增加 $\frac{1}{cwnd}$，呈現緩慢的線性成長，因此對於傳輸速率的影響遠不及 slow start 機制。除非在資料封包 *D* 之後，連續掉落大量資料封包或回應它們的 ACK，使資料封包 *D* 逾時，才會採用重新起始 slow start 階段的方法來處理。

我們假設，即使封包掉落，掉落封包之後的其它封包仍然能順利傳送，而不是大量的連續封包遺落。所以 Fast Retransmit 及 Fast Recovery 演算法上場，而不必再度啟動 slow start 機制。因此，每條 TCP 連線只必須承受一次 slow start 機制的 overhead。

而 slow start 機制 overhead 的計算方式為：

$$ssthresh = \left\lceil \frac{64\text{K}}{mss} \right\rceil$$

$$ss\_segs = \min(muws, ssthresh, window\_size, 100) \quad (1)$$

$$usefulstalltime(i) = \frac{segs[i] \times mss - 2 \times 40}{bw_{avail}} \quad (2)$$

$$wastedstalltime(i) = rtt - usefulstalltime(i) \quad (3)$$

$$slowstart_{overhead} = \sum_{i=1}^{stalls(ss\_segs)} wastedstalltime(i) \quad (4)$$

根據量測，大部分的伺服器都將每條連線所能發出的請求次數限制在 100 **次，我們本身實驗所選擇的伺服器也皆是如此，所以在運算式** (1) **中，將** *ss_segs*，**也就是** slow start **過程內，發送端所發出的資料封包數目限制在** 100 **以下。不過由於** *ss_regs* **同時也為** *muws*、*ssthresh* **及** *window_size* **的最小值，在絕大部分的情況下，***ss_regs* **遠小於** 100，**不會受到伺服器每條連線可接受** HTTP **請求次數的影響。**

運算式 (2)、(3)，**我們首先對照表** 5-1 **第二欄查出，第** *i* **個** stall **所能送出的封包數目** *segs[i]*。**表** 5-1 **第二欄所列的數值為，使用** delayed ACK **法則，但每一次都能夠在延遲時間內，收到來自資料發送端的另一個最大尺寸封包，形成每兩個資料封包回應一次的情況下，**slow start **階段中每個** stall **可以送出的封包數目以及截至該** stall **為止，送出的封包總數。**

在第 *i* **個** stall，**伺服端及客戶端總共只送出** *segs[i]* **個資料封包以及兩個回應封包，正常情形下，資料封包的長度為** *mss*，**回應封包長度為** IP **加上** TCP **標頭的長度** 40 **位元組，其它的時間都浪費掉了。每個** stall **時間約為** *rtt*，**所以每個** stall **浪費的時間** *wastedstalltime(i)* **就等於** *rtt* **減掉封包傳遞所使用的時間。**

已知 *ss_segs*，**就可在表** 5-1 **中查出** slow start **過程所需要的** stall **數目；再將** *wastedstalltime(i)* **累計，就可得到** slow start **機制的** overhead。

- **資料傳輸時間**

在整個檔案傳輸過程中，不計 TCP **連線／斷線等特殊用途的封包，**HTTP **請求及**

回應的區塊及封包數目為：

$$block_{num} = \left\lceil \frac{filesize}{blocksize} \right\rceil$$

$$segment_{num\_per\_reply} = \left\lceil \frac{256 + blocksize}{mss} \right\rceil \quad [5]$$

$$segment_{num} = block_{num} \times segment_{num\_per\_reply} + block_{num} \quad [6]$$

在運算式 (5) 中，我們假設 HTTP 回應封包所含的 HTTP 標頭長度平均為 256 位元組。運算式 (6) 計算回應及請求所需的封包數目。

每個封包都會包含各協定的標頭，由上式 (6)，這些封包標頭所帶來的額外傳輸時間為：

$$http\_header_{overhead} = block_{num} \times \frac{300}{bw_{avail}} \quad (7)$$

$$network\_header_{overhead} = segment_{num} \times \frac{40}{bw_{avail}}$$

因為 HTTP 請求的標頭長度通常都很短，而且為了減少 overhead，近來的 web 程式所發出的 HTTP 標頭有越來越精簡的趨勢 [Sper 95]。我們假設 HTTP 請求及回應標頭的總和約為 300 位元組。

- **區塊式傳輸的 overhead**

將檔案切分為許多區塊，一個個分開向伺服器取得，與直接向伺服器下載整個檔案比較，多了兩種額外負擔：

□閒置時間

□連線建立及 slow start overhead

閒置時間的計算方式十分容易，當伺服器傳送完某個區塊的資料後，直到接收到來自客戶端下一個區塊的請求動作，這之間的等待時間約為一個封包來回傳遞時間。所以閒置時間即為封包來回傳遞時間乘以區塊數目：

$$idletimes = rtt \times block_{num}$$

另一個額外的負擔是，TCP 連線建立後，客戶端會不斷地向伺服端要求區塊，而伺服器通常有每條連線請求次數的上限限制。當請求次數達到伺服器容許的上限值時，就會在回應封包內告知客戶端，此回應結束後，連線就會中止。客戶端若還有區塊仍須向該伺服器下載，就必須重新建立連線，承受再一次的連線建立及 slow start 負擔。

$$connection_{num} = \left\lceil \frac{block_{num}}{100} \right\rceil$$

$$connection_{setup} = rtt \times 2$$

假設伺服器都所能容許每條連線的最大請求次數為 100 次，就可由區塊數目除以 100 得到所需的連線數目。當伺服器強制斷線，再由客戶端重新建立連線，所浪費的額外時間約為兩倍的 *rtt*，如下圖所示。

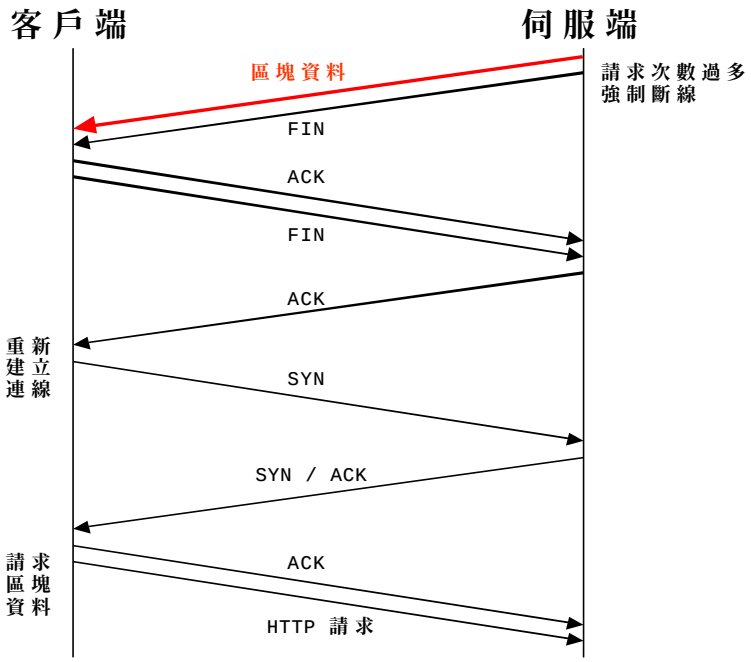

圖 6-3 / 伺服器強制斷線至重新建立連線的過程

連線重新建立的 overhead 很小，伴隨連線重新建立而來的影響是，由於 *cwnd*、*ssthresh* 等資訊都消失，必須重新經歷 slow start、congestion avoidance 階段來達

到穩定的傳輸效率。由於 slow start 機制的 overhead 不小，所以連線次數也會一定程度地影響整體的傳輸效率。

- **傳輸時間預估**

綜合以上的分析結果，若檔案大小為 *filesize* 位元組，區塊尺寸為 *blocksize* 位元組，我們可預估它所需的傳輸時間為：

$$
\begin{aligned}
transfer\_time = & \frac{filesize}{bw_{avail}} \\
& + \frac{idletimes}{2} \\
& + network\_header_{overhead} \\
& + http\_header_{overhead} \\
& + connection_{num} \times \left(connection_{setup} + slowstart_{overhead} \right)
\end{aligned}
$$

下載整個檔案所需的傳輸時間，即為檔案資料本身的傳輸時間，加上各種 overhead。在上式中，閒置時間除以二，是因為閒置時間的上半段，同時也是區塊資料的最後一個封包傳輸時間，已經加總進去，所以不再重覆計算。我們未考慮兩端主機的程式處理時間，事實上，與網路的傳輸速度比較，除非網路速度已達 Gigabit 等級，且檔案尺寸很小，否則主機的處理時間通常可以忽略不計。

6-3 模型驗證

為了驗證 6-2 節的分析，我們進行不同區塊大小的下載實驗。實驗對象為大小 2.4 MB 的檔案，實驗平台為 RetHat Linux，程式修改自 wget [wget] 程式，目的站台的選擇如表 6-2，當然，它們皆支援 HTTP 的範圍指定及持續連線兩項特性。

表 6-2 / 目的站台列表

站台	RTT（ms）	瓶頸頻寬（Mbps）
交通大學	7.2	9.0
台灣科技大學	8.5	9.5
中山大學	13.8	9.1

暨南大學	113.4	5.0

我們分別使用不同的區塊尺寸，由 1KB 至 1024KB，進行區塊式的檔案下載，傳輸效能與區塊大小的關係請見圖 6-4 及 6-5。圖 6-4 為傳輸速率（Mbps）與區塊尺寸關係，圖 6-5 為在不同區塊尺寸時，所能獲得的傳輸速率與最高傳輸效率比值。

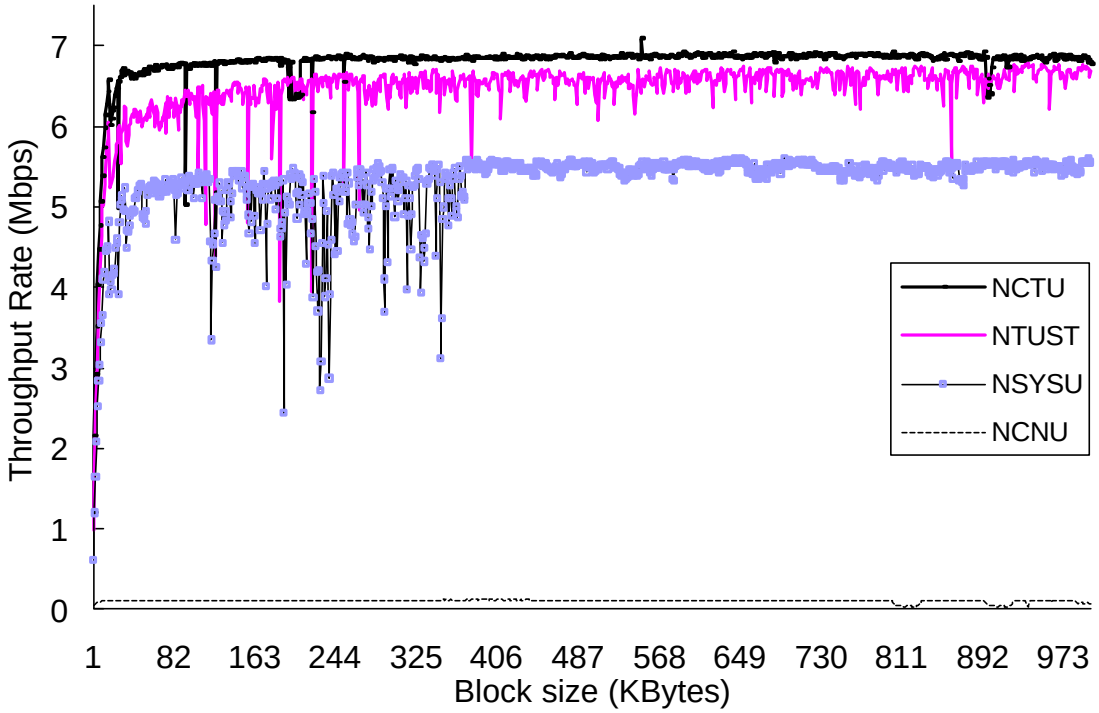

圖 6-4 / 傳輸速率－區塊尺寸關係圖

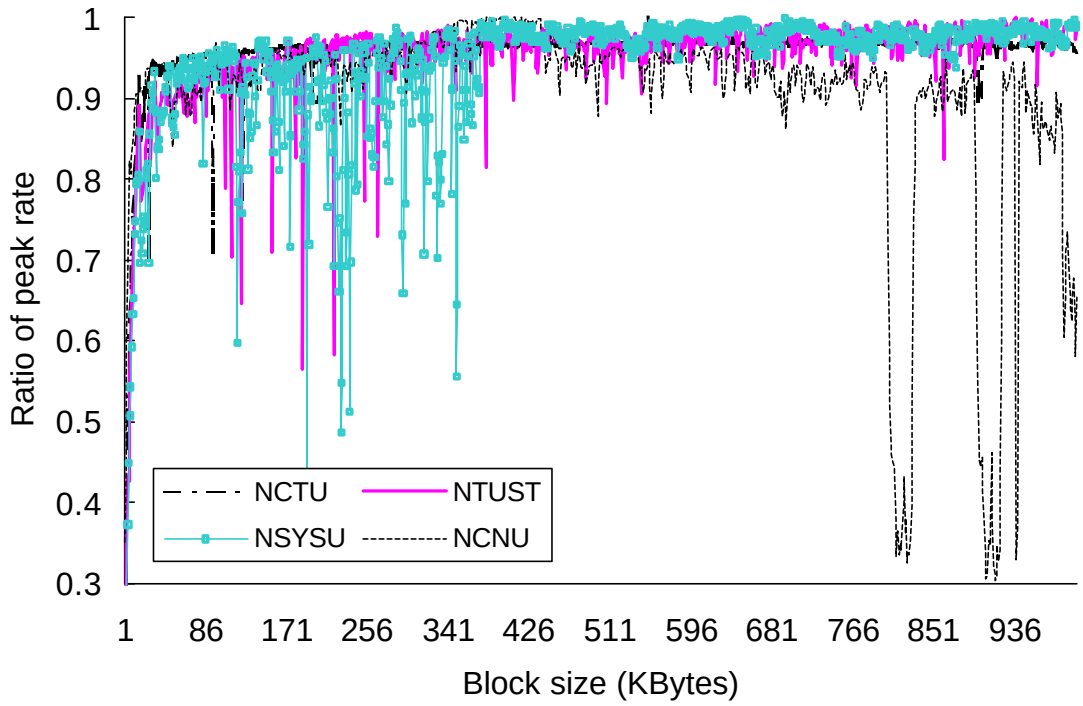

圖 6-5 / 傳輸速率與最佳傳輸速率比值－區塊尺寸關係圖

從上圖，可以發現由於區塊尺寸對於傳輸效率的影響，遠比檔案大小對傳輸效率的影響小多了。大約 40K 的區塊尺寸，就可達到最大傳輸率的九成。

以 6-2 小節所建立的模型，我們進行以各種區塊尺寸進行下載所得到的傳輸效能與預測值的比較，如圖 6-6、6-7 及 6-8。

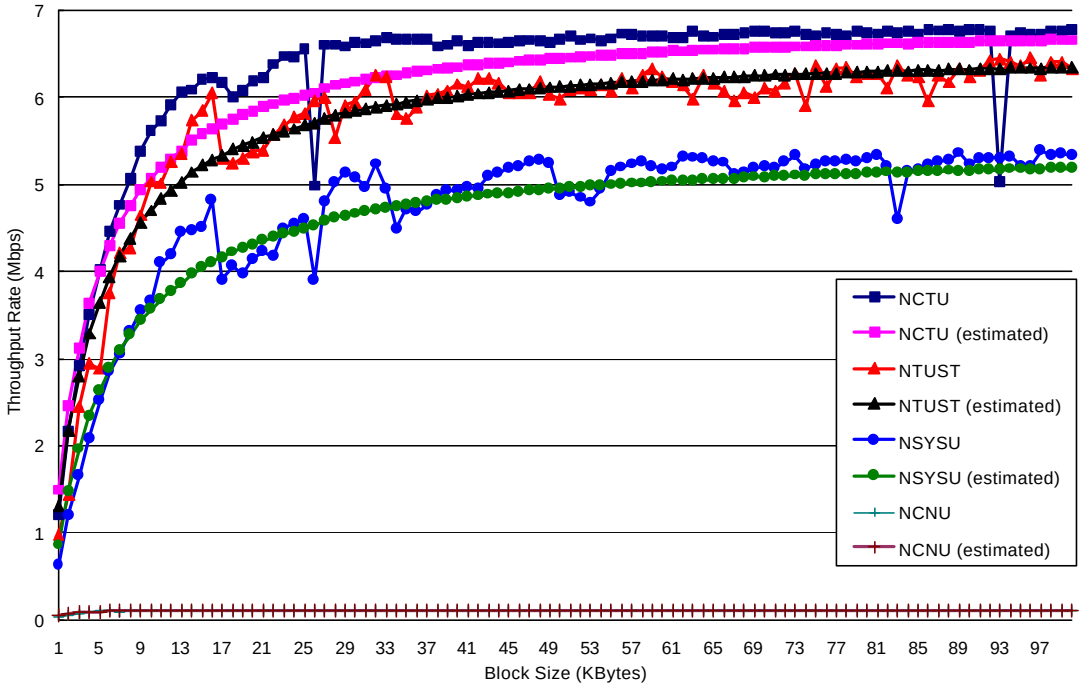

圖 6-6 / 預估的傳輸效能與實測結果比較

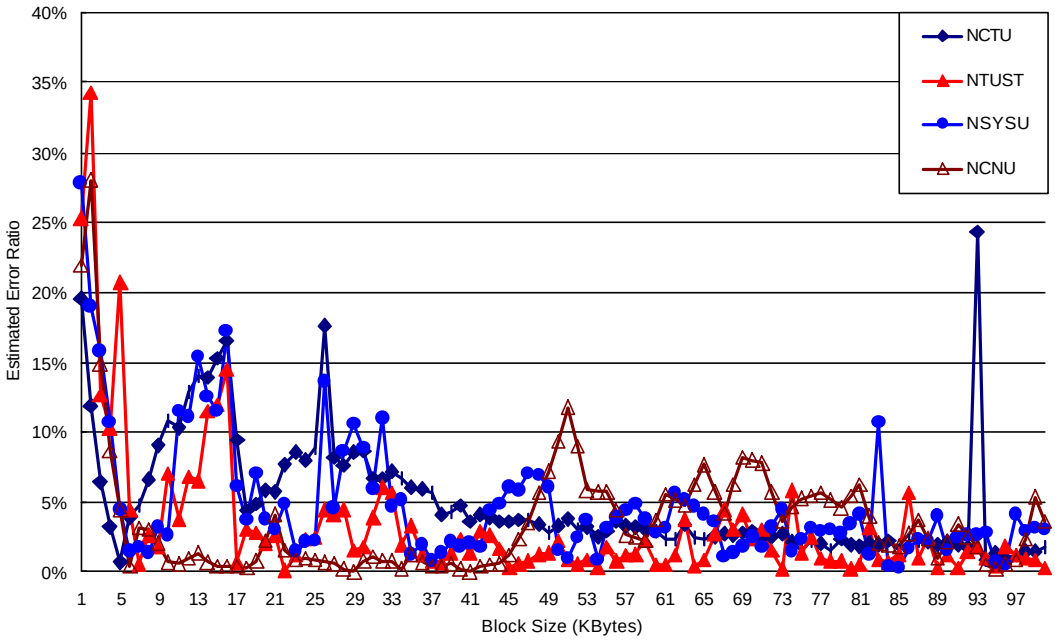

圖 6-7 / 預估的傳輸效能與實測結果的誤差率

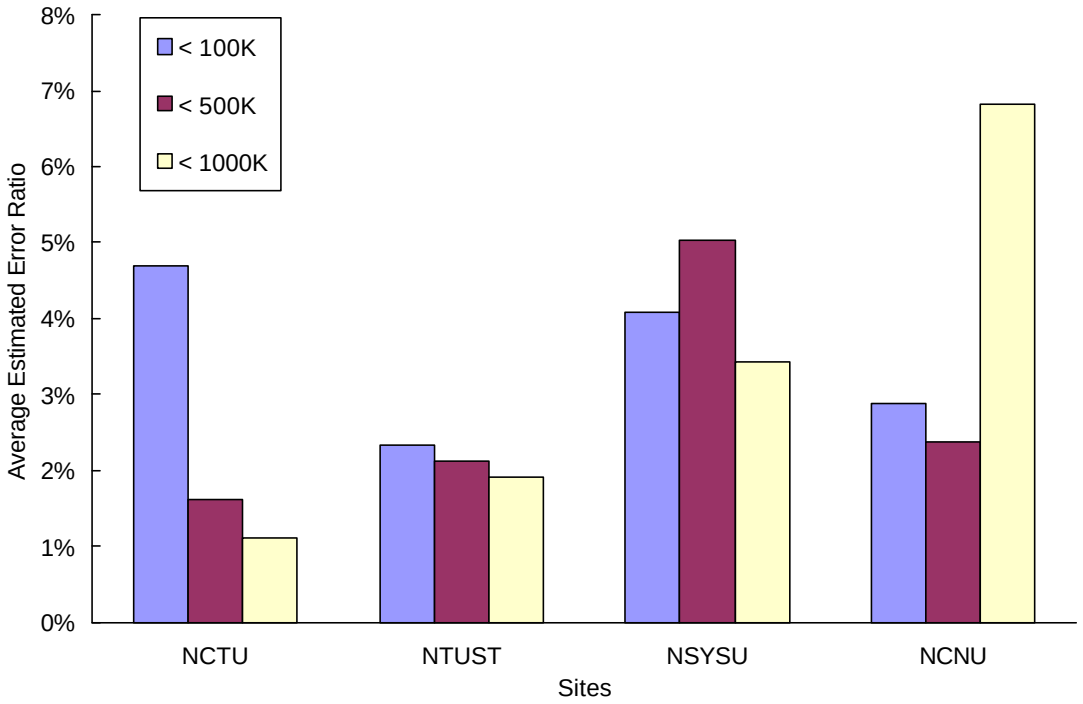

圖 6-8 / 預估的傳輸效能與實測結果的於不同區塊尺寸的平均誤差率

與實測結果相較驗證，可以發現在大部分的情形中，誤差率都小於 5%，由此可證明我們所推導的模型是合於實際情況的。

6-4 平行存取技術的區塊尺寸－下載時間模型

6-2 節中，我們已成功地推導區塊尺寸－下載時間模型，但僅限於單一站台、單一連線，將區塊循序式一個個從目的站台下載。而平行存取技術的行為是，同時與多個站台連線，同時從各個站台下載區塊資料。我們將 6-3 小節所建立的模型稍作修改，希望能將此模型推廣到平行存取技術上。

建立使用平行存取技術下載時間模型，必須定義及修改下列參數：

rtt	各連線路徑的封包來回傳遞時間平均值
bw_{avail}	各連線路徑的可用頻寬總和
mss	各連線路徑的 TCP segment 的最大長度平均值
muws	各連線路徑的最大的可用視窗尺寸平均值
$sites_{num}$	同時進行連線，下載同一個檔案的映射站台數目

為簡化模型，也因為區塊的分發情形無法準確地預估，我們假設所有區塊將平均分配給各個站台下載，因此各種網路特性均以平均值或總和來做計算。

以平行存取技術進行檔案下載花費的時間可大致分為兩部分：所有站台同時運作，全速下載資料的時間，我們稱此段時間為理想下載時間；以及某些站台已經完成工作，但仍有某些站台負責的區塊仍然未傳送完畢，所以必須等待，待到所有區塊傳送結束，才算完成檔案下載的動作，這段時間稱為結束等待時間。

接下來我們分別討論理想下載時間及結束等待時間的分析及預估。

- **理想下載時間**

多個站台同時進行連線，與單一站台連線的情況的異同是：

□若各映射站台未被同一個頻寬瓶頸限制住，通常可用頻寬會增加，可以大幅增快下載時間。

□區塊數目不變，但是封包數目由於各路徑 *mss* 值的不同，可能有些許增減。由於我們假設所有區塊平均分配給各個站台，所以以 *mss* 平均值來計算封包及 HTTP 標頭 overhead。

□工作量分攤的緣故，每個站台必須傳送的區塊數目減少，超出伺服器容許連線上限次數的機會降低，但是整體的連線數目並不會減少。

$$connection_{num} = \left\lceil \frac{block_{num} / \boldsymbol{sites_{num}}}{100} \right\rceil \times \boldsymbol{sites_{num}}$$

□在單一連線情形，每次閒置的狀態會造成額外的半個閒置時間浪費；不過在多站台連線的情形，假設每條連線皆進入閒置狀態 *n* 次，因為閒置狀態浪費的時間是針對某條連線而言，而連線同時進行，浪費的時間與單一連線進入閒置狀態 *n* 次是一樣的。

綜合以上的修正，我們可得到以下的運算式：

$$transfer\_time = \frac{filesize}{bw_{avail}} + \frac{idletimes}{2} \Big/ \boldsymbol{sites_{num}} + network\_header_{overhead} + http\_header_{overhead} + connection_{num} / \boldsymbol{sites_{num}} \times \left(connection_{setup} + slowstart_{overhead}\right)$$

下載時間的預估運算式做了兩個修正，一為閒置時間的計算，另一為 TCP **連線** overhead **的計算。這兩個修正意思上是相同的，閒置時間及** TCP **連線建立、**slow start **機制** overhead **計算的是浪費在某條** TCP **連線的時間，而不是現實世界的時間，因共有** $sites_{num}$ **條連線，所以這兩種** overhead **皆須除以** $sites_{num}$。

同樣的，我們以實際量測來驗證平行存取技術的區塊尺寸－下載時間模型的正確性及可用性。我們選定五個映射站台，每次隨機取出兩個、三個、四個、五個站台來進行各種檔案長度、各種區塊尺寸的平行存取檔案下載，經過分析統計，得到如圖 6-9、6-10 **的結果。**

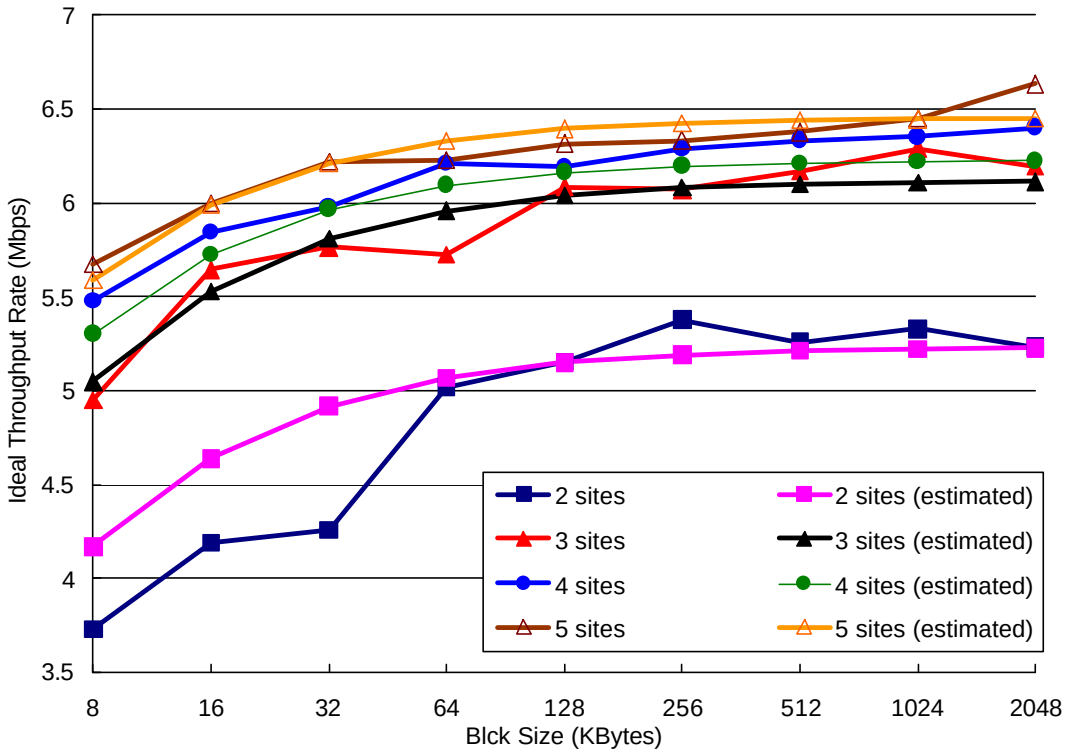

圖 6-9 / 理想傳輸效率與站台數目、區塊尺寸的關係，以及預估的理想傳輸效率（檔案長度為 16.6 MB）

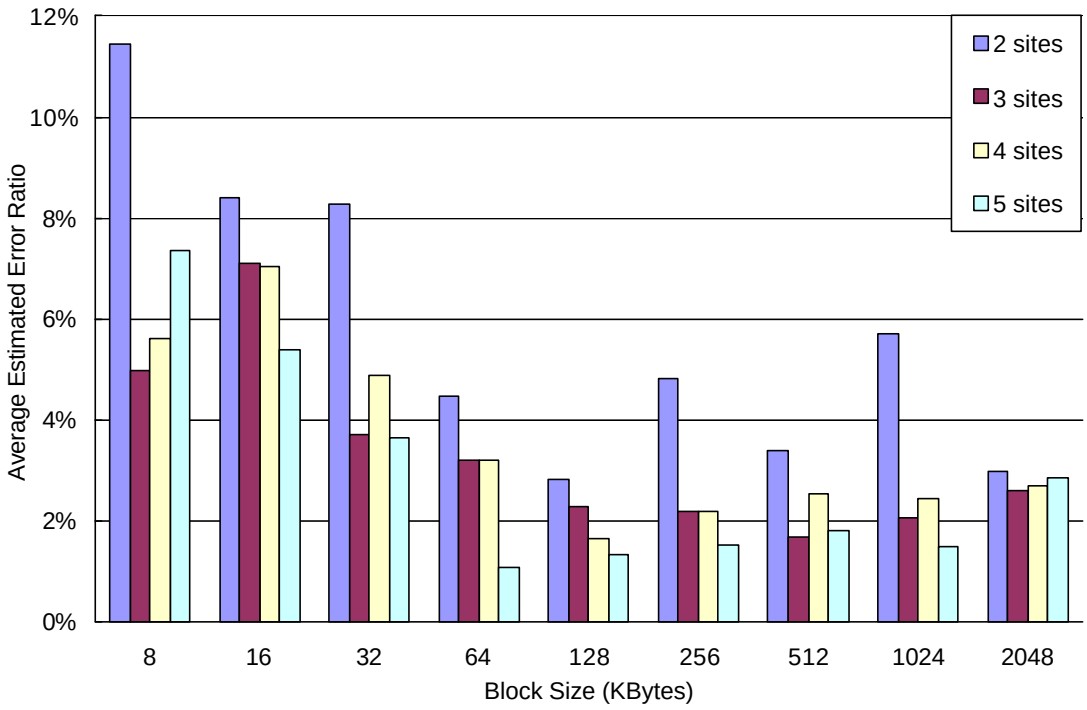

圖 6-10 / 預估的理想傳輸效率誤差與站台數目、區塊尺寸的關係

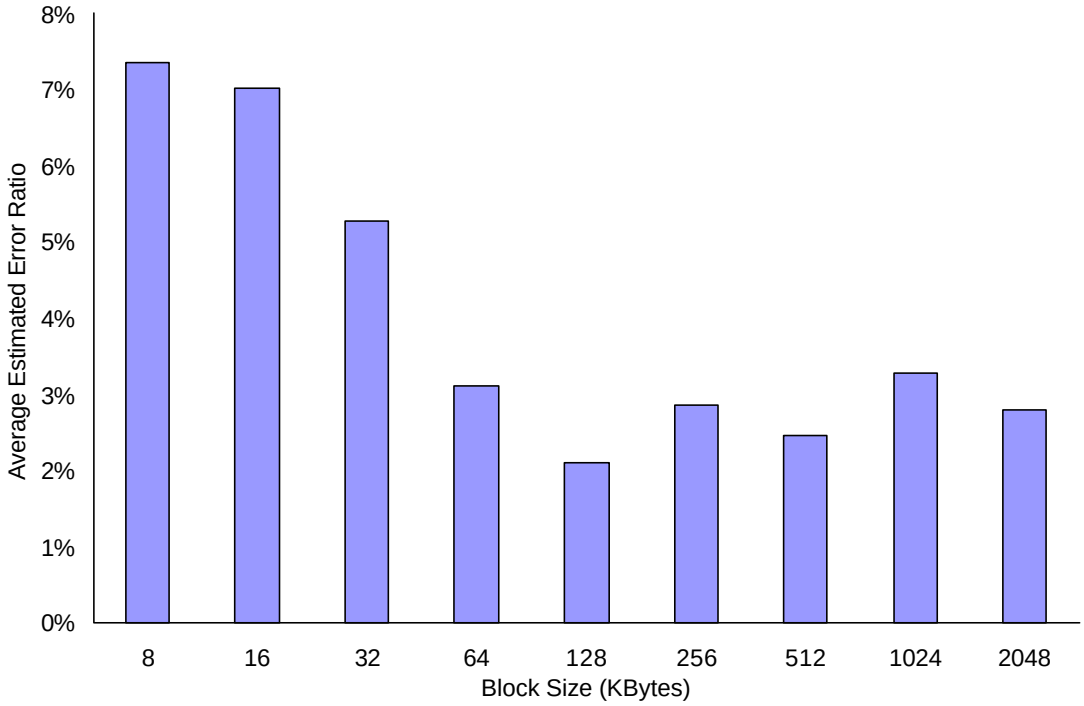

圖 6-11 / 區塊尺寸與理想傳輸效率平均誤差比率關係

平均而言，只要區塊尺寸為 32BK **或以上，不論檔案長度、連線站台數目，誤差率都小於** 5%。**而在實際應用上，區塊尺寸通常遠大於** 32KB，**所以此模型所估算出來的傳輸率也是十分值得參考的。**

- **結束等待時間**

在最理想的狀況下，所有站台同時完成資料的傳送，結束等待時間為零，檔案下載時間等於理想下載時間。不過這是少之又少的情況，通常站台會先後結束區塊傳送工作，使得檔案傳輸時間與理想下載時間有些差異，拉下平均傳輸速率。

在 4-1 **節中，我們已詳細介紹平行存取技術的流程。當所有區塊不是已經下載完成就是正在下載，當某個站台完成工作，在結束連線之前，會視其它下載中站台的下載情況（速率、剩餘長度、預估剩餘時間），決定是否幫忙，同時下載同一個區塊，希望比原負責該區塊的站台更早完成；或是將該站台的下載工作搶過來做。**

由 4-2 **節，我們可知結束等待時間大略與區塊尺寸、可用頻寬總和成正變，但由**

於網路狀況的不可預測性，我們很難估算準確的結束等待時間，能夠推導的只有，在網路狀況正常情況下，結束等待時間的上限值。

由於在結束等待時期，我們的演算法會盡量讓速度最高的站台負責最後幾個區塊的傳遞，讓速度較慢的站台閒置。我們定義最高的站台可用頻寬為 $bw_{fastest}$，以它來計算最後區塊的傳輸時間。

在最差的情況下，$sites_{num}-1$ 個站台已經閒置下來，並且中止連線，只剩下一個站台正在傳輸最後一個區塊，而且才正開始此區塊的傳輸動作。不過由於這種情形太極端，反而無法逼近實際的結束等待時間，因此我們假設當結束等待時期開始時，最後一個區塊已經傳送一半資料。這最後區塊的傳輸時間的計算方式為：

$$transfer\_time_{lastblock} = \frac{blocksize/2}{bw_{fastest}} + \frac{rtt}{2} - \frac{blocksize/sites_{num}}{bw_{avail}}$$

上式中，最後區塊的傳輸時間為半個區塊資料的傳輸時間加上 HTTP 請求時所造成一半的閒置時間浪費，再減掉此半個區塊原本就包含於理想下載時間內的傳輸時間。在此我們不考慮最極端的情況：若負責此最後區塊的站台正好在上一次或前幾次的區塊請求時被伺服器中止連線，此時就必須再考慮 TCP 連線建立時間及 slow start 機制對此區塊的傳輸時間影響。

圖 6-12 及 6-13 為平均結束等待時間與區塊尺寸關係圖。

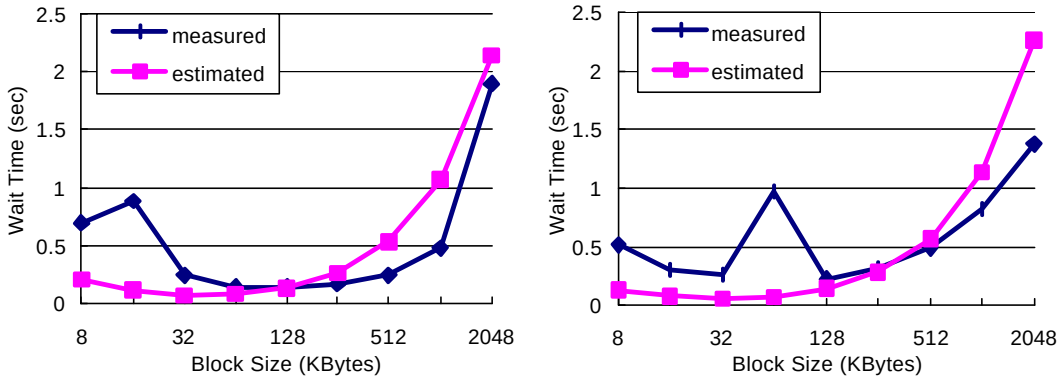

圖 6-12 / 兩個、三個連線站台的平均結束等待時間與區塊尺寸關係

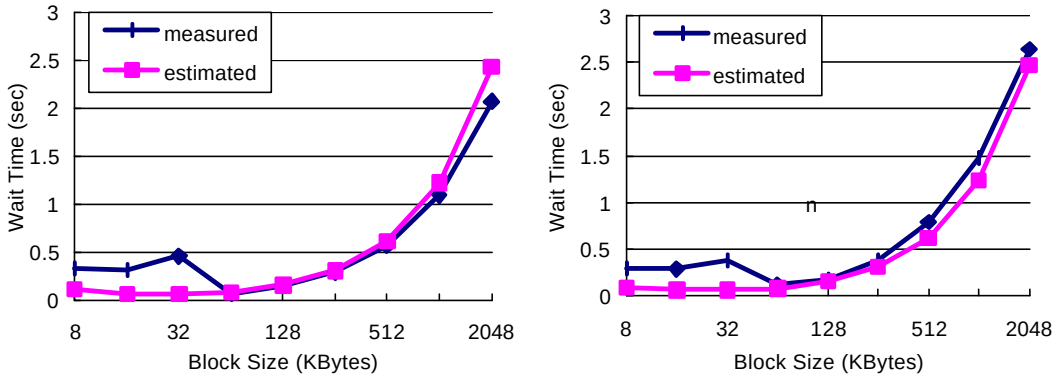

圖 6-13 / 四個、五個連線站台的平均結束等待時間與區塊尺寸關係

事實上，由於平行存取技術對於減少結束等待時間已做了不少努力，所以結束等待時間通常很短。在我們的實驗過程中，對於 16.6MB 的檔案，所用站台的可用頻寬總和約 8MB，結束等待時間絕大多數不超過一秒鐘，當區塊尺寸非常大（如 2MB）時，平行存取技術依舊表現良好，結束等待時間也甚少超過 5 秒鐘。

- **檔案下載時間**

將理想下載加上結束等待時間加總，就可得到檔案下載時間。對於相同的映射站台，同樣的檔案尺寸，理想下載時間通常與區塊尺寸成反變，而結束等待時間通常與區塊尺寸成正變，因此必須將兩種時間加總，才能得到最理想的區塊尺寸。

同樣的，我們以實驗驗證模型的正確性，圖 6-14 6-16 為實驗與估算值對照結果。

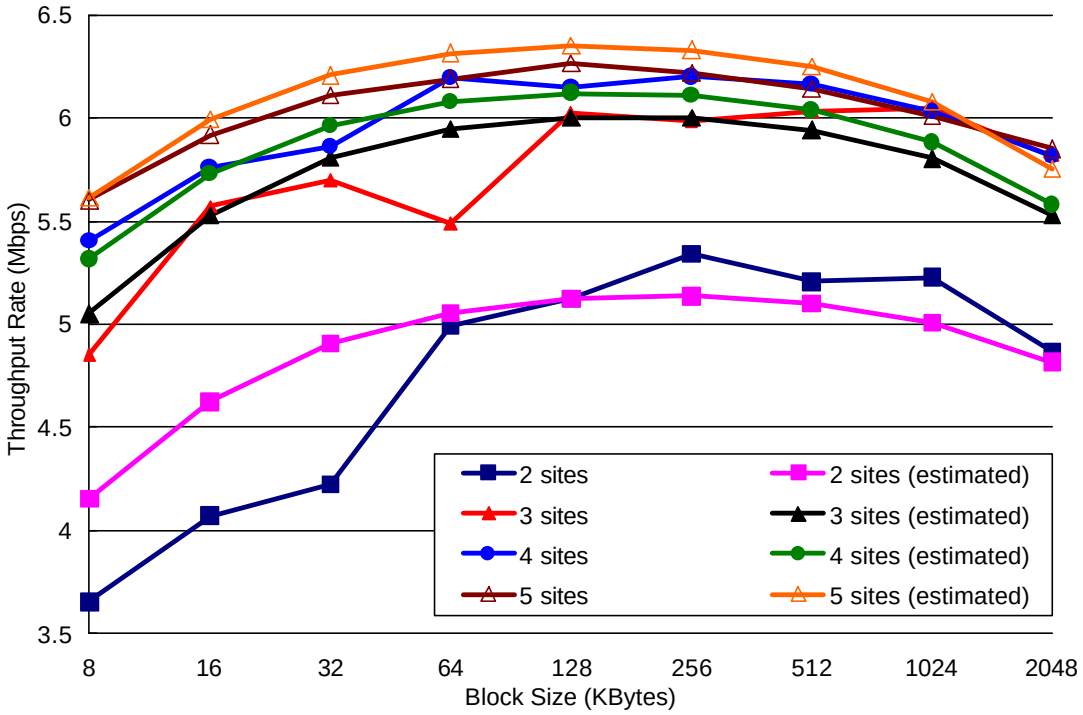

圖 6-14 / 傳輸效率與站台數目、區塊尺寸的關係，以及預估的傳輸效率（檔案長度為 16.6 MB）

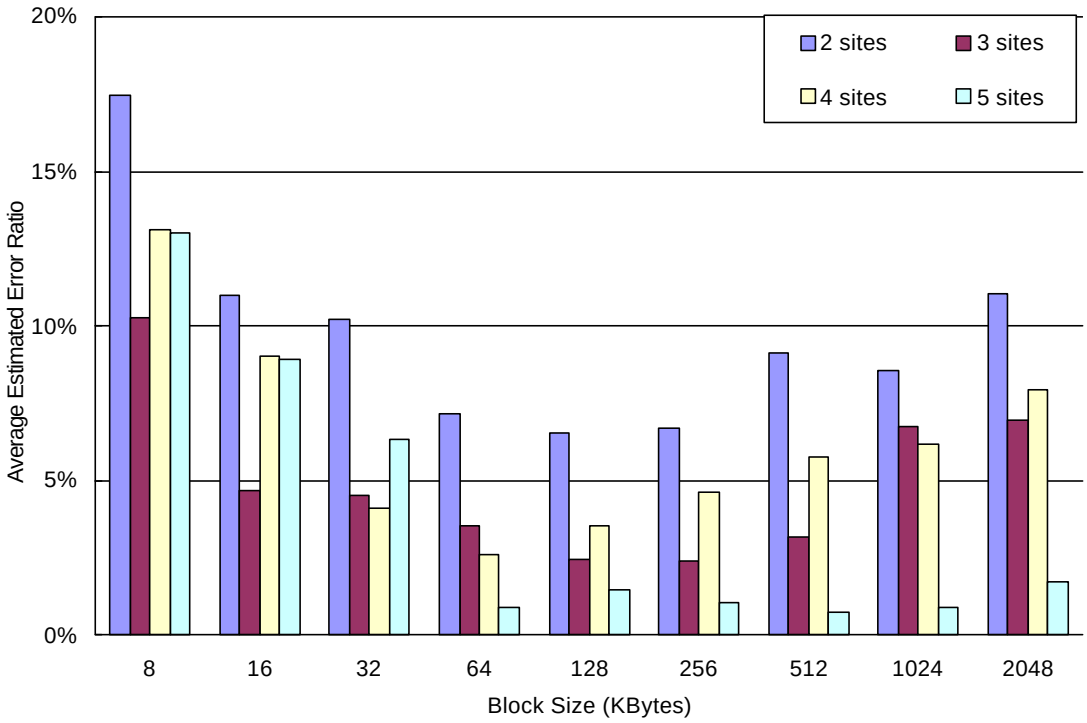

圖 6-15 / 預估的傳輸效率誤差與站台數目、區塊尺寸的關係

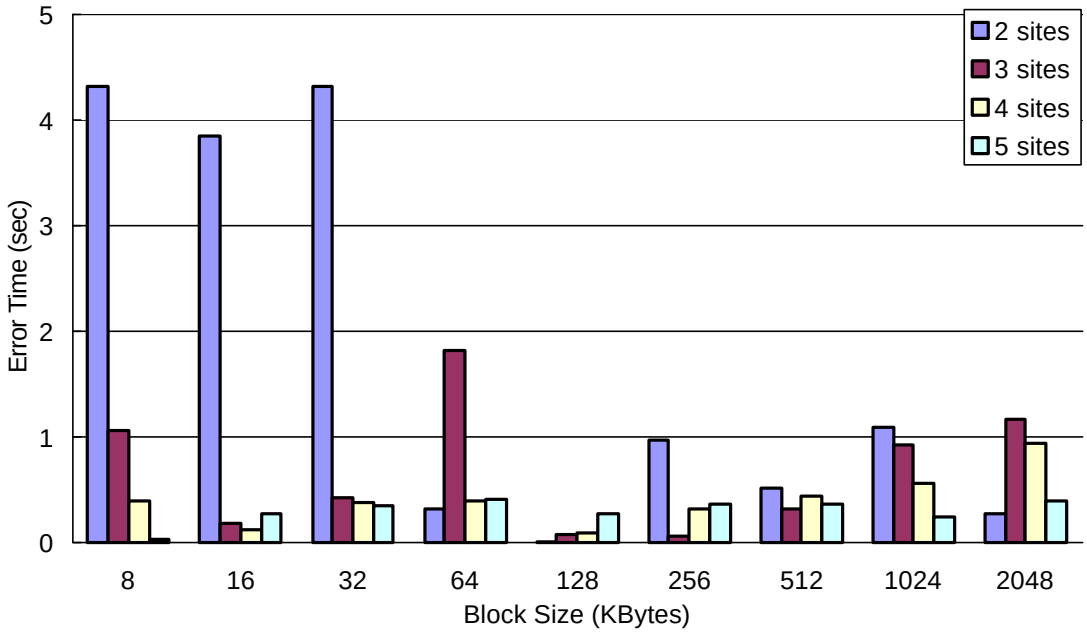

圖 6-16 / 傳輸時間預估誤差與站台數目、區塊尺寸的關係
（檔案長度為 16.6 MB）

6-5 動態平行存取技術

根據前面幾個小節所建立的模型，我們改良平行存取技術，發展出動態平行存取技術。動態平行存取技術是基於平行存取技術的區塊尺寸－下載時間模型而發展的，它會動態地根據各站台的連線及下載情形以時間預估演算法來計算最佳的區塊尺寸，企圖在各種情況下皆能得到合理的下載速率。

動態平行存取技術的改良之處為：

1. 在開始進行下載的同時，測量每個站台的封包來回傳遞時間。

- 記錄每個站台每個區塊的下載速率，以平均值做為每個站台連線路徑的可用頻寬。
- 下載開始時，向每個站台要求預設區塊尺寸的區塊資料。在所有站台成功取回它們的第一個區塊前，皆使用預設區塊尺寸向伺服器要求資料。
- 所有站台皆下載完成它們的第一個區塊後，每次站台完成區塊傳送，欲送出下一個請求前，先計算每個站台的平均封包來回傳遞時間及可用頻寬，求得

預估下載時間最少的區塊尺寸，再依此區塊尺寸向伺服器要求下載區塊。

在我們的評估實驗中，預設區塊尺寸設定為 64KB，可接受的最小區塊尺寸設定為為 32KB。分別從二、三、四、五個映射站台下載 64 KB、128 KB、256 KB、512 KB、1 MB、2 MB、4 MB、16 MB 等等各種尺寸的檔案，我們分別以動態平行存取技術，以及各種區塊尺寸的平行存取技進行比較。

圖 6-17 為下載各種尺寸檔案時，動態平行存取技術的平均表現，除了網路狀況不佳所造成的特例外，平均都能達到理想下載速率的九成。

圖 6-18、6-19 分別為各個時段動態平行存取技術的表現及固定尺寸平行存取技術在每個時段的最佳區塊尺寸的表現。相互比較的結果，若檔案大小超過一個程度（約 2　4 MB），固定尺寸平行存取技術可以得到較佳的結果，達到 3%以下的理想速率表現；而對於長度較小的檔案，表現較不穩定，時好時壞。

而動態平行存取技術在大檔案的表現雖然稍遜於固定尺寸平行存取技術的較佳結果，但各檔案尺寸的表現較平均，除了網路狀況不佳所造成的特例外，誤差比率都約在 15% 以內，換句話說，傳輸效率大約都在理想下載速率的 85% 以上。

而動態平行存取技術的最大優點是，可以省去選擇最佳區塊尺寸的麻煩，交由演算法根據網路情況決定。所以我們認為，動態平行存取技術在現實生活中有應用的價值。

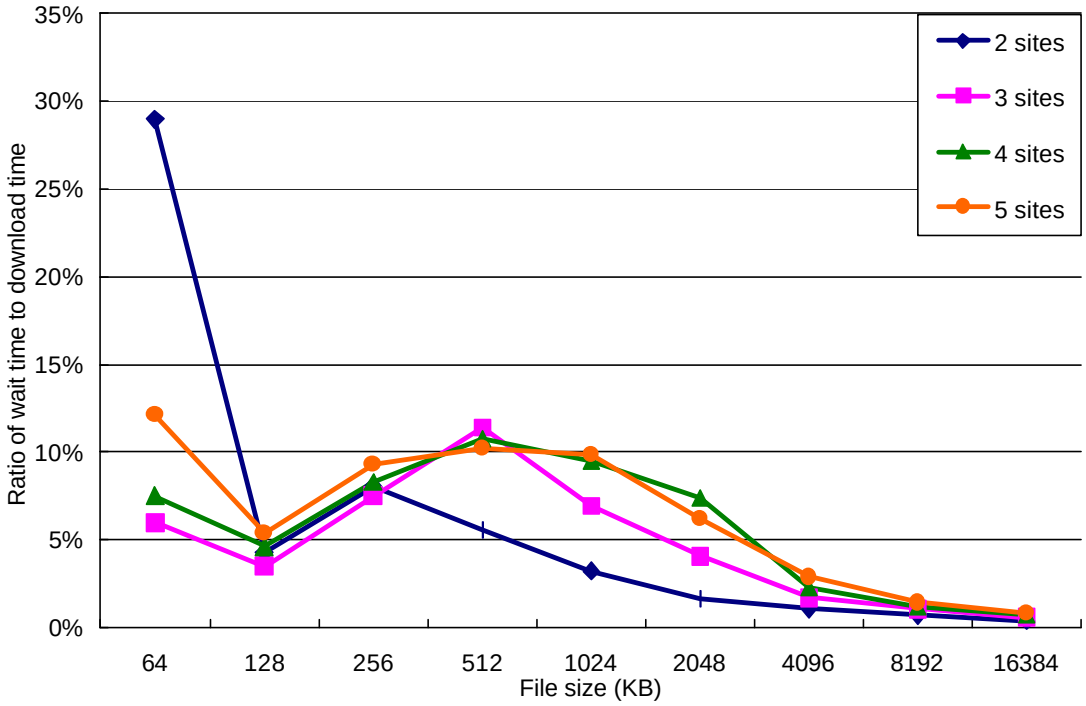

圖 6-17 / 動態平行存取技術的結束等待時間－整體下載時間比率與檔案尺寸的關係

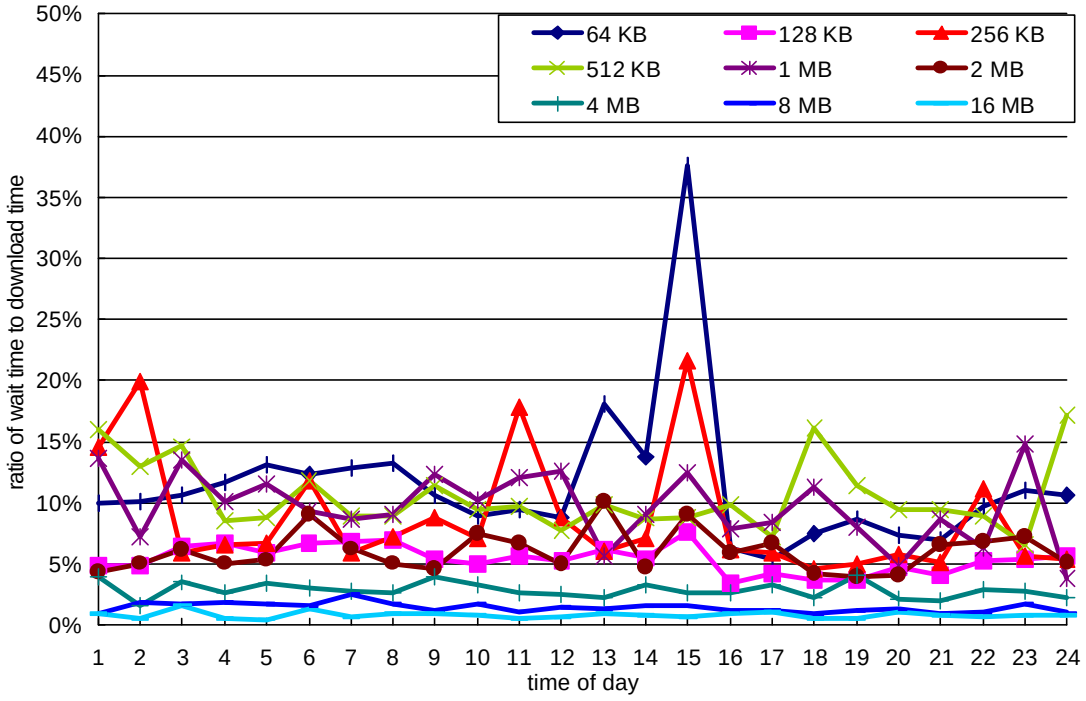

圖 6-18 / 各時段動態平行存取技術的結束等待時間－整體下載時間比率（5 個映射站台）

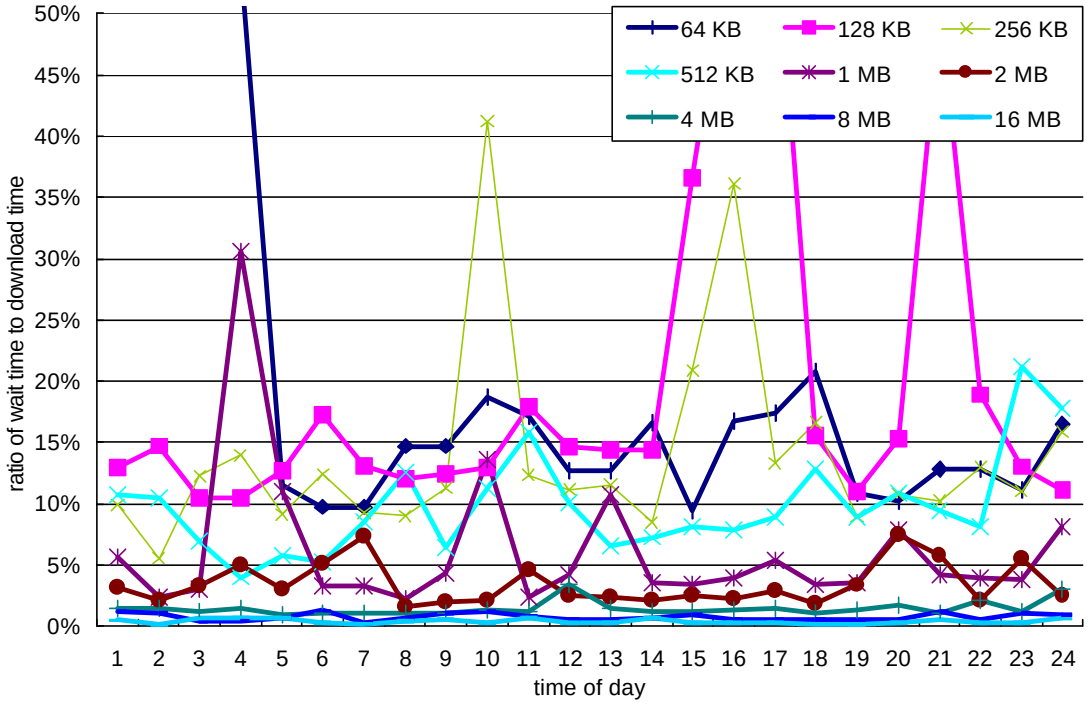

圖 6-19 / 各時段平行存取技術的最佳區塊尺寸的結束等待時間－整體下載時間比率（5 個映射站台）

第七章

結論及未來展望

目前網際網路上早有許許多多的文件及檔案被重覆地放置在不同的網站中，我們建立儘量使用平行存取技術來下載這些檔案。平行存取技術可以大幅地減少檔案下載時間、分散伺服器的負載以及避免困難的站台選擇問題，且不會對網路帶來額外的負擔。

我們發展出動態平行存取技術，動態平行存取技術可以在各種站台數目、連線情形、檔案尺寸等情況下，自動調整區塊尺寸，達到與理想下載速率相近的表現。

未來，我們希望在下列幾個方向上繼續努力：

- □改善動態平行存取技術的下載時間估算模型，我們希望能得到更好的預估準確度。
- □面對眾多的映射站台，往往只需要取連線情況最佳的前幾名，就可獲得十分接近理想值的下載速率。若同時與太多的站台進行連線，一來佔用用戶端主機的資源，二來也使得區塊的分派更困難，更容易導致額外頻寬的浪費。我們希望進行目的站台選擇的研究，發展快速準確的頻選擇演算法，讓動態平行存取技術能夠自動地選擇表現最佳的幾個站台來進行下載。
- □對於任一個檔案，目前並沒有方便的、完整的且廣為支援的映射站台資訊取得方法，我們希望發展這方面的技術，讓動態平行存取技術更有發揮的空間。

參考文獻

[Agui 84] L. Aguilar. “Datagram routing for Internet multicasting”, *SIGCOMM’ 84,*

pp 58 – 63. ACM, Jun. 1984.

[Amma 97] Russell Clark and Ammar, Mostafa. " Providing Scalable Web Service Using Multicast Delivery ", *Computer Networks and ISDN Systems Journal*, Vol. 29, pp841-858, 1997.

[Apac] "Apache Web Server", *http://www.apache.org/*.

[Arli 96] M. F. Arlitt, C. L. Williamson, "Web server workload characterization: The search for invariants", in *Proc. ACM SIGMETRICS*, May 1996, pp.126-137.

[Bern 95] J Bernabeu, M. Ammar, and M Ahamad, "Optimizing a generalized polling protocol for resource finding over a multiple access channel," *Computer Networks and ISDN Systems*, 1995.

[Bhat 97] Samrat Bhattacharjee, Mostafa H. Ammar, Ellen W.Zegura, Viren Shah, Zongming Fei, "Application-Layer Anycasting", *INFOCOMM97*, Japan, April, 1997.

[Brad 92] R. Braden, "Extending TCP for transactions - Concepts", *RFC 1379*, Nov. 1992.

[Brad 94] R. Braden, "T/TCP - TCP extensions for transactions functional specification", *RFC 1644*, Jul. 1994.

[Byer 99] John Byers, Michael Luby, and Michael Mitzenmacher, "Accessing multiple mirror sites in parallel: Using tornado codes to speed up downloads," in *INFOCOM 99*, Apr. 1999.

[Calf 94] K.C. Claffy and H. Braun. “Web traffic characerization: An assessment of the impact of caching documents from NCSA’s Web server”. In *Second World Wide Web Conference’ 94*. http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings, Oct. 1994

[CNet] "CNET DOWNLOAD.COM", *http://download.cnet.com/*.

[Cens] Censorware Project, "Size of the web: A dynamic essay for a dynamic medium", *http://censorware.org/web_size/*

[Crov 97] Mark Crovella and Robert Carter, "Dynamic server selection using bandwidth probing in wide-area networks," in *Proceedings of IEEE INFOCOMM*, 1997.

[Cunh 95] C.Cunha, A.Bestavros, and M.Crovella, "Characteristics of WWW client-based traces", *Tech. Rep. 95-010*, Boston Univ., Apr.1995

[Fan 98] Li Fan, Pei Cao, Jussara Almeida, and Andrei Broder, "Summary cache: A scalable wide-area web cache sharing protocol," Feb 1998, *SIG-COMM'98*.

[Fiel 97] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, T. Berners-Lee, et al., "RFC 2068: Hypertext transfer protocol – HTTP/1.1," Jan. 1997.

[Frys 96] H. Frystyk, R. Fielding, T. Berners-Lee, et al., "RFC 1945: Hypertext transfer protocol - HTTP/1.0," May. 1996.

[Gadd 97] Syam Gadde, Michael Rabinovich, and Jeff Chase, "Reduce, reuse, recycle: An approach to building large internet caches," in *The Sixth Workshop on Hot Topics in Operating Systems (HotOS-VI)*, May 1997.

[Heid 97] John Heidemann, Katia Obraczka, Joe Touch, "Modeling the Performance of HTTP Over Several Transport Protocols", Jun. 1997.

[Jaco 88] Van Jacobson, "Congestion Avoidance and Control", *SIGCOMM*, 1988.

[Jaco 90] Van Jacobson, "Modified TCP Congestion Avoidance Algorithm", April 30, 1990, *end2end interest mailing list*

[Kang 99] J. Kangasharju, K. W. Ross, and J.W. Roberts, "Locating copies of objects using the domain name system," in *Proceedings of the 4th International Caching Workshop*, San Diego, March 1999.

[Lawr 00] Dr. Lawrence G. Roberts, "Internet Growth Trends", *IEEE Computer, Internet Watch*, Jan. 2000

[Luby 97] M. Luby et al., "Practical loss-resilient codes," in *STOC*, 1997.

[Luig 97] Luigi Rizzo, "Effective erasure codes for reliable computer communication protocols," *Computer Communication Review*, vol. 27, no. 2, pp.24-36, April 1997.

[Lyco] http://ftpsearch.lycos.com/.

[Mehm 98] Mehmet Sayal, Yuri Breibart, Peter Scheuermann, and Radek Vingralek, "Selection algorithm for replicated web servers," in *Workshop on Internet Server Performance, SIGMETRICS*, Madison, USA, June 1998.

[Mogu 95] Jeffrey C.Mogul, "The Case for Persistent-Connection HTTP", *SIGCOMM 95* Cambridge, MA USA

[NTHU] "NTHU Computer Science Dept.", *http://www.cs.nthu.edu.tw/*.

[Neum 92] B. C. Neuman, "The virtual system model: A scalable approach to organizing large system", Ph.D. dissertation, Univ. Washington, Seattle, 1992.

[Pabl 00] Pablo Rodriguez, Andreas Kirpal, Ernst W.Biersack, "Parallel-Access for Mirror Sites in the Internet" in *Proceedings of IEEE INFOCOM*, March 2000.

[Padm 96] Venkata N.Padmanabhan, Jeffrey C.Mogul, "Improving HTTP Latency"

[Post 85] J. Postel, J. Reynolds, "RFC 959: File Transfer Protocol", Oct. 1985.

[Rich 94] W. Richard Stevens, "TCP/IP Illustrated, Volume 1", pp.289-291.

[Rous 98] Alex Rousskov and Duane Wessels, "Cache digest," in *3rd International WWW Caching Workshop*, June 1998.

[Sesh 97] S. Sesham, M. Stemm, and R. Katz, "Spand: Shared passive network

performance discovery," in *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, December

[Sper 95] E. E. Spero, "Analysis of HTTP performance problems," *http://sunsite.unc.edu/mdma-release/http-prob.html*, 1995

[Touc 95] J. Touch, "Defining 'high speed' protocols: Five challenges and an example that survives the challenges", *IEEE J. Select. Areas Commun.*,vol.13, pp.828-835, Jun. 1995

[Vern 97] Vern Paxson, "Measurements and Analysis of End-to-End Internet Dynamics", Ph.D. Thesis, Apr. 1997.

[Zong 98] Zongming Fei, S. Bhattacharjee, Ellen W. Zegura, and Mostafa H. Ammar, "A novel server selection technique for improving the response time of a replicated service," in *Proceedings of IEEE INFOCOM*, San Francisco, CA, USA, March 1998

[wget] wget, *ftp://gnjilux.cc.fer.hr/pub/unix/util/wget/*.