國立交通大學

資訊學院資訊科技（IT）產業研發碩士班

碩 士 論 文

具容錯性之使用點對點傳輸技術

巨量檔案發佈系統

The Study of Massive Deployment System Based on P2P Technology

with Fault Tolerance

研 究 生：詹益嘉

指導教授：吳毅成　教授

中 華 民 國 九 十 八 年 十 月

具容錯性之使用點對點傳輸技術巨量檔案發佈系統

The Study of Massive Deployment System Based on P2P Technology with Fault Tolerance

研 究 生：詹益嘉　　Student：Yi-Jia Zhan

指導教授：吳毅成　　Advisor：I-Chen Wu

國 立 交 通 大 學

資訊學院資訊科技（IT）產業研發碩士班

碩 士 論 文

A Thesis

Submitted to College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of Master

in

Industrial Technology R & D Master Program on

Computer Science and Engineering

Oct. 2009

Hsinchu, Taiwan, Republic of China

中華民國九十八年十月

具容錯性之使用點對點傳輸技術巨量發佈系統

研究生：詹益嘉　　指導教授：吳毅成

國立交通大學資訊學院產業研發碩士班

摘要

本篇論文以一個使用點對點傳輸技術之巨量傳輸系統[1] 為基礎，在協助企業節省網路頻寬、提升客戶下載體驗之外，針對提升系統容錯能力為主題，提供企業用戶一個更完整穩定的檔案發佈服務能力。

此系統命名為具容錯性之巨量檔案發布系統 (Fault-Tolerant Massive Deployment System, 簡稱為 FT-MDS)，主要伺服端元件有 FT-MDS Server 與 Super Seeder 以及資料庫，而在客戶端元件為 FT-MDS Client。本論文將針對伺服器端的容錯性設計，如伺服器端的主要元件 FT-MDS Server 意外斷線以及 Super Seeder 突然不能提供服務，改進這些問題而進行詳細研究探討。

The Study of Massive Deployment System Based on P2P Technology with Fault Tolerance

Student: Yi-Jia Zhan Advisor: I-Chen Wu

Industrial Technology R & D Master on
Computer Science College
National Chiao Tung University

Abstract

This paper describes a new architecture to improve the ability of fault tolerance in Massive Deployment System[1], which can provide a stable data download service. This new architecture system is called Fault-Tolerant Massive Deployment System, abbr. as FT-MDS. We want to solve some fatal problem which likes the MDS Server may be broken or lost all connection to Super Seeder and MDS Client. This paper focuses on the design of fault tolerance.

誌謝

這篇論文的完成要感謝非常多人的幫忙以及相挺。首先，我要感謝我的指導教授，吳毅成教授，是一位願意付出在學生時間比自己還多的人，老師的諄諄教誨，教我什麼是做研究。

我要感謝我的家人，全力的支持我在服完兵役後，繼續就讀研究所，以及我弟詹承浩的支持，跟我一起學習研究。我要感謝 P2P 組的各位，博班學長單益章、碩士班學長哲毅、鼎量、以及我的夥伴吳冠翬，在我遇到困難以及瓶頸的時候指導我以及告訴我很多需要痛到才學得到的知識。感謝兩位博班學長林秉宏及孫德中，時常給我一些中懇的建議。接著要感謝是我 96 級的各位同學：靖平、暐倫、宏軒、宜智、柏甫。我們一起走過熬夜寫作業，看日出的水深火熱碩士生活，我永遠不會忘記。感謝實驗室的各位學弟妹，郁雯、彥成、俊嶧、家茵、柏廷、忻芸，讓這七百多天的忙到不可開交的日子多了許多的歡笑及快樂回憶，還有感謝正宏學弟帶領我的健康生活快樂回憶，感謝各位。感謝鈊象電子股份有限公司獎助金，讓我能夠更專心於學業。

最後，謹以此論文獻給各位在我學習之路給予幫助以及勉勵的親朋好友們。

中華民國 98 年 10 月 於 國立交通大學工程三館 511 實驗室

目錄

圖目錄

表目錄

第一章、緒論

在本章會說明目前線上遊戲的發展，以及網路傳輸技術對於線上遊戲的影響，並且說明 P2P 傳輸技術用於遊戲檔案傳輸的應用，而接著會提出本篇論文的研究動機與目的，最後會對於本篇論文的大綱加以說明。

1.1 研究動機與目的

隨著網際網路的普及，根據財團法人台灣網路資訊中心 (TWNIC) 2009 年台灣寬頻網路使用調查報告[2] 指出，台灣總上網人口已經超過一千五百萬人，這約佔台灣總人口的百分之七十。並且在上網人口中約有三百五十萬人是網路遊戲的使用者。線上遊戲人口已經佔上網人口的百分之二十三，是網際網路相當熱門的應用。

而對於網路遊戲公司而言，良好的遊戲體驗除了遊戲本身的內容，還有伺服器穩定的提供服務。遊戲本身的內容部分，國內已經有相當多的業者投入，如鈊象電子、群想網路科技、以及宇峻奧汀等。而在伺服器穩定的提供服務部分，由於很多線上遊戲廠商仍採用傳統 Client/Server 架構的緣故，以 File Transfer Protocol (FTP)或是 Hypertext Transfer Protocol (HTTP)來提供檔案下載，而採用傳統

Client/Server 架構的公司，都會面臨到新開發遊戲檔案釋出或是遊戲檔案更新的時候，造成伺服器的瞬間負荷升高，網路的輸出頻寬也會瞬間的暴衝，而可能影響到公司的其他服務。面對這個問題，過去提出了使用點對點傳輸 (Peer-to-Peer)技術為基礎的巨量檔案發佈系統[1][3] ，藉著每位遊戲玩家貢獻一定比例的上傳頻寬，加速檔案的傳播以及分享，使得網路遊戲公司伺服器端輸出頻寬減少一定的比例，並且降低伺服器負載和頻寬的使用。而過去的系統還有一些問題需要克服，如增加系統的彈性以及可用性，讓系統可以更自動化的應付系統的意外錯誤。

本篇論文的目的在基礎於使用點對點傳輸技術巨量檔案發佈系統[1][3] 之上，提出容錯式架構。如過去所遇到的伺服器端有時會意外的發生斷線或無預警的停止提供服務，此時由於過去的巨量檔案傳播系統只有一套的伺服器，所以只要任何一個伺服端的元件無法提供服務時，整個系統將失去功能。而本篇論文的重點除了使用點對點傳輸技術的檔案發佈方式讓檔案可以快速傳播到可戶端之外，並且解決以上問題而穩定的提供檔案下載服務，使得除了滿足節省頻寬之外，也讓企業佈署、應用更加的具有彈性及便利。

1.2 論文章節說明

本論文將會在第二章介紹點對點傳輸技術相關的背景、目前發展的現況，以及使用點對點傳輸技術之巨量檔案傳輸系統；而第三章將會提出重新設計的具容錯性之巨量檔案傳輸系統 (FT-MDS)如何設計以及運作方式；第四章則會對於系統實作方面做說明，以及遇到問題的克服；第五章會說明實驗的設計以及結果的分析；最後，在第六章提出結論以及未來的展望。

第二章、背景說明

本章將會介紹目前關於點對點 (Peer-to-Peer)傳輸技術和相關應用，最後會介紹使用點對點傳輸技術之巨量檔案傳輸系統。

2.1 點對點 (Peer-to-Peer)傳輸技術

P2P (Peer-to-Peer) 是一種新的網路傳輸技術，藉由使用在 P2P 網路中客戶端的頻寬與計算能力，而非依賴少數幾台伺服器的服務，來快速完成目的。

剛開始的 P2P 技術設計並非為中央式，也就是不需中央伺服器，網路上每個客戶端皆有相同的地位，同時具備伺服端與客戶端的能力，這種 P2P 類型我們稱之為「純 P2P 技術」。

然而在之後因應用上的特性與需求，出現了所謂的「混合 P2P 技術」。此類 P2P 網路有著中央伺服器，用來負責保存每個客戶端的連線及檔案資訊，並且回覆來自客戶端對於這些資訊的請求。

純 P2P 以及混合 P2P 各有優缺點，也各有適用不同的應用環境。

目前的 P2P 技術發展，已逐漸地將兩者的優點結合，而套用在一些較受歡迎的 P2P 應用上。

2.2 點對點 (Peer-to-Peer)相關應用

本節將介紹幾個較為著名使用 P2P 技術的相關應用。

2.2.1 Napster

在 1999 年，19 歲的 Shawn Fanning 創造了 Napster[4] ，是世界上第一個使用 P2P 技術而且受到歡迎的音樂分享服務。Napster 影響了人們使用網路的方式，使得在網路上交換 MP3 音樂檔案變得相當容易而且流行。但是也因此遭受到唱片公司的控告。

正當 Napster 流行之際，美國法院在 2000 年下令 Napster 網站必須關閉並且停止音樂交換下載服務。雖然 Napster 因此消失了，但是卻開啟了後來眾多 P2P 檔案分享服務的先鋒。當 2003 年，Napster 恢復營運，並且改為付費音樂下載的服務。而免費音樂下載服務的 Napster 已經不存在了。

2.2.2 Gnutella

Gnutella[5] 是一種「純 P2P 技術」應用。剛開始是 AOL 美國線上的工程師在 2000 年撰寫的，並且以 GNU GPL 授權公開讓大家下載。雖然隔天就被 AOL 撤下，但 Gnutella 的 P2P 網路已經啟動。過了一段時間，Gnutella 的協定被反向工程破解，從此與 Gnutella 協定相容的客戶端便陸續誕生。

因為 Gnutella 沒有使用中央伺服器，只要網路上有兩個以上的客戶端，Gnutella 網路便會存在，因此很難被關閉。

現有的 Gnutella 客戶端大多為 Open Source，如 LimeWire、Gnucleus、Ares Galaxy、以及 Shareaza。

2.2.3 eDonkey2000

eDonkey2000[6] 簡稱 eDonkey 或 ed2k，由 Jed McCaleb 在 2000 年所撰寫出來，是一種使用 P2P 技術的檔案分享服務。eDonkey 使用中央伺服器用來尋找檔案，而檔案的傳輸則在客戶端之間進行。

eDonkey 是第一個將檔案切割成片段，讓客戶端能夠下載單一檔案內的不同片段，有效地讓頻寬使用分散至所有客戶端，而非擁有完整檔案的客戶端。

eDonkey 使用 MD4 摘要演算法來辨識文件，使得客戶端擁有的檔案有其獨一性，並可藉由向 eDonkey 的中央伺服器查詢，來得到欲下載檔案的來源資訊。

而在 2002 年，Hendrik Breitkreuz 啟動了 eMule[7] 計畫，目的是為創造相容 eDonkey、並且加入新功能、擁有更友善使用者介面的新軟體。

當 eDonkey 停止發展後，eMule 已成為最受歡迎的 eDonkey 軟體，擁有超過 80%以上的佔有率，且已被移植到各種平台上。

2005 年，美國唱片協會的要求下，eDonkey 網站關閉，也不再繼續發展。但 eDonkey 的 P2P 網路仍靠著 eMule 與眾多 eDonkey 相容客戶端，存活在於網路上。

2.2.4 BitTorrent

2002 年，Bram Cohen 在 CodeCon 發表 BitTorrent[8][9] 軟體以及 BitTorrent 協定，為一 P2P 檔案分享機制。跟一般 P2P 檔案分享軟體不同的是，BitTorrent 有著更利於檔案傳輸的特性。藉由 BitTorrent 協定中制訂的幾個規則，檔案的交換效率能更為提升。

BitTorrent 的規則如下：

1. Rarest Piece First Policy

   優先傳送最稀有的檔案片段。

2. Strict Priority

   盡快完成目前正在下載的檔案片段。

3. Tit-for-Tat Choking Policy

   依照對方給予的檔案片段數量，給予回報。

4. Optimistic Unchoking

   定時隨機挑選連線，以尋找能力更好的對象。

5. Anti-Snubbing

   若一段時間內，皆未從某對象收到片段，便不再上傳給它。

另外還有處理第一片檔案片段的 Random First Piece 以及最後數片檔案片段的 Endgame Mode。這些規則讓 BitTorrent 的客戶端，

能夠盡可能地使用自己的上傳頻寬，以提升整體傳輸速度。

目前，有許多社群維護著自己的 BitTorrent 軟體，在這些社群的努力下，各項新功能被加入，BitTorrent 仍在快速成長中。

2.3 巨量檔案發佈系統 (MDS)系統

MDS 巨量發佈系統[1][3] ，依照 2.1 節所述，為一「混合型 P2P」應用。由中央伺服器保存、維護客戶端資訊，並回覆來自客戶端對彼此資訊的請求。

與其他相類似的 P2P 檔案分享機制相比，MDS 在檔案的發佈上，採用中央伺服器控管的方式，由此伺服器來決定現在要傳送的檔案，並主動將其資訊傳送客戶端。

而 MDS 為了能更快速地發送檔案至客戶端，加入了另一個，擁有完整檔案的伺服器，藉由有效利用此伺服器的頻寬，加上客戶端之間的 P2P 檔案片段交換行為，讓整體下載更加快速。

為了便於使用，我們將 MDS 分割為兩部分。一部份為系統核心，含有上述伺服器與客戶端元件，不需額外設定；而另一部份則為使用

MDS 的公司自行架設的元件，可依需求進行設定與調整。

MDS 架構，如圖 2-1 所示：

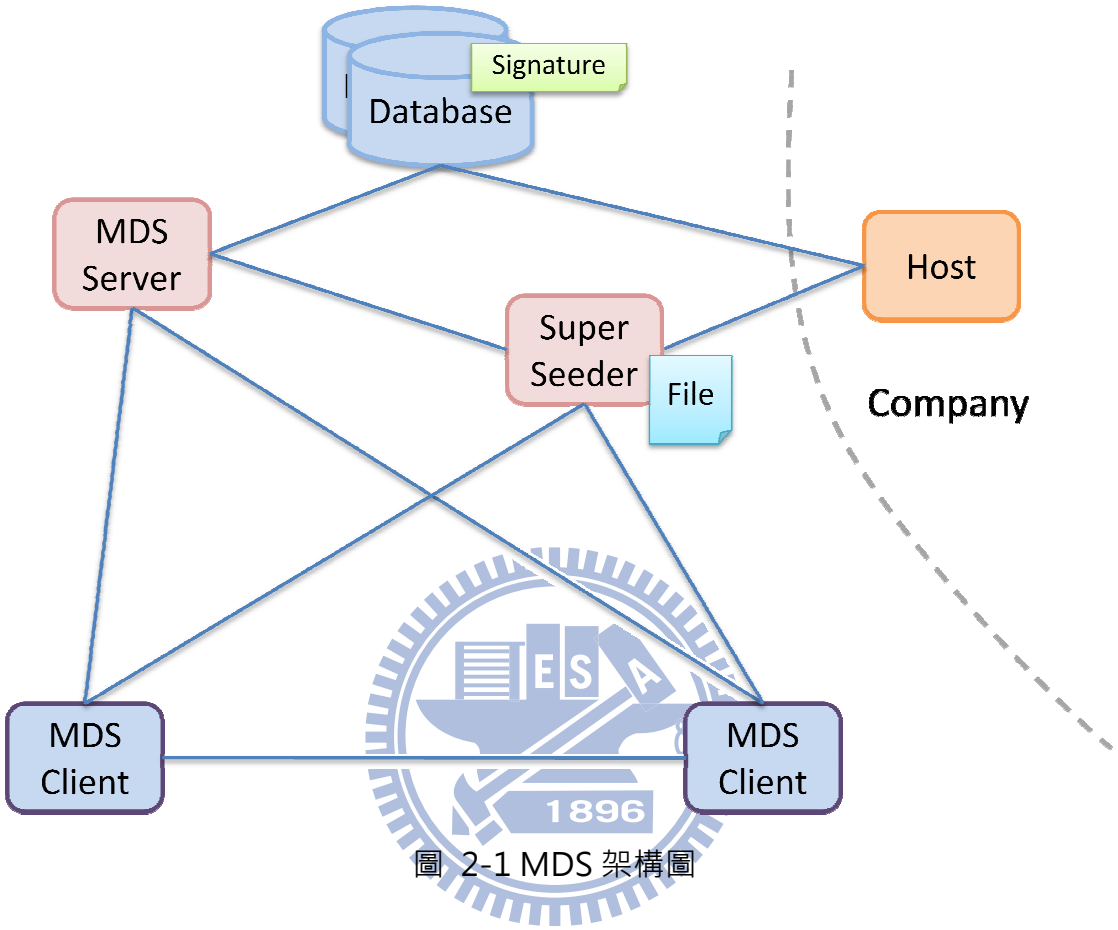


圖 2-1 MDS 架構圖

其中右側虛線以左，為 MDS 核心元件；而虛線以右則為使用 MDS 系統之公司，所需架設與操作的元件。

本篇論文將以此架構為雛型，加入容錯性重新設計，保有原有的 P2P 連線特性，並且當伺服器端有元件故障而無法繼續提供服務時，尤其它的伺服器元件繼續提供服務。

第三章、系統設計

本章節將對具容錯性之點對點傳輸技術巨量檔案發佈系統(With Fault Tolerance Massive Deployment System，簡稱為FT-MDS)的系統架構、運作流程以及各種機制作說明介紹。

3.1 FT-MDS 系統架構

依照 2.3 節所述，我們將重新設一 P2P 巨量檔下載系統架構。首先，我們來看新的架構圖，如下圖 3-1：

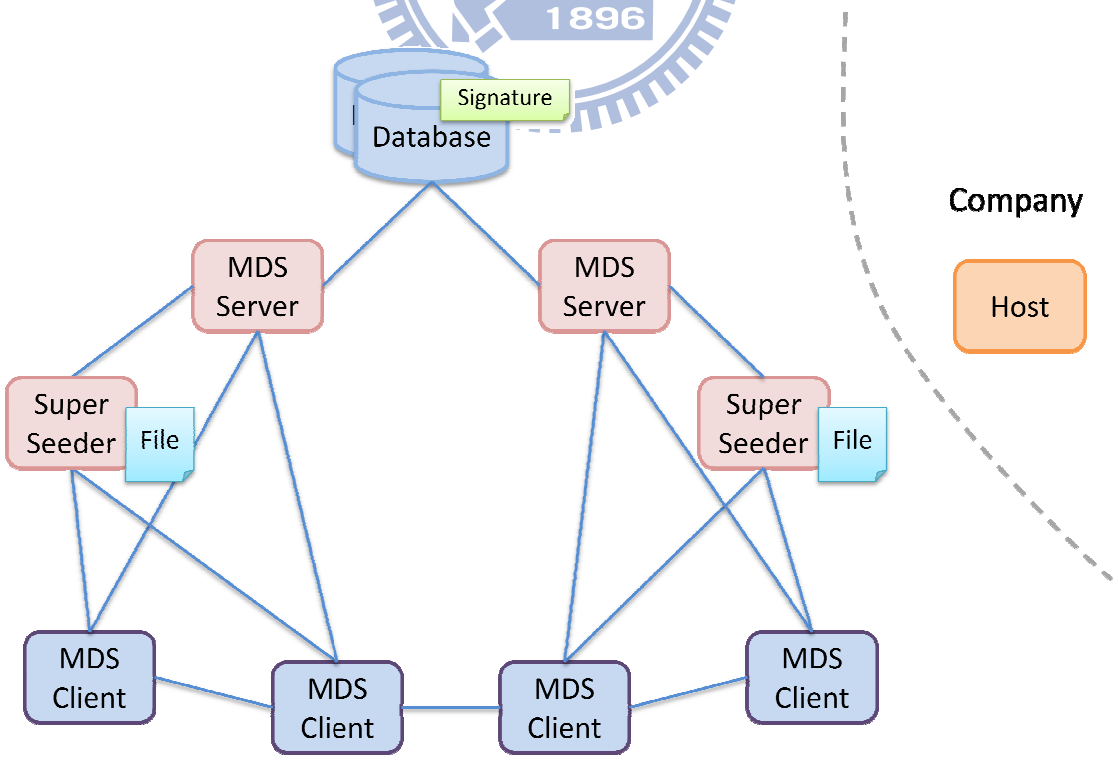


圖 3-1 FT-MDS 架構圖

在上圖 3-1 中，右側虛線以左為 FT-MDS 核心元件；而虛線以右則為使用 FT-MDS 系統之企業，所需架設與操作的元件。

各元件說明如下：

- File：

  企業要發佈給客戶端的檔案，可以為單一檔案或是一個資料夾的檔案。

- Signature：

  企業端由 Host 元件對想發佈的檔案 File 所產生的檔案簽章，含有該 File 的檔案結構、檢查數據以及檔案傳輸所需要的參數。

- Host：

  提供給企業操作 FT-MDS 所使用，主要有三項功能：

  - 製作檔案簽章 File Signature。
  - 將檔案簽章儲 File Signature 存至資料庫 Database。
  - 將想發佈的檔案傳至 Super Seeder。

- MDS Server：

單一 MDS Server 元件具有以下功能：

- 作為 Super Seeder 以及 MDS Client 之間的協調元件。
- 將要發佈檔案 File 的當案簽章傳 File Signature 送給 Super Seeder 以及 MDS Client。
- 傳送 MDS Client List 給 MDS Client。
- 具備負載紀錄，由 MDS Client 產生負載平衡機制。

- Super Seeder：

  在下載系統中擔任發佈檔案的元件，具備以下功能：

  - 擁有完整的發佈檔案 File。
  - 傳送檔案片段至 MDS Client。

- MDS Client：

  客戶端元件，具備以下功能：

  - 測試 MDS Server 負載值。
  - 向 Super Seeder 要求發佈檔案 File 片段。
  - 向其他 MDS Client 作發佈檔案的檔案片段交換。

- Database：

儲存 FT-MDS 系統相關資訊，以及所有的 MDS Server、Super Seeder 以及 MDS Client 的資料，並且提供伺服器資訊以供達到容錯式設計。以下為主要的儲存項目：

- 發佈檔案 File 的檔案簽章 File Signature。
- MDS Server 的清單。
- Super Seeder 的清單。
- MDS Client 的資訊以及清單。
- 系統服務紀錄以及報告。

3.2 FT-MDS 系統及檔案準備流程

使用 FT-MDS 系統發佈檔案，在企業端需要使用到 Host 這個元件製作發佈檔案的檔案簽章 File Signature，並且將發佈檔案 File 傳至 Super Seeder、檔案簽章 File Signature 傳至資料庫。

檔案發佈的準備流程如下圖 3-2 所示：

1. Host 將要發佈的檔案 File 製作出檔案簽章 File Signature。
2. Host 將檔案簽章 File Signature 儲存至資料庫。
3. Host 將檔案 File 透過 FTP Protocol 傳送至 Super Seeder。

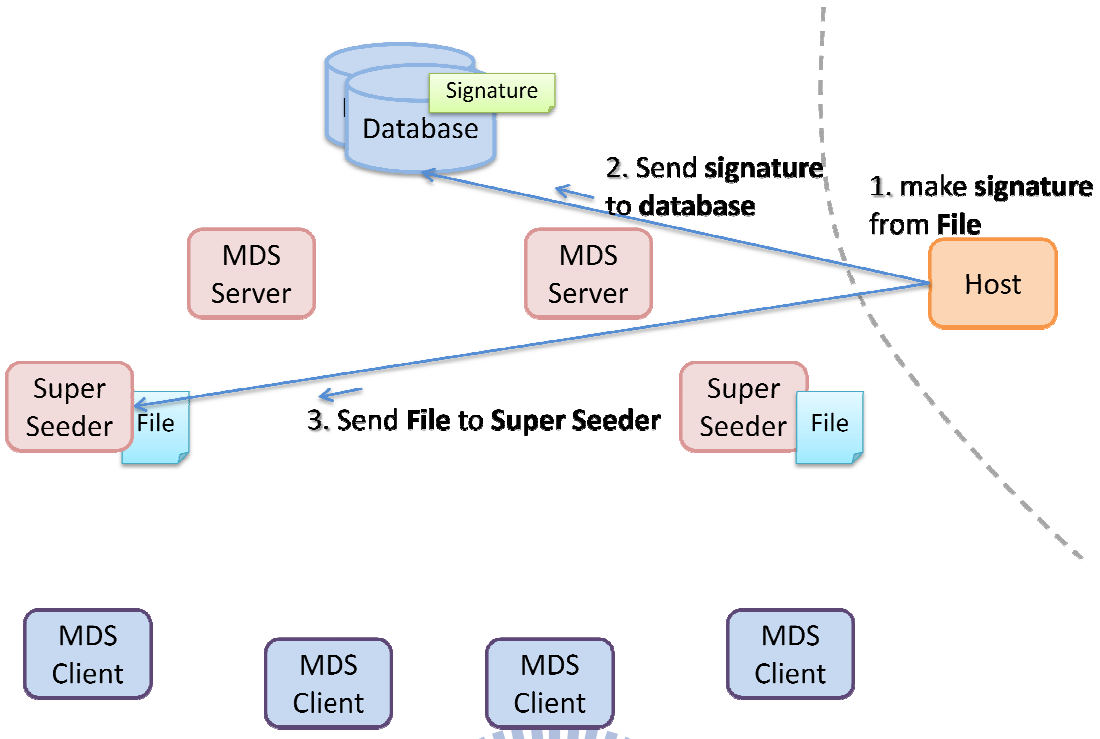


圖 3-2 FT-MDS 準備流程圖

當 Host 完成將發佈檔案傳送至 Super Seeder 並且檔案簽章 File Signature 已經傳至資料庫，FT-MDS 系統檔案發佈的準備流程就已經就緒了，此時就可以啟動 MDS Server 以及 Super Seeder 準備進行檔案的傳輸服務。

3.3 FT-MDS 檔案傳輸流程

已經完成準備流程之後，須先將 MDS Server 啟動，由 MDS Server 來管理 Super Seeder 以及 MDS Client 的連線和資訊。而在檔案傳輸的過程中，MDS Server 以及 Super Seeder 會一直保持連線，用來監控 Super Seeder 的服務狀態。

檔案傳輸的流程如下圖 3-3，圖 3-4，圖 3-5 以及圖 3-6：

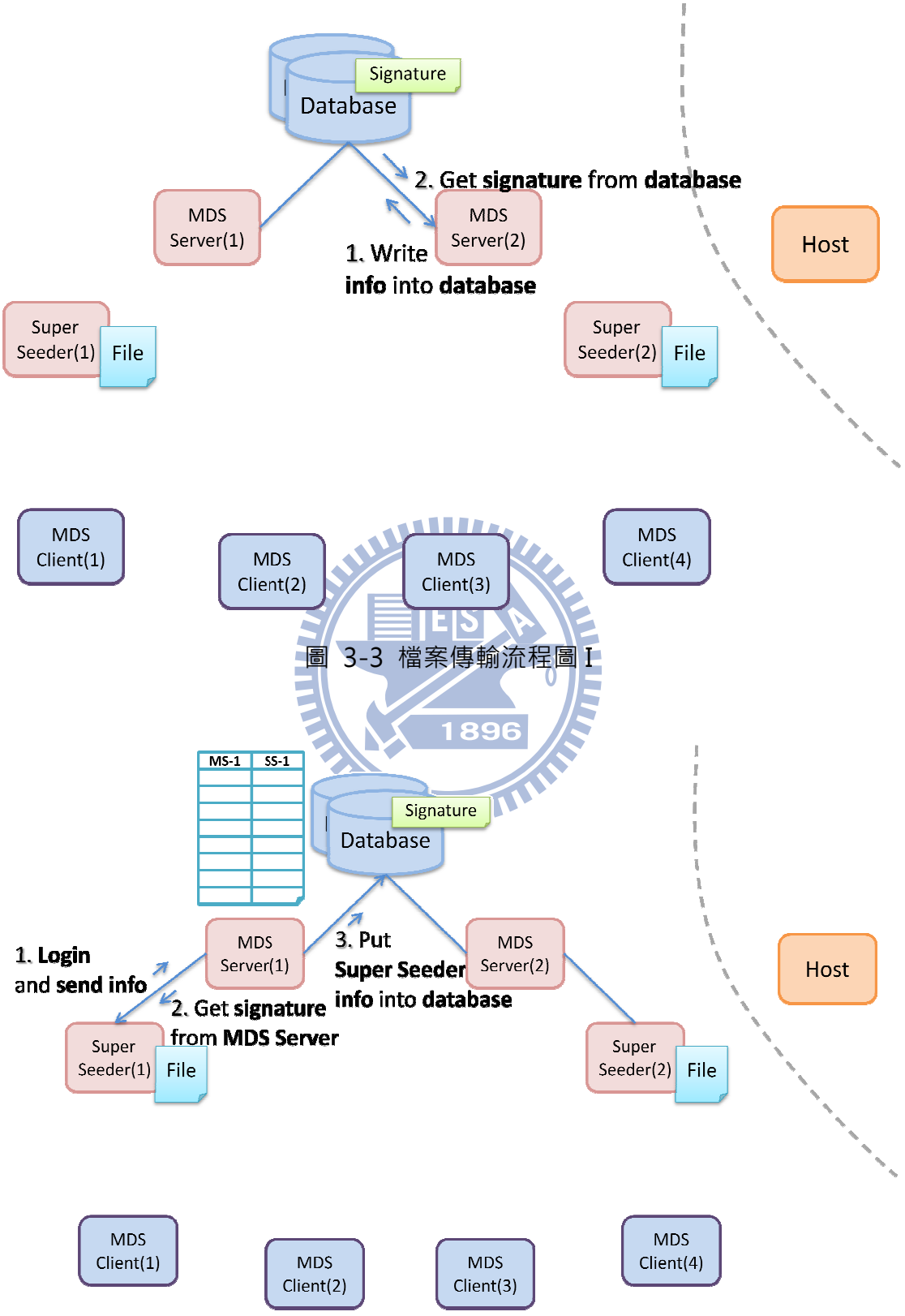


圖 3-3 檔案傳輸流程圖 I

圖 3-4 檔案傳輸流程圖 II

如圖 3-3，圖 3-4 所示：

1. MDS Server 連線至資料庫，將 MDS Server 本身的資訊寫入資料庫， 並且取得檔案簽章 File Signature 以及初始化系統。
2. 接著 Super Seeder 啟動並且登入 MDS Server，取得要服務的檔案簽章 File Signature，然後依照 File Signature 的資訊檢查由 Host 傳送過來的發佈檔案 File 完整及正確性。

而此架構下，將具有多套的 MDS Server 以及 Super Seeder，依照前面兩個步驟逐一的啟動並且等待 MDS Client 的登入。

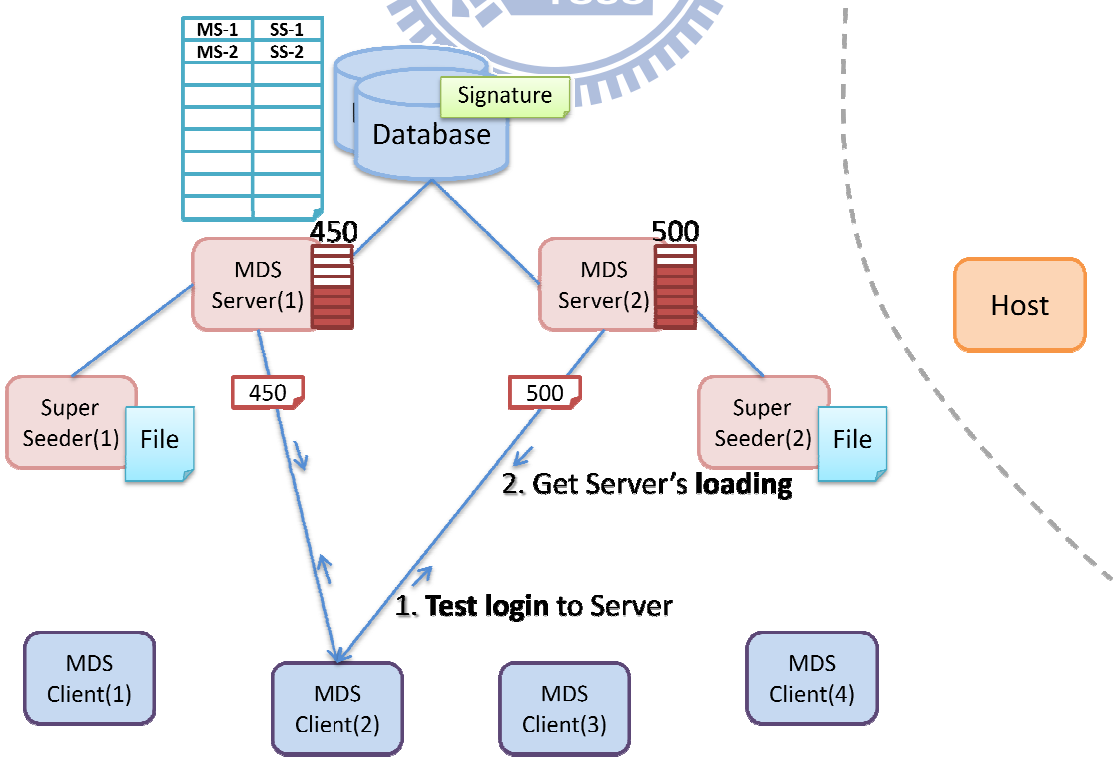


圖 3-5 檔案傳輸流程圖 III

如上圖 3-5 所示：

1. MDS Client 啟動後，根據伴隨的 MDS Server List，逐一的發送測試連線封包。
2. MDS Server 收到測試連線封包之後，會回傳系統的負載值。

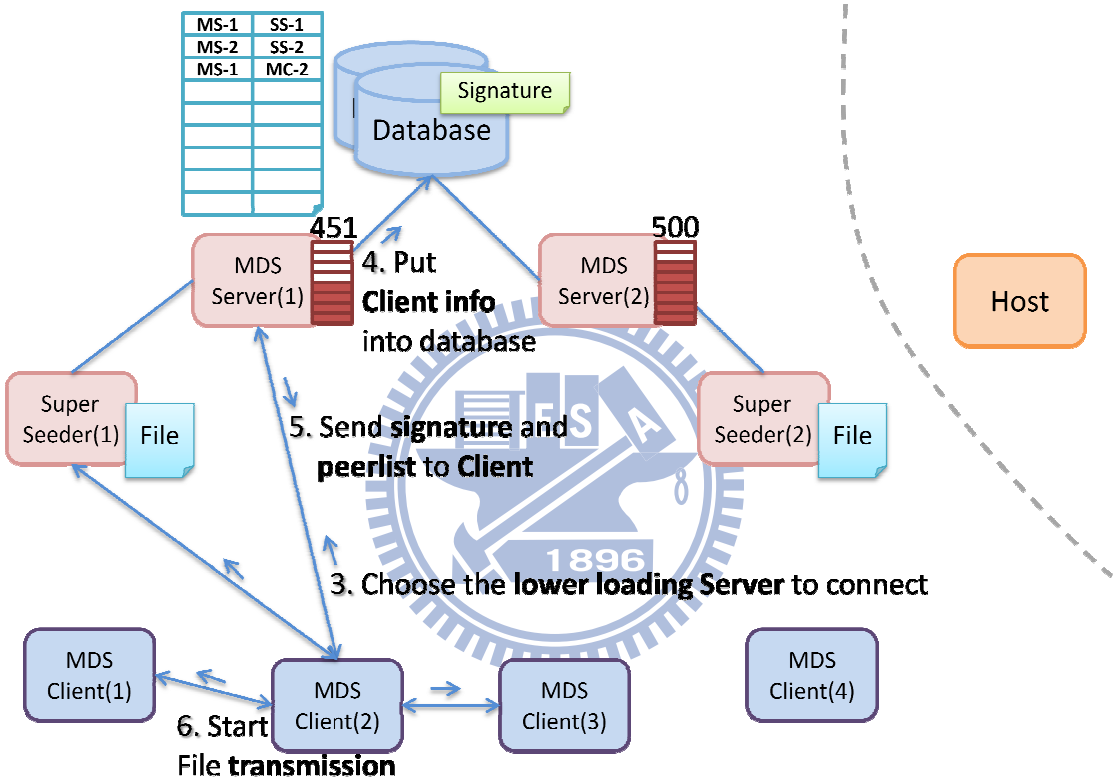


圖 3-6 檔案傳輸流程圖 IV

如圖 3-6 所示：

1. MDS Client 根據 MDS Server 傳回的負載值，選擇有較小負載值得 MDS Server 作一般性的登入，並且更新 MDS Server List。

2. 當 MDS Server 確認連線後，將會傳送發佈檔案的檔案簽章 File Signature 以及 MDS Client List 給 MDS Client，而 Super Seeder 就放在 MDS Client List 的第一項。
3. MDS Client 將會先連線至 Super Seeder，取得一小部分的檔案 File 片段，接著連線到 MDS Client List 的其他 MDS Client 交換檔案。

經過一段時間後，系統將達到一個簡易的負載平衡狀態，如圖 3-7 所示：

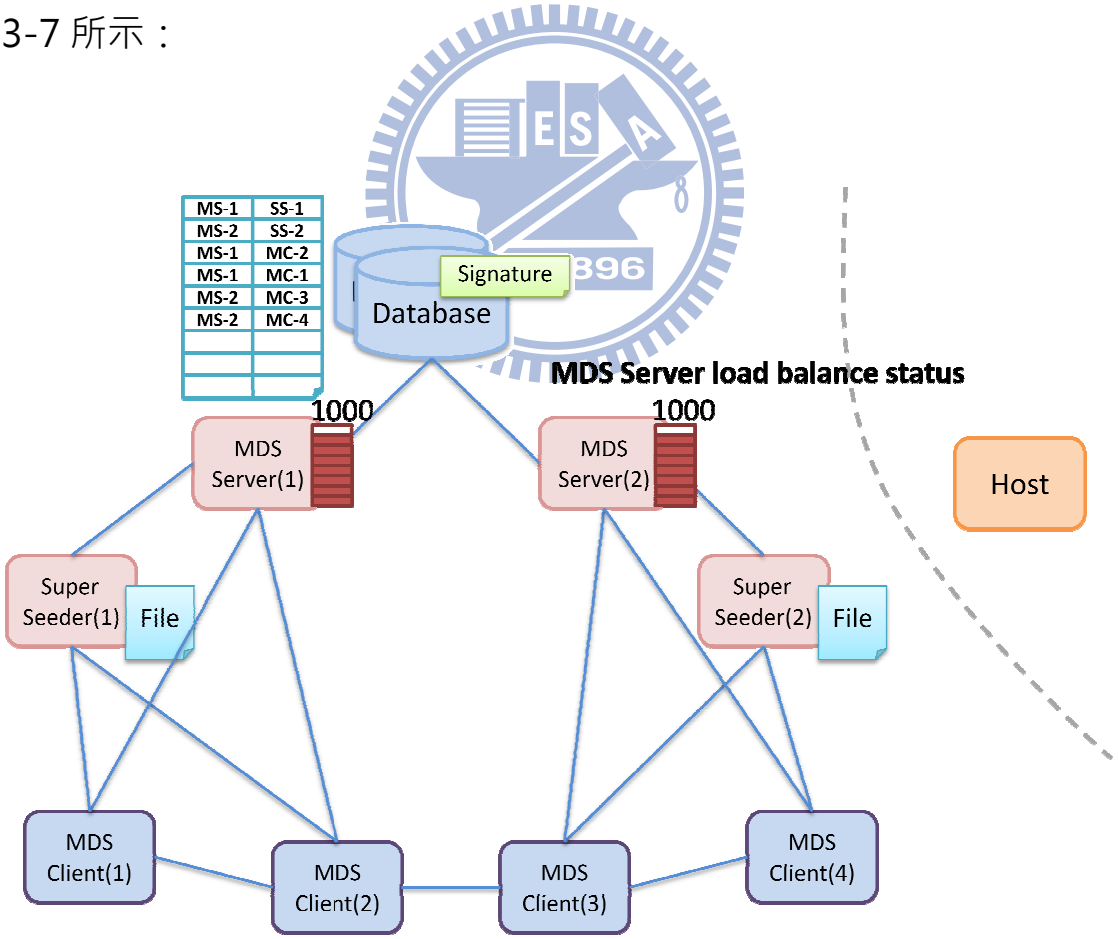


圖 3-7 FT-MDS 負載平衡狀態

而接下來會針對 FT-MDS 可能會遇到的連線問題，提出解決的方式。

3.4 TF-MDS 錯誤處理流程

本節將會對於可能遇到的連線失效問題，提出解決方案。

3.4.1 MDS Server 錯誤處理

如下表格 1 所示，對於 MDS Server 而言，所有可能發現的斷線組合，我們將針對當發生斷線錯誤時，FT-MDS 將如何處理這些意外狀況，並且讓系統持續提供服務。

表格 1 MDS Server 連線錯誤表

	Database	Super Seeder	MDS Client
MDS Server	X	X	X
MDS Server	X	X	O
MDS Server	X	O	X
MDS Server	X	O	O
MDS Server	O	X	X
MDS Server	O	X	O
MDS Server	O	O	X
MDS Server	O	O	O

3.4.1.1 MDS Server 失去所有連線

當 MDS Server 發生當機狀況，或是失去了所有的連線，如下圖 3-8 所示：

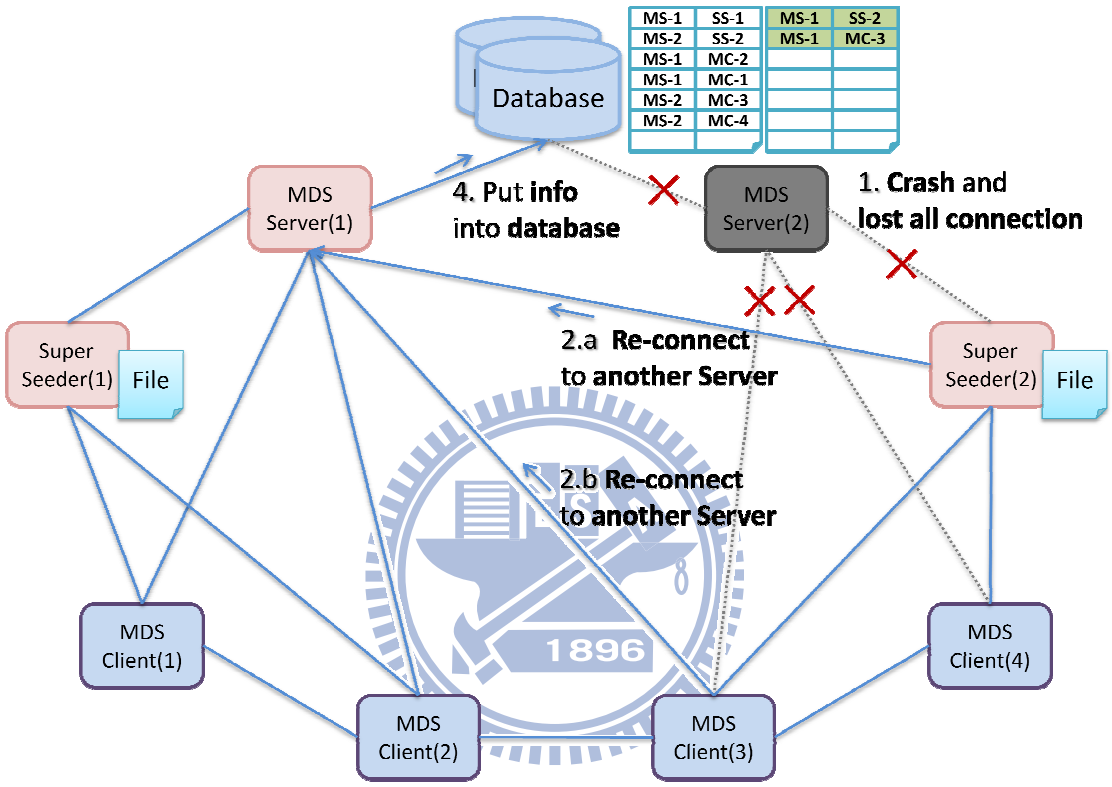


圖 3-8 MDS Server 失去所有連線

此時 FT-MDS 的處理流程如下：

1. MDS Server 發生錯誤，失去所有的連線。
2. 此時由於 Super Seeder 之間保持的通訊連線斷線，Super Seeder 將會馬上發現，等待 MDS Server 一段時間而無法恢復連線時，Super Seeder 會連線至自己所具有的 MDS

Server List 第二個順位的 MDS Server 去連線。

3. 當發生 MDS Server 斷線後，MDS Client 再下一次向 MDS Server 要求 MDS Client List 時將會發現無法連線，此時會連線至自己的 MDS Server List 第二個順位的 MDS Server 連線。

而當失去連線的 MDS Server 在一段時間後，若恢復連線時，則要對剛剛斷線時所造成的一些錯誤資料做處理，如下圖 3-9 所示：

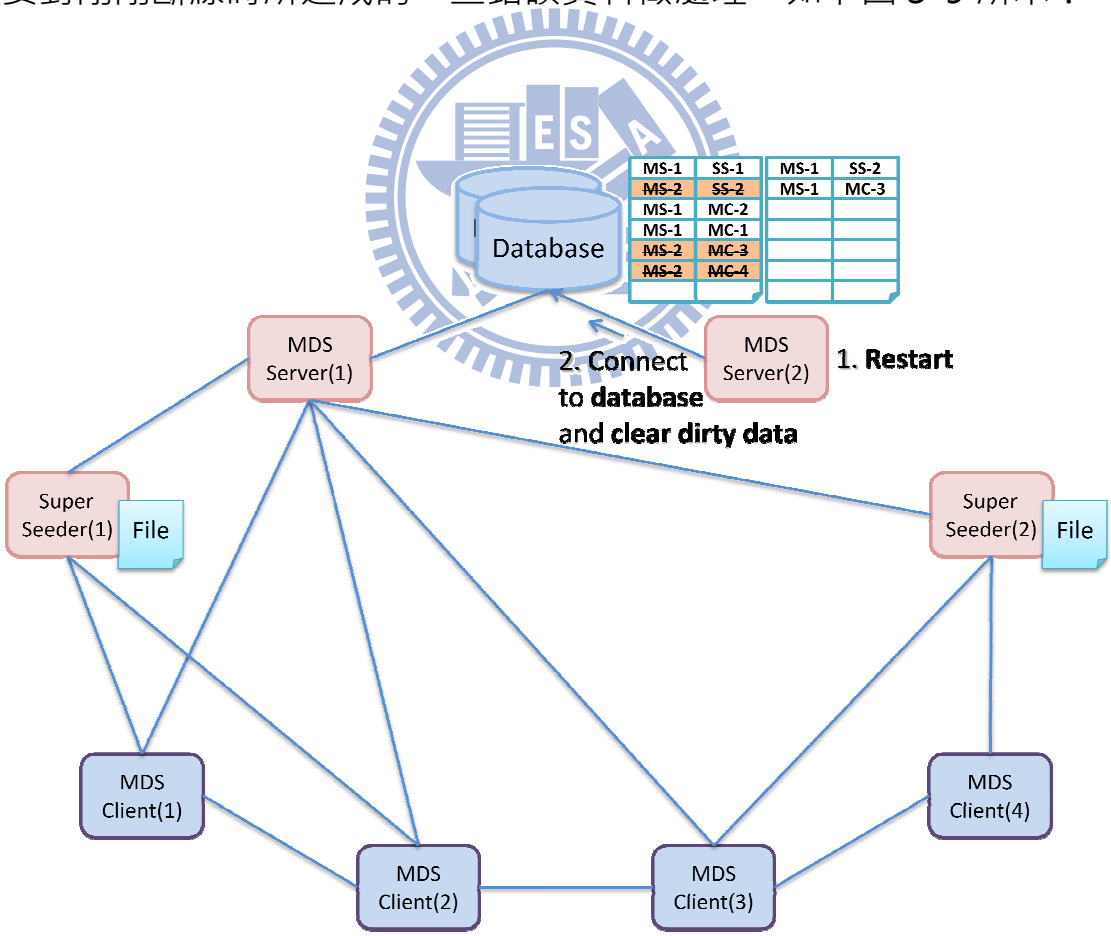


圖 3-9 MDS Server 回復可用狀態

回復連線後處理流程如下：

1. 重新啟動 MDS Server 或解除失去連線因素。
2. 連線至資料庫，將自己之前所寫下而未清空的 Super Seeder 以及 MDS Client 資訊清除。

3.4.1.2 MDS Server 只有跟資料庫連線

當 MDS Server 發生連線異常，MDS Server 只剩下與資料庫的連線正常，如下圖 3-10 所示：

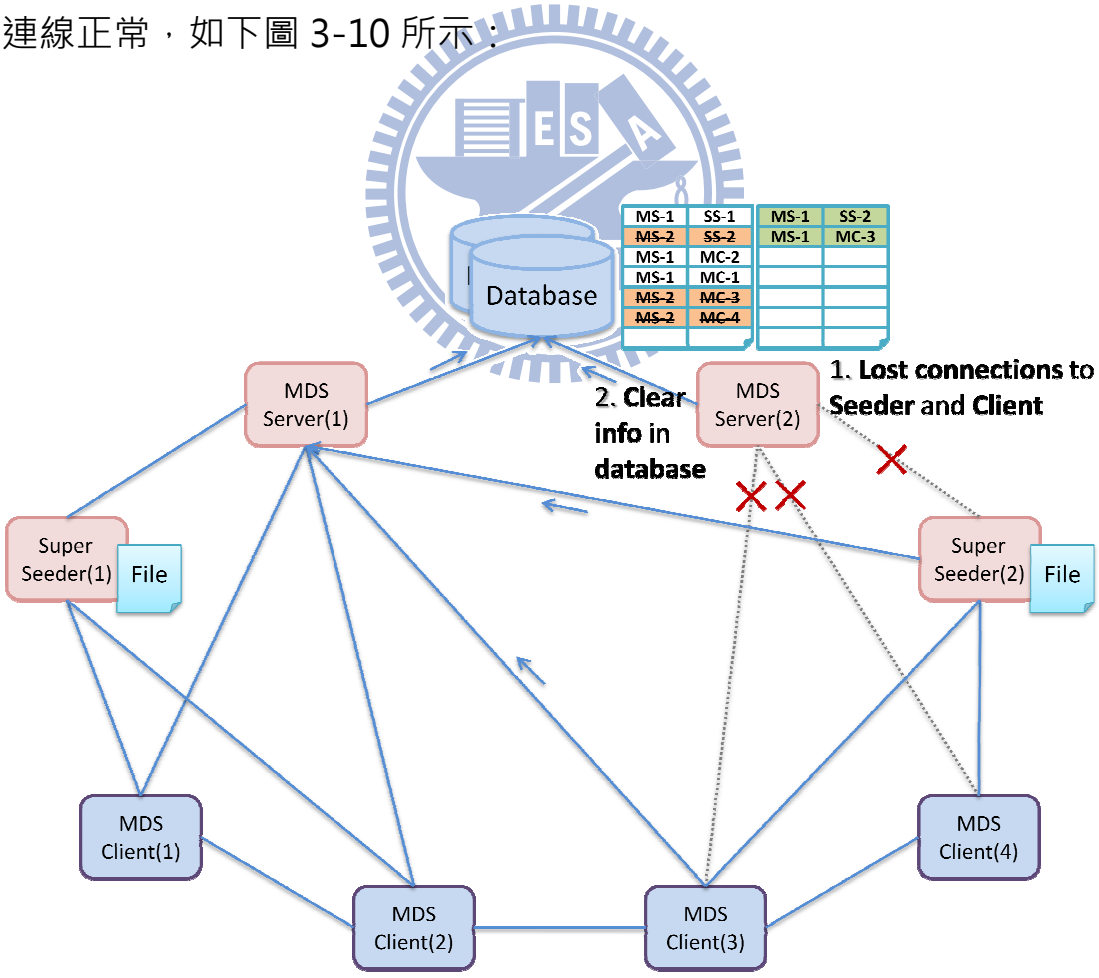


圖 3-10 MDS Server 只跟資料庫連線

當發生此狀況時，處理流程如下：

1. 首先 Super Seeder 會發現無法與 MDS Server 連線。而連線至自己的 MDS Server List 下一個順位的 MDS Server 進行登入並且等待服務。
2. 而 MDS Server 也認為 Super Seeder 以及 MDS Client 的資訊從資料庫中清除。
3. 接著 MDS Client 發現 MDS Server 無法連線之後，也連線到自己的 MDS Server List 中的下一個順位 MDS Server 繼續取得服務。

而此時的 MDS Server 會從資料庫中去取得別的可用 Super Seeder 資訊，當有新的 MDS Client 成功連線至此 MDS Server 時，即可繼續提供服務。

3.4.1.3 MDS Server 只有跟 Super Seeder 連線

當 MDS Server 發生連線異常，MDS Server 只剩下與 Super Seeder 的連線正常，此時狀況如下圖 3-11 所示：

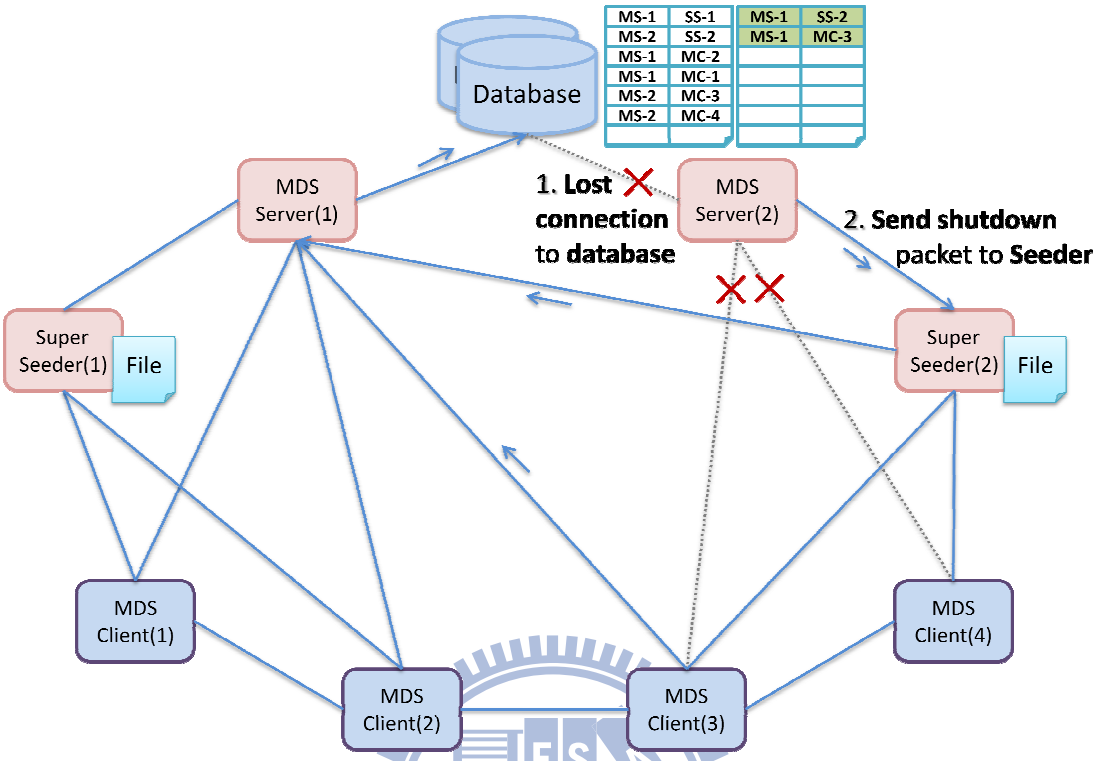


圖 3-11 MDS Server 只跟 Super Seeder 連線

此狀況的處理流程如下：

1. 首先當 MDS Server 無法連線到資料庫時，由於無法取得下載檔案的檔案簽章，無法確認此次將提供何種版本或是何種檔案的下載服務，此時 MDS Server 將發送斷線資訊給 Super Seeder 以及 MDS Client。
2. 當 Super Seeder 收到 MDS Server 斷線訊息，則重新連線至自己的 MDS Server List 中的下一個順位 MDS Server，繼續等待提供服務。

3. MDS Client 收到 MDS Server 的斷線訊息後，則回到重新連線機制，對 MDS Server List 中的所有 MDS Server 作負載測試而登入別的 MDS Server。

3.4.1.4 MDS Server 只有跟 MDS Client 連線

當 MDS Server 發生連線異常，MDS Server 只剩下與 MDS Client 的連線正常，如下圖 3-12 所示：

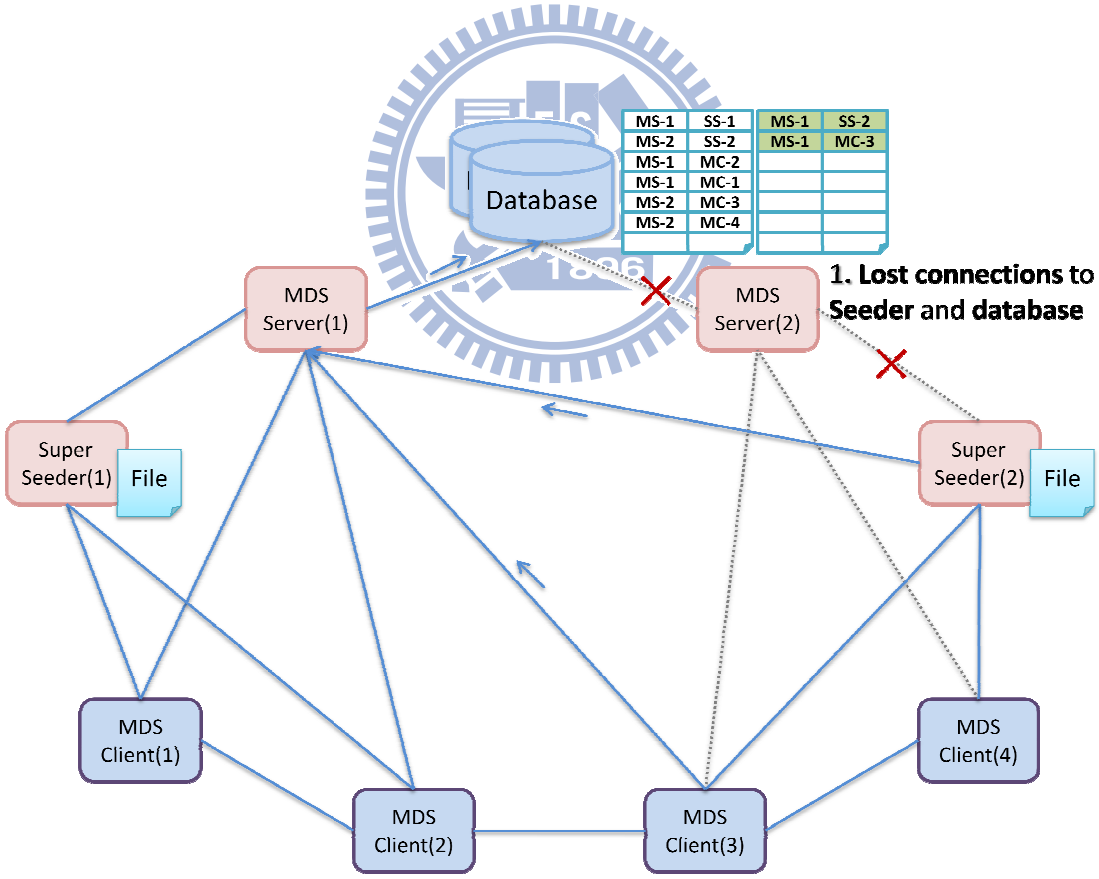


圖 3-12 MDS Server 只跟 MDS Client 連線

此狀況的處理流程如下：

1. 由於 MDS Server 發現無法與資料庫連線，並且失去與 Super Seeder 的連線，所以 MDS Server 發送斷線訊息給所有跟它連線的 MDS Client，以確保 MDS Client 可以拿到最新的 MDS Client List 達到良好的傳輸品質。
2. Super Seeder 無法連線到 MDS Server 後，則向自己的 MDS Server List 中的下一個順位 MDS Server 進行連線並且繼續等待提供服務。
3. MDS Client 收到 MDS Server 的斷線訊息後，則重新回到登入狀態，向自己的 MDS Server List 中的所有 MDS Server 作負載的測試，並且登入繼續取得服務。

3.4.2 Super Seeder 錯誤處理

本節將介紹 Super Seeder 會遇到的斷線狀況如下表格 2，並且針對斷線的狀況提出解決的方式。

表格 2 MDS Server 連線錯誤表

	MDS Server	MDS Client
Super Seeder	X	X
Super Seeder	X	O
Super Seeder	O	X
Super Seeder	O	O

3.4.2.1 Super Seeder 失去所有連線

Super Seeder 失去所有的連線，如下圖 3-13 所示：

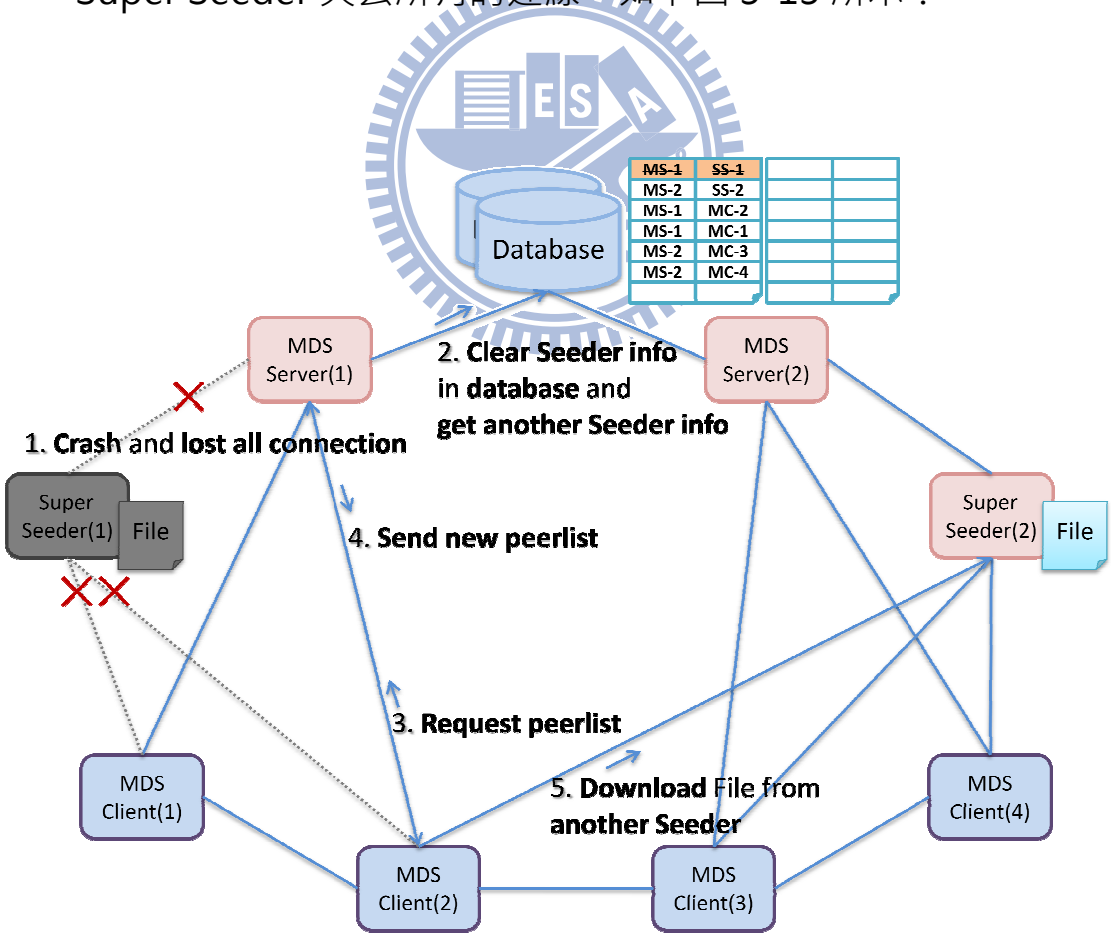


圖 3-13 Super Seeder 失去所有連線

此狀況的處理流程如下：

1. Super Seeder 失去所有連線。
2. MDS Server 發現後會清除 Super Seeder 在資料庫的資訊，並且拿到可以提供服務的 Super Seeder 連線資訊。
3. 當 MDS Client 在下一次跟 MDS Server 要求 MDS Client List 的時候，MDS Server 所送出的新 MDS Client List 的第一項將會是新的可以提供服務的 Super Seeder 的連線資訊。

而當 Super Seeder 回復到可服務的狀態後，會從自己的 MDS Server List 去連線登入系統，並且等待提供服務。

3.4.2.2 Super Seeder 只有跟 MDS Client 連線

另一種狀況是 Super Seeder 只有跟 MDS Client 連線，如下圖 3-14 所示：

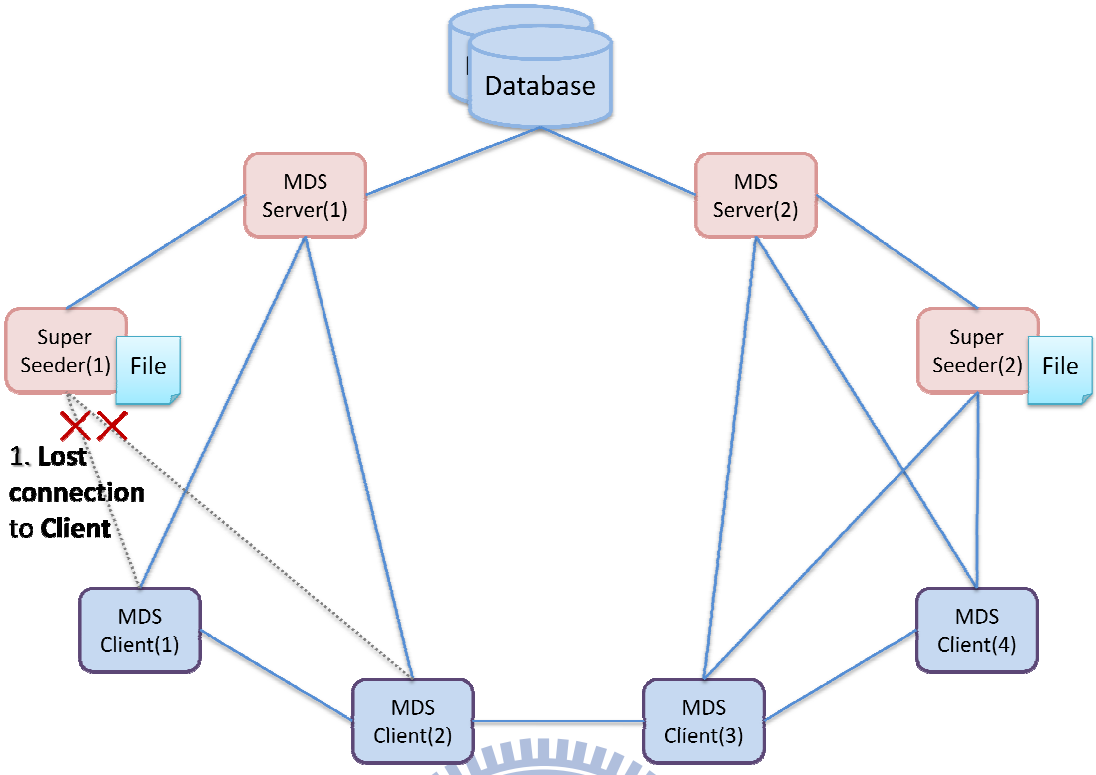


圖 3-14 Super Seeder 只有跟 MDS Server 連線

此狀況由於 MDS Server 與 Super Seeder 保持連線，所以仍然能夠繼續提供檔案傳輸服務，對於此狀況伺服端元件並無做相關的處理。

第四章、系統實作

本章節將會說明系統上實作的所有細節，包含程式架構以及實作的內容。

4.1 程式架構

在程式架構上，由於 MDS Server、Super Seeder 皆由過去巨量檔案發佈系統承接而來，主要的程式架構仍是以滿足大量的下載需求為主，而 MDS Server 與 Super Seeder 均會承載大量的連線與封包處理，因此繼續延用較高的 I/O 架構。以過去的網路程式而言，處理同時連線 (Concurrent connections) 的網路程式皆會使用 select() 系統呼叫[8]，達到快速處理多個 socket 的能力。然而在大部分的系統中，為了不讓 select() 超過其能負載的範圍，皆定義了 FD_SETSIZE 這個巨集常數來限制其能力，以 FreeBSD 與 Linux 來說，其值為 1024。也就是說，select() 最多只能夠同時處理 1024 個 socket，這對於大量連線的巨量檔案發佈系統並不足夠。

而還有另一種 poll() 系統呼叫[9]，雖然沒有限制最大能處理的 socket 數目，但實際上當連線的數量達到兩千個左右的連線數時，速度便相當慢了，原因在於掃瞄每個 socket 需要花費大量時間。

因此，FT-MDS 延續過去 MDS 必須使用快速處理超大量 socket，

且反應速度夠快的系統呼叫。在著名的 The C10K Problem[10] 網頁中，提出了種種的 I/O 處理策略，我們這次依然選擇 Event-Driven 的方式，並使用 Non-blocking I/O (非阻塞式 I/O) 以及 Readiness Change Notification (就緒狀態改變通知) 的高效能系統呼叫：FreeBSD 的 kqueue()[11] 以及 Linux 的 epoll()[12]。

在 MDS 的處理連線核心中而為了讓程式能夠跨平台，我們使用了 libevent 這套 Open Source 開放原始碼函式庫[13]。libevent 在編譯函式庫時，能針對編譯的系統環境，選擇效能最佳的系統呼叫，像在 FreeBSD 的 kqueue()、Linux 的 epoll()、Solaris 的 /dev/poll[14] 與 event ports[15]，以及傳統的 select() 與 poll()。因此，我們的程式可以在不修改程式碼的情況下，輕鬆地在多個平台上編譯、執行。

此外，libevent 作者採用 BSD License (3-clause BSD License) 授權協議[16]，若將 MDS 運用於企業內，便不易產生侵犯授權爭議。這也是使用 libevent 的優點之一。

4.2 實作內容

MDS Server 與 Super Seeder 以 C++ 實作而成，是一個 single-threaded 的單一執行緒程式，並且以 single-process 單一程

序服務所有連線。由於 MDS Server 與 Super Seeder 皆為需處理大量同時連線的程式，因此兩者共用相同、使用 libevent[13] 撰寫的程式核心。目前 MDS Server 與 Super Seeder 已在下列系統中測試：

- FreeBSD 7.0-Release (amd64 64bit)
- Debian Linux 2.6.18 Release 4.0 (i386 32bit)

因程式在 UNIX 環境執行運作，所以兩者皆設計以 daemon process 在系統背景執行，透過指令稿與額外的控制程式，發送 POSIX Signal 來控制 MDS Server 與 Super Seeder 進行啟動、停止、重新啟動、回報狀態等動作。並使用 syslog 服務記錄程式輸出的訊息，使其和 UNIX 系統整合，提高使用便利性。

以下說明針對 MDS Server 及 Super Seeder 個別的實作問題與解決方法。

4.2.1 MDS Server

MDS Server 在系統中扮演中樞控制的角色，提供 Super Seeder、MDS Client 登入，傳送所需資訊，並維護其清單以及控管簡易的負載平衡數值等功能。

以下介紹與系統穩定度相關的 Super Seeder 與 MDS Client 狀態維護，清除已發生問題之 Super Seeder 與 MDS Client 所設計的 KeepAlive[17] 機制。

4.2.1.1 Super Seeder 與 MDS Client 狀態維護

Super Seeder 除了在登入時會傳送其服務資訊給 MDS Server 外，在運行的過程中，也會定時傳送 Report 資訊給 MDS Server，內容為負載 (loading)、每秒處理幾個片段要求 (requests/sec) 以及頻寬使用率 (bandwidth usage)。MDS Server 會依照目前所有 Super Seeder 的負載狀況，來決定要指派哪一個 Super Seeder 給 MDS Client。

而 MDS Client 在登入時會傳送 NAT[18][19][20] 與頻寬測試資訊以及目前狀態。其狀態分為兩種：

- 已完成所有 File 片段 (Seeder)
- 未完成所有 File 片段 (Leecher)

當 MDS Client 開始或繼續下載一個新發佈的 File 時，狀態為

Leecher；而當 MDS Client 完成所有 File 片段時，會傳送已完成資訊給 MDS Server，讓 MDS Server 更新此 MDS Client 的狀態。

MDS Client 狀態不同，在挑選、組合 MDS Client List 時的策略也有所不同。例如：當某個 MDS Client 已經成為 Seeder 後，MDS Server 就不需要再給它內含 Seeder 的 MDS Client List，因為它已經不需要其他 MDS Client 傳送 File 片段給它。MDS Server 這時會給他全部都是 Leecher 的 MDS Client List，讓它盡全力去傳送 File 片段；而在 MDS Client 仍是 Leecher 時，MDS Server 會給它一部份 Seeder、一部份 Leecher 的 MDS Client List。除了讓它能和 Seeder 們要求 File 片段外，也能和其他 Leecher 交換彼此沒有的 File 片段。

4.2.1.2 KeepAlive 機制

在經過一段時間的運作後，總是會發生一些例外狀況，如 Super Seeder 所在的伺服器當機，或者某 MDS Client 所在的使用者電腦無預警斷電，甚至是某一家 ISP 業者發生網路故障等。這些會造成不正常斷線、甚至不知道有沒有斷線的狀況，導致 MDS Server 沒有偵測到 Super Seeder、MDS Client 的離開，而繼續維護其狀態。

這些不正確的狀態資訊，會讓其他 MDS Client 取得不可用的 Super Seeder 與 MDS Client List，因此要設法讓 MDS Server 知道每個 Super Seeder、MDS Client 是否真正存活。

在這邊，我們讓 MDS Server 記錄 Super Seeder 與 MDS Client 最後傳送資訊的時間戳記 (timestamp)，並定時檢查目前時間與該時間戳記的差值。超過預定的時間 (如 600 秒) 時，發送 KeepAlive 封包，並期待回傳 KeepAlive Acknowledge 封包。若一段時間過後仍未收到 KeepAlive Acknowledge，則判定對方已發生問題，MDS Server 將主動切斷連線，並釋放其所使用資源。不使用 TCP Socket Option 中的 SO_KEEPALIVE[17] 選項，是因為此機制需要 2 小時內皆沒有任何傳輸，才會發出探測封包。兩小時是相當久的時間，若同時有許多 MDS Client 發生問題，對系統的效能會有相當大的影響。雖然有方法能夠更改此時間，但各種平台上上的支援程度並不相同，因此決定自行實作。

4.2.2 Super Seeder

Super Seeder 為系統中最為忙碌的元件，需要不停地處理來自 MDS Client 的 File 片段請求，並回傳該片段給 MDS Client。Super

Seeder 與 MDS Client 之間的行為參考自 BitTorrent 協定[13]，含有一系列的狀態與資料封包格式。而為了保持 Super Seeder 的穩定，以及定時回報狀態給 MDS Server 外，也需要控制其輸出的頻寬。以下將說明 Super Seeder 如何與 MDS Client 通訊、並靈活地回報狀態給 MDS Server，以及如何控制輸出頻寬。

4.2.2.1 與 MDS Client 通訊

BitTorrent 協定主要用於交握、確認已有 File 片段、以及要求 File 片段、傳遞 File 片段。各封包有其順序，依序如下：

1. **Handshake：**

   Super Seeder 與 MDS Client 之間傳送的第一個封包，內含軟體版本資訊及目前所使用 File Signature 的一組雜湊值 (hash)。我們將原本 BitTorrent 的 Handshake 擴充，額外加入發送端的 NAT 類型與頻寬，未來可利用此資訊進行更細微的最佳化。此封包由 MDS Client 主動送出，Super Seeder 收到後回傳自己的 Handshake 封包，完成交握。

2. **BitField：**

   BitField 內記載發送端所擁有的 File 片段資訊，一個 bit 表示一

個片段。Super Seeder 在完成交握後，將傳送每個 bit 都是 1 的 BitField 封包給 MDS Client，告知對方自己擁有全部 File 片段。

3. **Interested：**

   此封包由 MDS Client 發出，表示想要從 Super Seeder 下載 File 片段。

4. **UnChoke / Choke：**

   Super Seeder 收到來自 MDS Client 發出的 Interested 後，可選擇回覆 UnChoke 或 Choke。UnChoke 表示同意 MDS Client 開始請求 File 片段；而 Choke 表示禁止 MDS Client 請求 File 片段。此機制是 BitTorrent 協定中用來控制傳輸的，有一套 Choking Policy 來控制。然而 Super Seeder 在目前的設計中，是公平傳送 File 片段給每個發出要求的 MDS Client 的，因此當 Super Seeder 收到 Interested 封包時，一律回覆 UnChoke，讓 MDS Client 能夠來請求 File 片段。

5. **Request：**

   MDS Client 發出的 File 片段請求，內含片段編號 (piece number)、開始位置 (offset) 以及長度 (length)。雖然將 File 分成若干片段來傳送，但實際上每次請求的長度，才是真正在網

路上傳送的大小。在我們的系統中，長度均為 16384 Byte (16 KB)。

6. **Piece：**

   Super Seeder 傳送給 MDS Client 的 File 片段，內含片段編號 (piece number)、開始位置 (offset) 以及長度 (length)，最後接上 File 片段資料。MDS Client 可從封包內容得知自己已經收到了哪一個片段的部分內容。

每一個 MDS Client 與 Super Seeder 皆需經過上述封包流程，在開始發送 Request 片段請求後，便不停地重複此步驟，直到完成所有 File 片段。在完成所有 File 片段、成為 Seeder 之後，便會主動與 Super Seeder 斷線。

4.2.2.2 狀態回報

為了精確監控 Super Seeder 的運作狀況，每一秒鐘便會執行一個用來統計資訊的函式，而每執行五次此函式，便會將這五次所統計資料的平均值，傳送給 MDS Server。內容含有負載 (loading)、每秒處理幾個片段要求 (requests/sec) 以及頻寬使用率 (bandwidth usage)。

若 Super Seeder 目前並不忙碌，每五秒傳送一次 Report，便顯得有點浪費了。因此我們設計了一個閒置計數器，此計數器數值越大，表示 Super Seeder 越不忙碌。藉由此計數器，可以讓 Super Seeder 在忙碌的時候，每五秒便傳送一次 Report 給 MDS Server，以即時監控狀態；而在漸漸不忙碌的時候，拉長傳送 Report 的時間間隔；直到完全不忙碌的時候，傳送幾次 Report，讓 MDS Server 維護的 Super Seeder 狀態符合現況後，便停止傳送 Report。

4.2.2.3 控制輸出頻寬

MDS 系統的目的之一，便是讓使用此系統的公司、企業，能夠降低頻寬上的花費。倘若 Super Seeder 不限制輸出，讓系統持續佔用頻寬直到滿載，便失去使用 MDS 的意義。因此，讓 Super Seeder 能夠有效地控制輸出頻寬，是相當重要的。

由於傳送給 MDS Client 的 File 片段為主要輸出項目，且長度也比通訊用封包長了許多倍，為了不影響其他通訊，輸出控制只針對傳送 File 片段所用的 Piece 封包。

Super Seeder 控制輸出頻寬的步驟流程如下：

1. 設置一輸出計數器，初始值為零。
2. 每當送出一個 Piece 封包時，便累加其長度於輸出計數器。
3. 當寫入事件發生時，檢查輸出計數器，若小於預先設定的頻寬限制大小，便寫入 Socket；若大於，則不寫入 Socket，並將此連線暫停，使其不會再發生，並將其儲存至一清單。
4. 每一秒將輸出計數器列印到記錄檔，並將輸出計數器歸零，然後將步驟三中，儲存在已暫停連線物件清單內，每一個連線啟動。由於輸出計數器已歸零，下次發生寫入事件時，若輸出計數器小於頻寬限制，便可寫入 Socket。

由於 MDS Client 會偵測 Super Seeder 回傳 Piece 封包的速度，來調整發出 Request 片段請求的頻率，因此不必擔心連線內會囤積太多未處理的資料，而影響 Super Seeder 的效能與資源使用。

第五章、實驗結果

在本章節中會討論我的實驗設計，實驗結果以及結果分析。

5.1 實驗設計

在實驗環境的設計上，我們模擬一間遊戲公司使用具容錯性的 MDS 發布一個新的遊戲更新檔案，而客戶端則是模擬一般的家用 ADSL 環境。為了增快實驗的速度，我們將網路的傳輸速度設定為真實環境的五倍，並且在不受外界干擾的 Gigabit 內部網路進行實驗。

實驗環境設定如表格 3 至表格 5。

表格 3 實驗環境設定一、Client/Server 架構環境設定

設定項目	設定說明
實驗架構	Client / Server
傳輸檔案大小	50MB
Server 網路環境	上傳 2048KB/sec
Client 網路環境	下載 500KB/sec，使用 public IP
Client 數量	20 台，所有 Client 設定相同

表格 4 實驗環境設定二、MDS 架構環境設定

設定項目	設定說明
實驗架構	單一 MDS Server/單一 Super Seeder
傳輸檔案大小	50MB
Super Seeder 網路環境	上傳 2048KB/sec
MDS Client 網路環境	上傳 100KB/sec、下載 500KB/sec 使用 public IP
MDS Client 數量	20 台，所有 Client 設定相同

表格 5 實驗環境設定三、具容錯性 MDS 架構環境設定

設定項目	設定說明
實驗架構	兩台 MDS Server/兩台 Super Seeder
傳輸檔案大小	50MB
Super Seeder 網路環境	兩台上傳速度均為 2000KB/sec
MDS Client 網路環境	上傳 100KB/sec、下載 500KB/sec 使用 public IP
MDS Client 數量	20 台，所有 Client 設定相同

而在實驗的設計分成以下三大部分，第一部分是模擬傳統 Client/Server 的架構，而第二部分是不具容錯性的 MDS 架構，最後是具容錯性的 MDS 架構實驗。

實驗設計說明如表格 6。

表格 6 實驗內容說明

	實驗說明
實驗一	模擬傳統 Client/Server 架構
實驗二	MDS P2P 架構
實驗三	具容錯性 MDS P2P 架構

5.2 實驗結果與分析

5.2.1 實驗一、模擬傳統 Client/Server 架構

實驗架構如圖 5-1：

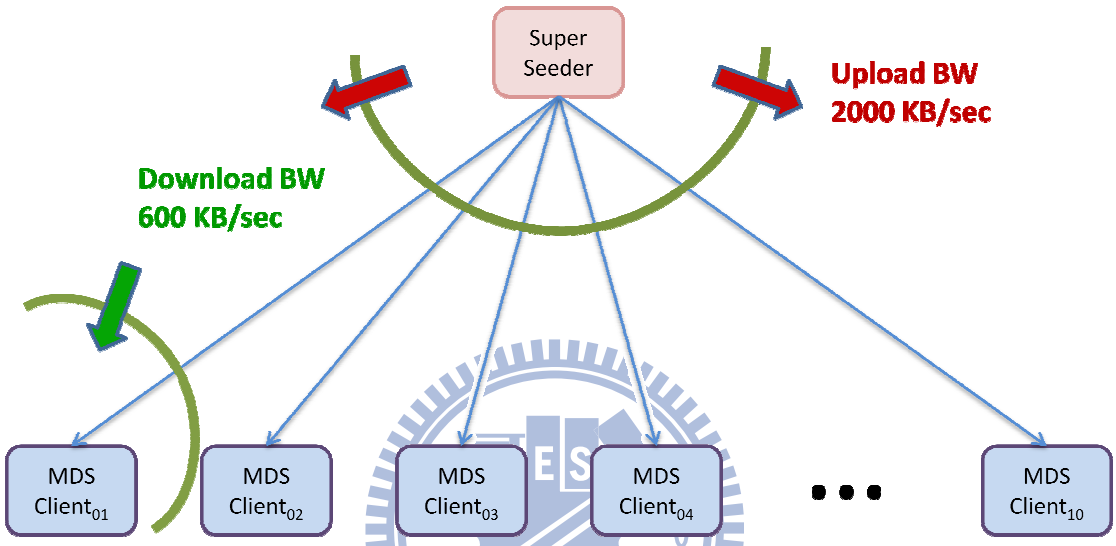
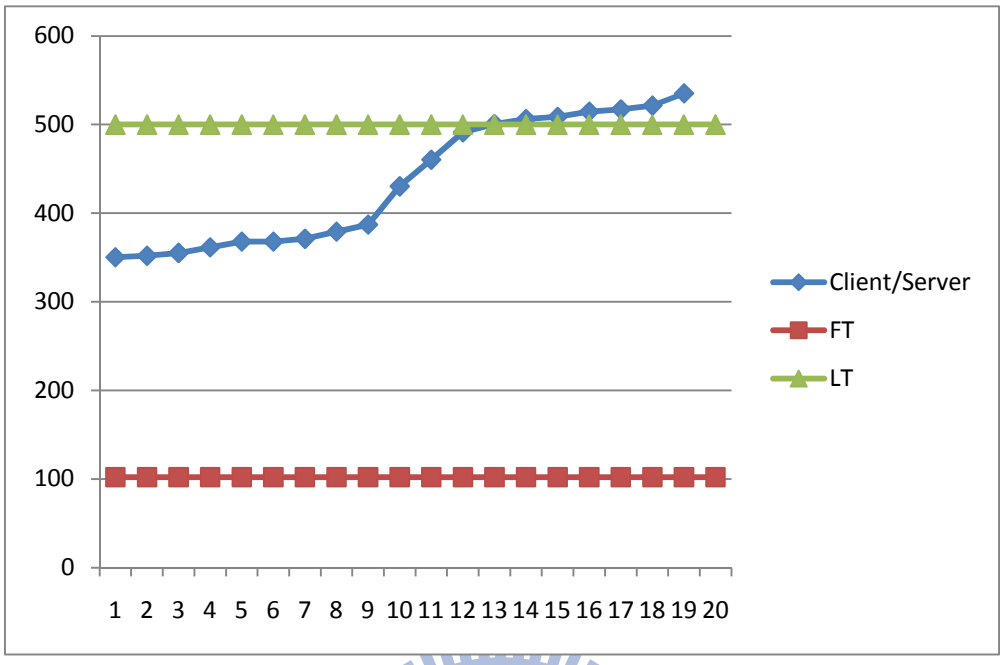


圖 5-1 實驗一 模擬傳統 Client/Server 架構

模擬傳統 Client/Server 架構實驗，主要是為了和 MDS P2P 以及具容錯性的 MDS P2P 作為本質上的比較，實驗結果如表格 7。

表格 7 中橫軸為 MDS Client 完成下載 File 的順序，左至右為最快至最慢；縱軸則為 MDS Client 完成 File 所花費的秒數。由於橫軸為快至慢的順序，因此曲線的走向必為左低右高。

表格 7 實驗一結果 模擬傳統 Client/Server 架構



下方標示 FT 的直虛線，為單一 MDS Client 最快完成時間，其計算方式為：

$$\frac{50\text{MB}}{500\text{Kb/s}} = 102.4\ \text{sec}$$

此虛線表示，無論系統效能再好，也無法超越單一 MDS Client 能夠達到的最快完成時間。

而 LT 為傳統 Client / Server 架構的理論整體最快完成時間，計算方式為：

$$\frac{50\text{MB} \times 20}{2048\text{Kb/s}} = 500\ \text{sec}$$

由表 7 可以看出來，實驗結果的曲線最後與理論最快完成時間交錯，證明這個實驗是有合乎預估的結果。

5.2.2 實驗二、MDS P2P 架構

實驗架構如圖 5-2：

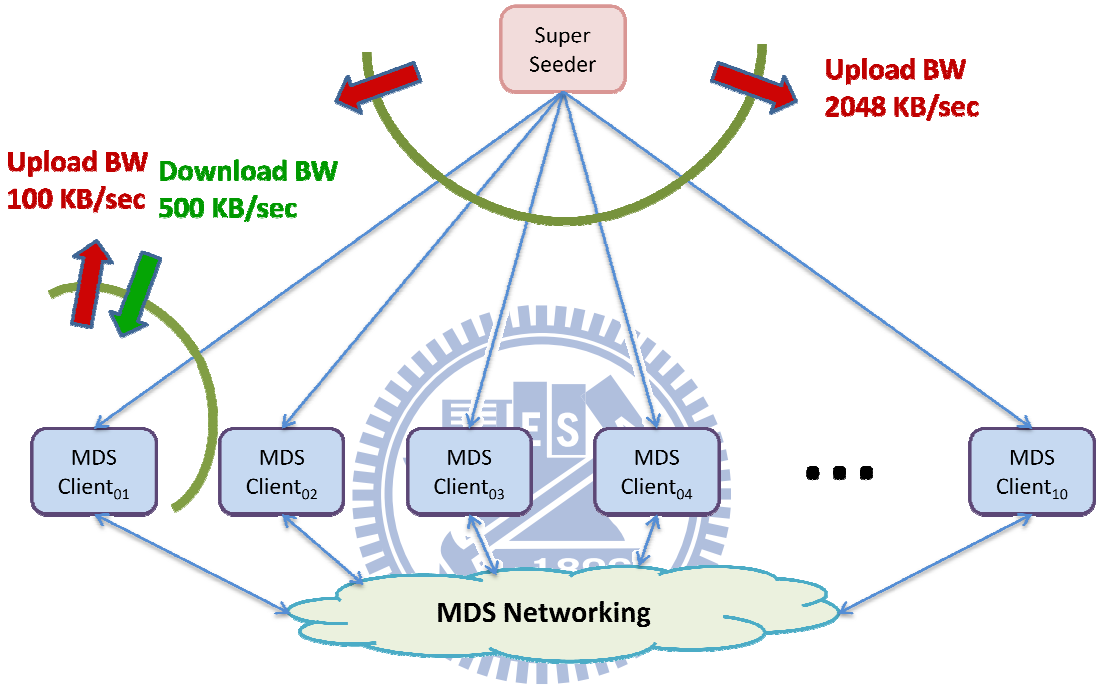
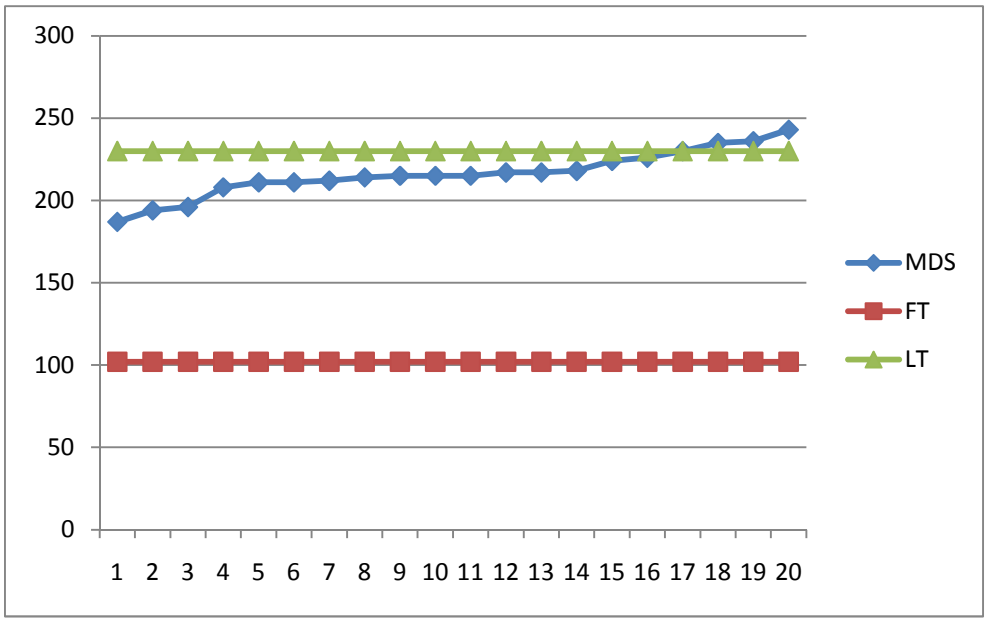


圖 5-2 實驗二 MDS P2P 架構

實驗結果如表格 8 所示：

表格 8 實驗二結果 MDS P2P 架構



下方標示 FT 的直虛線，為單一 MDS Client 最快完成時間，其計算方式為：

$$\frac{50\text{MB}}{500\text{Kb/s}} = 102.4\ \text{sec}$$

此虛線表示，無論系統效能再好，也無法超越單一 MDS Client 能夠達到的最快完成時間。

而 LT 為傳統 Client / Server 架構的理論整體最快完成時間，計算方式為：

$$\frac{50\text{MB} \times 20}{2048\text{Kb/s} + 100\text{Kb/s} \times 20} = 230\ \text{sec}$$

由表 8 可以看出來，實驗結果的曲線最後與理論最快完成時間交錯，證明這個實驗是有合乎預估的結果。

5.2.3 實驗三、具容錯性 MDS P2P 架構

這部分的實驗將對具容錯性 MDS P2P 架構與 FT-MDS P2P 來比較之外，並且針對各種錯誤狀況實驗，實驗列表如表格 9 所示。

表格 9 具容錯性 MDS P2P 架構實驗列表

	實驗說明
實驗三之一	MDS Server 上線兩分鐘後斷線
實驗三之二	Super Seeder 上線兩分鐘後斷線

實驗架構圖如圖 5-3 及圖 5-4：

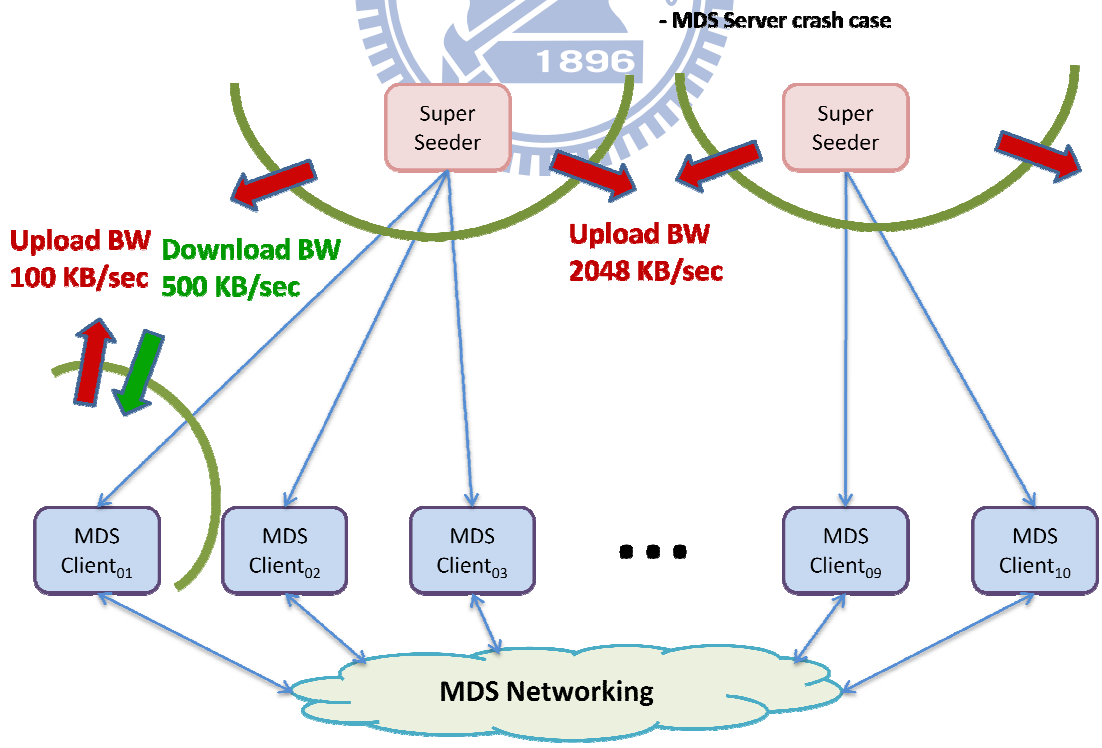


圖 5-3 實驗三之一 模擬一 MDS Server 斷線

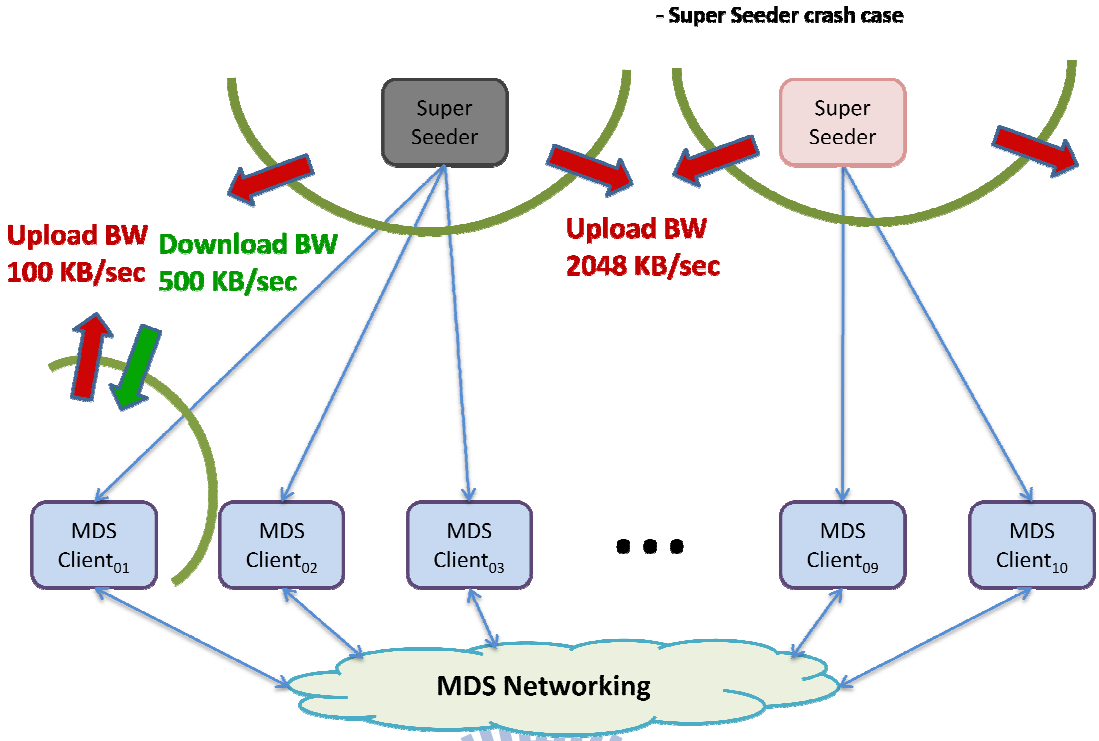
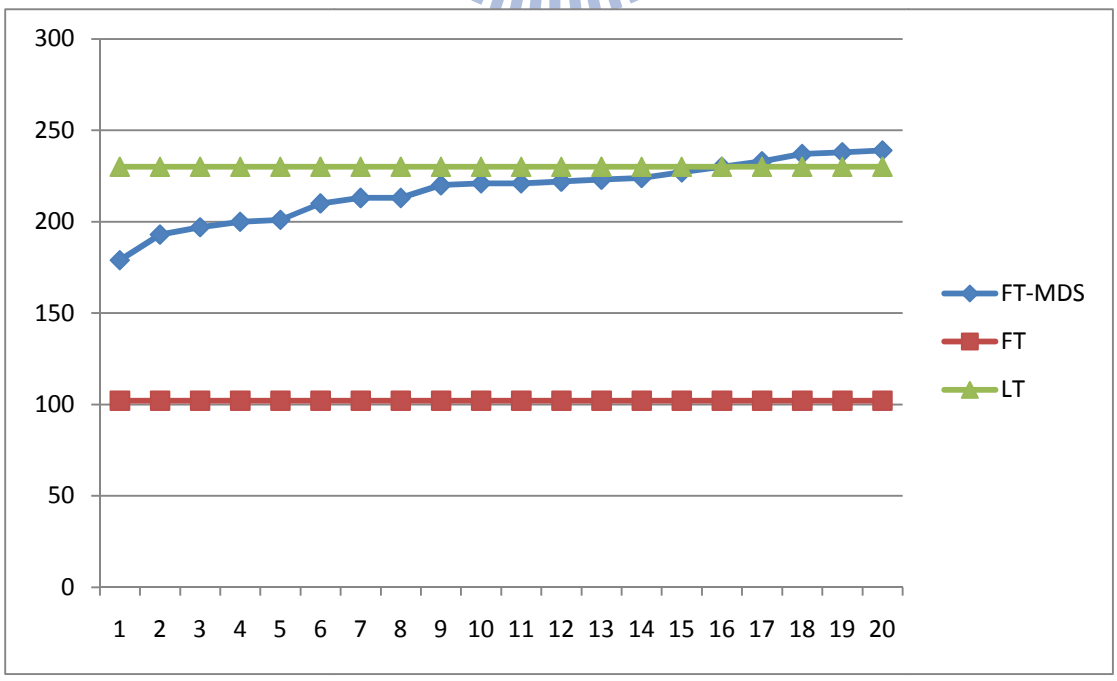


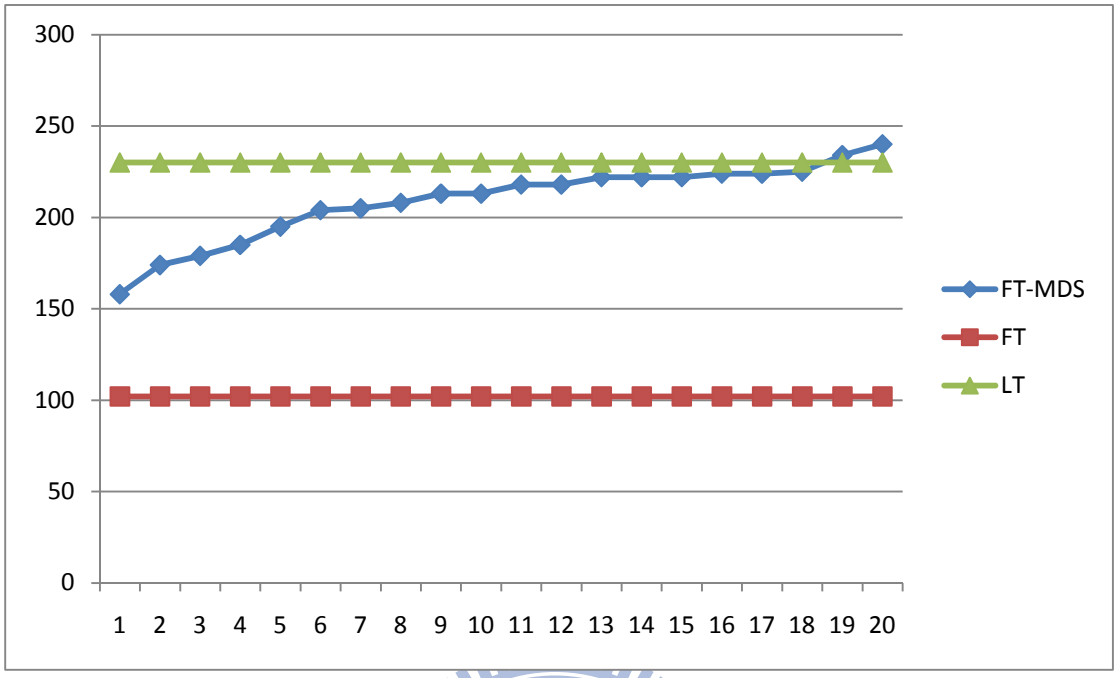
圖 5-4 實驗三之二 模擬— Super Seeder 斷線

實驗結果如表格 10，表格 11 所示：

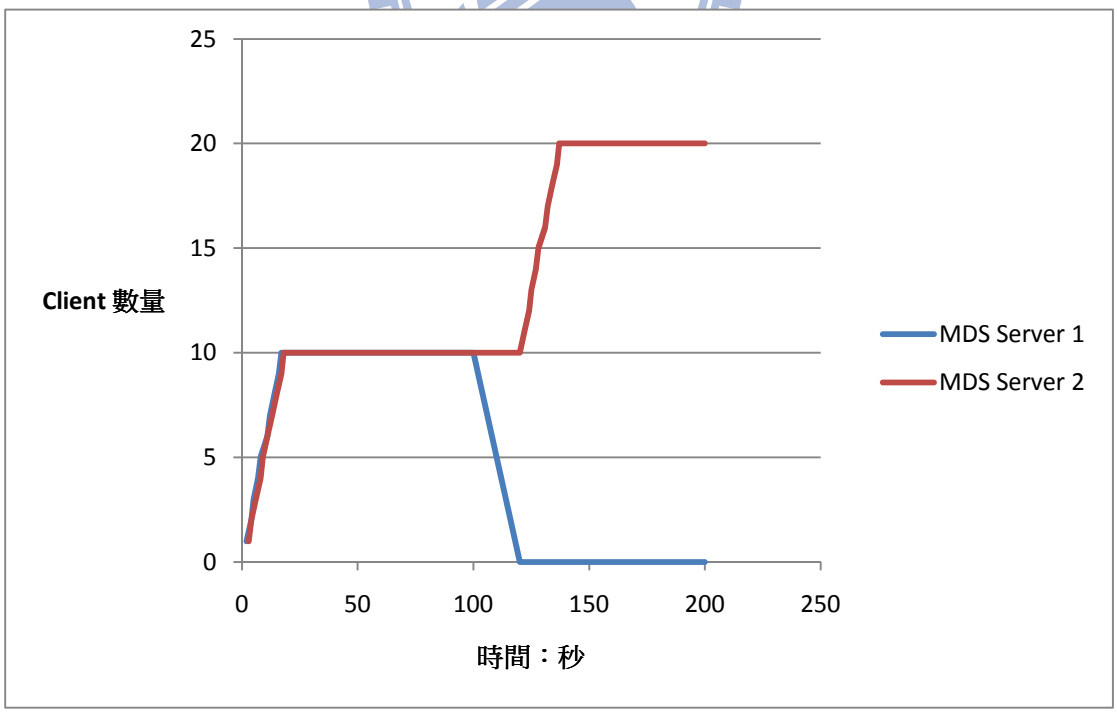
表格 10 實驗三之一結果 MDS Server 中途斷線



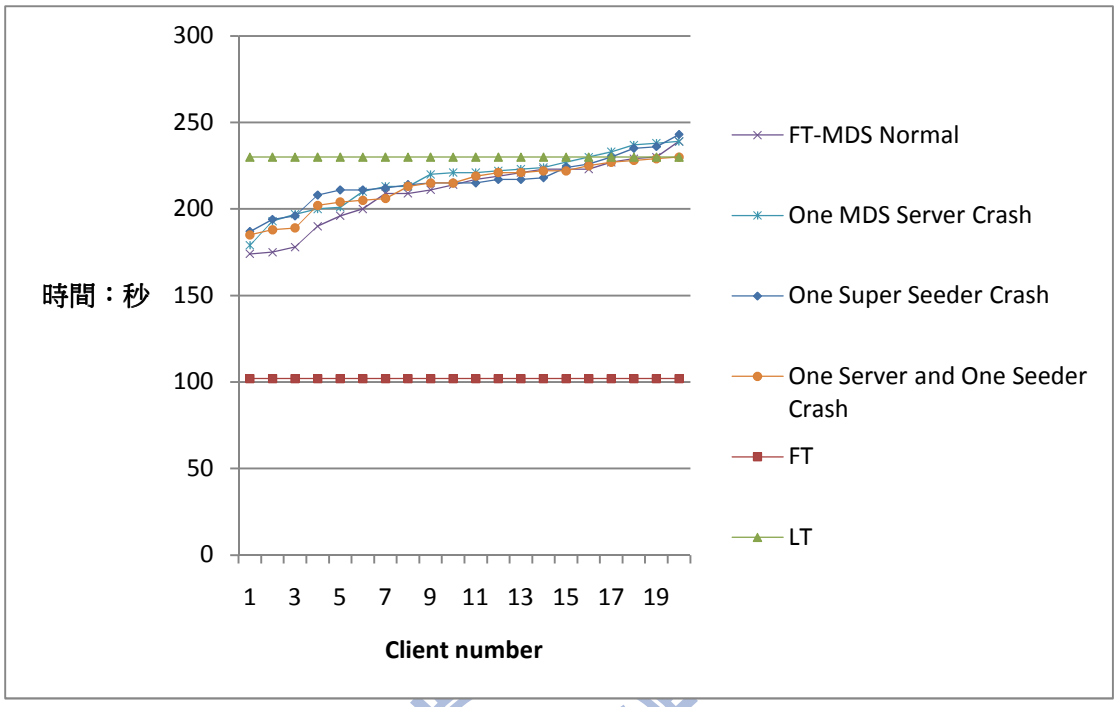
表格 11 實驗三之二結果 Super Seeder 中途斷線



表格 12 實驗三之一 一台 MDS Server 斷線狀況的兩台 MDS Server 負載狀況



表格 13 實驗結果整理



上表格 12 是在實驗三之一裡面，當其中一台 MDS Server 斷線後，兩台 MDS Server 所表示的 MDS Client 負載數量。由表格 13 可以得到此實驗證實當其中一台 MDS Server 發生斷線錯誤之後，MDS Client 的確陸續的登入到另外一台 MDS Server 取得下載服務。

而在上表格 13 中，首先第一條 FT-MDS Normal 的線段表示是具容錯性的巨量檔案發佈系統的客戶端下載完成時間曲線，而第二條 One MDS Server Crash 則是指在實驗經過兩分鐘後，將其中一台

MDS Server 斷線的客戶端下載完成時間曲線，第三條 One Super Seeder Crash 則是表示在實驗經過兩分鐘後將一中一台 Super Seeder 斷線的客戶端下載完成時間曲線，第四條線則是表示在實驗經過兩分鐘後，將其中一台 MDS Server 以及 Super Seeder 斷線之後，客戶端完成下載的時間曲線。從實驗結果表格 12 看出來，在一般的狀況之下，FT-MDS 系統有最佳的下載時間表現，並且合乎理論值，而在處理一台 MDS Server 斷線、一台 Super Seeder 斷線、以及一台 MDS Server 及一台 Super Seeder 斷線的狀況下，表現略有慢於一般的下載狀況，不過整體效能看起來，容錯機制所造成的額外負擔(overhead)，仍然能夠保有原本的執行效率。

第六章、結論與未來展望

6.1 結論

本論文實作出一套具有容錯性的巨量檔案發佈系統，解決了以下的問題，單一 MDS Server 當機或斷線時，將無法繼續提供服務。而藉由多套的 MDS Server 加上 MDS Server List 的使用可以讓系統持續提供服務直到剩下最後一套的伺服器。

過去只提供一個 Super Seeder 提供服務，當 Super Seeder 發生錯誤將會導致整體系統效能的降低。而現在搭配多套 Super Seeder，只要有一台 Super Seeder 可以使用，整體的系統將持續提供服務。

目前使用多套的 MDS Server 以及 Super Seeder 再加上 MDS Server List 的使用，使得系統可以提供更穩定的服務，再加上藉由 P2P 技術加速傳播檔案使得本系統更具可用性。

我們確實解決了以下問題：

1. 當系統中有其中一台 MDS Server 斷線後，會經由容錯機制使得 Super Seeder 以及 MDS Client 可以連線到其他的 MDS Server 繼續提供下載服務以及下載功能。
2. 當系統中有其中一台 Super Seeder 斷線後，會經由容錯機制，

MDS Server 可以發現有 Super Seeder 發生斷線錯誤，並且即刻置換提供服務的 Super Seeder，讓整體系統效能不受影響。

3. 即使系統發生一台 MDS Server 以及一台 Super Seeder 發生斷線錯誤後，經由容錯機制仍然可以在短時間之內，MDS Client 選擇其他可用的 MDS Server 來取得服務。
4. 當 MDS Server 發生斷線錯誤之後，再回到可提供服務狀態，提供系統清除垃圾資訊功能，避免系統有過多不能使用的垃圾資訊。
5. 當 Super Seeder 發生斷線錯誤之後，在回到系統之後，MDS Server 可以即刻的將 Super Seeder 切換進服務狀態，讓系統每個元件都可以平均負載的提供服務。
6. 整體系統即使只剩下一台 MDS Server 以及一台 Super Seeder 仍然可以提檔案下載服務，並且維持 P2P 的檔案傳輸特性，依然可以節省企業頻寬達到最快檔案發佈功能。

6.2 未來展望

在我們實作具容錯性的巨量檔案發佈系統的時候，在 MDS Server 部分有做了一個簡易的負載平衡機制，這個負載平衡只是很

簡易的方式並且不是即刻的負載平衡，未來可以在負載平衡的機制上，再進一步的研究，使得整體系統可以更快的達到負載平衡。

而目前的系統均為指令稿方式執行，系統已經趨近完善，未來也可以透過整合介面控制多套的 MDS Server 以及 Super Seeder 並且即時的監控所有 MDS Client 的檔案傳輸狀況，來確保所有的客戶端都能夠快速的取得檔案。

參考資料

[1] Che-Yi Lin, “The Study of Massive Deployment System Based on P2P Technology – The Servers and System", Master Thesis, National Chiao Tung University, August, 2008.

[2] 梁德馨,"九十八年度台灣無線網路使用調查報告", 財團法人台灣網路資訊中心 (TWNIC), July, 2009.

[3] Chou, Ting-Liang, "The Study of Massive Deployment System Based on P2P Technology - Clients and Performance Analysis", Master Thesis, National Chiao Tung University, August, 2008.

[4] Shawn Fanning, "Napster", http://www.napster.com, Napster, Inc., 1999, 2004.

[5] Justin Frankel, Tom Pepper, "Gnutella", http://en.wikipedia.org/wiki/Gnutella, 2000.

[6] MetaMachine.com, "eDonkey2000 (eDonkey, ed2k)", http://www.edonkey2000.com, 2000.

[7] eMule-Team, "eMule", http://www.emule-project.net, 2002.

[8] Berkeley Software Distribution, 4.2BSD, select(2), http://www.freebsd.org/cgi/man.cgi?query=select"

[9] AT&T, Inc., System V UNIX, poll(2), http://www.freebsd.org/cgi/man.cgi?query=poll

[10] Dan Kegel, "The C10K Problem", http://www.kegel.com/c10k.html, September, 2006.

[11] FreeBSD 4.1, *kqueue(2),* http://www.freebsd.org/cgi/man.cgi?query=kqueue.

[12] Linux 2.5.44, *epoll(4),* http://www.xmailserver.org/linux-patches/epoll.txt.

[13] Niels Provos, "libevent API", http://monkey.org/~provos/libevent, 2008.

[14] Sun Microsystems, Inc., Solaris, */dev/poll,* http://access1.sun.com/techarticles/devpoll.html.

[15] Sun Microsystems, Inc., Solaris, *event ports,* http://developers.sun.com/solaris/articles/event_completion.html.

[16] Regents of the University of California, 3-clause BSD License, http://www.linfo.org/bsdlicense.html, 1990.

[17] R. Braden, RFC 1122, TCP Keepalive, http://www.faqs.org/rfcs/rfc1122.html, October 1989.
K. Egevang, P. Francis, "RFC 1631 The IP Network Address Translator (NAT)", http://tools.ietf.org/html/rfc1631, May, 1994.
[18] J. Rosenberg, J. Weinberger, C. Huitema, R. Mahy, "*RFC 3489,*STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)", http://tools.ietf.org/html/rfc3489, March, 2003.
[19] Bryan Ford, Pyda Srisuresh, Dan Kegel, "Peer-to-Peer Communication Across Network Address Translators", http://www.bford.info/pub/net/p2pnat, February, 2005.
[20] J. Rosenberg, R. Mahy, P. Matthews, "IETF Behave Draft, Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)", http://tools.ietf.org/id/draft-ietf-behave-turn-09.txt, July, 2008.