國立交通大學

電機學院通訊與網路科技產業研發碩士班

碩 士 論 文

串流伺服器資料解析及封包包裝

Parsing and Packetization in Streaming Server

研 究 生：陳瑩甄

指導教授：張文鐘　教授

中 華 民 國 九 十 六 年 八 月

串流伺服器資料解析及封包包裝
Parsing and Packetization in Streaming Server

研 究 生：陳瑩甄　　Student：Ying-Jen Chen

指導教授：張文鐘　　Advisor：Wen-Thong Chang

國 立 交 通 大 學

電機學院通訊與網路科技產業研發碩士班

碩 士 論 文

A Thesis

Submitted to College of Electrical and Computer Engineering

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Industrial Technology R & D Master Program on

Communication Engineering

August 2007

Hsinchu, Taiwan, Republic of China

中華民國九十六年八月

串流伺服器資料解析及封包包裝

學生：陳瑩甄　　指導教授：張文鐘 博士

國立交通大學電機學院產業研發碩士班

摘　　要

現今在即時多媒體應用上的主要課題在於即時(Real-time)的傳輸實況影音與已儲存的多媒體檔案，而多媒體串流技術可以讓使用者在下載多媒體檔案的同時可以先觀賞到已下載的部份，此項技術使得使用者不需要花費下載時間與硬體空間來儲存多媒體檔案。

而一個多媒體串流伺服器包含了三個主要的模組，這三個模組各別為:「標準連線程序建立」、「RTSP 信息溝通協調」與「封包包裝和傳送」。而在本論文中主要討論是「封包包裝和傳送」模組，此一模組是整個伺服器運作的重心所在，最主要在於實現檔案資料的解析與特定的封裝演算法，產生適合網路環境的封包並且傳輸。

從 IETF 所制定的標準中可以知道 RTP 對不同影音檔案皆有其建議的封裝方式，本篇論文主要是針對這些封裝方式做探討，著重於影音檔案資料 parser 的運作機制，且提出一個架構在「封包包裝和傳送」模組內的 RTP 封包演算法，並藉由分析 Live555 此套 open source 的架構做為輔助及驗證 RTP 封包演算法是如何被實現的，在本論文中，所要探討的主要在於已儲存的多媒體影音檔案的部份，並以 MPEG 家族的影片格式做為實驗對象。

Parsing and Packetization in Streaming Server

student：Ying-Jen Chen　　Advisors：Dr. Wen-Thong Chang

Industrial Technology R & D Master Program of
Electrical and Computer Engineering College
National Chiao Tung University

ABSTRACT

Today, the main points of real-time multimedia applications are the transmission and the play of live video or stored files. Streaming technology allows people to enjoy the multimedia contents while downloading them. With the advantages of this technology, users don't need to spend extra time when downloading multimedia contents or spare extra memory space to save them.

A streaming server will consists of three important modules, they are " Set up a standard connection procedure "," RTSP Signaling Negotiation " and " packet Packetization and Transmission " respectively.

This discourse focuses on the module of "packet Packetization and Transmission " which is the key point to operate well in server, and to realize the analysis of file information and specific packetization algorithm which produces the packet adapting the internet environment.

Because of standard define by IETF, we can know that different multimedia files have different RTP payload formats. The topics of this thesis mainly discuss the multimedia files' parsing, Packetization, and transmission modules. Finally, we analysize the open source "Live555" to verify how the above three modules are implemented.

誌謝

能夠順利完成此篇論文，最要感謝的人是我的指導教授 張文鐘博士。老師在我二年的碩士生涯中，不管在學業或生活上都給予了相當多的幫助與指導，尤其在整理論文以及準備口試的期間中，難為老師如此辛苦與花費許多時間，不斷的為我指出研究中的盲點和問題的關鍵，使得此篇論文能更趨於完備。

另外要感謝的，是實驗室裡的同學們，這兩年一起修課、在實驗室唸書寫報告趕論文的日子裡，感謝他們為我帶來了美好的回憶。另外，要特別感謝的是孟潔、素仙與小邱，在我研究上遭遇到問題時，經由與你們的討論才能逐一的解開我的困惑與滯礙，讓我能夠順利的往前前進。

最後，我要感謝我的家人，父母親、姐姐們和其他好友們的支持與鼓勵，總讓我能心無旁騖地從事研究工作，並在我遇到挫折與低潮時讓我能夠重新振作。

感謝所有幫助過我、陪伴我走過這一段時光的師長、同儕、朋友和家人。

謝謝！

誌於 2007.08 風城 交大

Ying-jen

目錄

圖目錄

表目錄

第一章 緒論

1.1 研究背景

現今在即時多媒體應用上的主要課題在於即時(Real-time)的傳輸實況影音與已儲存的多媒體影音檔案，在本論文中，所要探討的主要在於已儲存的多媒體影音檔案的部份。在網際網路上，目前傳輸已儲存的多媒體影音檔案可分為兩種方式，一種為下載模式(download mode)，另一種為串流模式(streaming mode)；在下載模式中，使用者必需要將整個影音檔案完整的下載後才能夠完全播放此一影音檔案，雖然可以確保播放多媒體影音檔的輸出品質，但在網際網路上傳輸檔案所需花費的時間常常是相當冗長且讓使用者感到難以接受，尤其多媒體影音檔案容量通常都非常的大，且大多時候所下載的影片檔只是單純的用來觀賞及娛樂，並沒有保存的必要性，對於許多無法裝設大容量硬碟的裝置，像是行動電話、PDA...等，要將整個檔案下載完畢並不可能或是較為困難，且即使下載完畢也顯得浪費空間資源；同時在網路的頻寬限制下，也需要一個更好的方式來解決漫長的等待時間，因此發展出將多媒體影音檔切割成許多封包的格式，使用連續流動(Flow)的方式來傳輸，這就是串流模式。

串流模式是一種具有即時(Real-time)特性的多媒體傳輸方式，其隨著通訊網路而蓬勃發展，它的發展目地在於如何透過通訊網路即時傳輸多媒體的音訊和視訊資料，而客戶端不需要花費大量的時間等待影音檔案下載完畢，只需要下載極小部份的資料便可以立即開始播放；其優點除了可以縮短客戶端下載所需要的時間外，更可以節省龐大的磁碟暫存空間，達到智慧財產權的保護等。

現今的串流模式在通訊網路上有兩種主要的傳輸方式，第一種是以 HTTP/TCP 為基礎，使用 HTTP 協定可以讓串流媒體順利的通過防火牆的攔阻，而 TCP 是一種傳輸層協定（transport-layer protocol），主要是提供較高可靠性的資料傳輸，但缺點在於 TCP 通訊協定會導致傳輸速度的減慢，使得整個傳輸頻寬加大；第二種方式是使用 RTP/UDP，UDP（user datagram protocol）有別於 TCP，是一種不可靠協定，在傳輸上不能保證每個被丟出的封包一定會到達目的地，在接收的順序上也沒有按照傳送順序接收，而 RTP(Real Time Protocol)，是由 IETF 中的 AVT 這個 working group 所制定，並定義於 RFC-3550[1]；根據 RFC-3550，RTP 為即時資料傳輸提供點對點（end-to-end）網路發送服務，包含了 payload type identification、sequence number 和 timestamp，利用 sequence number 來進行封包的重建，而 timestamp 則帶有各影像、音訊封包所應該播放的時間資訊，如此便可以增加即時傳輸的品質。

且在現今的商業市場裡，以串流模式傳輸多媒體影音的伺服端並沒有一個統一或是權威性的標準規格出現，也因此各家廠商所推行的伺服器大多都只能對應自己所開發的標準或是特定的通訊協定，形成百家爭鳴的狀況，也使得一套伺服器通常都只能對應某一套編碼/解碼格式；目前市場中較為主流的多媒體串流影音伺服器分別為 Microsoft 的 Media ServicesR、RealNetworks.com 的 RealSystemR 和蘋果電腦的 Quick Time ServerR。由於沒有一套統一性規格的多媒體串流伺服器及編解碼標準，且各家產品各有千秋，也因為商業上的激烈競爭要有所統合也較為困難，因此 Open Source 在整合出統一的多媒體串流伺服器上將扮演著舉足輕重的角色。

1.2 研究動機

一個多媒體串流伺服器包含了三個主要的模組，這三個模組各別為:「標準連線程序建立」、「RTSP 信息溝通協調」與「封包包裝和傳送」。「連線建立標準程序」模組主要使用 BSD Socket API 來完成一個伺服器的初始化設置，也就是依串流程式的需求來建立 TCP/UDP 模式的 socket。「RTSP 信息溝通協調」模組主要允許串流伺服器經由 RTSP 信令來完成與用戶端的連線溝通[2]。

而「封包包裝和傳送」模組，是整個伺服器運作的重心所在，最主要在於實現特定的封裝演算法，產生適合網路環境的封包並且傳輸。在此模組中，伺服器必須要能夠分辨客戶端所要求串流傳輸的影音檔案格式，因為多媒體影音檔案格式類型眾多，像是 mpeg、m4v、avi...等，這些檔名皆是代表不同的包裝方式，表示其壓縮編碼的方法各有其獨特的地方，而在一個隨選即播的視訊系統裡，儲放在伺服器端的檔案會是各種包裝格式的其中一種，伺服器必須要有能力分辨出不同的格式，進而從檔案中抽取出視訊流與音訊流，並依據檔案格式的特性進行資料切割且包裝成特定型式的封包來進行傳送，而這樣的一個結構化的封包產生便是經由某一種特定的封裝演算法來獲得，也是「封包包裝和傳送」此一模組的功能所在。

封裝演算法最主要的目的在於讓多媒體串流伺服器根據網路使用環境來選擇適當的封包包裝方式，藉以達到封包使用率以及封包錯誤恢復能力(error-resilience)之間的最佳化關係[2]。因此在有線、無線、不同傳輸速率或是使用不同通訊協定的網路環境下就各有適當的封裝演算法。而本篇論文中所要探討的是一個串流多媒體伺服器如何實現 RTP

封裝演算法，如何解析檔案資料及將不同編碼格式的傳輸資料包裝成一個 RTP 封包可以負載的形式。

一個 RTP 封包分成檔頭(header)和負載資料(payload)兩部份，RTP packet 的檔頭格式在 RFC3550 中有明確的定義，而負載資料的格式的切割方式會依據不同的多媒體資料而有不同的定義，像是在 RFC 2250 [3]就定義了 MPEG-2 的 RTP 封包包裝方式，而 MPEG-4 則定義在 RFC 3016[4]中。而在「封包包裝和傳送」模組中首先會遇到的問題在於伺服器端要如何辨認出它所要串流的影音檔案包裝格式為何?伺服器端是如何從這些影音檔案中抽取出視訊流和音訊流?又是如何對不同的影音檔案的資料進行切割成合適的 RTP payload ?唯有將這些問題解決，才能產生出一個個的 RTP 封包進行傳輸，而達到串流的目的。

在 IETF 所制定的標準中可以知道 RTP 對不同影音檔案的包裝方式皆有其建議的封裝方式，本篇論文主要是針對這些封裝方式做探討，著重於影音檔案資料的解析分割也就是 parser 的運作機制，且提出一個架構在多媒體串流系統上的 RTP 封包演算法，並藉由分析 Live555 此套 open source 的架構做為輔助，用以驗證在真實的網路世界裡 RTP 封包演算法是如何被實現的，串流影音檔案格式則主要以 MPEG 家族的影片格式為實驗對象。在我們所討論的這個演算法中，所含蓋的範圍非常的廣泛，包括檔案格式的辨認、影像及聲音的資料處理、封包的組成與傳輸...等，這些每一部份都是很重要的環節，需要我們一一的去剖析並了解每個環節的因果關係，這樣才可以開發出可重覆使用的軟體模組，使其能夠擴充及包容更多不同檔案類型，達到整合的目地。

1.3 論文章節提要

在深入探討一個實體多媒體串流伺服器的 RTP 封包演算法之前，我們必須要先了解網路上的即時傳輸基本概念以及多媒體影音檔的編碼格式，因此在第二章，會依序的介紹串流媒體（Streaming Media）與協定、即時傳輸協定（Real-Time Transport Protocol）、多媒體影音檔的編碼格式與 RTP 對不同多媒體檔案的包裝格式說明；而在第三章中將說明在一多媒體串流伺服器上的 RTP 封包演算法完整運作流程，包含檔案格式的辨認、RTP 封包的產生、檔案資料如何經由 parser 而分離出視訊流及音訊流...等，在此部份會描述如何包裝 MPEG-4 simple profile 及 MPEG-2 system 於 RTP payload 內作為例子。第四章主要是分析 Live555 伺服端的架構作為驗證；第五章為實驗結果；第六章則是為本篇論文作出結論。

第二章 多媒體串流系統相關背景知識概論

在本章節中將逐一的說明在多媒體串流系統中的「封包包裝及傳送」模組所使用到的各項背景知識，包括了串流媒體與通訊協定、多媒體檔案的編碼方式和 MPEG 家族的 RTP payload 包裝方式說明。

2.1 串流媒體與協定

2.1.1 串流媒體文件格式說明

一個多媒體串流系統可以透過網際網路存取遠端檔案，也可以從攝影機或是麥克風擷取獲得。一個串流檔案的文件格式包含了兩層涵意，一種是文件系統格式(container type)，即是定義如何將各種媒體組織在一起，可能是單獨的音訊和視訊，如 MP3、M4V，或者是結合了音訊與視訊，像是 AVI、MPEG、QuickTime...等；另一個就是媒體內容本身的編碼方式(codec type)，也就是文件中的各種媒體內容是通過什麼樣的方式來完成編碼的，比如常見的 MPEG1、MPEG2、MPEG4、AC3 等等。

通常一個多媒體串流都是由多個通道(channel)組成，一個 channel 就是建構單獨的一層 track，每一 track 都代表著一種資料類型，例如：audio、video。而一個 mpeg 檔就是由音訊頻（audio track）和視訊頻（video track）組合而成。

在多媒體串流系統運作上，伺服端須要知道客戶端要求傳輸的媒體檔的文件系統格式，才能依據不同的文件格式特性進行解析去分離出音訊頻和視訊頻，進而得到每一 track 的資料結構也就是編碼格式；伺服

端會因應不同的編碼格式對媒體檔進行解析與包裝。

2.1.2 串流媒體通訊協定

目前網路封包技術的標準有 TCP/UDP、RTP/RTCP...等，然而對於一個多媒體串流系統而言，若是使用 TCP(Transmission Control Protocol)傳輸封包，客戶端將會因為網路上的封包遺失，而時常面臨難以忍受的畫面延遲。這是因為 TCP 所提供的是一個較高可靠性的資料傳輸，TCP 會嘗試回復遺失的封包，而封包回復的作法就是要求來源端重新傳送一次封包。如果網路上的封包遺失嚴重，客戶端則會發出多次封包重送的要求。此時的客戶端必須在等待重送的封包抵達之後才能進行進一步的影音服務。等待的時間是漫長的，對於使用者而言這是讓人感到不悅的使用經驗。為了避免這樣的狀況，多數人選擇使用 UDP 傳送封包。

有別於 TCP 的協定，UDP(User Datagram Protocol)在傳輸上不能保證每一個被送出的封包一定會到達目的地，在容易遺失封包的環境下 UDP 並不是個可靠的協定。UDP 不提供封包遺失偵測的功能，也不支援重送，但在網路擁塞的狀況下，多媒體串流系統使用 UDP 的機制能避免客戶端等待過久的時間。

即時傳輸協定（RTP: Real-Time Transport Protocol）在 UDP 的上一層工作，它是項能彌補 UDP 不足之處的協定，且也利用了 UDP 的多工(multiplexing)及錯誤位元檢測(checksum service)兩項功能。事實上，多媒體串流系統傳輸多媒體內容封包之前，須先做適當的切割，並包裝成為 RTP 封包，接著利用 UDP 與 IP（Internet Protocol）傳輸協定依序傳送到網路上，而 RTP 的資訊即是被封裝在 UDP 的封包裡，RTP 為即時資料傳輸提供端點對端點（end-to-end）網路發送服務，也

提供了下列功能：content type identification, sequence numbering, timestamping...等。下個小節中將會詳述。

2.1.3 即時傳輸協定（RTP: Real-Time Transport Protocol）

RTP[1]是為了處理具有即時特性的資料，而制訂一個屬於應用層的端對端(End-to-End)傳輸通訊協定。它主要的目的在於支援即時性的影音資料傳輸，將媒體檔案切割成數份的封包，搭配 RTP header，再交給較底層的傳輸協定如 UDP 傳送。而一個 RTP header 中包括承載格式（Payload Type）、序號（Sequence Number） 和時間戳記（Timestamp）等欄位，序號和時間戳記可以讓接收端重新排列出封包的正確順序，以及資料應該被播放的正確時間點，例如：當多個封包中的時間戳記相同時，接收端就可以利用序號來偵測是否有封包遺失。

RTP 此一協定並不具有確認傳送的時間與保證傳送服務的品質（QoS）的機制，這些功能只能依賴著較低層級的網路服務來提供。RTP 不會去防止封包不按照順序傳送，也不會保證封包一定會送達；其封包裡所包含的序號，會幫助接收端依照發出的順序重建資料，決定封包適當的位置。

2.1.3.1 RTP Packet format

Figure 2.1 RTP Packet Format

如同 figure2.1 所表示，RTP 封包是由 Header 和 Payload 兩部份所組成，RTP 封包的 Header 格式在 RFC 3550 中有明確的定義，在

2.1.3.2 節中會有詳細的說明。Payload 可為視訊或是音訊的資料，而不同格式的檔案資料要如何置入 payload 部份中才能使用 RTP 傳輸影音資料則另有規範，比如 RFC 2250 中規範了 MPEG 1/2 payload，而 MPEG-4 AV payload 則定義在 RFC3016 中，此部份在之後的章節將會有更進一步的說明。

2.1.3.2 RTP Header Information

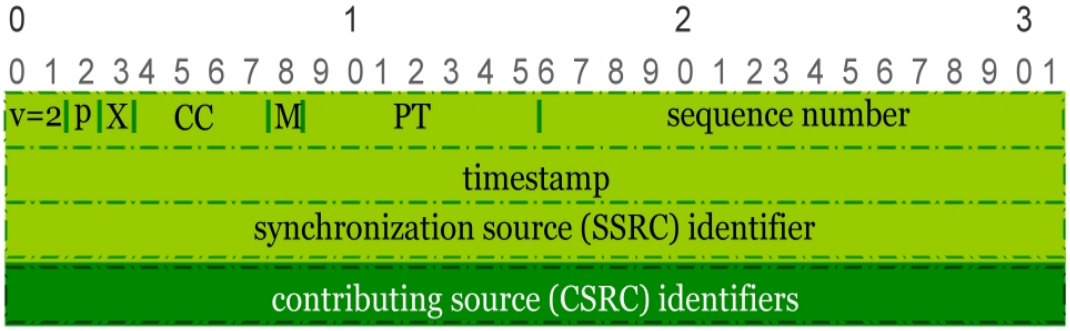

Figure 2.2[1] RTP Header Format

如 figure2.2 所表示，RTP header 欄位的前 12 個位元組是每一個 RTP 封包都具備的，而 CSRC 辨別碼欄位是否存在則端看封包是否有經過混合器而定。下面將依序介紹每個欄位的代表的意義：

- Version (V)：2 bits

  指出代表 RTP 協定的版本。目前所使用的版本為 2。

- padding (P)：1 bit

  表示封包的尾端是否有 padding。Padding 的原因通常是為了加密，或者是為了交由較底層的協定進行傳輸。Padding 最後一個位元組代表 padding data 有多少個位元組，padding data 並不屬於承載資料的其中一部份，可用於各種加密演算法。

- extension (X)：1 bit

表示是否進行 RTP header extension，當此 bit 設定為 1 時，表示在 RTP header 後面還會在跟隨額外的 header。

- CSRC count(CC): 4 bits

由於多媒體串流技術支援多個來源，換句話說一個 RTP 封包可能會由兩個以上的來源經由 Mixer 組合而成，CSRC(Contributing Source identifier)代表參與此次串流通訊來源的 ID，而 CC 則是表示此 RTP 封包是包含了多少個 CSRC。

- Marker (M)：1 bit

這個 bit 是由規格文件（profile）所定義，用來標示重要事件的旗標，例如在此封包中的 payload data 有無 frame 的邊界。

- Payload type (PT)：7 bits

這個位元組是用來辨認 RTP 承載的內容型態，相關定義會在 2.3 節中說明。

- Sequence number：16 bits

每送出一個 RTP 封包，就會在序號欄位上加一，序號的起始值是在 0~65535 中亂數決定的;在串流一開始時亂數決定 sequence number 是為了避免加密的明文攻擊（known-plaintext attack）。藉由此一數字，可以幫助接收端偵測出所收到的封包是否有遺失，以及將順序錯誤的封包進行重新排列。

- Timestamp：32 bits

Timestamp 是表示 RTP 資料封包的第一個位元組取樣的瞬間。而這個瞬間則是由時鐘線性的時間變化量所取得，可以用來同步化並計算抖動時間偏移（jitter）的情形。時鐘的頻率是依承載所攜帶的資料格式而不同，也可能會在規格文件或是定義承載格式的規格中規定，以 MPEG-4 來說，其頻率為 90kHz。

在傳輸即時擷取的影音資料時，timestamp 所反應的是 payload

data 第一個 byte 的取樣時間；當所要傳送的資料是已經儲存好的影音檔案時，則這個值所表示的為 payload data 的播放時間點，當連續好幾個 RTP 封包的 payload data 是屬於同一張影像時，則 timestamp 的值都會相同，且其初始值和 sequence number 相同都是經由亂數產生。

- SSRC：32 bits

Synchronization Source (SSRC)是用來辨識 RTP 資料來源。不同的資料流會有個別的 SSRC 值，這個辨別碼在任何一個 RTP session 中都是隨機亂數產生，且是獨一無二的。

- CSRC list :0 到 15 個項目，每項需要 32 bits

CSRC list，當來源資料流被合併時，存放合併過的資料流的 SSRC 值，因此這個欄位並不一定存在。辨別碼的數目是由 CC 欄位提供，由於 CC 只允許 4 個位元，故存放數目由 0~15 個 SSRC，如果有超過 15 個傳送端的話，也只能記錄 15 個。

2.1.4 實時流協議 (RTSP :Real Time Streaming Protocol)

RTSP(Real Time Streaming Protocol)[5]是由 RealNetworks 和 Netscape 共同提出的，此一協議定義了一對多應用程序如何有效地通過 IP 網絡傳送多媒體數據。RTSP 在整個傳輸體系結構上位於 RTP 和 RTCP 之上，它使用 TCP 或 RTP 完成數據傳輸。RTSP 支援客戶端和伺服端間雙向溝通，兩端都可以發出請求，使用者可以透過 RTSP 下指令給伺服器，像是“PLAY”、“PAUSE”等動作。

RTSP 可以稱為一個對多媒體服務做遠端控制的通訊協定，RTSP 在語法上與 HTTP 非常相近，都是屬於純文字的通訊協定，其請求的訊息格式為被請求者的 URL 位址以及協定版本，接著是 header 及訊息本文。訊息本文可能包含一“會議描述(session description)”— 描

述該會議中串流媒體的訊息，此一會議描述的格式並未限定，但目前大部分皆以 SDP[6]為其訊息格式。運用 RTSP 協定中所定義的各種方法(method)，可以達成對串流媒體的各種控制，RTSP 協定中的 method 會在第四章中說明。

2.1.5 SDP (Session description Protocol)

SDP [6]是一種以文字格式描述多媒體會議(session)的通訊協定，嚴格說 SDP 並不算是一種通訊協定，而在內涵上比較傾向於類似 HTML 的標記語言。SDP 可以被包含在 SIP、RTSP、SAP、HTTP 甚至 E-Mail 等各種協定中；SDP 可以包含各種有關於 session 的描述，像是媒體種類(Video/audio/shared application 等)、媒體傳輸協定(RTP/UDP/IP 等)、媒體格式(H.261 video/MPEG video/G.723.1 audio 等)、還有一些有關安全方面的訊息(如該媒體會議的加密值)等資訊。

整個 SDP 的內容可以歸納成三部份:

- 第一部份包含了 session 名稱和目的以及該會議活動的時間。
- 第二部份則是組成該會議的媒體種類與接收這些媒體的控制信息。
- 第三部份與控制信息相關。

2.2 多媒體檔案編碼格式介紹

現今在網際網路上常見的串流媒體編碼格式包括了有需要較高頻寬的 MPEG-1、MPEG-2 外，另外還有 Real Video、QuickTime、WMV、MPEG4 及 H.264...等格式，如 table2.1 所示。

在此小節中將介紹由 ISO 13818-1[7]所定義的 MPEG-2 system 包括了現今 DVD 所使用的 mpeg-2 program stream 以及用於數位廣播電

視上的 transport stream；除此之外也會說明規範於 ISO 14496-2[8]中的 mpeg-4 simple profile。

Table2.1 串流媒體類型

編碼格式 (Codec Type)	副檔名 (Content Type)	說明
MPEG 動態壓縮標準	.mpg (.mp3)	包含了 MPEG-1 和 MPEG-2。 MPEG-1 是最早定義出的影音格式，最熟知的有 MP3、Vedio CD；MPEG-2 並不是用來取代 MPEG-1 的格式，二種規格自有其不同的應用層面，最常見的的應用為數位電視、DVD Video 與 SVCD。
	.m4v	MPEG-4 是目前最新的壓縮格式標準，主要針對低速頻寬的影音視訊編碼的目的，讓影音檔案更小，但品質幾乎與 DVD 無異，在網路上的傳輸更便利。
	.mp4	為音樂檔案的一種，依循著 MPEG 所制訂的 MPEG-4 影音格式編碼，以 Advanced Audio Coding（AAC）Codec 編碼技術更有效地將 CD 音樂轉檔壓縮而成的數位音樂格式檔案。
Quick Time	.mov	由 Apple 公司開發出來的檔案結構格式，QuickTime 沒有定義其視訊檔案實際的壓縮技術，而只是定義了視訊的架構。
ASF	.asf	Advanced Streaming Formate 的簡稱，是微軟針對串流影音應用所提出的影音新規格。ASF 檔案可以是任何的編碼方式為基礎的影音編碼，而且仍然是 ASF 格式。然因為 ASF 檔案主要是為了提供在網路上直接觀看，所以品質較 VCD 來的差。
WMV	.wmv	由微軟公司主導所發展出來的多媒體視訊格式，也是微軟所有視訊編碼方式的通稱，該檔案的特點是檔案小，有利於網路上的即時傳送播放，是一種網路上常見的視訊格式。

續 Table2.1

編碼格式 (Codec Type)	副檔名 (Content Type)	說明
RM	.rm .rmvb	Real Networks 公司所制訂的檔案格式，是一種可以應用在網路上的多媒體串流檔案格式，也是最早在網路上流行的多媒體格式檔案。優點包括了：壓縮率高、檔案小的特性。包括了三種主要檔案類型：（1）Real Audio：主要用來傳輸接近 CD 音質的音頻資料；（2）Real Video：主要用來傳輸連續的視訊資料；（3）Real Flash：高壓縮比的動畫格式。

2.2.1 MPEG-2 System Introduction

MPEG-2 Standards 是由 Moving Picture Expert Group(MPEG)所制定，並由 International Standards Organization (ISO)所出版。目前共分成九個部份列於 table2.2；在 ISO 13818-1[7]中定訂了兩種 Multiplexing 模式: Program stream 與 Transport stream，如 figure2.3 所示。Multiplexing 是指將壓縮過後的視訊流、音訊流、使用者資訊與控制資料...等結合成一數據流，讓此數據流可以適用於傳輸或是儲存。

Table2.2 MPEG-2 Standards -ISO/IEC 13818

編號	文件內容
1	System
2	Video
3	Audio
4	Conformance testing
5	Software simulation
6	Extensions for DSM-CC
7	Advanced Audio Coding (AAC)
9	Extension for real time interface for systems decoders
10	Conformance extensions for Digital Storage Media Command and Control (DSM-CC)

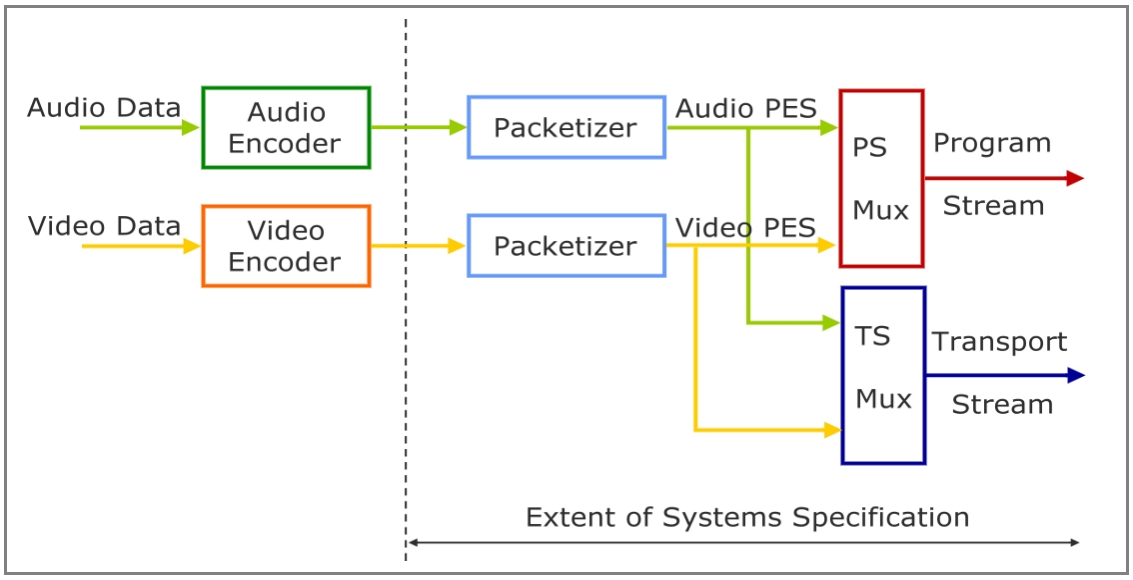

Figure 2.3 MPEG-2 System Model

2.2.1.1MPEG-2 Program Stream

由 figure2.4 可以得知，一個完整的 Program Stream (PS)是由多個 pack 所組成的。一個完整的 pack packet 會包有 system header 與多個 elementary stream，但 system header 不一定會存在。而一個 elementary stream 在加上一個適當的 header 後形成 PES (Packetized Elementary Stream)。

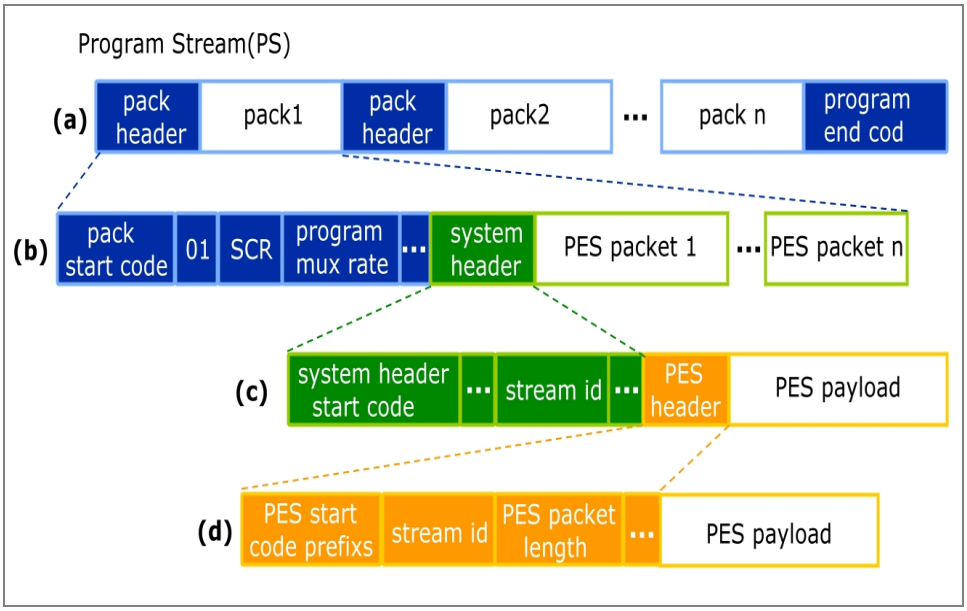

Figure 2.4 Relationship between PES packets and Program

每一個 PS 中的所有 PES 都具有共同的時間基準(time base)，且只能包含一個節目，通常是由一個視訊流搭配兩個音訊流所構成；由於這些 PES 享有共同的時間基準，也才能夠同步地對視訊流和音訊流進行解碼動作。PS 封包的大小並不固定，也因為不同長度的特性，使得 PS 並不適合用於需要錯誤校正的系統，另外也由於其低通訊負載的特性，在一些如 CD-ROM 等無錯誤或近似無錯誤的環境中是相當適合的。

依據 ISO13818-1[7]中所定義，PES 組合成一個 PS 前必須要在加上 Pack header 和 System header，每一個 header 都是由一列固定的起始碼開始，起始碼是一串特殊的編碼值，其可以區分出資料流的每層架構，而 PS 可視為總共分成三個階層，最外層為 Pack header 是由 pack_start_code:0x000001BA 開始，而第二層 System header 則是由 system_header_start_code:0x000001BB 開始，且在 System header 中的 stream_id 欄位會定義此一 PS 的視訊流和音訊流的類型。最裡層的 PES header 開始於 packet_start_code_prefix:0x00000001，接在 PES 起始碼後的欄位為 stream_id 和 PES_packet_length，如 figure2.4 中的(d)所示；其中 stream_id 所裝載的為此 PES 的 payload 類型資訊，從 table2.3[7]中可以知道，當 stream_id 的值在 0xC0~0xDF 間 payload 為 audio stream，如果位於 0xE0~0xEF 間的話 payload 則是 video stream。

PES header 中的 stream_id 欄位除了告訴解碼器其 payload data 的資料流為視訊或是音訊外，它還必須要告訴解碼器此資料流的編碼格式，舉例來說:當 PES header 中的 stream_id=0xE0 時，表示此一 PES 所攜帶的 payload data 為視訊流，編碼方式為 MPEG1 或是 MPEG2；如果 PES header 的 stream_id=0xBC，表示這個 PES 為 program

stream map，它所挾帶 stream_type 和 elementary_stream_id 這兩個欄位的訊息可以告知解碼器這一個 PS 中的 PES 彼此之間的關係，假設 program_stream_map 的 elementary_stream_id=0xE0，stream_type 為 16，依據 Table2.4[7]可以知道 stream_type 為 16 為保留型別，定義為 mpeg-4 video codec，因此只要是 PES header 中的 stream_id 為 0xE0 的話，代表此視訊流編碼格式為 mpeg-4。

Table 2.3 [7] Stream_id assignments

Stream_id	Stream coding
1011 1100	Program_stream_map
1011 1101	Private_stream_1
1011 1110	Paddinf_stream
1011 1111	Private_stream_2
110x xxxx	ISO/IEC 13818-3 or ISO/IEC 11172-3 audio stream
1110 xxxx	ITU-T Rec.H.262｜ISO/IEC 13818-2 or ISO/IEC 11172-3 video stream
1111 0000	ECM_stream
1111 0001	EMM_stream
1111 0010	ITU-T Rec.H.222.0｜ISO/IEC 13818-1 Annex A or ISO/IEC 13818-6 DSMCC stream
1111 0011	ISO/IEC_13522_stream
1111 0100	ITU-T Rec.H.222｜type A
1111 0101	ITU-T Rec.H.222｜type B
1111 0110	ITU-T Rec.H.222｜type C
1111 0111	ITU-T Rec.H.222｜type D
1111 1000	ITU-T Rec.H.222｜type E
1111 1001	Ancillary_stream
1111(1010-1110)	reserved data stream
1111 1111	program_stream_directory

Table 2.4 Stream type assignment (program stream map)

Stream_id	Description
0x00	ITU-T｜ISO/IEC Reserved
0x01	ISO/IEC 11172 video
0x02	ITU-T Rec.H.262｜ISO/IEC 13818-2 or ISO/IEC 11172-2 constrained parameter video stream
0x03	ISO/IEC 11172 audio
0x04	ISO/IEC 13818-3 audio
0x05	ITU-T Rec.H.222.0｜ISO/IEC 13818-1private_sections
0x06	ITU-T Rec.H.222.0｜ISO/IEC 13818-1PES packets containing private data
0x07	ISO/IEC 13522 MHEG
0x08	ITU-T Rec.H.222.0｜ISO/IEC 13818-1 Annex A DSM CC
0x09	ITU-T Rec.H.222.1
0x0A	ISO/IEC 13818-6 type A
0x0B	ISO/IEC 13818-6 type B
0x0C	ISO/IEC 13818-6 type C
0x0D	ISO/IEC 13818-6 type D
0x0E	ISO/IEC 13818-1 auxiliary
0x0F-0x7F	ITU-T Rec.H.222.0｜ISO/IEC 13818-1 Reserved
0x80-0xFF	User Private

2.2.1.2 MPEG-2 Transport Stream

Transport Stream 簡稱 TS，其主要是使用於數位廣播電視上，也就是 HDTV 的傳輸格式；TS 由一道或多道節目(program)組成,每道節目由一個或多個原始流和一些其他流複合在一起，包括視頻流、音頻流、節目特殊信息流（PSI）和其他數據包。不同的節目可以擁有不同的時間基準(time base)，且 TS 封包具有固定的長度，可以容易的讓解碼器得知一張照片的開始與結束，也比較容易在封包遺失後進行補救。

參考 figure2.5 可以得知， TS 和 PS 不同，TS 是由長度固定為 184

bytes 的 PES 再加上 4 bytes 的 TS header 所組成，共 188 bytes。TS header 包含了 8 bits 的 sync_byte (0x47) 和 13 bits 數據包識別號 PID…等。PID 欄位的訊息可以告知解碼器 TS packet 的 payload 類型，可以參考 table2.5[7]，從 PID 可以判斷其後面負載的數據類型是視頻流、音頻流、PSI 還是其他數據包。

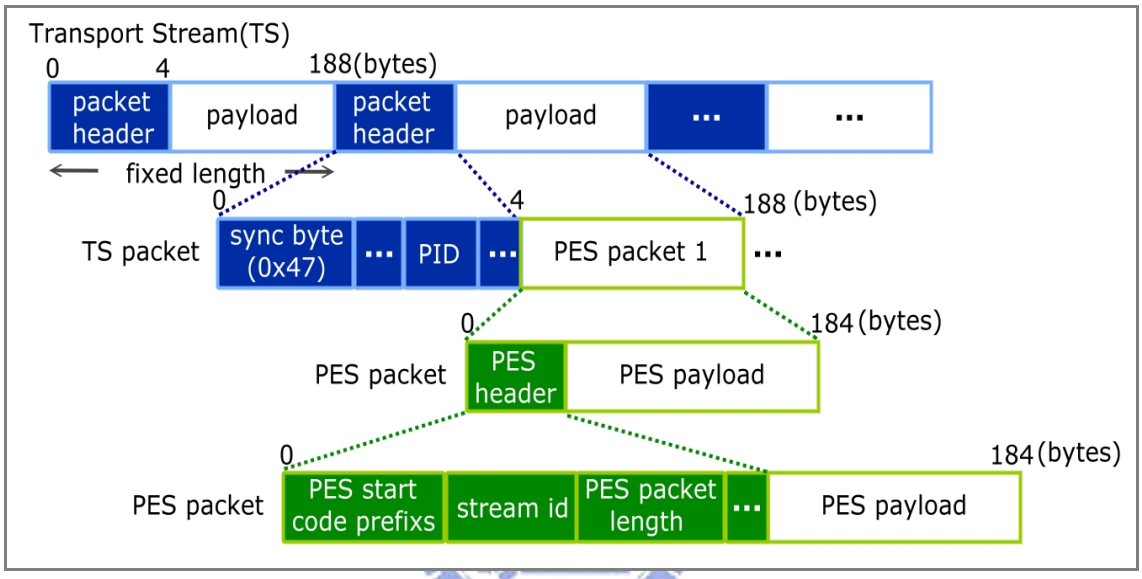

Figure 2.5 Relationship between PES and Transport Stream

Table 2.5[7] PID Table

Value	Description
0x0000	Program Association Table
0x0001	Conditional Access Table
0x0002-000F	Reserved
0x0010-0x1FFE	May be assigned as network_PID, Program_map_PID,elementary_PID, or for other purpose
0x1FFF	Null packet

2.2.2 Mpeg-4 Introduction

MPEG-4 是 MPEG-1 和 MPEG-2 這兩種視訊標準的後繼者，是目

前市場上資訊產品應用裡，最為熱門的技術之一。在網際網路的應用領域裡，MPEG-4 影音串流傳輸的技術研發是一個企業激烈競爭的戰場。目前有非常多的企業組織加入開發 MPEG-4 影音串流的多媒體應用，不過每家公司都採用自訂的 Codec 或特有的檔案格式及系統架構，因此對於 MPEG-4 多媒體呈現的支援度有異，例如大部份都沒有支援互動式場景呈現的能力，而且支援的影音編碼亦不相同。在此小節中將描述一個 mpeg-4 simple profile 的 elementary stream 也就是 M4V，其 header 包含那些重要的資訊。

2.2.2.1 M4v Structure

傳統的視訊壓縮技術是以一張畫面作為壓縮單位，而 MPEG-4 則將畫面再切割成更小的單位作為壓縮單位，稱為物件(Object)，因此本來是一張張的畫面視訊序列(video sequence)會被劃分為數個以物件為主的視訊物件序列(Visual Object Sequence)。

MPEG-4 視覺場景 (visual scene) 可能包含一個或多個視訊物件，每個視訊物件都可藉由時間和空間資訊加以描述，包括它們的形狀、移動和紋理。而 MPEG-4 video stream 會提供階層式的視覺場景描述，起始碼 (start codes) 則是特殊的編碼值，它們可以存取位元串流的每一層階層架構。參考 figure2.6，MPEG-4 階層架構中的各層包括：

- 視覺物件序列 (Visual Object Sequence，簡稱 VS)

  它是完整的 MPEG-4 場景，可能包含任何 2-D 或是 3-D 的自然或合成物件以及它們的加強層 (enhancement layer)。

- 視訊物件 (Video Object，簡稱 VO)

  視訊物件會連結至場景中的某個 2D 元素，矩形圖框就是最簡單

的例子；它也能是任意形狀的物件，對應於場景中的某個物件或是背景。

- 視訊物件層 (Video Object Layer，簡稱 VOL)

視訊物件支援可延展 (scalable) 以及不可延展 (non-scalable) 兩種編碼模式，實際編碼模式則由視訊物件層所代表的應用決定。視訊物件層能支援可延展性編碼。

- 視訊物件平面群 (Group of Video Object Planes，簡稱 GOV)

視訊物件平面群是可選用的功能，它會提供視訊物件平面被獨立編碼的各點，讓位元串流中能夠加入多個隨機存取點。

- 視訊物件平面 (Video Object Planes，簡稱 VOP)

視訊物件平面是在時間取樣的視訊物件，它們可以獨立取樣，也可以利用移動補償值進行取樣。矩形可以代表傳統的視訊圖框。視訊物件平面的使用方法有很多種，最常見的做法是讓它們包含某個視訊物件的時間取樣值的編碼視訊資料。每個視訊物件平面都包含多個巨集區塊 (macroblock)，每個巨集區塊則會包含四個 8x8 亮度區塊 (luminance block) 以及兩個 8x8 色度區塊 (chrominance block)。

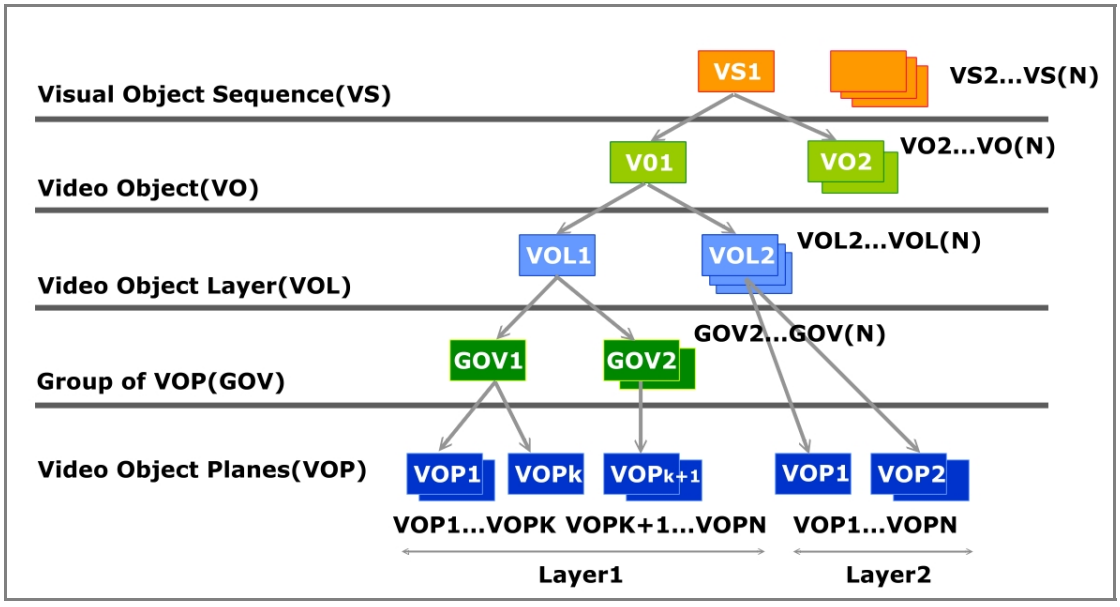

Figure 2.6 MPEG4 video bitstream logical structure

2.2.2.2 M4V Syntax

參考 Figure 2.7[8] (A)，一個完整的 M4V 是由 Visual Object Sequence Header、Visual Object Header、Video Object Layer Header 以及 Elementary Stream Visual Object 所組成;前面三個 header 會攜帶解碼器所需的一些資訊，像是照片的尺寸、量化矩陣的參數...等的資料，這些資料稱為 Elementary Stream Descriptors(ESDS data)，而每個 header 都有其固定的起始碼的位元值，可以參考 table 2.6[8]；當 ESDS data 結束後會進入 Video Object Plane，也就是每張 frame 的起始位置，藉由 Video Object Plane 的 header 可以判別此張 frame 類型為 I、P 中的那一項，而 Video Object Plane 的 start code 即為此張 frame 的起始位置。在 figure 2.7(B)明白表示出一個完整的 M4V 檔案是由 ESDS data 和 I frame 與 P frame 所組成。詳細的 header 解說可以參考 3.2.1 小節。

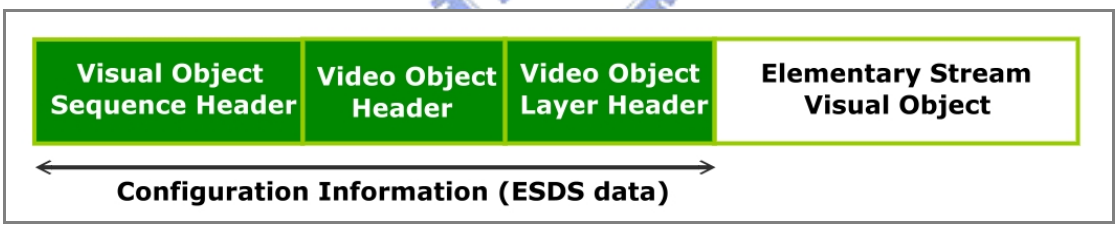

Figure 2.7[8] (A) Structure Of M4V

Figure 2.7 (B) Structure Of M4V

Table 2.6 [8] MPEG-4 Start Code

Name	Start code value (hexadecimal)
Video_object_start_code	00 through 1F
Video_object_layer_start_code	20 through 2F
Reserved	30 through AF
Visual_object_sequence_start_code	B0
Visual_object_sequence_end_code	B1
User_data_start_code	B2
Group_of_vop_start_code	B3
Video_session_error_code	B4
Visual_object_start_code	B5
Vop_start_code	B6
.....	.....

2.3 RTP Payload Format for MPEG-2 System and M4V

從 2.1.3 節中可以知道 RTP 封包是由 RTP header 與 payload 所組成。RTP header 中的 Payload type(PT)的欄位中裝載了此一 RTP 封包所負荷的資料型態的資訊，也就是 payload 的類型資訊；PT 欄位訊息由 7 bits 來表示，表示資料的編碼格式有 128 種的可能性，而 payload data 可能為 128 種中的其中一種的型態，相關定義在 RFC 1890 中，而每一種編碼類型都有其不同的 RTP 包裝方式，這些資訊被定義在各個 RFC 的文件中，table 2.7 列出較常見編碼類型其 PT 值以及其相對應的 RFC 文件，詳細的資料請參見附錄 A。而在本節中將說明 MPEG-2 與 M4V 的 RTP payload 包裝方式。

Table 2.7[12] Payload Type Table

PT	Name	Type	Clock rate (Hz)	Audio channels	References
7	LPC	Audio	8000	1	RFC 3551
8	PCMA	Audio	8000	1	RFC 3551
14	MPA	Audio	90000		RFC 2250, RFC 3551
26	JPEG	Video	90000		RFC 2435
31	H261	Video	90000		RFC 2032
32	MPV	Video	90000		RFC 2250
33	MP2T	Audio/ Video	90000		RFC 2250
34	H263	Video	90000		
72~76	reserved				RFC 3550
96~127	dynamic				RFC 3551
dynamic	H263-1998	Video	90000		
dynamic	BMPEG	Video	90000		

2.3.1 RTP Payload Format for MPEG-2 System

在 RFC 2250[3]中描述了傳輸單獨的視訊流和音訊流所採用的 RTP 封裝方式。以下將分別介紹視訊流及音訊流的包裝格式。

2.3.1.1 RTP Payload Format for MPEG-2 Video Stream

MPEG-2 視訊流定義在 ISO 13818-2[9]中，其結構如 figure 2.8 所示，MPEG-2 視訊流是由 I、P 與 B frame 所組成，每個 frame 由許多的 Slice 組成，Slice 可以切割成許多的 Macroblock，Macroblock 是組成 frame 的最小單位，且同一個 Slice 的所有 Macroblock 都會位於 frame 的同一個水平位置上，可以參考 figure 2.9。而在每張 I frame 前都會有 ESDS data，包含了 sequence_header、group of picture header...等，這些 header 將攜帶描述此視訊流的相關資訊如:frame

rate、frame size...等。

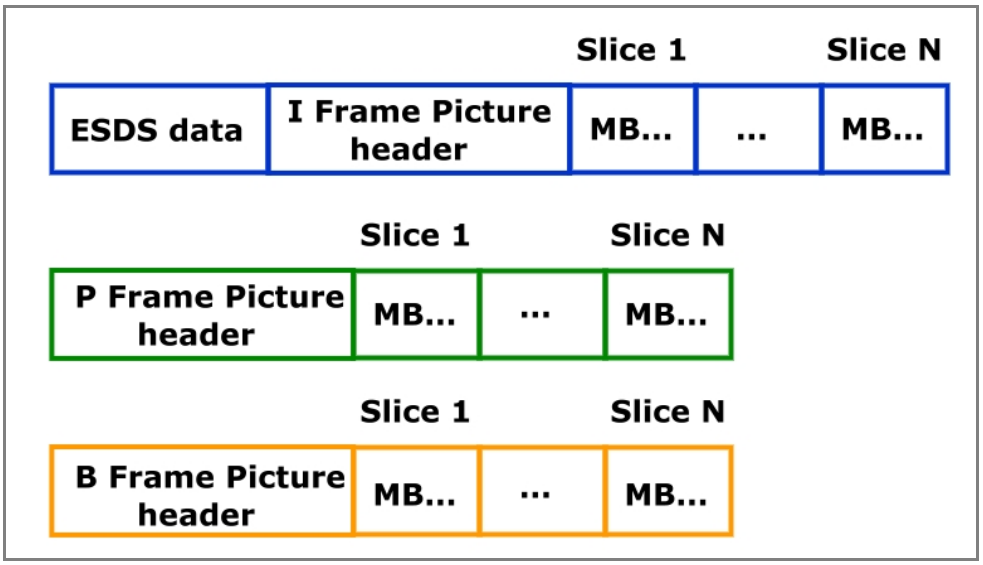

Figure 2.8 MPEG-2 Video Stream Example

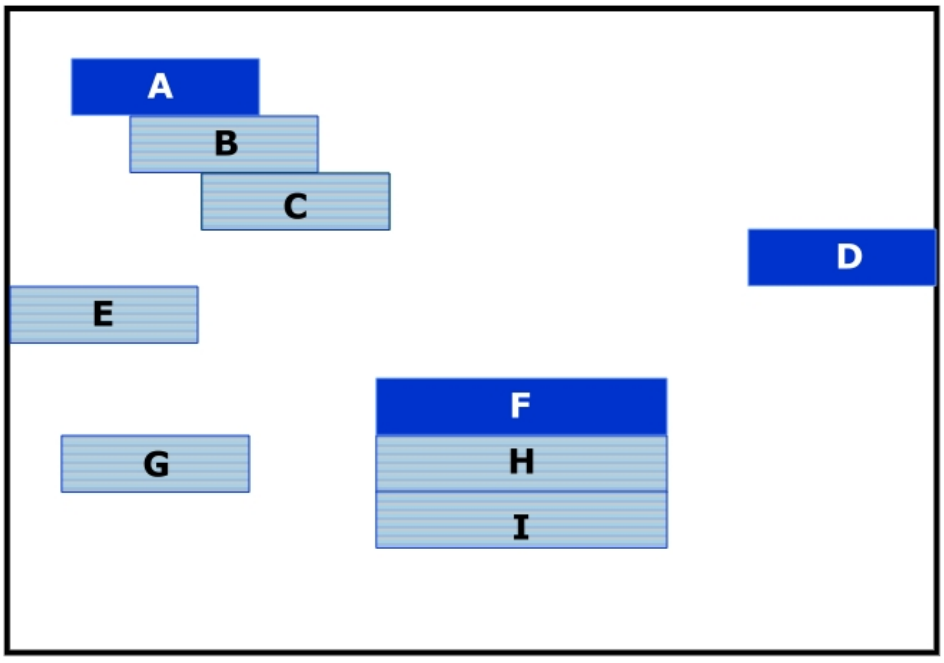

Figure 2.9 Slice Distributed In Frame Schematic Drawing

而 RTP 對 MPEG-2 視訊流的封裝遵循著以下四點的原則:

- MPEG-2 的 Video_Sequence_Header 出現的時候，將總是在一個 RTP payload 的開始處。

- MPEG-2 的 GOP_header 出現的時候，將總是在一個 RTP payload 的開始處，或跟隨在一個 Video_Sequence_Header 的後面。
- MPEG-2 的 Picture_header 出現的時候，將總是在一個 RTP payload 的開始處，或跟隨在一個 GOP_header 的後面。
- 每一個封包還必須包含一個整數數目的視頻片斷(Video slices)。

從上述四點可以得知所有的 ESDS data 都必須要被放入在同一個封包中，因此 RTP payload 的部分最小的長度約為 261 bytes，包含了所有的 ESDS data，而另外一點須要注意的是不管 payload 的部份為視訊流或是音訊流，在 RTP 固定的 header 之後都會跟隨著 4 bytes 長的 specific header，可以參考 figure 2.10 的說明。

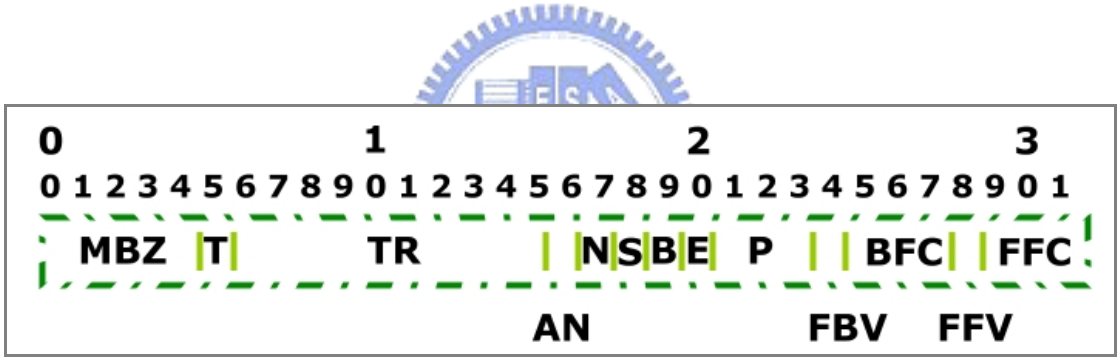

Figure 2.10 MPEG-2 Video Stream -Specific Header

- MBZ (5 bits)

  Unused. 必須為 0。保留作為將來使用。

- T (1 bits)

  MPEG-2 視訊流的延伸 header，位元值為 1 時，表示 Specific Header 後會跟隨另一 header。

- TR (10 bits)

  Temporal Reference. 當前圖片的 GOP 暫存，這個值的範圍從 0-1023，而且對所有給定圖片之 RTP 而言為常數。

- AN(1 bit)

標示錯誤回復狀態的位元(大小為1bit)，當MPEG-2下的圖片header資訊改變時設定為1；而在MPEG-1下或者該位元未被使用時設定為0。

- N(1 bit)

當AN被設為1且在MPEG-2下被使用，除此之外，必須為0。當先前傳送圖片的header不能重建當前圖片的header時設為1，這種情況會發生在當前的圖片與先前圖片是用不同設定的編碼。這個N位元對所有相同圖片的RTP而言必須為常數,如此從一個圖片接收的任何封包，才能偵測圖片資訊是否有必要重建(N=1),或是之前的圖片(N=0)。

- S (1 bit)

Sequence header present.當payload中有MPEG sequence header出現時此欄位會設成1。

- B (1 bit)

Beginning of slice (BS).如果payload內有slice start code 、Video_Sequence_Header、GOP_header 或者是 Picture_Header 的話，此欄位會設置成1。

- E (1 bits)

End of slice (ES). 如果payload 內有MPEG slice最後區段資料則設為1。

- P (3 bits)

Picture Type. I (1), P (2), B (3) or D (4).

- FBV (1 bit)

全部pel的順向導引。

- BFC (3 bits)

逆向框架的代碼。

- FFV (1 bit)

全部 pel 的逆向導引。

- FFC (3 bits)

  順向框架的代碼。

2.3.1.2 RTP Payload Format for MPEG-2 Audio Stream

MPEG-2 音訊流通常都是以 frame by frame 的方式儲存，且通常每個 frame 的大小都不會超過 LAN/Ethernet 的 MTU，所以在 RTP payload 的部份通常都是包含了一個單獨音訊流的 header 及 bit-stream 的部份。而音訊流的 specific header 定義如 figure 2.11。

Figure 2.11 MPEG-2 Audio Stream -Specific Header

- MBZ (14 bits)

  Unused. 必須為 0。保留作為將來使用。

- Frag_offset (18 bits)

  數據封包造成音訊框架位元組的偏移量。

2.3.2 RTP Payload Format for M4V

在 RFC3016 中定義了如何切割 MPEG-4 視訊流於 RTP payload 當中，但在此要先說明 MPEG-4 視訊流中 video packet 的觀念，在 MPEG-4 video 的壓縮中，使用 video packet 的目的是為了幫助解碼器進行錯誤的恢復，一個 VOP 中可能會含有多個 video packet，可以參考 figure 2.12；在網際網路的傳輸過程中常常會有封包遺失的情況發

生，如果遺失的封包中含有 VOP header 的資料時，將會使得屬於這個 VOP 所有的 Macroblock 都無法進行解碼，而 video packet 可以解決此一問題，依據 ISO14496-2[8]，一個具有 video packet 的 MPEG-4 simple profile 會如 figure 2.12 所示，video packet 是由許多 Macroblock 所組成，當解碼端讀取到 video packet 時，會開始讀入 Macroblock，每讀取完一個 Macroblock 時解碼端會判別接下來是否會出現 video packet header 或是 VOP header，若是這兩者都沒有出現的話則解碼端就會判斷接下來仍是 Macroblock。

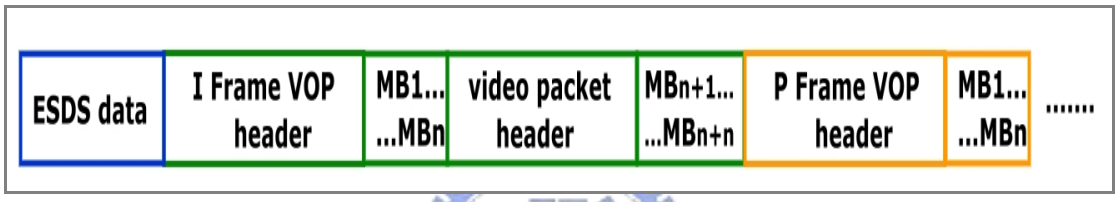

Figure 2.12 MPEG-4 Video Stream-Video Packet

在 RFC3016 中基本是建議再一個 RTP 封包裡面只放入一個 VOP，這是因為在 RTP header 中的 Timestamp 欄位值會隨著不同的 VOP 而變動，但是在實際的應用上此種作法會造成頻寬的浪費，因為有時候 VOP 會只有 VOP header 但卻沒有 video packet，有時候某種形狀的 VOP 其 video packet 會很小，在這樣的情形之下，若是仍舊讓一個 RTP 封包只包有一個 VOP 的話將會產生 overhead 的效應，因此 RFC3016 中允許多個 VOP 串聯再同一個封包中，因此在 RFC3016 中也對 Timestamp 有額外的定義產生如下:

- 若封包中含有多個 VOP，則 Timestamp 的值定義為這些 VOP 中最早出現的一個，而其他的 Timestamp 則可由 VOP header 中的資訊進行推算。
- 假如此一封包中只包含了 configuration information，則

Timestamp 定義為下一個出現的 VOP。

- 若此封包中只含有 visual object sequence end code，則 Timestamp 定義為上一個出現的 VOP。

而在 RFC3016 中定義了對 MPEG-4 視訊流的封裝遵循著以下的原則：

- Configuration information 與 Group_of_Video_Object_Plane ( ) 應該要放在 RTP payload 的最前面。
- 若有多個 header 放在 RTP payload 中，則要依據 syntax 的順序排放，而 visual_sequence_end_code 是排在最末。
- 同一個 header 不能拆解存放到兩個封包中。
- 除非一個 VOP 的資料量太小，否則盡量一個 VOP 存放在同一個封包中。
- 盡量讓同一個 video packet 放在同一個封包中。

綜合以上五點規則，在 RFC3016 中列舉了幾個 RTP 封包的例子，可以參考 figure 2.13[4]所示。以下分別簡述其優缺點:

- Figure 2.13(d) 一個封包中裝載了一個 video packet。

  優:適用於易遺失封包的網路環境，因為即使其中一個封包遺失，但是其後封包中的 video packet header 中包含的 HEC(Header Extension Code)機制仍然可以使得解碼器找到目前 RTP 封包中的 Macroblock 位於 VOP 的何處，並且解碼器亦可由 VOP 表頭中得知解碼的方式。

  缺: 因為一個 RTP 封包中只裝有一個 video packet，故 UDP/IP/RTP header 的 overhead 將會使得網路傳輸量變大。

- Figure 2.13(e) 多個 video packet 放入同一個封包中。

  優:適用於網路頻寬較差的情形下，因為可以減少因 UDP/IP/RTP header 所產生的 overhead 的問題。

  缺:會減弱因封包遺失後解碼端進行錯誤恢復的能力。

- Figure 2.13(f)

  優:沒有 video packet header 的 overhead，也就是不採用 video packet 的作法，目的是希望藉此提高封包使用率。

  缺：僅限用於無錯(error-free)的網路環境下，因為此種裝載方法每當封包遺失後便幾乎沒有錯誤恢復的能力。

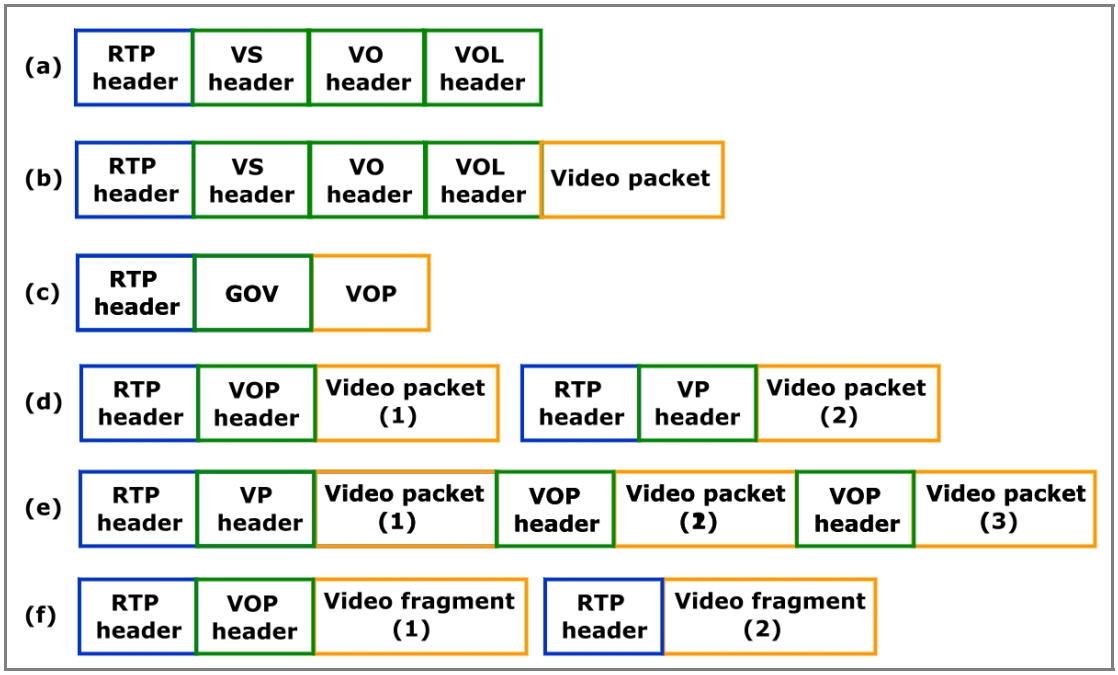

Figure 2.13[4] RTP Packet Example

第三章 RTP 封裝演算法的設計原理介紹

在一個隨選即播的串流系統裡，伺服器會提供本地端的檔案列表給予客戶端選擇，當伺服器端收到客戶端的播放請求時，並不知道其所請求播放檔案類型為何，因此伺服器需要有一個機制可以去辨別客戶端所選擇要傳送的檔案格式類型，進一步的將檔案中的視訊流和音訊流分離成獨立的媒體以及辨認出各自的編碼類型，再來依據各媒體的編碼特性對資料進行解析及切割，並將其包裝成一個個的封包透過網際網路傳輸到客戶端。而這樣的一個流程在多媒體串流系統中，是由「封包包裝和傳送」此一模組負責做統籌處理，此模組是整個系統內部運作的重心所在，最主要在於實現特定的封裝演算法，產生適合網路環境的封包並且傳輸。

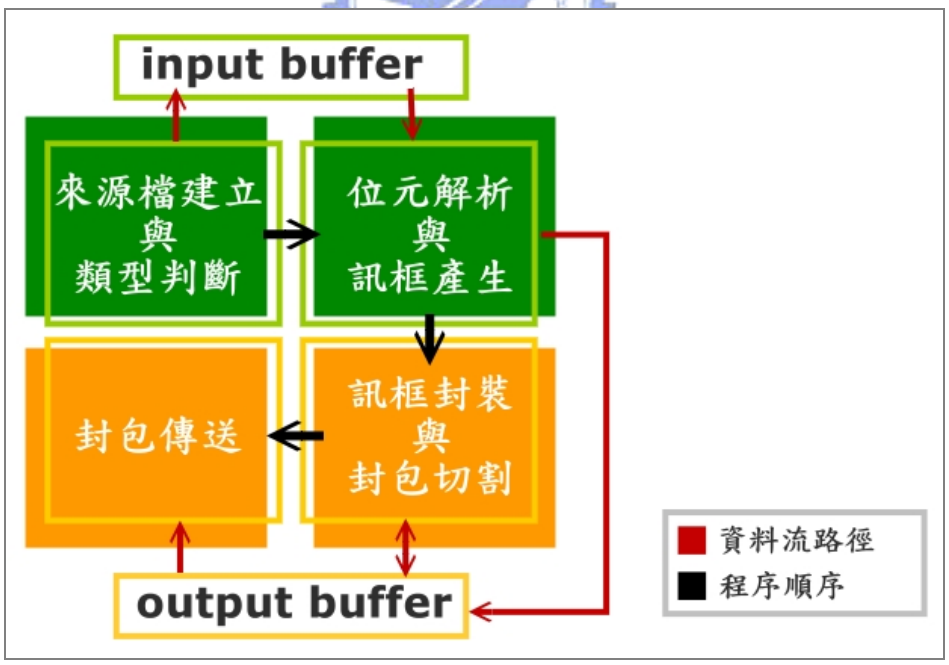

Figure3.1『封包包裝和傳送』模組方塊圖

參考 figure3.1，一個『封包包裝和傳送』模組可以視作成四個區塊，分別是來源檔案建立與類型判斷、位元解析與訊框產生、封包切割與包

裝以及封包傳送，在本章中將個別解說每一個區塊所進行的工作內容與方式，並在第四章以 Live 555 Stream Server 說明『封包包裝和傳送』模組的實現方式。

3.1 來源檔案建立與類型判斷

當伺服端收到客戶端請求播放某一個影音檔案的要求後，伺服端會找到此一檔案並開啟，在此階段必須要知道檔案的類型，要先知道檔案的 container type 後，再進一步拆解出檔案中所包含的音訊流及視訊流的 codec type，才能在進行「位元解析與訊框產生」階段時選用適當的解析器與訊框產生器。

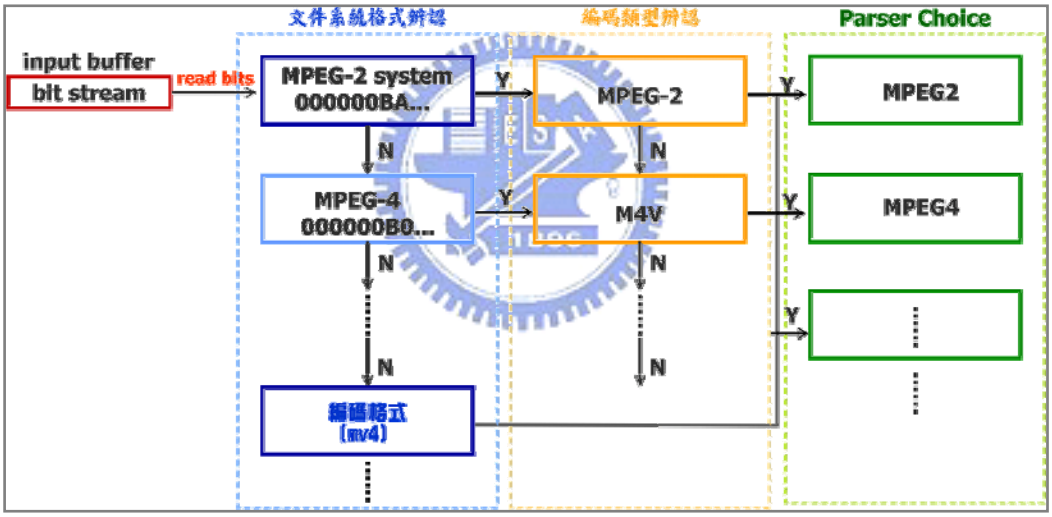

Figure3.2(A) 檔案格式判別說明圖-Method1

通常判斷檔案的類型所採取的方式有兩種，第一種是先讀入檔案開頭的一段位元流到暫存器內，進行 container type search，因為不同類型的檔案，都有其固定的語法及編碼值，以 MPEG2 的檔案為例，其檔案開頭的編碼值為「000000BA...」，因此只要識別此段位元流到對應的編碼值相似度最大的格式類型，便可以得知此影音檔案的 container type 為何，尋找到 container type 後會再進一步辨識檔案中 elementary stream 中的 codect type，進而可以選用相應的解析器與訊框產生器，

如 figure3.2 所示。而如果檔案本身為 codec type，因在 container type search 中找不到對應的 container type，就會直接辨識 codec type，再選用相應的解析器與訊框產生器。

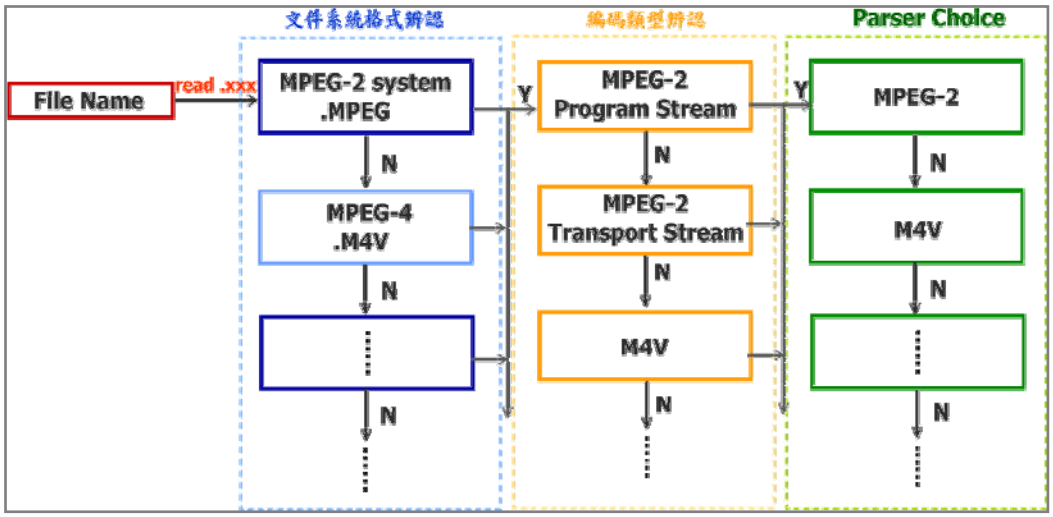

Figure3.2(B) 檔案格式判別說明圖-Method2

另一種作法是直接讀取副檔名，由 2.2 節的說明中，可以知道每一種 container type 及 codec type 在作業系統上都有其相對應的副檔名，如副檔名為.mpeg 檔案對應的即是 MPEG-2 的 container type，而.m4v 檔案對應的則是 MPEG-4 的 codec type，因此伺服器可以以讀取副檔名的方式來判斷要選擇何種解析器與訊框產生器，如果讀入的副檔名為 container type 的話，則要先解析出檔案中的 elementary stream 的 codec type 才能進行解析器與訊框產生器選擇。此種方式的優點在於不必先讀入檔案內容便可立即知曉檔案的編碼格式並直接進入下一階段，也可以節省判斷的時間與暫存器的使用，但其缺點是在面臨偽檔名或是錯誤的副檔名時，便會產生無法正確解析檔案內容的問題，因此在精確度上並不如第一種方法來的好。

3.2 位元解析與訊框產生

當檔案開啟並找到對應系統格式後，便須要選擇相應的位元流解析

器(parser)來分析 elementary stream 中 header 所攜帶的資訊，進而抽取出音訊流及視訊流，並設置訊框產生器(framer)來分離出音訊流或是視訊流中的 header 和訊框資料；一般而言會將解析器與訊框產生器視為一體也就是 parser，因為都是對檔案的位元流進行分析並找出所需要的資料，且其可以視作一個模組但其中一些細節部份須要依據每種編碼規格而有所變動，下面將解說整個解析器與訊框產生器的設計及運作方式，並以下面兩個例子說明:

1.M4V: 解析 elementary stream 中 header 所攜帶的訊息並產生 frame。

2.MPEG-2 Program Stream:從 container type 辨認 elementary stream 的 codec type。

3.2.1 M4V 檔案解析

從 2.2.2 小節中可以知道，M4V 是單純的視訊檔，其整個檔案位元流架構共可以分成六個階層，每一階層都有其起始碼(start code)，利用起始碼這個特殊編碼值可以劃分出六個區塊如 table3.1 所示，其中前面三個區段(VS+VO+VOL)為組態資訊，是重要的解碼資訊，可以視作一體，而 frame 則是存在 VOP 這個區段中。

Table3.1 MPEG-4 位元流結構表

	Start code
MPEG-4 configuration information(組態資訊)	
Visual Object Sequence (VS)	000001B0
Visual Object (VO)	000001B5
Video Object Layer (VOL)	00000120~2F
Elementary stream data(基本流資料)	
Group Of Video Object Plane (GOV)	000001B3
Video Object Plane (VOP)	000001B6
場景結束	
Visual Object Sequence End Code	000001B1

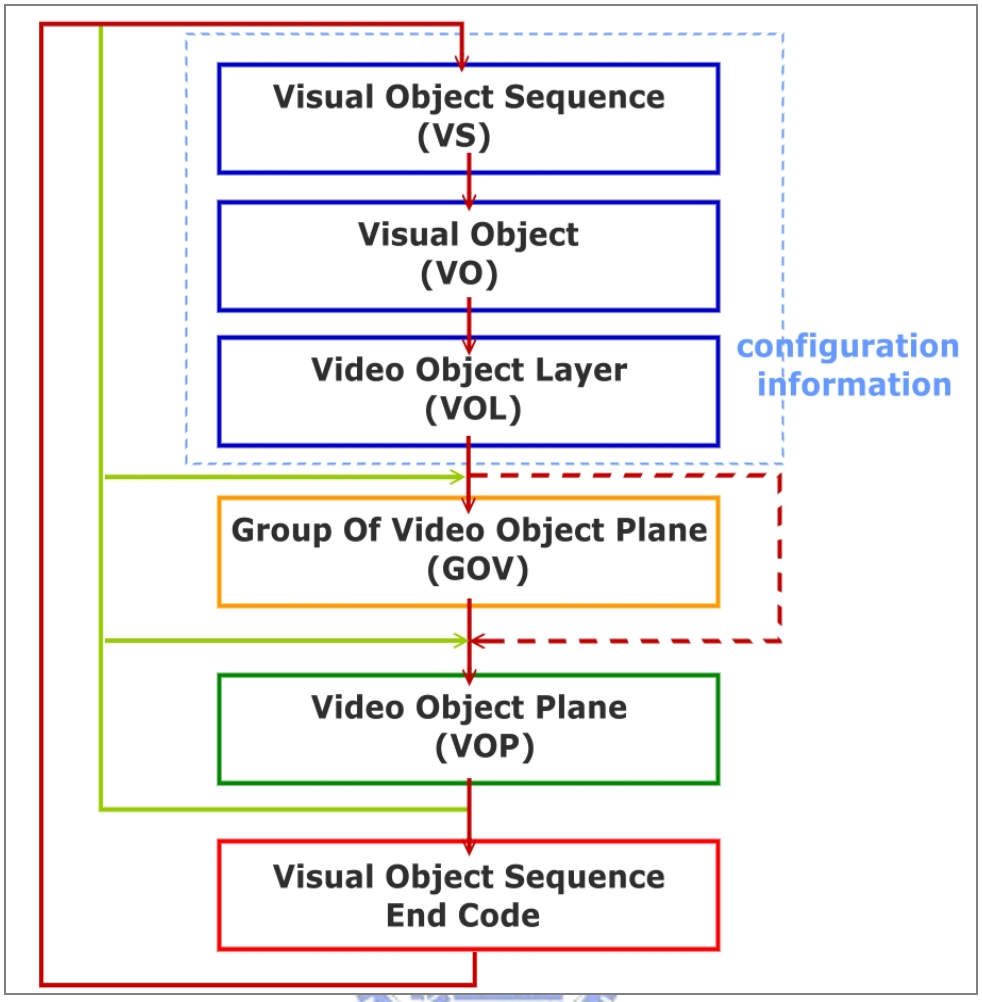

Figure3.3 MPEG-4 parse model

參考 figure3.3 所示，每一個區塊可以視作一個獨立的處理程序，但彼此之間具有層次關係，因此每一個處理程序進行完畢後便會視其之後的 start code 的值而將 parser 狀態設至相對應的處理程序，箭頭所指的方向是下個會執行的程序，如果是多重路徑(像是青綠色箭頭線段)，則表示下個要執行的程序是其中的一條路徑；在每一個程序中除了會進行分析 header 所攜帶的訊息外，同時也會將檔案的位元流切割成一塊塊的區塊存放在輸出緩衝區中，每一個區塊資料都是由起頭的 start code 到下一階層的 start code 之間的位元流，而這些區塊資料即是 frame，在下一階段封包包裝中便是將這些 frame 包入 packet 中。的以下分別說明六個處理程序的內容。

1. Visual Object Sequence (000001B0)

開始從來源檔案的輸入緩衝區中讀入 MPEG-4 視訊位元流，此時會不斷的搜尋視訊位元流，直到找到 Visual Object Sequence start code:“000001B0”後才會真正的進入解析的處理程序中；參考 figure3.4，Visual Object Sequence 這層資料區段的總長是由 header information 的 5 個 bytes 再加上長度不定的“User data” 資料流所組成。Header information 包括 4 bytes 的 Visual Object Sequence start code 和 1 byte 的“profile_and_level_indication”；“profile_and_level_indication”是用來告訴解碼器其所用的解碼工具類型為何及其支援程度。

另外在 Visual Object Sequence 這層中還會包含“User data”的部份，這部份是編碼此檔案的使用者自行定義的資訊，是給解碼器的訊息，在 MPEG-4 編碼系統裡是選用的功能，所以不一定會存在，也因此“User data”會被解析器視為普通的資料位元流，其 start code 的編碼值也不在解析器識別範圍內。

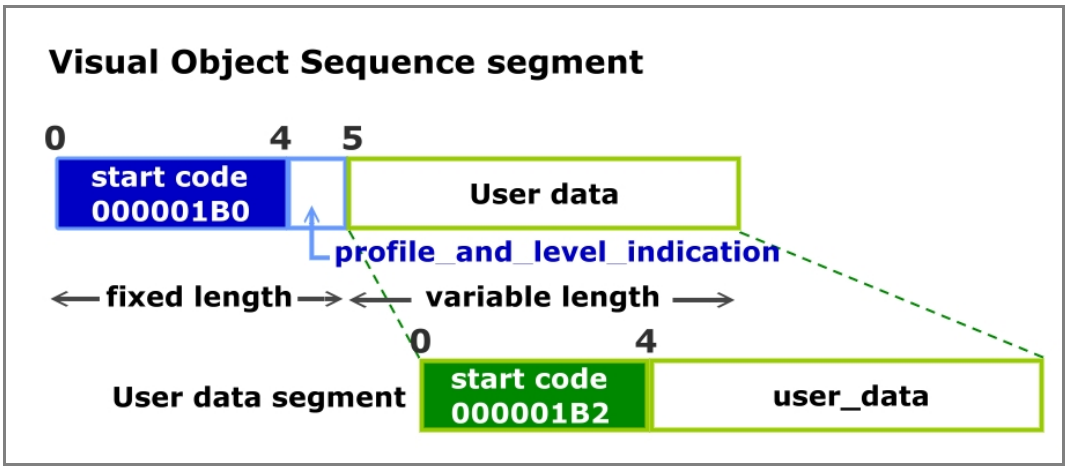

Figure3.4 Visual Object Sequence segment

解析器會將 Visual Object Sequence header information 及之後的

“User data”位元流儲存起來直到在視訊位元流中看見 Visual Object start code 後才會停止且結束這個階段的工作，並將解析器狀態設為下個處理程序: Visual Object。

2. Visual Object (000001B5)

Visual Object 整個區段結構如 figure3.5 中的(a)部份所示，整個區段的資料大小為固定的 header information 在 9~10 bytes 間，有時會另外再加上“User data”部份而使區塊資料長度增加。Visual Object 區塊資料長度會有所變異的原因主要是“is_visual_object_identifier”此一欄位所攜帶的資訊不同所致；在 Visual Object 區段中的第五個 byte 中的第一個 bit 即是“is_visual_object_identifier”，參考 figure3.5 的(b)，當此位元為 1 時，須進一步的找出在跟隨在後 4 bits 的“visual_object_verid”和 3 bits 的“visual_object_priority”的資訊，兩者相關訊息意義請參考 table3.2[8]。

“is_visual_object_identifier”與“visual_object_verid”再加上“visual_object_priority”共為 1 byte 也就是 Visual Object 區段中的第五個 byte。接下來第六個 byte 中的前 4 bits 為“visual_object_type”，其所表示的訊息意義請參考 table3.3[8]，而第七至十這 4 bytes 的編碼值是一組 start code，代表著“visual_object_type”所選用的型態訊息區段開端，一般情況中 visual object type 所選用的型態通常為 video ID，從 ISO/IEC 14496-2[8]中可以查閱到:

	Code value
if (visual_object_type==“video ID”){	
video_object_start_code	00000100~1F
VideoObjectLayer()	
}	

也因此第七至十這 4 bytes 通常為 video_object_start_code。

Table3.2[8] Meaning of visual_object_verid and visual_object_priority

visual_object_verid: This is a 4-bit code which identifies the version number of the visual object. When this field does not exist, the value of visual_object_verid is '0001'. **visual_object_priority**: This is a 3-bit code which specifies the priority of the visual object. It takes valuesbetween 1 and 7, with 1 representing the highest priority and 7, the lowest priority. The value of zero is reserved.

visual_object_verid	Meaning
0000	reserved
0001	Object type listed in ISO/IEC 14496-2 Table 9-1
0010	Object type listed in ISO/IEC 14496-2 Table V2-39
0011-1111	reserved

Table3.3[8] Meaning of visual object type

visual_object_type: The visual_object_type is a 4-bit code given in table which identifies the type of thevisual object.

visual_object_verid	Meaning
0000	reserved
0001	Video ID
0010	Still texture ID
0011	mesh ID
0100	FBA ID
0101	3S mesh ID
01101	reserved
...	...
1111	reserved

如果"is_visual_object_identifier"位元值為 0 時，則表示沒有做任何特別的設定，因此緊跟在後的 4 bits 會是"visual_object_type"，而不會存在"visual_object_verid"和"visual_object_priority"兩個欄

位，如 figure3.5(c)所示，“is_visual_object_identifier”與“visual_object_type”會組合成為一個 byte，所以 Visual Object 區段資料大小會變成 9 bytes。

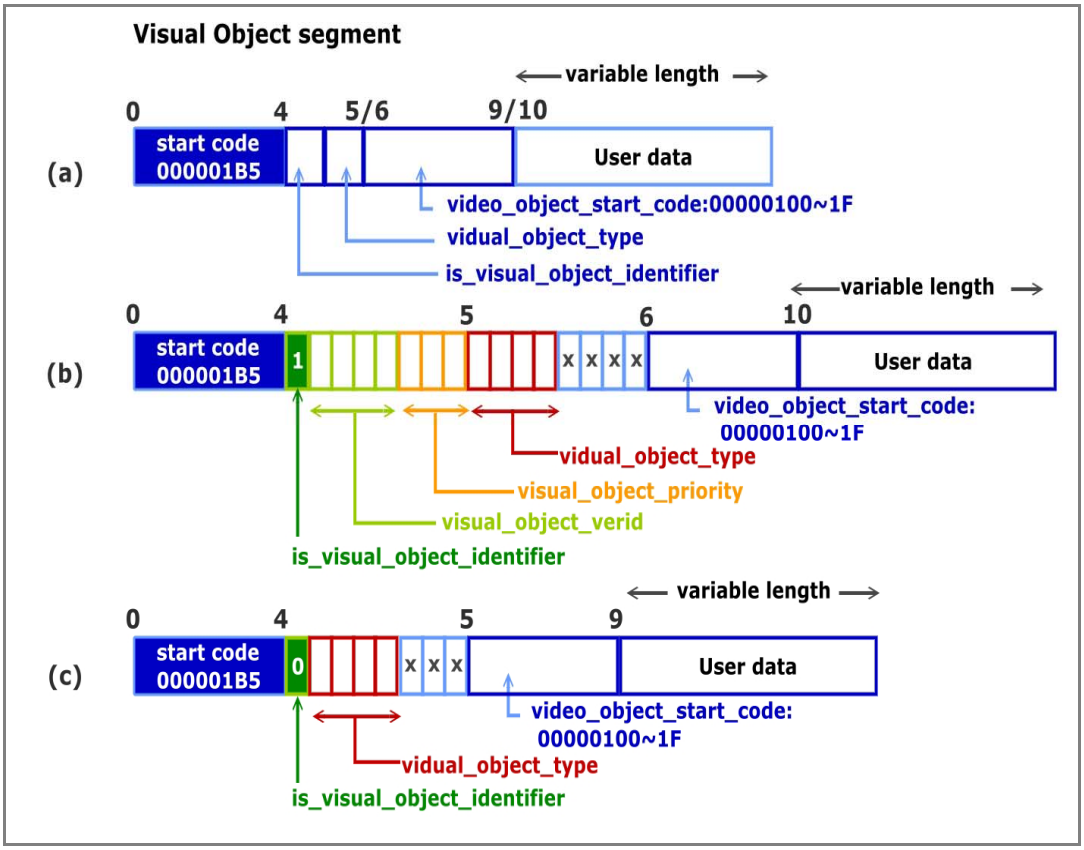

Figure3.5 Visual Object Segment

而 Visual Object 區段資料儲存動作會在存入 video object start code(或是其它型態的 start code，請參考[8])至輸出緩衝區後結束，且解析器也會將狀態設為下個處理程序: Visual Object Layer 並停止且結束這個階段的工作。

3. Video Object Layer (00000120~2F)

Video Object Layer 此一區段的資料量並不固定，基本的 header information 佔有 16~20 bytes，而隨著 header 中每一個參數的意義不

同會再增加更多的資訊位元，有時也會再加入額外的“User data”的資訊位元流，因此整個 Video Object Layer 區段資料是由 header information 再加上“User data”所組成。當解析器進入 Video Object Layer 這個處理程序中時，首先會進行 VOL 的 header 分析，header 的各個欄位的參數請參考 figure3.6 和 table3.4，在 figure3.6 中所示的是固定的參數欄位，在 table3.4 中則是因參數設定而額外增加的相關參數。在這階段中最主要是找 vop_time_increment_resolution、fixed_vop_rate 和 fixed_vop_time_increment 三個參數並從中分析相鄰兩張 VOP 之間的時間間隔資訊。

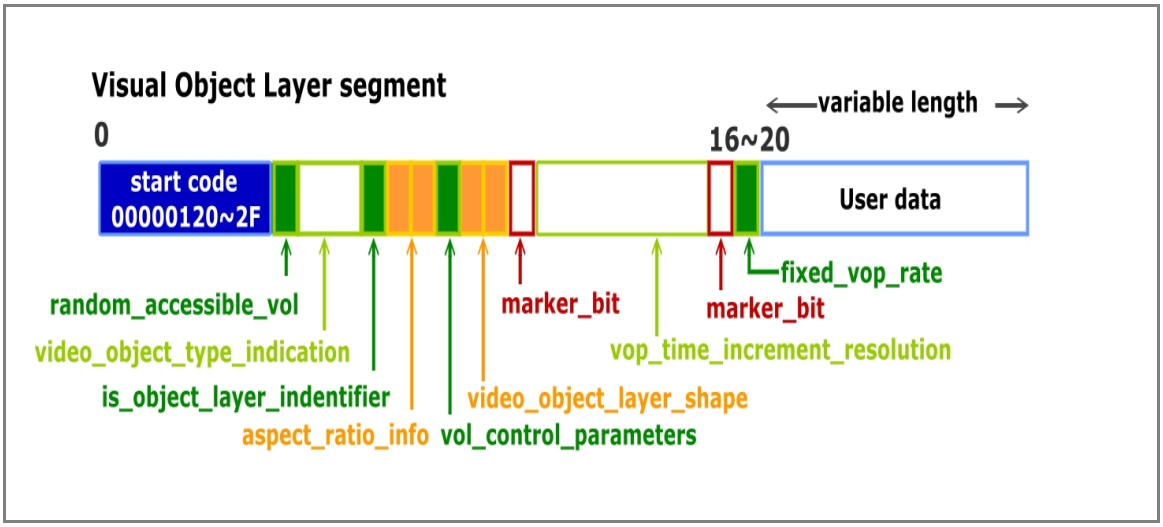

Figure3.6 Visual Object Layer Segment

當解析器分析 header information 後，會將 header 與之後的位元流儲存至輸出緩衝區直到看見下一個 start code 才停止，並結束這個階段的工作，解析器會依據所看到的 start code 的碼值將狀態設定為 Group Of Video Object Plane 或是 Video Object Plane 兩者其中之一的處理程序。

Table3.4[8] Visual Object Layer parameters

Parameter	Extra Parameter	No. of bits
is_object_layer_identifier=1	video_object_layer_verid	4
	video_object_layer_priority	3
aspect_ratio_info ==“extended_PAR”	par_width	8
	par_height	8
vol_control_parameters=1	chroma_format	2
	Low_delay	1
	vbv_parameters	1
vbv_parameters=1	first_hafe_bit_rate	15
	marker_bit	1
	latter_half_bit_rate	15
	marker_bit	1
	first_half_vbv_buffer_size	15
	marker_bit	1
	latter_half_vbv_buffer_size	3
	first_half_vbv_occupancy	11
	marker_bit	1
	latter_half_vbv_occupancy	15
	marker_bit	1
fixed_vop_rate = 1	fixed_vop_time_increment	1-16
video_object_layer_shap	...	...

4. Group Of Video Object Plane (000000B3)

由於 GOV 在 MPEG-4 編碼系統中是選用的功能，並非每一個視訊位元流中都具有此區塊，因此這個階段的處理程序並不會常態性的執行。此程序最主要是分析一群 VOP 中每張 VOP 被獨立編碼的各個時間點，讓位元串流中能夠加入多個隨機存取點。整個區塊的資料長度由 7 bytes 的 header information 與不定長度的“User data ”所組成。解析器會將這 7 bytes 的 header information 與之後的“User data ”存入輸出緩衝區直到看見 Video Object Plane start code 為止。此階段結束後會將解析器的狀態設成下個處理程序:Video Object Plane。

5. Video Object Plane (000000B6)

在 Video Object Plane 的區塊資料可以視為 7 bytes 的 header information 與之後的 VOP 資料位元流所組成。參考 figure3.7 所示，這 7 bytes 由“Video Object Plane start code”、“vop_coding_type”、“modulo_time_base”和“vop_time_increment”所組成；如 table3.5[8] 所示，“vop_coding_type”位元值決定該 VOP 的編碼種類為 I、P、B frame 其中一種。而“modulo_time_base”和“vop_time_increment”則代表此一 VOP 播放的時間資訊；當位於 Video Object Layer header 中的“fixed_vop_time_increment”的值不存在時，則 VOP 的播放時間相關參數會採用“modulo_time_base”和“vop_time_increment”的值，“modulo_time_base”表示本地端的基底時間，以秒為單位，而“vop_time_increment”則是播放的時間間隔，以毫秒固定的增加。若“fixed_vop_time_increment”有值存在則會以它為主，而忽略“modulo_time_base”和“vop_time_increment”這兩個參數的值。

解析器會不斷的將 Video Object Plane 區塊中的 header information 和其後的 VOP 資料位元流存入輸出緩衝區直至看見下個狀態的 start code 時才會停止，此時會設置一個旗標，這個旗標的狀態是用來表示在緩衝區中的這段位元流是一張完整的 VOP，此一旗標在『封包切割與包裝』中扮演著重要的角色。

Table 3.5[8] Meaning of vop_coding_type

vop_coding_type	coding method
00	intra-coded(I)
01	presictive-coded(P)
10	bidirectionally- presictive-coded(B)
11	sprite(S)

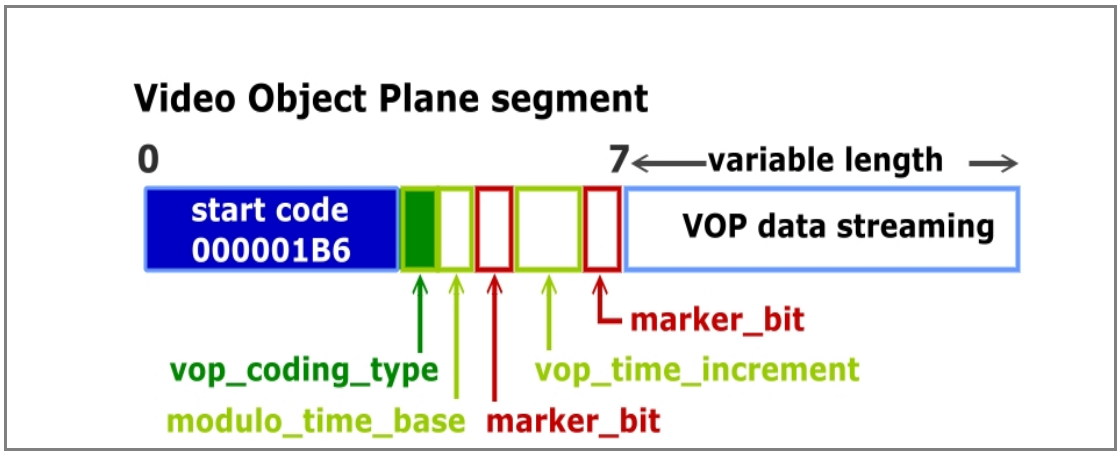

Figure 3.7 Video Object Plane Segment

當這個階段結束會依之後的 start code 的值將 parser 狀態設成下面處理程序其中一種:

(1.) Visual Object Sequence End Code

表示一個場景完整的結束。但此編碼值並非一定會存在。

(2.) Visual Object Sequence

代表又進入一個新的場景，回到起始的處理程序重新開始。

(3.) Group Of Video Object Plane

表示一群 VOP 的開端，再次分析每張 VOP 被獨立編碼的時間點。

(4.) Video Object Plane

代表又是一張新的 VOP 的開始。

6. Visual Object Sequence End Code (000001B1)

代表一個場景的結束，整個區段的資料只有 Visual Object Sequence End Code:000000B1 4 個 Bytes 而已，所以此區塊資料會和上個階層 VOP 的資料區塊合併為一，同樣的也會設置一個旗標，表示在緩衝區中的資料是完整的 frame。此時會再將解析器的狀態設成起始處理程序:Visual Object Sequence。

3.2.2 MPEG-2 檔案解析

一個 MPEG-2 的檔案包含了音訊與視訊，從 2.2.1.1 小節中可以知道整個 MPEG-2 檔案的結構，可以將其視為三個階層:第一層為 Pack header，第二層為 System header，第三層為 PES，其中真正的 frame 資料是存在 PES Packet 中，在此小節中主要說明如何從 MPEG-2 Program stream 中的 PES Packet 分離出音訊流和視訊流，而音訊流和視訊流的解析則不多作說明，因其解析方式和 MPEG-4 雷同，有所差異的部份只是在於兩者的編碼語法上。

如同在 3.2.1 節中所提到一樣,可以將 Pack header、System header 與 PES 視為三個獨立的處理程序，但彼此之間具有層次關係，因此每一個處理程序進行完畢後便會視其之後的 start code 的值而將狀態設至相對應的處理程序，不過有一點最大不同的地方在於此三個程序所解析的資料並不是真正所需要的資料，所以並不會儲存到輸出緩衝區中，參考 figure3.8 所示，整個 MPEG-2 Program Stream 的解析模式可以劃分成三個大區塊，最上層的區塊(MPEG Program Stream Parser)包含了 Pack header、System header 與 PES Packet 三個獨立的處理程序，在這個區塊中將辨別出 PES 所攜帶的 payload 為視訊流或是音訊流後，將依照辨別出的資訊進入(MPEG Video Stream Parser)和(MPEG Audio Stream Parser)其中一個裡，而這三個區塊會各自會設定本身 parser 的狀態，以便在整個 loop 運行不會有所混亂，下將說明(MPEG Program Stream Parser)裡三個處理程序的內容。

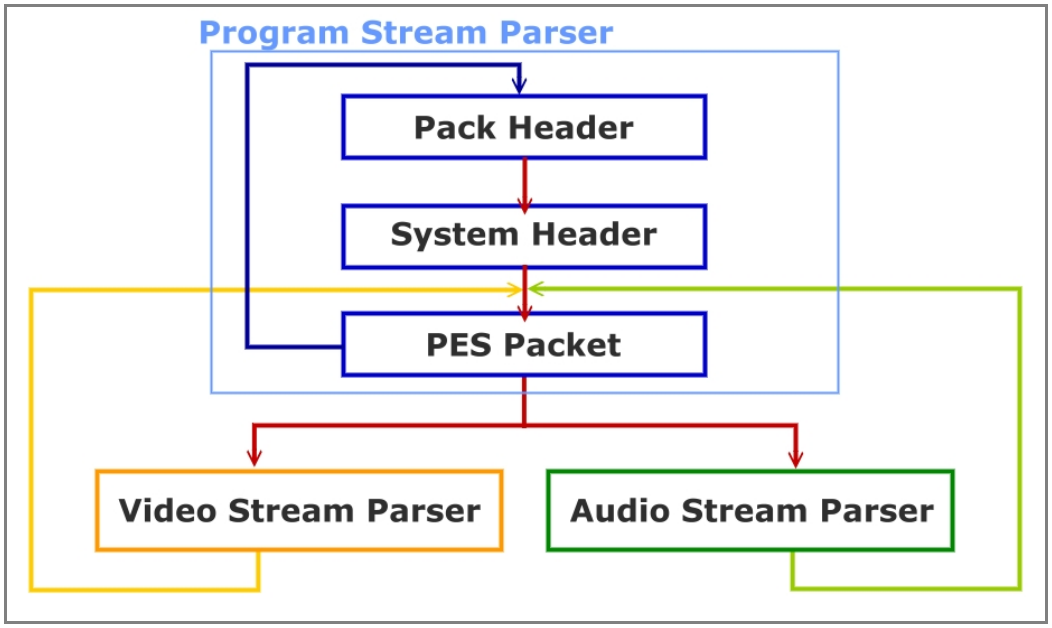

Figure3.8 MPEG-2 Program Stream Parse Model

1. Pack Header (000001BA)

開始從來源檔案的輸入緩衝區中讀入 MPEG-2 位元流，此時會不斷的搜尋位元流，直到找到 Pack Header start code:"000001BA"後才會真正的進入解析的處理程序中；在此 header 所挾帶的資訊中，可以判斷編碼類型為 MPEG1 或是 MPEG2。而在位元流中讀到 System Header start code 後便會結束這個階段的工作並將 parser 狀態設成下個處理程序: System Header。

2. System Header(000001BB)

在此階段最主要是以 System Header 本身的位元流長度來判別編碼格式的正確與否。並在讀到 PES Packet start code:00000100 後結束此階段的程序並將 parser 狀態設成下個處理程序: PES Packet。

3. PES Packet(00000100)

在這個程序中最主要的任務便是解析 sream_id 的值，經由

stream_id 的值便可知曉此一 PES packet 的類型為音訊流或是視訊流，且在這 header 中也會告訴解碼器此一 packet 的大小為何；當 stream_id 告知了 PES packet 中所裝載的類型為 video 或是 audio 後，便會進入相對應的區塊繼續解析並產生 frame 並放入到不同的輸出緩衝區中。而 MPEG Program Stream Parser 的狀態會一直停留在此，直到此一 Pack packet 被解析完畢，才會將 parser 狀態設成起始程序:Pack Header。

3.3 封包切割與包裝

當檔案資料經過解析並產生一段段的 frame 放在輸出緩衝區後，接下來便是要將這些 frame 打包放入封包中並加上適宜的 header 以進行傳送。

通常在網際網路上傳輸封包的大小都有限制，參考 table3.6，像是在 Ethernet 上封包最大能傳輸的限制為 1500 Bytes，1500 Bytes 為 MTU（Maximum Transfer Unit，最大傳輸單位），在 OSI mode 所定義的 Data-link layer 中 MTU 代表的是傳輸的最大限制，而在 Network layer 中則會因 Data-link layer 的 MTU 而對 frame 做切割的動作。

Table3.6 MTU limited table

Network Category	MTU(Bytes)
FDDI	4352
Ethernet	1500
IEEE 802.3/802.2	1492

如 figure3.9 所示，在乙太網路上傳輸的 IP 封包具有三層 header，分別是 IP header、UDP header 和 RTP header 共佔 40bytes 的大小，也就是說真正傳送的資料量(RTP payload)為 1500 Bytes(MTU)減去

header 的長度 40 Bytes 為 1460 Bytes，如果一個多媒體串流伺服器故意設定 RTP payload 的大小為 1470 Bytes 則會造成系統額外的負擔，因為 Network layer 會把一個 1470Bytes 的 frame 切割成二個封包，除了造成傳送端的麻煩，亦會造成接收端在封包重組上的負擔。

Figure3.9 A RTP Packet

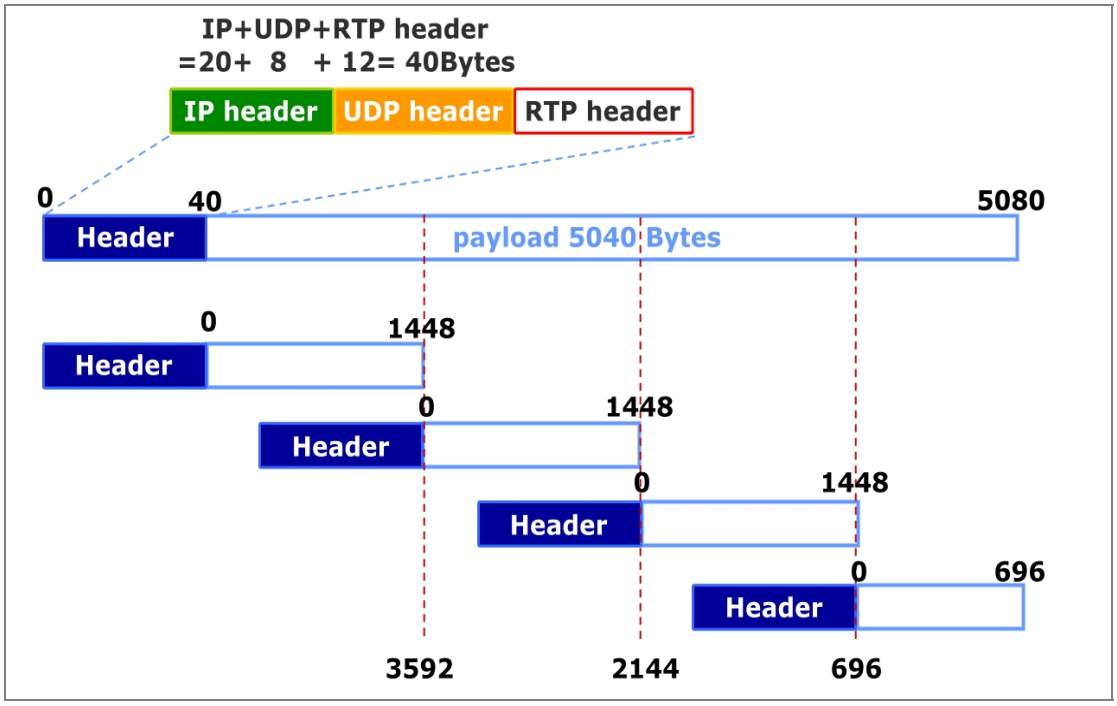

Figure3.10 frame 切割與封裝示意圖

而從 2.3 節的說明可以知道，不管是 M4V 或是 MPEG-2 的封包封裝都是以一張圖片(或是一段音訊)的大小和 MTU 兩者之間較小的值作為實際傳送封包的大小，以 M4V 來說明，當一張 VOP 過大，無法全部放入同一個封包中時，則會對 VOP 進行切割，將其分割並封裝成好幾個封包來進行傳送；假設多媒體串流伺服器設定的 RTP payload MTU 大小為 1448 Bytes，現有一張 VOP 大小為 5040 Bytes，因此原始封包

的大小為 5040 Bytes 再加上所有的 header (IP+UDP+RTP)長度 40 Bytes 總共為 5080 Bytes，若須經過乙太網路傳輸，因為伺服端對 payload 大小有所設定的關係，因此在經過切割後每個封包最大 payload 的長度為 1448 Bytes，所以會產生如 figure3.10 中所示的四個新的封包，每一個封包都會再配置 header 後在進行傳送，所以一個封包的總長度為 1488 Bytes，這個作法即是封包的封裝與切割方式。

3.4 封包傳送機制

當檔案資料經過前面敘述的一連串程序形成一個個的封包放置在輸出緩衝區後，需要一個機制去判斷在緩衝區的資料是否可以傳送，這個機制並不是指傳送時間的控制，而是針對封包的資料內容做判別，在此所須要對緩衝區內的資料偵測的事項為下列條件，只要符合其中一項便達到傳送的標準:

- Frame 經過切割後，符合預先設定的封包大小。
- 當 Frame 沒有超過預先設定的封包大小時:
  - 封包內為 frame 的最後一段資料，且並未包含其它新的 frame。
  - 封包內為一個完整的 frame。

第四章 多媒體串流系統與 RTP 封裝演算法

在第三章中說明了整個「封包包裝和傳送」模組的工作內容及運作方式，在本章中將分析 Live555 Server 的架構，簡單說明整個多媒體串流系統的結構，並著重在「封包包裝和傳送」模組實現的方式，主要討論影音檔案如何經過解析與切割以及如何包裝成合適於網路傳輸的封包。

Live555 是一套提供 Source code 的多媒體串流的 C++函式庫，這套函式庫實做了 RTP/RTCP/RTSP/SIP 等標準協定。Live555 source code 可以在 Unix、Window...等平台下進行編譯，且 Live555 為 GNU Lesser General Public License(LGPL)自由軟體。

Live555 source code 包含了伺服端以及客戶端部分，這套函式庫可以用來傳送和接收多媒體串流資料。配合各種編碼器可以整合成為 video on demand 串流模組，也可配合相關的解碼器以及應用程式整合成為串流播放模組，如 VideoLan(VLC Player)以及 MPlayer。Live555 目前可以支援多種 codec 的標準 RTP 封裝及解封裝，如 AC3、AMR、H.261、H.263、JPEG、MP3、MPEG-1-2、MPEG-2TS、MPEG-4 ES、MPEG-4 Audio、CELP 以及 WAV 等。在以下各小節中將分別說明 Live555 伺服端中如何對資料進行解析與封包包裝。

4.1 多媒體串流系統伺服端架構

多媒體串流系統伺服端主要是由三個模組部份所組成:「標準連線程序建立」、「RTSP 信息溝通協調」以及「封包包裝和傳送」。其中

「連線建立標準程序」模組主要使用 BSD Socket API 來完成一個伺服器的初始化設置，也就是依照串流程式的需求來建立 TCP/UDP 模式的 socket ；「RTSP 信息溝通協調」負責了伺服端和客戶端兩者的連線溝通，而「封包包裝和傳送」則是實現特定的封裝演算法，產生適合網路環境的封包並且傳輸。

如 figure4.1 與 4.2 所示，伺服端對客戶端所請求播放影音檔案的整個處理流程最主要是由「RTSP 信息溝通協調」與「封包包裝和傳送」兩個模組所實現。在第三章中對於「來源檔案的建立與類型判定」討論裡提出了兩種實現方式，第一種為讀取檔案開頭的小段位元流後與各類編碼語法進行比對來判斷類型，第二種則是以檔案的副檔名(Content Type)做為辨別依據，而在本章中所論述的實現方法是採用第二種，並將此部份從「封包包裝與傳送」模組中劃分出放入「RTSP 信息溝通協調」模組內。因此在「RTSP 信息溝通協調」模組中的“DESCRIBE”最主要是因應客戶端的要求，檢查在伺服端是否有客戶端所要求的檔案存在，並解析此檔案的文件格式，進而選擇相對應的解析器與 RTP 封包包裝模式。而“PLAY”則是客戶端告知伺服端可以開始發送 RTP 封包的請求，當伺服端收到此依請求時，便會進入「封包包裝與傳送」模組中，建立 RTP 封包、裝載資料並傳送，在此階段中所使用的解析器與 RTP payload 的裝載機制的模式是依據在「RTSP 信息傳送協調」模組選擇而來的。

伺服端對客戶端所請求串流播放的媒體檔案從檔案確認存在與否、檔案類型判斷、檔案內容解析及封包建立傳送...等等，整個處理程序可以分成下面四點:

- Set up source file
- Build RTP packet and set RTP header
- Pack frame in RTP packet
- Send RTP packet

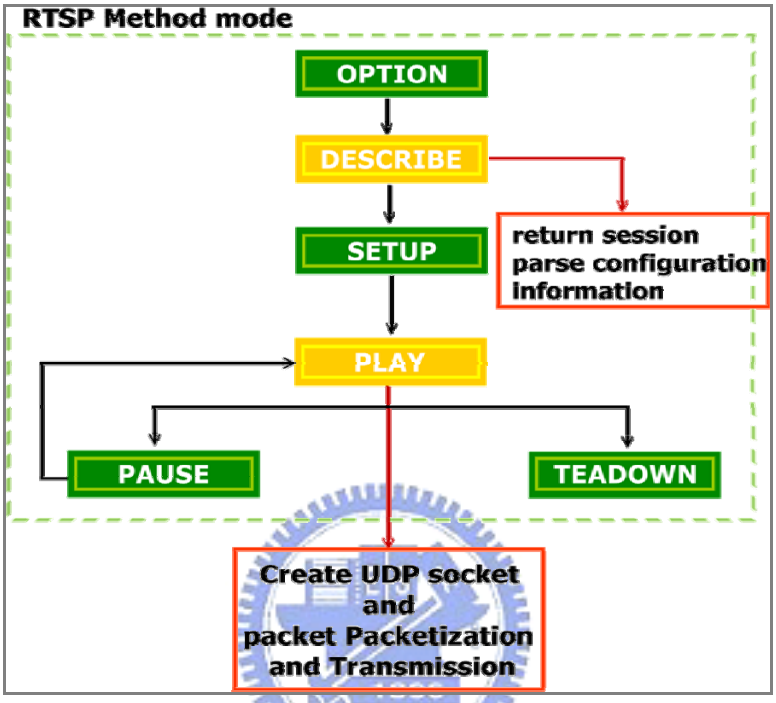

Figure4.1 RTSP Signaling Negotiation-RTSP Method Chart

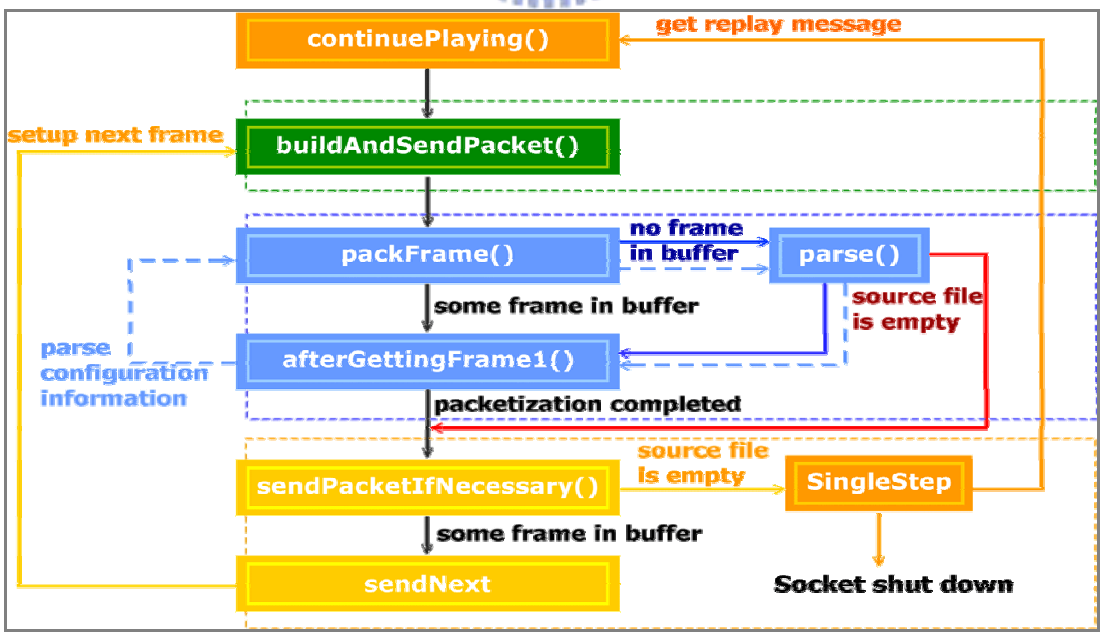

Figure4.2 packet Packetization and Transmission Chart

在下面的節次中將以figure4.1與4.2中所示流程分別說明上列四點

的實作細節，並以串流 MPEG4 類型的檔案做為輔助例子說明。

4.2 Input / Output Buffer Architecture

在開始說明整個實作細節前，首先要先說明在多媒體串流系統伺服端內部檔案資料存放的輸入與輸出緩衝區結構。在此系統中只有簡單的兩個緩衝區，第一個為存放檔案資料流的輸入緩衝區，第二個為存放經過解析器處理的視訊位元流或是音訊位元流以及 RTP 封包。以下將各別說明其結構、存取與實現方式。

4.2.1 Input Buffer Architecture-StreamParser ( )

輸入緩衝區 fCurBank 由 StreamParser ( )所建置，設置的目的主要是存放串流媒體檔案的資料流，為解析器所處理的資料來源處。在 fCurBank 中包含兩個動態陣列，分別為 fBank[0]和 fBank[1]，空間大小皆為十五萬位元組。當從檔案中讀入資料流時，首先會將 fBank[0]的空間填滿，之後便停止讀入的動作，當解析器所須要拿取的資料大小大於 fBank[0]內剩餘的資料量時，便會再次從檔案中讀入資料流填滿 fBank[1]，也就是說伺服端每次都從檔案內讀出 150000 bytes 做解析，fBank[0] 和 fBank[1]兩者為交替使用。

解析器對 fCurBank 進行讀出位元資料的動作通常是一次以 4 bytes 或 1 bytes 取出，但也可以是其它不同的 bytes 數取出，取出大小端看解析器處理分析的程序為何，主要是藉由 table4.1 中的函式進行運作。每當從 fCurBank 中取出資料時，會先行檢查 fCurBank 中的資料量是否足夠，如果不足，便會再從檔案中讀入 150000 bytes；當解析器將資料讀出後，fCurBank 也會將緩衝區目前偏移量以 fCurParserIndex 紀錄下來。

Table 4.1 fCurBank 存取相關 function

	Function	Instruction
1	ensureValidBytes( )	檢查 fCurBank 的狀態
2	ensureValidBytes1( )	從檔案讀入資料流
3	test4Bytes( )	從 fCurBank 取出 4 bytes，但不紀錄偏移量
4	getBytes( )	從 fCurBank 取出任意 bytes 數
5	get4Bytes( )	從 fCurBank 取出 4 bytes
6	get2Bytes( )	從 fCurBank 取出 2 bytes
7	getByte( )	從 fCurBank 取出 1 bytes

4.2.2 Output Buffer Architecture-OutPacketBuffer ( )

輸出緩衝區 fBuf 是由 OutPacketBuffer( )所建置，主要是擺放 RTP 封包與解析器處理後的資料。fBuf 是一個動態陣列，預設大小為 60816 bytes，可以裝載大小為 1448 bytes 的 RTP 封包共 42 個，1448 bytes 是預設傳輸的 RTP 封包的最大值。參考 figure4.3 所示，fBuf 使用的方式，一開始會先置入 RTP header，並紀錄 RTP header 的起始位址，這也是 RTP 封包的起始位址(fPacketStart)，且會以 fCuroffset 變數紀錄 fBuf 中的偏移量，fCuroffset 表示封包要傳送的長度。之後會放入解析器產生的 frame，相關儲存函式可以參考 table4.2，每次儲存的 bytes 是看解析器由 fCurBank 中取出多少，當存入 fBuf 後，會紀錄目前緩衝區的偏移位址(fTo)；每次結束一段解析的程序，解析器會回傳此次解析的資料量(framesize)，假使尚未達到傳送標準，則會將原先的 fCuroffset 的值再加上 framesize 形成新的偏移量。

如果 RTP header 加上 frame 的大小超過預設的封包的大小，則會對 frame 進行切割的動作，並紀錄 frame 溢位資料量(overflowBytes)，frame 切割的方式請見 4.5.2 節；每做一次切割就會在剩餘的 frame 之前在加入新的 RTP header，直到整個 frame 傳送完畢，因此需要設置

多個指標或是變數來紀錄資料在 fBuf 中的偏移量以及 fBuf 的狀態。

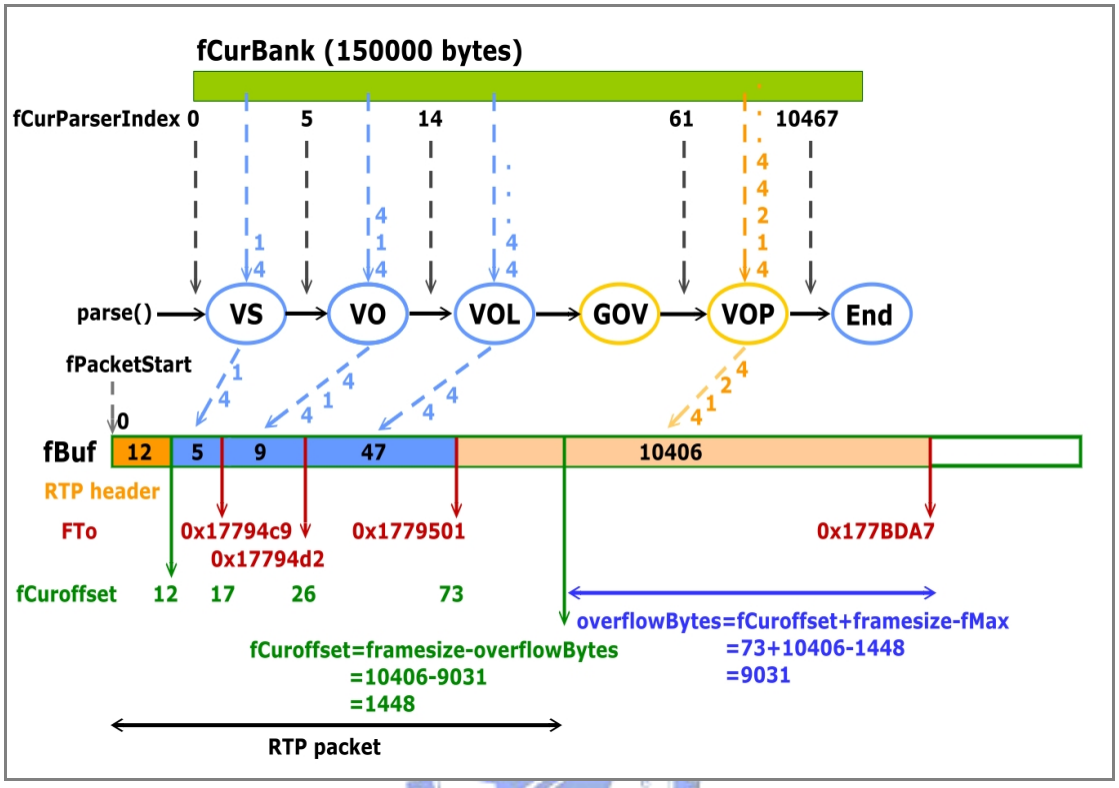

Figure4.3 Input/Output Buffer-使用示意圖

Table4.2 fBuf 儲存函式

	Function	Instruction
1	saveByte ( )	將 1 bytes 的資料存入 fBuf
2	save4Bytes ( )	將 4 bytes 的資料存入 fBuf
3	saveToNextCode( )	先判別要存入 fBuf 的 4bytes 是否為 start code，若不是則存入 fBuf 中。

4.3 Set up source file

「RTSP 信息溝通協調」主要負責伺服端和客戶端間連線溝通，在這階段中會解析 RTSP 信息封包，主要由 incomingRequestHandler1() 來解析出客戶端送出的要求，並呼叫相關處理函式，相關程序流程可以參考 figure4.2；在此關於來源檔案的建立、解析器與封包包裝格式的選

擇以及「封包包裝與傳送」模組的啟動主要來自於“DESCRIBE”和“PLAY”的請求。Table4.3 中簡單說明了 RTSP 信息模式，在下面二個小節會在對“DESCRIBE”和“PLAY”作進一步說明。

Table4.3 RTSP Method

RTSP 信息	簡要說明
OPTION	客戶端向伺服端詢問支援哪些RTSP信息;伺服端收到此信息後必須傳送封包把它所支援的RTSP信息告知用戶端。
DESCRIBE	客戶端利用DESCRIBE這個信息向伺服端要求有關影音檔案的相關資料; 伺服端收到這個信息後便先行解析該檔案，找出該檔案的影音編碼資訊並以SDP協定回傳。
SETUP	SETUP 是客戶端將它設定的傳送機制(machanism)傳給伺服端，以便伺服端依照此機制開啟一個新的串流連線，倘若此機制伺服端不支援，連線便結束。
PLAY	客戶端經由 PLAY 告知伺服端可以開始傳送 RTP 封包，而伺服端會依照 SETUP 時所設定的串流機制來傳送封包。
PAUSE	客戶端要求伺服端暫時停止串流的動作，但伺服端並不會釋放目前所有的串流資源。
TEARDOWN	客戶端要求伺服端釋放掉串流佔用的資源，也就是TCP/UDP socket的關閉動作。

4.3.1 handleCmd_DESCRIBE ( )

客戶端利用“DESCRIBE”向伺服端要求播放影音檔案的詳細描述資訊，當伺服端接受這個請求後，會執行 handleCmd_DESCRIBE( )，其處理的事項可以參考 figure4.4 的流程。handleCmd_DESCRIBE ( ) 一開始會先讀取用戶端要求播放的影音檔案檔名，抓取檔案名稱字串中的副檔名部份比對出相應的處理程序，以 test.m4v 為例，會對映到 MPEG4VideoFileServerMediaSubsession 此一處理程序，這表示之後整個串流程序中對檔案的處理由它來做統籌，在這個處理程序中主要包

含了檔案的存在與否的確認及開啟，以及解析器和 RTP 封包包裝模式選用與啟動。

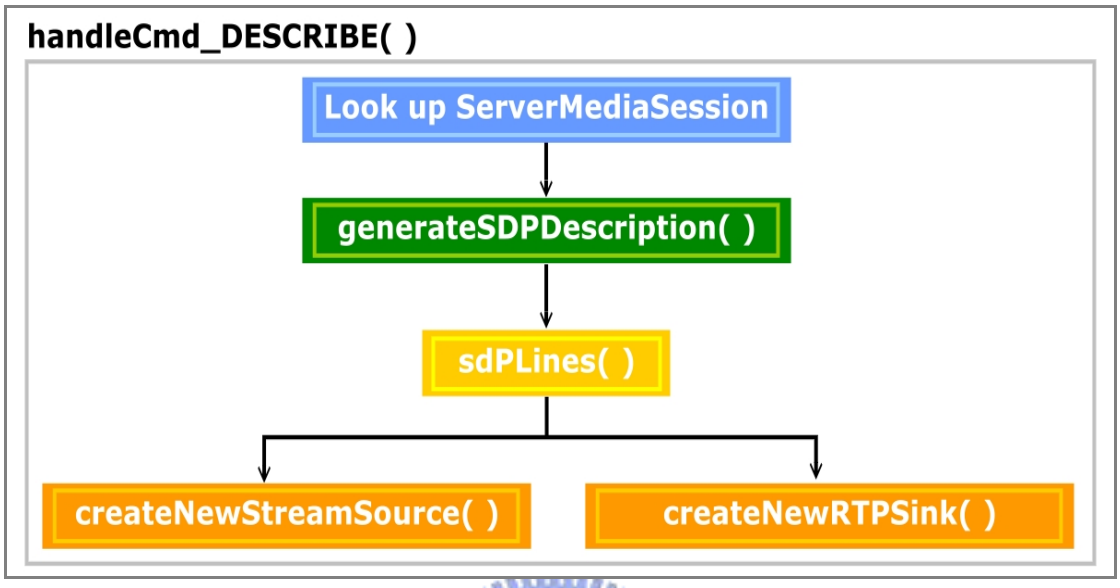

Figure4.4 handleCmd_DESCRIBE ( ) Flow Chart

當選擇好處理程序後，接下來伺服端要傳送關於檔案的相關資訊給客戶端，此時會以 SDP 協定描述的方式將資訊放在封包中傳送，因此會呼叫 generateSDPDescription ( )來產生關於描述多媒體會議(session)的文字檔，而 generateSDPDescription( )會呼叫 sdpLines( )，sdpLines( )在此階段中是一個重要的程序，主要處理的事項為進入一開始所選擇好的處理程序:MPEG4VideoFileServerMediaSubsession 中，依序進行:

- 建立 stream source–createNewStreamSource ( ):
  - (1.) 利用 fopen( )開啟準備串流的檔案，此時只是利用 fid 這個指標告訴伺服端檔案是否存在以及檔案所存放的位置；並沒有將檔案內容讀入到伺服端的輸入緩衝區中。
  - (2.) 啟動相應的解析器，完成其初始化，在此所啟動的解析器為 MPEG4VideoStreamFramer::parse ( )；並且經由呼叫 StreamParser ( )執行建置輸入緩衝區 fCurBank。

- 建立 RTP 封包包裝的相關環境-createNewRTPSink( ):

主要因應所選檔案類型啟動相應的 RTP 封包包裝環境，並在啟動的過程中回傳多媒體會議(session)描述文字檔中所需要的訊息，像是:rtpPayloadType、rtpTimestampFrequency、rtpPayloadFormmatName...等；在此所啟動的 RTP 封包包裝環境為：MPEG4ESVideoRTPSink，回傳的訊息中像是 rtpPayloadType 為 96、rtpTimestampFrequency 等於 90000 與 rtpPayloadFormmatName 是"MP4V-ES"。另外，會先預先建置或設定下列各項:

(1.) setPacketSizes( )-RTP 封包大小設定:

包含 RTP header 所允許傳送的 RTP 封包大小範圍為 1000~1448 bytes，最大值限制為 1448 bytes，也就是說封包大小超過 1448 bytes 便須要對 frame 進行切割處理。

(2.) OutPacketBuffer ( )-建置輸出緩衝區 fBuf。

(3.) 利用 random( )產生 RTP header 部分欄位的初始值:

a. Timestamp:

產生 fTimestampBase 作為 Timestamp 的初始值。

b. SSRC

c. Sequence Number

(4.) Special case :

如果串流的媒體檔案的編碼語法結構中包含了組態資訊(configuration information)，像是 MPEG-4ES，則在此會先執行 startPlaying( )中的 continuePlay ( )進入「RTP 封包包裝與傳送」模組，最主要的目地是要進入 parse( )內先行解析出檔案資料的組態資訊回送給客戶端，並不是真正開始要傳送 RTP 封包，先行解析出的組態資訊會同時放入一個暫存的記憶

體空間和輸出緩衝區中，回傳訊息時是由暫存的空間拿取資料進行傳送，而在輸出緩衝區的資料會在等到收到客戶端傳送“PLAY”請求後才啟用。

假如串流的媒體檔案包含了音訊與視訊，比如 MPEG-2 檔案，則會為視訊流和音訊流各自建置解析器與 RTP 相關環境，並且也會建立兩個輸出緩衝區將音訊流與視訊流各別放入，因此在「封包包裝與傳送」模組運作上能輕易的辨認目前裝載的資料類型，進而設定 RTP header 的欄位。

4.3.2 handleCmd_PLAY ( )

客戶端透過傳送“PLAY”這個訊息向伺服端請求開始傳送封包，當伺服端接受“PLAY”這個請求時，會執行 handleCmd_PLAY ( )。handleCmd_PLAY ( )會呼叫 startStream( )來處理伺服端要回應給客戶端的訊息資訊，它會將 Sequence Number 和 Timestamp 的初始值回傳給客戶端，並會執行 startPlaying( )中的 continuePlay ( )，代表要開始傳送封包，也表示從「RTSP 信息協調溝通」進入到「封包包裝與傳送」的階段。

4.4 Build RTP packet

當伺服端進入此階段，表示要開始傳送 RTP 封包，從開始傳送第一個封包一直到整個影音檔案被傳輸完畢，整個處理程序是一個大型迴圈，如 figure4.2 所示，處理程序會不斷的從 buildAndSendPacket( ) 執行到 sendPacketIfNecessary ( )，再從頭開始，直到確認來源檔案已經傳送完畢才會停止。

4.4.1 continuePlay ( )

continuePlay ( )是整個「封包包裝與傳送」的進入點，當continuePlay ( )被呼叫執行時表示要開始傳送第一個 RTP 封包，因此會呼叫 buildAndSendPacket ( )來建立 RTP 封包。

4.4.2 buildAndSendPacket ( )

buildAndSendPacket ( )主要是建立 RTP 封包，其方式是先在輸出緩衝區(fBuf)設置 12 bytes 的 RTP header，並依照傳輸檔案的格式選擇是否要再多增加 4 bytes 的 specialHeader 欄位，specialHeader 裝載的資訊是用來告知客戶端所接收到的 RTP 封包中的 payload 為音訊流或是視訊流，以 M4V 檔案來說，就不須要此欄位，因為它只有視訊流；而 MPEG-2 的檔案就須要這一個欄位，因為檔案中包含了視訊流與音訊流。RTP header 的設定方式參考 table4.4 和 figure4.5。

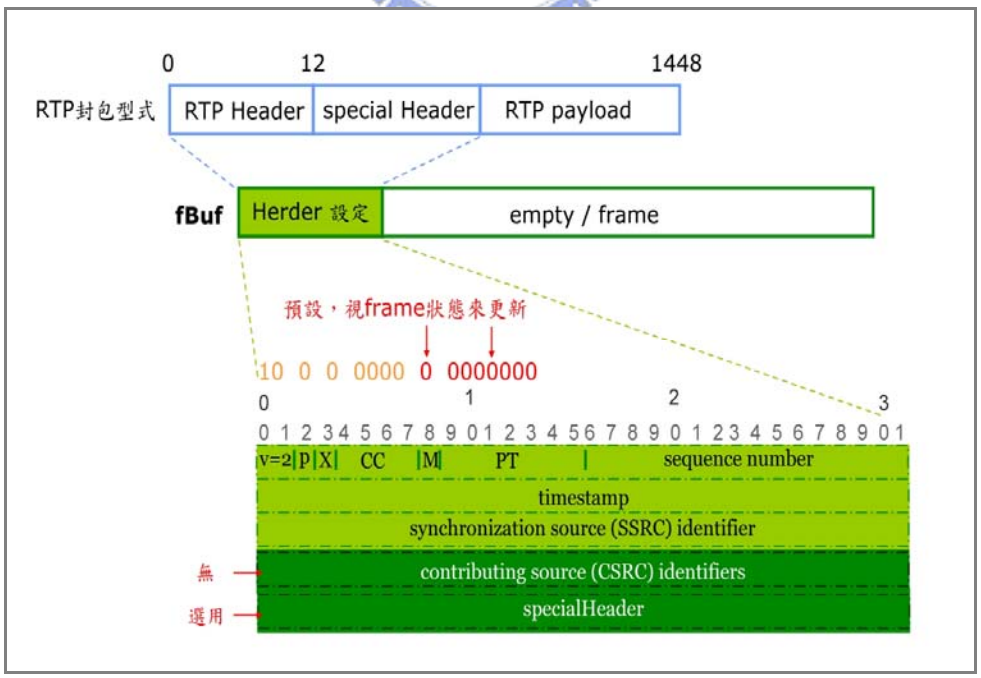

Figure4.5 RTP header 設定示意圖

Table4.4 RTP Header 設定

內容固定的欄位		
Field	**Value**	**Account**
V	10	指出代表 RTP 協定的版本，目前的版本為 2。
P	0	表示封包中沒有 padding data。
X	0	表示 RTP header 之後沒有跟隨額外的 header，在此把 specialHeader 視作 payload data 的一部份，因此 X 的位元值不會因為 specialHeader 存在與否做特別設定。
CC	0000	表示沒有不同媒體來源的封包經由 Mixer 混合在此一封包中，因此也無需 CSRC list。
非固定內容的欄位：表示欄位內的位元值會隨著 payload 的狀態更改。		
Field		**Account**
PT		表示封包的承載資料類型，會隨著封包裝載的資料類型改變 PT 的位元值。
M (Maker Bit)		初始值設定為 0，當在封包中裝載的是一張完整的 frame 或是一張 frame 末端的資料時，便會重新設定為 1，由 doSpecialFrameHandling( )來檢查 frame 的狀態以及設定 M 的位元值。
Sequence Number		初始值為 random( )產生的一個數值，每送出一個封包，Sequence Number 的值便會增加 1。
Timestamp		Timestamp 的數值是由 frame 的 presentation time 乘以 encoding type 的 clock rate 再加上一個基底值 fTimestampBase 計算來的。 由 doSpecialFrameHandling( )來設定 Timestamp 的欄位，在此階段中會先預留 4 bytes 的空間。

續 Table4.4

Field	Account
SSRC	由 random( )產生的亂數值。在同一個 session 中，每個 RTP 封包的 SSRC 欄位值都是固定的。

Timestamp 的設定在 4.5.2 節中會再作更進一步說明。

4.5 Pack frame in RTP packet

設定 RTP header 後，接下來要處理 payload data 的部份，產生完整的 RTP 封包進行傳送。

4.5.1 packFrame ( )

packFrame( )主要負責檢視輸出緩衝區(fBuf)的狀態，依據 fBuf 的狀態採取不同的處理方式，當 fBuf 內沒有任何的 frame 或是在 frame 中沒有看到預期的位元值，以 MPEG-4 的 RTP 封包包裝模式為例，如果在 fBuf 中沒有看到 VOP 的 start code:000001B6，就表示目前的 frame 中還沒有真正要傳送的資料存在，在這兩種情況下都會呼叫 parse( )，對來源媒體檔做解析以產生新的 frame 放到 fBuf 中，之後交由 aftergGettingFrame1( )處理，進行封包包裝的處理程序。

當 fBuf 中存有 frame 時，會紀錄 fBuf 中 frame 的相關資訊，包含了 frame 的大小，presentation time，並會檢視 frame 在 fBuf 中的位置是否有跟隨在 RTP header 之後，因為如果 frame 太大超過可傳輸封包大小最大的限制時，伺服端會對 frame 進行切割，所以每當送出一個封包後便會重新在剩餘的 frame 前插入新的 RTP header，因此須要檢查剩餘的 frame 是否有置放在新的 RTP header 之後，如果沒有的話，會將 frame 搬運到 RTP header 的後方；當收集完 fBuf 中 frame 的資

訊後，便會由 afterGettingFrame1( )來處理 fBuf 中的 frame，進行封包的包裝。

4.5.1.1 parse( )

在本文中一直提及的解析器是交由 parse( )所執行；parse( )是一個總稱，從 3.2 節中可以知道不同的檔案類型會啟動不同的解析器對檔案進行處理，parse( )所執行的任務如下:

- 檔案內容讀取

  當解析器經由 table4.1 中三至七項的函式讀取輸入緩衝區(fCurBank)資料流時，若 fCurBank 是無資料的狀態或是解析器所要求的資料數量大於 fCurBank 的資料量時，則會呼叫 ensureValidBytes1( )進行讀檔的動作，也就說檔案內容真正的讀取是由解析器所啟動。

- 位元解析與訊框產生

  每種編碼格式都有其對應的處理程序，在 3.2 節中說明了 M4V 檔案與 MPEG-2 檔案解析的方式，在此以 M4V 檔為例解說明 MPEG4VideoStreamFramer::parse ( )對整個檔案資料處理的方式，而詳細位元解析意義請參考 3-2 節的說明。

  參考 figure4.6 所示，在 parse( )中可以依 MPEG-4 的編碼結構區分成六個區塊:

  - parseVisualObjectSequence( )
  - parseVisualObject( )
  - parseVideoObjectLayer( )
  - parseGroupOfVideoObjectPlane( )
  - parseVideoObjectPlane( )
  - parseVisualObjectSequenceEndCode( )

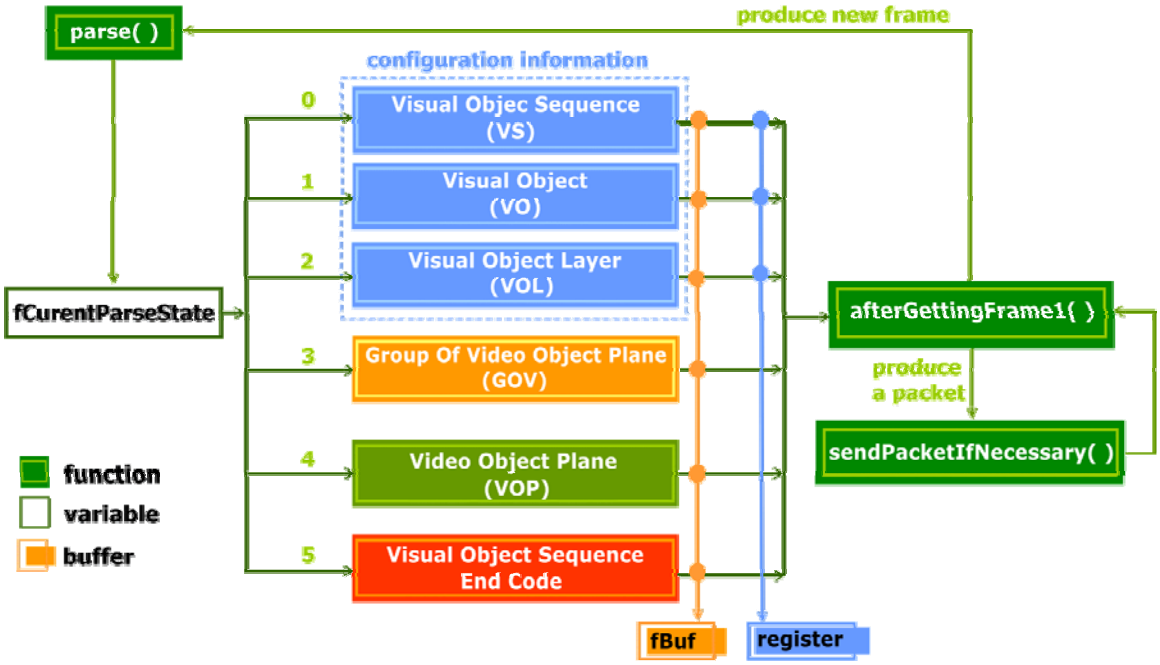

Figure4.6 parse ( ) model

Table4.5 fCurrentParseState 與 process 對映表

Value	State / function
0	PARSING_VISUAL_OBJECT_SEQUENCE parseVisualObjectSequence( )
1	PARSING_VISUAL_OBJECT parseVisualObject( )
2	PARSING_VIDEO_OBJECT_LAYER parseVideoObjectLayer( )
3	PARSING_GROUP_OF_VIDEO_OBJECT_PLANE parseGroupOfVideoObjectPlane()
4	PARSING_VIDEO_OBJECT_PLANE parseVideoObjectPlane( )
5	PARSING_VISUAL_OBJECT_SEQUENCE_END_CODE parseVisualObjectSequenceEndCode( )

每一個區塊為一個小型的處理程序，而 fCurrentParseState 代表著目前解析器要進行的處理程序的狀態，table4.5 為 fCurrentParseState 的狀態值與六個區塊程序

的對映關係，在最初解析器被啟動的時候，fCurrentParseState 被預設為“PARSING_VISUAL_OBJECT_SEQUENCE”，因此開始進入 parse( )時，會先進入 parseVisualObjectSequence( )裡。在 parseVisualObjectSequence( )中一開始會從輸入緩衝區(fCurBank)中讀取 4 bytes 來辨別是否為 Visual Object Sequence start code:000001B0，由於是第一次進入到此程序中，此時 fCurBank 中並沒有任何的資料，因此會由 ensureValidBytes1( )先進行讀檔的動作，把檔案資料放入緩衝區中後，再繼續回到程序中進行解析。

在 parseVisualObjectSequence( )中，解析器會不斷的讀入 4 bytes 來做判別，直到看見 Visual Object Sequence start code 後才會停止，並會先將 Visual Object Sequence start code 存入輸出緩衝區(fBuf)中，才會繼續對之後的位元流進行分析與儲存的動作，解析過程中所讀入的位元流的 byte 數的多寡是依照編碼的語法結構而決定，當解析器看見 Visual Object start code:000001B5 時會結束 parseVisualObjectSequence( )的運作，並將 fCurrentParseState 設定成下一個處理程序的狀態值:“PARSING_VISUAL_OBJECT”。

當結束 parseVisualObjectSequence( )後，會先進入到 afterGettingFrame1 ( )中檢查 fBuf 的 frame 狀態，由於此時 fBuf 中的 frame 並沒有 VOP 的 start code:000001B6 的存在，因此不符合封包傳送的標準，所以又會再回到 parse( )中繼續進行檔案資料的解析及產生新的 frame，此時會依據 fCurrentParseState 來決定進入那一個程序中。

上述為 parseVisualObjectSequence( )的執行模式,其它程序也都依循這樣的模式進行運作，關於 fCurrentParseState 詳細的設定順序可以參考 3.2.1 小節 M4V 檔案解析以及 figure3.3 的說明。

另外 Visual Object Sequence 、Visual Object 以及 Video Object Layer 為 MPEG-4 檔案的組態資訊，是伺服端在回送 RTSP 信息時須要傳送的資訊，因此這三個區段的資料都會由 appendToNewConfig( )另外存入到一個暫存記憶體的空間中以供 handleCmd_DESCRIBE ( )使用。

當進入 parseVideoObjectPlane( )，會先解析 7 bytes 的 header 資訊，第一次讀入 4 bytes 為 start code，第二次讀入 1 bytes，辨認 vop_coding_type，之後再讀入 2bytes，找出時間資訊的參數 modulo_time_base 與 vop_time_increment。解析 vop_coding_type、modulo_time_base 和 vop_time_increment 是為了計算 presentation time。

解析 header 後，會從 fCurBank 中不斷的讀入 4 bytes 的資料並判別其是否為其他程序的 start code，每一次都從 fCurBank 中讀取 4 bytes 是因為 start code 這個特殊編碼值的長度在 MPEG-4 編碼語法中都是 4 bytes 的緣故；如果讀入的位元值不是 start code 就會存入 fBuf 中，假設是的話，表示已經讀到一張 VOP 的底端，伺服端會把 fPictureEndMarker 這個旗標設為 true 的狀態，表示一張 VOP 的結束，並會依所解析的 start code 設定 fCurrentParseState 的狀態。在解析器進

入 parseVisualObjectSequenceEndCode ( )這個程序時，也會對 fPictureEndMarker 這個旗標的設定，此時表示了一個場景的結束。而在此階段中會依解析得到的時間參數執行 computePresentationTime( )計算 presentation time。

由於每執行完一個區塊程序即會執行 afterGettinFrame1( )來審視 fBuf 中 frame 的狀態，檢視 frame 中有無 VOP 的 start code，如果有 VOP start code 的存在時，便會開始進行封包的包裝與傳送，因此除了串流媒體檔案開頭的組態資訊外，解析器每次解析的資料量大小單位為一張 VOP 的大小。

4.5.2 afterGettingFrame1 ( )

afterGettingFrame1 ( )最主要是將 fBuf 中的 frame 裝載成一個 RTP 封包，並呼叫 sendPacketIfNecessary ( )進行傳送。其運作機制可分成三個階段如下所述:

- Stage 1-fBuf 資料量檢查

先檢查 fBuf 裡的資料量(RTP header +specialHeader +frame)大小是否有超過預設封包的最大值(fMax=1448 bytes)，如果有超過，表示有溢位情況發生，則會對 frame 進行切割；如果 fBuf 的資料量大小沒有大於 fMax，則再檢查 fCurFragmentationOffset 這個變數是否有值，fCurFragmentationOffset 是紀錄 frame 經過切割後在 fBuf 中的偏移量，如果 fCurFragmentationOffset 大於零，則表示目前在 fBuf 中的 frame，是經過切割後的 frame 的尾端部份，因此會設定旗標 fPreviousFrameEndedFragmentation 為 true 表示在 fBuf 中的資料是 frame 末端了。

- Frame 切割方式說明:

frame 的切割是由 numOverflowByte ( )所執行，參考 figure4.7 所示，伺服端首先會計算出 fBuf 中 frame 溢位資料(overflowBytes)的大小，是由下式完成:

overflowBytes= fCuroffset+framesize-fMax

fCuroffset 是紀錄目前 fBuf 中尚為送出的 frame 長度，framesize 為每次解析器處所產生的新的 frame 的大小值，所以 fCuroffset+framesize 為 fBuf 中資料的總大小值， fMax 為預設封包最大的長度 1448 bytes。

接著會計算要傳送封包的長度:

fCuroffset= framesize- overflowBytes

重新計算 fCuroffset 的值，也就是對 frame 進行切割，fBuf 前 1448 bytes 為所要傳送的封包。而在進行切割後會使用 setOverflowData( )紀錄 fBuf 中相關的資訊，包含了封包的起始與結束位址，被切割後剩餘的 frame 的起始位址以及剩餘 frame 大小的資料量(overflowBytes)...等。

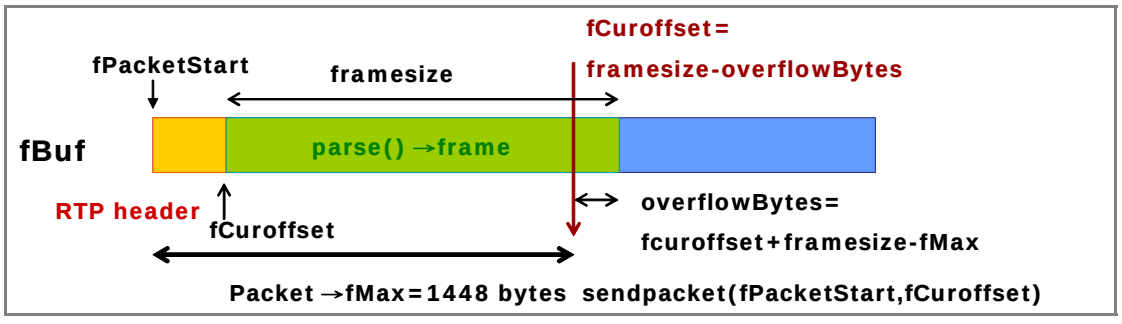

Figure4.7 fragment Frame

- Stage 2-doSpecialFrameHandling ( )

doSpecialFrameHandling ( )主要是檢視 fBuf 中的 frame 狀態，並依據 frame 的狀態更新 RTP header 中 Maker 和 Timestamp 的欄位。

(1.) 如果在 fBuf 中的 frame 是沒有經過切割的，也就是 fCurFragmentationOffset 為零時，會檢查 frame 中是否有 VOP start code，並把結果紀錄在旗標 fVOPIsPresent 中。

(2.) 如果在 fBuf 中，已經沒有溢位狀態存在，也就是 overflowBytes 為零，旗標 fPictureEndMarker 也為 true，則會把 RTP header 欄位中的 Maker bit 重新設定為 1。

(3.) 設定 RTP header 欄位的 Timestamp。

Timestamp 的產生和 presentation time 息息相關，也就是說 timestamp 的值是經由對 presentation time 做一特定計算而來。當解析器產生一張 VOP 的 frame 同時，伺服端也會得到此 VOP 的 presentation time，因為每兩張 VOP 之間的 presentation time 增加的時間間隔是固定的，都是以固定的毫秒增加，所以 Timestamp 的值也會隨著 presentation time 的變化而線性增加。當客戶端收到的 RTP 封包中 Timestamp 欄位的值都相同時表示這些封包裝載的都是同一張 VOP 的位元流。

Timestamp 計算由 convertToRTPTimestamp( ) 執行，其計算方式是由 VOP 的 presentation time 乘上 90000 再加上一個初始值 fTimestampBase 而來。90000 是 encoding type 的 clock rate，初始值 fTimestampBase 是由 random ( )產生。

- Stage 3-packet 傳送與否檢查

在這一階段中會根據前兩階段中對 fBuf 中的 frame 所做的分析與設定來判定是否可以進行 RTP 封包傳送。如果以下四個條件其中一項成立便會執行 sendPacketIfNecessary ( )來進行傳送。

- 滿足預設的封包大小，預設大小值在 1000~1448 間。
- 沒有超過最大的封包大小值，最大值為 1448。
- 如果 fPreviousFrameEndedFragmentation 為 true，表示封包中所裝載為 frame 的末端資料流，且這個封包中並沒有包含另一張 frame 的資料流，也可以進行傳送。
- 如果是單獨的一張圖片或是一段音訊流，也就是 fVOPIsPresent 值為 true 時，則未達到預設的封包大小設定也可以傳送。

如果上述條件都不成立，就會再回到 packFrame( )中運行，繼續產生新的 frame。

4.6 Send RTP packet

在這階段中表示輸出緩衝區(fBuf)已經有包裝完成的封包可以傳送，因此伺服端會啟動 sender 來傳送封包並且判斷檔案中是否還有資料要包裝成封包進行傳送。

4.6.1 sendPacketIfNecessary ( )

sendPacketIfNecessary ( )負責傳送 RTP 封包。由於輸出緩衝區(fBuf)只有一個，也就是說已經建制好的 RTP 封包和剩餘尚未包裝的 frame 都放在同一個緩衝區中，因此必須要告知伺服端的 sender 要傳送的 RTP 封包的起始位址以其所要傳送的範圍；當每送出一個 RTP 封

包時，則 Sequence Number 便會接著加一，且會再次檢查 fBuf 的狀態，當 fBuf 中還存有尚未被裝入 RTP 封包的 frame 時，則伺服端會重新設定下一個封包起始的位置，如 figure4.8 所示；也就是在 fBuf 剩餘的資料前插入新的 RTP header，並呼叫 sendNext ( )建置下一個 RTP 封包傳送。

下一個封包起始位址的設定的方式是將 fBuf 中剩餘 frame 的起始位置往前挪動 RTP header 加上 specialHeader 大小的空間，並且重新紀錄封包的起始位置，也就是說先在剩餘的 frame 前留下 12 或 16 bytes 的空位，等待 buildAndSendPacket( )來重新設定 RTP header 與 specialHeader 的欄位值。

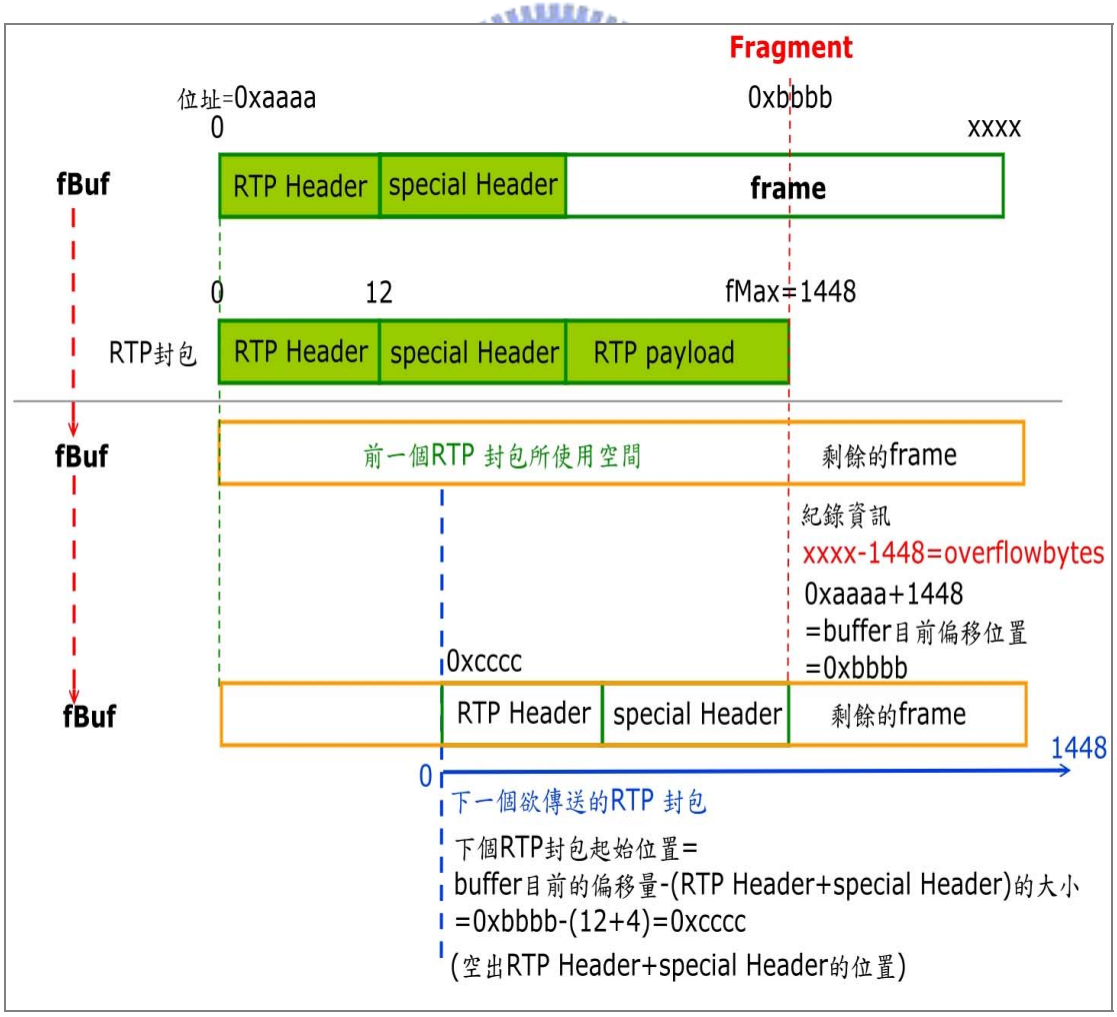

Figure4.8 Set up Next Packet

4.6.2 sendNext ( )

當 sendNext ( )運作時，表示要建立一個新的 RTP 封包，因此會呼叫 buildAndSendPacket( )，重新進行整個建置過程，包含 RTP header 設定，payload 裝載...等。

而在新建一個 RTP 封包的過程中，如果 output buffer 沒有資料，則會再 parse 新的資料，此時，會先檢查 input buffer 有沒有資料，如果沒有會進行讀檔動作，如果檔案指標已到檔案底部，表示整個檔案都被讀出，則會將 fNoFramesLeft 設成 true，表示已沒有資料要傳送。會再由 parse()回到 sendPacketIfNecessary ( )，進入 singlestep()等待下一個狀態。

4.6.3 SingleStep ( )

當整個串流的媒體檔案被完整傳輸到客戶端後，伺服端會離開「RTP 封包包裝及傳送」這一階段，回到 SingleStep ( )中等待下一個指令的指示，有可能是 user 要求要重新播放或者是關閉連線…等。

第五章 Experimental Results

在本章中以 Live555 做為伺服端，而客戶端使用 VideoLanClient (VLC)進行實驗，以下將各別呈現當伺服端與客戶端進行連線時，伺服端各部運作的實驗數據，串流媒體檔案為 test.m4v。

5.1 RTSP 信息溝通協調

在以下將列出 Live555 伺服端的「RTSP 信息溝通協調」模組中每個 Command function 在接收到客戶端的相對應的請求時所回應的內容，如伺服端收到“SETUP”這個請求時會利用 handleCmd_SETUP ( ) 做出回應。

5.1.1 handleCmd_OPTION ( )

客戶端利用 OPTION 來詢問伺服端支援那一些功能，伺服端則會將它所支援的功能送出，如 figure5.1 中的綠框所示。

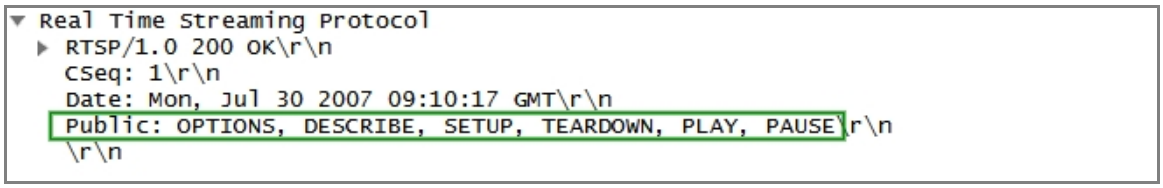

Figure5.1 OPTION - response information

5.1.2 handleCmd_DESCRIBE ( )

客戶端用“DESCRIBE”請求伺服端以何種描述方式來描述播放資訊，描述的方式有三種:application/sdl、application/rtsl、application/mheg，例如客戶端請求伺服端以 application/sdp 的方式來描述:

rtsp://140.113.13.82/test.m4v

則客戶端的請求會如 figure5.2 所示。

```
▼ Real Time Streaming Protocol
  ▶ DESCRIBE rtsp://140.113.13.82/test.m4v RTSP/1.0\r\n
    CSeq: 2\r\n
    Accept: application/sdp\r\n
    User-Agent: VLC media player (LIVE555 Streaming Media v2006.10.27)\r\n
    \r\n
```

Figure5.2 Client request- OPTION

由於所傳送的為 MPEG-4 ES 類型的檔案,所以伺服端會先進入 parse( ) 中解析影音檔案的 configuration information，並回應給客戶端，figure5.3 為伺服端進入 parse( )後解析檔案位元流後將 configuration information 抽出後暫存至記憶體中的數據圖，而伺服端會回應如 figure5.4 所示，。

```
parsestate=PARSING_VISUAL_OBJECT_SEQUENCE
Get source File data=150000

parsestate=PARSING_VISUAL_OBJECT_SEQUENCE
**********M4V configuration information**********
0 0 1 B0 F5
**************************
------------------------------------------------
parsestate=PARSING_VISUAL_OBJECT
**********M4V configuration information**********
0 0 1 B0 F5 0 0 1 B5 9 0 0 1 0
**************************
------------------------------------------------
parsestate=PARSING_VIDEO_OBJECT_LAYER
**********M4V configuration information**********
0 0 1 B0 F5 0 0 1 B5 9 0 0 1 0 0 0 1 20 8 C4 9D C0 0
43 A9 C0 9 50 0 B0 D4 97 53 C 1F 4C 2C 10 78 71 F 0 0
1 B2 65 6D 34 76 20 34 2E 33 2E 32 2E 38 0 C9 FF 0
**************************
```

Figure5.3 M4v configuration information 解析數據圖

在 figure5.3 中，綠色框線部份為進入 parse ( )程序中時先將檔案

資料讀入至輸入緩衝區中，接下來再依 MPEG-4 ES 語法結構分段解析出所需要的資訊，橘色框線為每次新增至暫存記憶體空間的資料。在 parsestate 為 PARSING_VIDEO_OBJECT_LAYER 階段結束時表示所有的 configuration information 解析完，由 handleCmd_DESCRIBE ( ) 回送給客戶端。

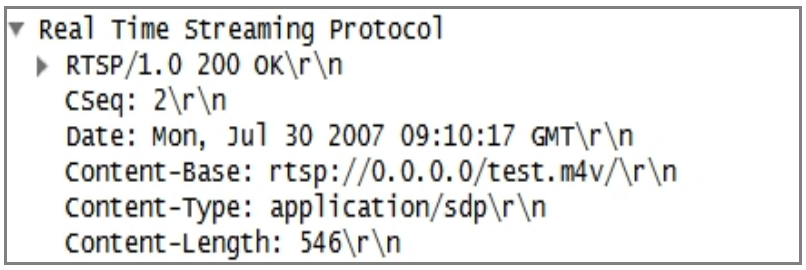

Figure5.4 (A) DESCRIBE - response information

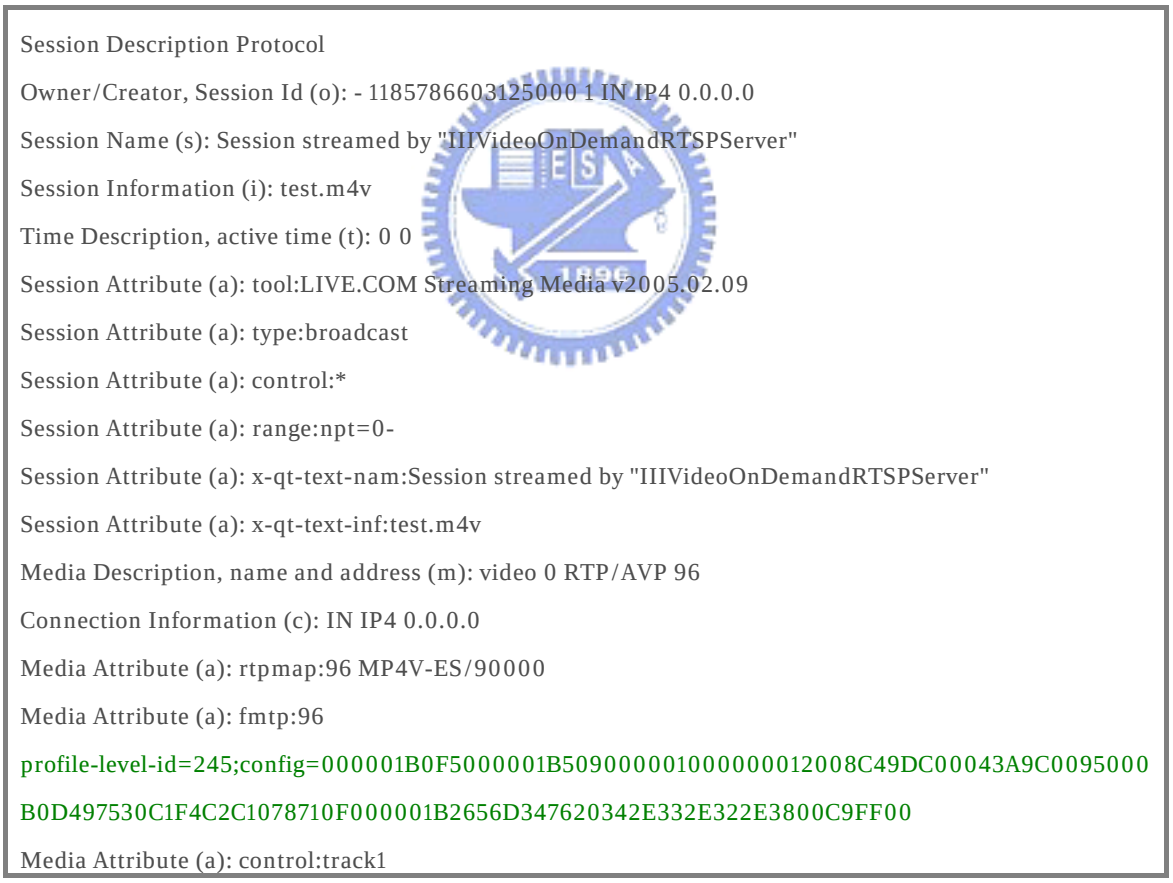

Figure5.4 (B) DESCRIBE - response information

在 figure5.4(B)中的綠色字體部份即為伺服端所回傳的 configuration information 部份，如果所串流的檔案類型為 MPEG-2，

則不會出現組態資訊的部份，如 figure5.4(C)所示。

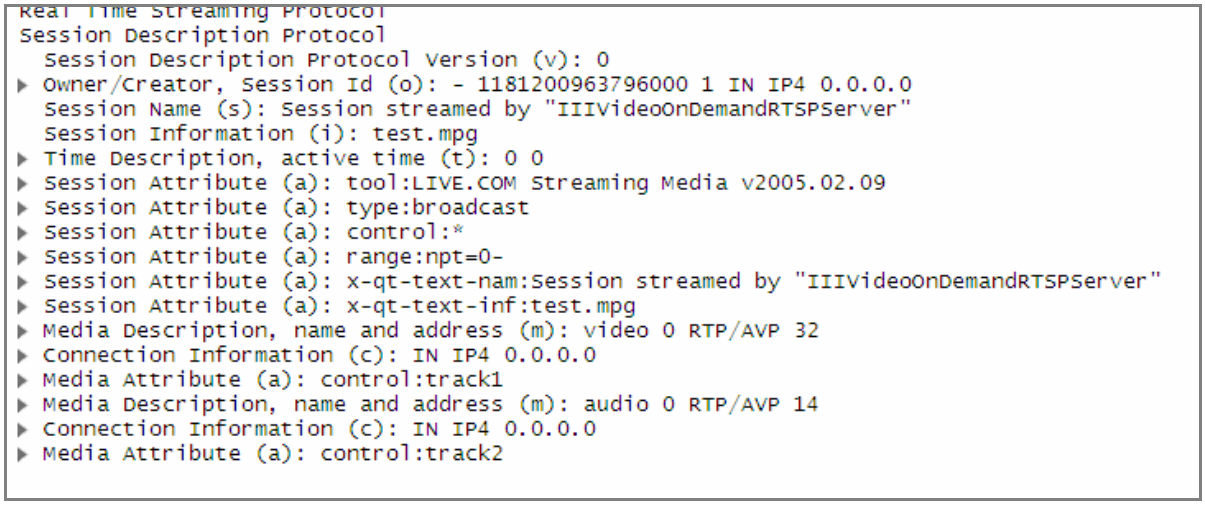

Figure5.4 (C) DESCRIBE - response information

5.1.3 handleCmd_SETUP ( )

客戶端使用“SETUP”向伺服端描述其串流的傳輸機制，如 figure5.5 所示客戶端向伺服端請求建立一個:

rtsp://0.0.0.0/test.m4v

的 RTSP 連線，而 RTP 封包則是在 port 2210 與 2211 進行接收。

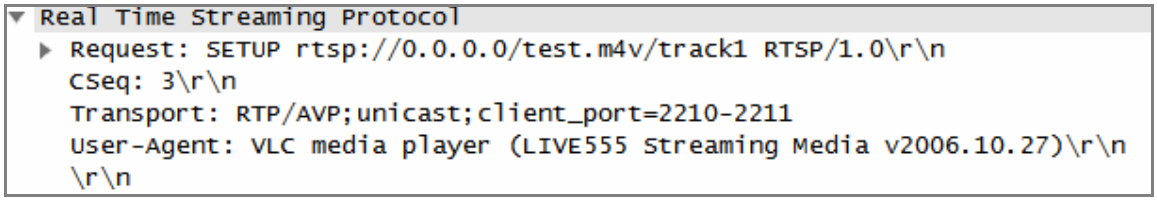

Figure5.5 Client request- SETUP

而伺服端會回應從 port 2500 與 2501 送出封包。如 figure5.6 所示。

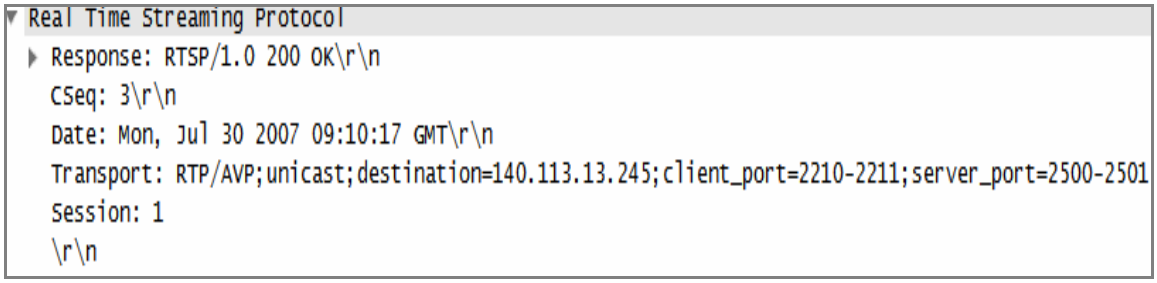

Figure5.6 SETUP - response information

5.1.4 handleCmd_PLAY ( )

客戶端使用“PLAY”向伺服端請求開始傳送封包，如 figure5.7 所示，其中 npt=0 表示請求從檔案起始處開始播放。而 figure5.8 則是伺服端回應給客戶端的訊息。

```
Real Time Streaming Protocol
  Request: PLAY rtsp://0.0.0.0/test.m4v/ RTSP/1.0\r\n
  CSeq: 4\r\n
  Session: 1
  Range: npt=0.000-\r\n
  User-Agent: VLC media player (LIVE555 Streaming Media v2006.10.27)\r\n
  \r\n
```

Figure5.7 Client request- PLAY

```
Real Time Streaming Protocol
  Response: RTSP/1.0 200 OK\r\n
  CSeq: 4\r\n
  Date: Mon, Jul 30 2007 09:10:17 GMT\r\n
  Range: npt=0.000-\r\n
  Session: 1
  RTP-Info: url=rtsp://0.0.0.0/test.m4v/track1;seq=24333\r\n
  \r\n
```

Figure5.8 PLAY - response information

5.1.5 Packet header information

```
Real-Time Transport Protocol
  [Stream setup by RTSP (frame 66)]
  10.. .... = Version: RFC 1889 Version (2)
  ..0. .... = Padding: False
  ...0 .... = Extension: False
  .... 0000 = Contributing source identifiers count: 0
  0... .... = Marker: False
  .010 0000 = Payload type: MPEG-I/II Video (32)
  Sequence number: 50021
  Timestamp: 1763532033
  Synchronization Source identifier: 1468872339
RFC 2250 MPEG1
  MBZ: 0
  T: 0
  Temporal Reference: 0
  AN: 0
  New Picture Header: 0
  Sequence Header: False
  Beginning-of-slice: True
  End-of-slice: False
  Picture type: I-Picture (1)
  FBV: 0
  BFC: 0
  FFV: 0
  FFC: 0
  MPEG-1 stream: 000001061BE2D706801E00E810BCC8618378140C492827AD...
```

Special Header

Figure5.9 RTP Packet Header Information-MPEG-2

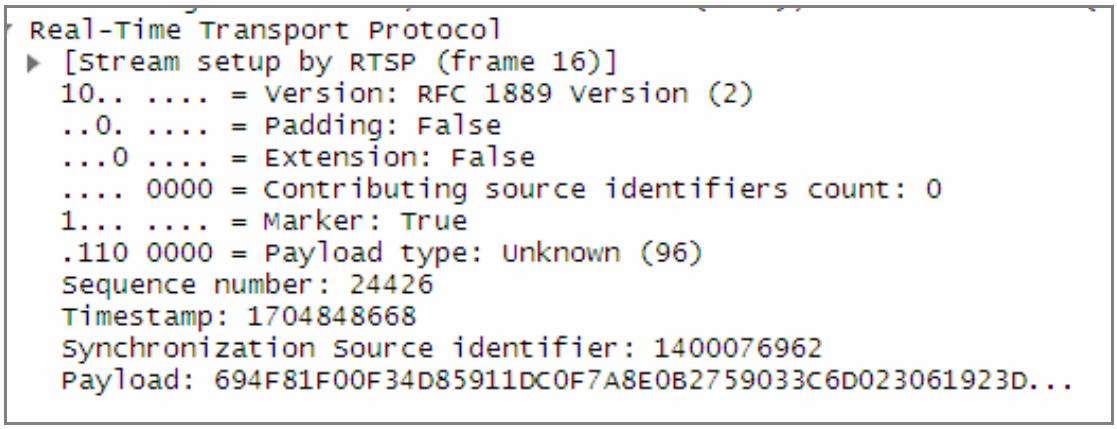

Figure5.10 RTP Packet Header Information-M4V

Figure5.9 為 MPEG-2 檔案 RTP 封包的 header 訊息，而 figure5.10 為 M4V 檔案 RTP 封包的 header 訊息，兩者相比較可以發現，在 MPEG-2 的 RTP 封包訊息中多出了 special header 的部份，這是因為 MPEG-2 檔案中包含了音訊與視訊，而依據 RFC2250 的包裝方式，其在 RTP header 後須要在跟隨 4 bytes 的 special header，可以參考 figure5.9 所示。

5.2 封包包裝和傳送

在 5.2.1 中將列出與說明整個「封包包裝和傳送」模組中資料處理的數據，而在 5.2.2 中則會說明 frame 切割和封包傳送的結果以及 Output Buffer 指標在模組運作時移動方式。

5.2.1 Parsing Results

在此小節中將列出整個「封包包裝和傳送」模組中對一影音檔案進行解析與包裝的數據整理，figure5.9 為原始影音檔案 test.m4v 的內容，而 table5.1 則是在伺服端進行測試的數據整理；table5.1 中列出在「封包包裝和傳送」模組中每一程序所進行的事項，主要列出 parse( )切割檔案後每個區段的資料，表格中包含了程式執行狀態以及所處理過的資

料內容，整個數據表的資料區段為檔案一開頭的 configuration information 至第一張 VOP 的結束。

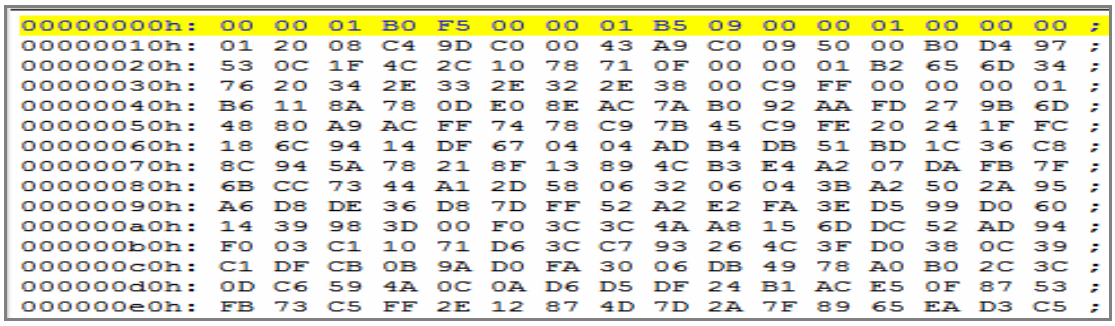

Figure5.11 Source File

Table5.1 Parsing Results

1 Function	**buildAndSendPacket()**
處理事項/狀態	Set RTP Header fCuroffset=12
Size(bytes)	12
資料內容	RTP Header=80605F0D 包含: RTPPayload=96 (Hex=60) Sequence Number=24333 (Hex=5F0D)
	Timestamp=00000000 SSRC=1400076962 (Hex=53737AA2)
2 Function	**packFrame( )→parse( )**
處理事項/狀態	parsestate=PARSING_VISUAL_OBJECT_SEQUENCE →ensureValidBytes1( ) →fread ( )
Size(bytes)	150000
資料內容	將檔案資料讀入 Input Buffer 內
3 Function	**parse( )**
處理事項/狀態	Parsestate=PARSING_VISUAL_OBJECT_SEQUENCE fCuroffset=17
Size(bytes)	5
資料內容	0 0 1 B0 F5
4 Function	**packFrame( )→afterGettingFrame1( )**
處理事項/狀態	Insert timestamp into the RTP packet
Size(bytes)	4
資料內容	Timestamp=1704743563

續 Table5.1

5 Function	**packFrame( )→parse ( )**
處理事項/狀態	parsestate=PARSING_VISUAL_OBJECT fCuroffset=26
Size(bytes)	9
資料內容	0 0 1 B5 9 0 0 1 0
6 Function	**packFrame( )→afterGettingFrame1( )**
處理事項/狀態	Insert timestamp into the RTP packet
Size(bytes)	4
資料內容	Timestamp=1704743563
7 Function	**packFrame( )→parse( )**
處理事項/狀態	parsestate=PARSING_VIDEO_OBJECT_LAYER fCuroffset=73
Size(bytes)	47
資料內容	0 0 1 20 8 C4 9D C0 0 43 A9 C0 9 50 0 B0 D4 97 53 C 1F 4C 2C 10 78 71 F 0 0 1 B2 65 6D 34 76 20 34 2E 33 2E 32 2E 38 0 C9 FF 0
8 Function	**packFrame( )→afterGettingFrame1( )**
處理事項/狀態	Insert timestamp into the RTP packet
Size(bytes)	4
資料內容	Timestamp=1704743563
9 Function	**packFrame( )→parse( )**
處理事項/狀態	parsestate=PARSING_VIDEO_OBJECT_PLANE fCuroffset=10479
Size(bytes)	10406
資料內容	0 0 1 B6 11 8A 78 D E0 8E AC 7A B0 92 AA FD 27 9B 6D 48 80 A9 AC FF 74 78 C9 7B 45 C9 FE 20 24 1F FC .....
10 Function	**packFrame( )→afterGettingFrame1( )**
處理事項/狀態	Insert timestamp into the RTP packet
Size(bytes)	4
資料內容	**Timestamp=1704746566 (Hex=659C5E46)**

在 table5.1 中的“Size”欄位表示每次新增至緩衝區中的資量量也是 parse ( )在每一階段中處理的資料量，而 fCuroffset 表示為 frame 存入

Output Buffer 後的偏移量，且在 table5.1 中可以清楚看出:

- 伺服端一開始先在 Output Buffer 中先放入了 12 bytes 的 RTP header，其中除了 Timestamp 欄位尚未有正確的數值外，其餘欄位都已經填入。
- 接下來進入裝載 payload data 階段，伺服端會利用 parse ( )將 frame 放入 Output Buffer 中，在 table5.1 的 3、5、7 中，因為 frame 中的位元值經過 4、6 及 8 中的查驗並沒有 VOP 的 header 存在，因此會繼續利用 parse( )產生新的 frame，而 timestamp 的值也因為尚未解析至 VOP 的部份，所以一直維持初始值的狀態。
- 直到 9，parse( )將第一張 VOP 放入 Output Buffer 中，因此到 10 時會有新的 Timestamp 產生並會更新 RTP header 中 Timestamp 的欄位。而在這階段中由於 fCuroffset 為 10479，表示在緩衝區中存有 10479 筆的 data，因此在此階段會進行 frame 的切割與封包的傳送，在下個小節中將進行說明。

5.2.2 Packetizing Results

在 table5.2 中，藍色部份表示伺服端以 parse ( )產生新 frame 放到輸出緩衝區中的數量及放置的起始位址，在“Packet Size”欄位的資料大小有包含 12 bytes 的 RTP header，“Overflow Bytes”欄位表示 frame 被切割後剩下的長度，“Packet Address Range”欄位表示封包被傳送的起始和結束位址，以及伺服端所設置的下個要被傳送的封包的起始位址。

Table 5.2 Packetizing State

N0.	Packet Size	Timestamp	Overflow Bytes	Packet Address Range(HEX)		
				Start	End	Next Start
	10479	1704746566	10479	16C8028		
1	1448	1704746566	9031	16C8028	16C85D0	16C85C4
2	1448	1704746566	7595	16C85C4	16C8B6C	16C8B60
3	1448	1704746566	6159	16C8B60	16C9108	16C90FC
4	1448	1704746566	4723	16C90FC	16C96A4	16C9698
5	1448	1704746566	3287	16C9698	16C9C40	16C9C34
6	1448	1704746566	1851	16C9C34	16CA1DC	16CA1D0
7	1448	1704746566	415	16CA1D0	16CA778	16CA76C
8	427	1704746566	0	16CA76C	16CA917	16C8028
	1480	1704749569	1480	16C8028		
9	1448	1704749569	44	16C8028	16C85D0	16C85C4
10	56	1704749569	0	16C85C4	16C85FC	16C8028
	189	1704752572	189	16C8028		
11	201	1704752572	0	16C8028	16C80F1	16C8028
	2743	1704755575	2743			
12	1448	1704755575	1307	16C8028	16C85D0	16C85C4
13	1319	1704755575	0	16C85C4	16C8AEB	16C8028
	2037	1704758578		16C8028		
14	2037	1704758578	601	16C8028	16C85D0	16C85C4
15	613	1704758578	0	16C85C4	16C8829	16C8028
	1453	1704761581	1453	16C8028		
16	1448	1704761581	17	16C8028	16C85D0	16C85C4
17	29	1704761581	0	16C85C4	16C85E1	16C8028
	544	1704764584		16C8028		
18	556	1704764584	0	16C8028	16C8254	16C8028
	5513	1704767587	5513	16C8028		
19	1448	1704767587	4077	16C8028	16C85D0	16C85C4
20	1448	1704767587	2641	16C85C4	16C8B6C	16C8B60
21	1448	1704767587	1205	16C8B60	16C9108	16C90FC
22	1217	1704767587	0	16C90FC	16C95BD	16C8028
	684	1704770590	684	16C8028		
23	696	1704770590	0	16C8028	16C82E0	16C8028
	3288	1704773593	3288	16C8028		

續 Table5.2

N0.	Packet Size	Timestamp	Overflow Bytes	Packet Address Range(HEX)		
				Start	End	Next Start
24	1448	1704773593	1852	16C8028	16C85D0	16C85C4
25	1448	1704773593	416	16C85C4	16C8B6C	16C8B60
26	428	1704773593	0	16C8B60	16C8D0C	16C8028
	10094	1704776596	...	16C8028		
27	1448	1704776596	8658	16C8028	16C85D0	16C85C4
28	1448	1704776596	7222	16C85C4	16C8B6C	16C8B60
29	1448	1704776596	5786	16C8B60	16C9108	16C90FC
30	1448	1704776596	4350	16C90FC	16C96A4	16C9698
31	1448	1704776596	2914	16C9698	16C9C40	16C9C34
32	1448	1704776596	1478	16C9C34	16CA1DC	16CA1D0
33	1448	1704776596	42	16CA1D0	16CA778	16CA76C
34	54	1704776596	0	16CA76C	16CA917	16C8028
	...	...	...	16C8028		

由於「封包包裝與傳送」模組是先建置 12 bytes 的 RTP header，在呼叫 parse ( )產生 payload data 放在 RTP header 之後，因此緩衝區中的 frame 長度都有包含 RTP header。當 frame 的長度大於 fMax(等於 1448 bytes)也就是封包大小預設的最大值時，就會對 frame 進行切割，從 table5.2 中的數據可以看出，第一份 frame 的大小遠大於 fMax，因此被切割成八個封包被傳送出，每當送出一個封包，伺服端便會將指向封包起始位址的指標(fPacketStart)往前移 12bytes，以便插入 RTP header，以第一個封包來說，其資料範圍為起始位址 0x16C8028 至結束位址 0x16C85D0，0x16C85D0 也為剩餘的 frame 的起始位址，此時伺服端會將 fPacketStart 值設為 0x16C85D0 減去 12 的大小也就是 0x16C85C4，這樣在下一個封包建立時，RTP header 便會從 0x16C85C4 的地方開始置入，figure5.10 是 fPacketStart 在輸出緩衝區中從第一至第十個封包的變化情形。當一張完整的 VOP 被傳送完畢後，會再度啟

動 parse ( )產生新的 frame，此時 fPacketStart 的位址會指向輸出緩衝區的起始位址。

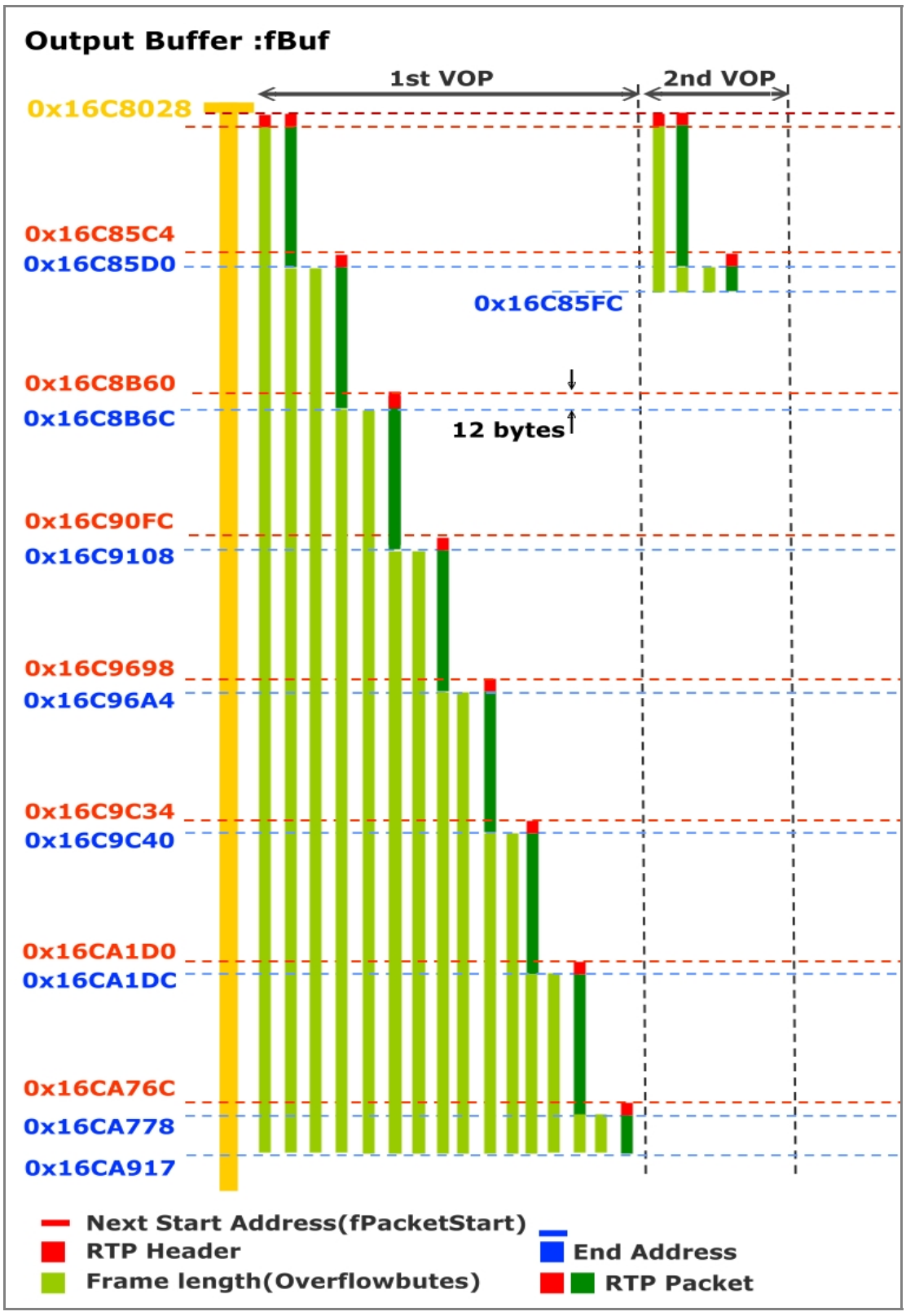

Figure5.12 Output Buffer State

第六章 結論

檔案包裝格式的拆解、資料的解析以及網路傳輸是多媒體串流伺服器內部運作最主要的核心，也是本文中所論述的「封包包裝與傳送」模組，一個理想的多媒體串流伺服器必須要能夠精確的判斷檔案的格式，並依據不同類型的包裝格式拆解出其視訊流與音訊流，進而包裝成符合 RTP payload 格式的封包，從本文的討論中可以看出一個多媒體串流伺服器的實作包含了許多領域的知識，像是網路程式設計、通訊協定以及各種影音的壓縮解碼知識，而將這些知識結合起來後還必須考慮實際的應用環境，在有限的資源下使得串流系統發揮最大的效能比。

Parse 是一項階層性的工作，每個階層彼此互相具有關聯性，從連線開始後對檔案文件格式拆解、編碼格式的辨認、parser 的選擇…等，整個流程是一個龐大的 state machine，從對一未知型態的檔案進行處理到解析及分割成一個個封包傳送，須要複雜的實施方式來實現每一階層的連慣性，因此 link-list 組態的形成對 parse 的 process 非常的重要。傳送媒介的不同會影響 packet 的大小，因此不同的傳送機制也會影響 parse 機制的運作，而產生不同的程序問題。

在未來，希望能夠藉著對於「封包包裝與傳送」模組基本架構的了解，進而開發出可重覆使用的軟體模組，使其能夠擴充及包容更多不同檔案類型，達到整合的目地，並更進一步的往串流程式軟硬體間的協同設計方向實現。

第七章 參考文獻

[1] H.Schulzrinne, S.Casner, R.Frederick, and V.Jacobson, “RTP:A Transport Protocol for Real-Time Applications”, Audio Visual Working Group Request for Comment RFC 3550, IETF, July 2003

[2] 吳宗修，「串流伺服器特性剖析」，國立交通大學，碩士論文，民國 95 年。

[3] D. Hoffman,G. Fernando,"RTP Format for MPEG1/MPEG2 Video" ,RFC 2250,January 1998 3.

[4] Y. Kikuchi(Toshiba),T. Nomura(NEC),S. Fukunaga(Oki),Y. Matsui (Matsushita),H. Kimata(NTT), “Payload Format for MPEG-4 Audio/Visual Streams“,RFC 3016,November 2000

[5] H. Schulzrinne, A. Rao, and R. Lanphier, "Real Time Streaming Protocol (RTSP)", RFC 2326 ,April 1998.

[6] M. Handley, V. Jacobson, “SDP: Session Description Protocol”,RFC2327, April 1998

[7] ISO/IEC 13818-1 Generic coding of moving picture and associated audio information: system

[8] ISO/IEC 14496-2 Information technology-coding of audio-visual object-Part2:Visual

[9] ISO/IEC 13818-2 Generic coding of moving picture and associated audio information: Video

[10] Live555 http://www.live555.com/liveMedia

[11] 張為棟，『整合式多媒體串流平台的發展與實作』，國立交通大學，碩士論文，95 學年度。

[12]http://blog.csdn.net/lengxingfei/archive/2005/08/24/463553.aspx

Appendix A　RTP Payload Type List Table

以下將列出 RTP header 中 PT 欄位位元值的對應編碼格式，以及相關的 RFC 文件。

Table A-1[12] Payload Type Table

PT	Name	Type	Clock rate (Hz)	Audio channels	References
0	PCMU	Audio	8000	1	RFC 3551
1	1016	Audio	8000	1	RFC 3551
2	G721	Audio	8000	1	RFC 3551
3	GSM	Audio	8000	1	RFC 3551
4	G723	Audio	8000	1	
5	DVI4	Audio	8000	1	RFC 3551
6	DVI4	Audio	16000	1	RFC 3551
7	LPC	Audio	8000	1	RFC 3551
8	PCMA	Audio	8000	1	RFC 3551
9	G722	Audio	8000	1	RFC 3551
10	L16	Audio	44100	2	RFC 3551
11	L16	Audio	44100	1	RFC 3551
12	QCELP	Audio	8000	1	
13	CN	Audio	8000	1	RFC 3389
14	MPA	Audio	90000		RFC 2250, RFC 3551
15	G728	Audio	8000	1	RFC 3551
16	DVI4	Audio	11025	1	
17	DVI4	Audio	22050	1	
18	G729	Audio	8000	1	
19	reserved	Audio			
25	CellB	Video	90000		RFC 2029
26	JPEG	Video	90000		RFC 2435
27					
28	nv	Video	90000		RFC 3551
31	H261	Video	90000		RFC 2032
32	MPV	Video	90000		RFC 2250

續 Table A-1

PT	Name	Type	Clock rate (Hz)	Audio channels	References
33	MP2T	Audio/ Video	90000		RFC 2250
34	H263	Video	90000		
72 - 76	reserved				RFC 3550
96 - 127	dynamic				RFC 3551
dynamic	GSM-HR	Audio	8000	1	
dynamic	GSM-EFR	Audio	8000	1	
dynamic	L8	Audio	variable	variable	
dynamic	RED	Audio			
dynamic	VDVI	Audio	variable	1	
dynamic	BT656	Video	90000		
dynamic	H263-1998	Video	90000		
dynamic	MP1S	Video	90000		
dynamic	MP2P	Video	90000		
dynamic	BMPEG	Video	90000		

- RFC 文件

[RFC 2029] RTP Payload Format of Sun's CellB Video Encoding.
[RFC 2032] RTP Payload Format for H.261 Video Streams.
[RFC 2190] RTP Payload Format for H.263 Video Streams.
[RFC 2198] RTP Payload for Redundant Audio Data.
[RFC 2250] RTP Payload Format for MPEG1/MPEG2 Video.
[RFC 2343] RTP Payload Format for Bundled MPEG.
[RFC 2429] RTP Payload Format for the 1998 Version of ITU-T Rec. H.263 Video (H.263+).
[RFC 2431] RTP Payload Format for BT.656 Video Encoding.
[RFC 2435] RTP Payload Format for JPEG-compressed Video.

[RFC 2508] Compressing IP/UDP/RTP Headers for Low-Speed Serial Links.
[RFC 2658] RTP Payload Format for PureVoice(tm) Audio.
[RFC 2733] An RTP Payload Format for Generic Forward Error Correction.
[RFC 2736] Guidelines for Writers of RTP Payload Format Specifications.
[RFC 2762] Sampling of the Group Membership in RTP.
[RFC 2793] RTP Payload for Text Conversation.
[RFC 2833] RTP Payload for DTMF Digits, Telephony Tones and Telephony Signals.
[RFC 2862] RTP Payload Format for Real-Time Pointers.
[RFC 2959] Real-Time Transport Protocol Management Information Base.
[RFC 3009] Registration of parityfec MIME types.
[RFC 3016] RTP Payload Format for MPEG-4 Audio/Visual Streams.
[RFC 3047] RTP Payload Format for ITU-T Recommendation G.722.1.
[RFC 3095] RObust Header Compression (ROHC): Framework and four profiles: RTP, UDP, ESP, and uncompressed.
[RFC 3119] A More Loss-Tolerant RTP Payload Format for MP3 Audio.
[RFC 3158] RTP Testing Strategies.
[RFC 3189] RTP Payload Format for DV (IEC 61834) Video.
[RFC 3190] RTP Payload Format for 12-bit DAT Audio and 20- and 24-bit Linear Sampled Audio.
[RFC 3267] Real-Time Transport Protocol (RTP) Payload Format and File Storage Format for the Adaptive Multi-Rate (AMR) and Adaptive Multi-Rate Wideband (AMR-WB) Audio Codecs.
[RFC 3389] Real-time Transport Protocol (RTP) Payload for Comfort Noise (CN).
[RFC 3497] RTP Payload Format for Society of Motion Picture and Television Engineers (SMPTE) 292M Video.
[RFC 3558] RTP Payload Format for Enhanced Variable Rate Codecs (EVRC) and Selectable Mode Vocoders (SMV).
[RFC 3545] Enhanced Compressed RTP (CRTP) for Links with High Delay, Packet Loss and Reordering.
[RFC 3550] RTP: A Transport Protocol for Real-Time Applications.

[RFC 3551] RTP Profile for Audio and Video Conferences with Minimal Control.
[RFC 3555] MIME Type Registration of RTP Payload Formats.
[RFC 3557] RTP Payload Format for European Telecommunications Standards Institute (ETSI) European Standard ES 201 108 Distributed Speech Recognition Encoding.
[RFC 3640] RTP Payload Format for Transport of MPEG-4 Elementary Streams.
[RFC 3711] The Secure Real-time Transport Protocol (SRTP).
[RFC 3816] Definitions of Managed Objects for RObust Header Compression (ROHC).