

國立交通大學

電機學院 IC 設計產業研發碩士班

碩士論文

串流伺服器特性剖析

Analysis of Streaming Server's Properties



研究生：吳宗修

指導教授：張文鐘 教授

中華民國九十六年一月

串流伺服器特性剖析

Analysis of Streaming Server's Properties

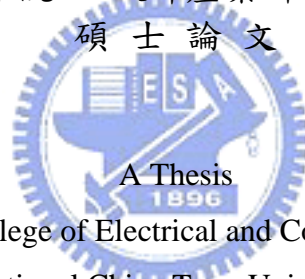
研究生：吳宗修

Student : Jong-Shou Wu

指導教授：張文鐘

Advisor : Wen-Thong Chang

國立交通大學
電機學院 IC 設計產業研發碩士班
碩士論文



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
IC Design

January 2007

Hsinchu, Taiwan, Republic of China

中華民國九十六年一月

串流伺服器特性剖析

摘要

一個串流伺服器主要由三個重要的模組所組成，這三個模組分別是『標準連線程序建立』、『RTSP 信令溝通協調』以及『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 信令溝通協調』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，此部分的重點在於設定一個狀態機的機制來進行 RTSP 的信令。在『封包包裝和傳送』模組中，多媒體來源檔案的取得以及切割是經由封裝演算法來執行之，而一個結構化封包的產生即是經由某一種特定的封裝演算法而得到的，舉例來說，Live555 串流伺服器所使用的 M4V 封裝演算法即是採用 RFC 3016 標準所建議的方式 F。

在多工方面，Live555 基於單一行程以及事件觸發的機制來達到串流伺服器的多工，此種多工型態所衍生的程式執行流程以及效能問題也是我們在本論文中所必須論述到的重點。

Analysis of Streaming Server's Properties

Abstract

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.

" Set up a standard connection procedure "module uses the BSD Socket API to establish socket for the server, it can set up a TCP or UDP socket in accordance with the demand for our streaming program. In " RTSP Signaling Negotiation " module , it allows the Streaming Server to negotiate with Client via RTSP Signaling, the key issue is to set up a state machine for RTSP Signaling. In " packet Packetization and Transmission " module, multimedia source is retrieved and packetized through Packetization Algorithm , and we can generate a standard structurization packet with a specific PA, for example, the PA of MPEG-4 Video used in Live555's Streaming Server is based on the TYPE(F) of RFC 3016.

For multiple clients connection at the same time , Live555 is based on Single-Process and Event-Triggered mechanism to reach the Streaming Server's I/O multiplexing , and the

relevant issues caused by specific multiplexing classification such as program flow path and performance are also significantly that we have to expound in this thesis.



誌謝

能順利完成這篇論文，最感謝的人是我的指導教授 張文鐘 博士。在這二年的碩士生涯中，老師平易近人的態度是我前所未見的，特別是趕論文的那段時間，真的是難為老師如此辛苦地花時間幫我找出問題的關鍵點，在此真的要再次謝謝老師的指導。同時也謝謝 余孝先副所長、范國清教授以及黃仲陵教授於口試時的指導，有了您的指導才使得這篇論文更趨完備。

另外要感謝的還有幾個實驗室裡的好朋友，包含小邱、瑩甄，特別感謝素仙在最後趕論文的那段時間從早到晚幫我作程式流程的追蹤。跟你們在這兩年一起修課、打球的時光是我最美好的回憶。

最後，我要感謝的是我的父母親，有了你們的支持以及鼓勵我才能順利地完成碩士學業，除了感謝還是感謝。



誌於2007.1 風城 交大

Ashou

目錄

中文摘要	i
英文摘要	ii
誌謝	iv
目錄	v
圖目錄	viii
表目錄	xi
一、	緒論.....	1
1.1	研究背景與動機.....	1
1.2	論文架構.....	3
二、	連線建立與 RTSP 信令溝通.....	4
2.1	Live555 串流伺服器組成.....	4
2.2	連線建立的標準程序.....	6
2.2.1	Socket.....	9
2.2.2	Bind.....	11
2.2.3	Listen.....	12
2.2.4	Accept.....	13
2.2.5	Connect.....	15
2.2.6	Recv/Recvfrom.....	16
2.2.7	Send/Sendto.....	17
2.2.8	Close.....	18
2.3	串流伺服器的溝通協調.....	18
2.3.1	Live555 RTSPClientSession.....	18
2.3.2	Live555 RTSP 連線建立流程.....	21
2.3.3	RTSP DESCRIBE.....	23
2.3.4	SDP.....	24
2.4	Server 與 Client 互動之流程.....	30
2.4.1	RTSPClient 動作流程.....	30
2.4.1.1	openConnectionFromURL(url).....	31
2.4.1.2	sendRequest.....	32
2.4.2	Live555 RTSP 動作流程.....	33
2.4.2.1	RTSP OPTIONS.....	34
2.4.2.2	RTSP DESCRIBE.....	36
2.4.2.3	RTSP SETUP.....	37
2.4.2.4	RTSP PLAY.....	39
2.4.2.5	RTSP PAUSE.....	40
2.4.2.6	RTSP TEARDOWN.....	41

2.4.3	client getResponse.....	43
三、	封包包裝及傳送.....	47
3.1	RTSP PLAY 信令的執行(handleCmd_PLAY).....	47
3.1.1	伺服器媒體連線會談 (ServerMediaSession)	50
3.1.2	伺服器的媒體底層連線會談 (ServerMediaSubSession)	51
3.2	訊框的切割、包裝以及傳送.....	56
3.2.1	IP、Fragment and RTCP.....	57
3.2.2	建立影像來源(createNewStreamSource)	62
3.2.2.1	開啟檔案並讀取檔案內容 (ByteStreamFileSource.createNew).....	62
3.2.2.2	位元流解析器/訊框產生器 (MPEG4VideoStreamFramer.createNew).....	64
3.2.3	影像訊框的組成(createNewRTPSink).....	65
3.2.4	影像封包的切割與包裝(MultiFramedRTPSink).....	66
3.2.5	RTP Payload format for MPEG4-RFC3610 (createNewRTPSink).....	71
3.2.5.1	RTP.....	71
3.2.5.2	Live555 RTSPServer payload format for MPEG4 (MPEG4ESVideoRTPSink.cpp).....	73
3.2.6	取得各種影音格式的訊框(doGetNextFrame).....	74
四、	串流伺服器實作細節.....	78
4.1	串流伺服器的多工.....	78
4.1.1	Multi-Process， Concurrent Servers.....	78
4.1.1.1	Fork()說明.....	79
4.1.1.2	Fork()虛擬碼範例.....	80
4.1.2	Single-Process (Event-Based)， Concurrent Servers....	80
4.1.2.1	Select()說明.....	82
4.1.2.2	Select()虛擬碼範例.....	84
4.1.3	多工串流方式的效能比較.....	85
4.1.3.1	Socket API 的作業模式比較.....	85
4.1.3.2	fork 與 select 分析比較.....	86
4.1.4	串流伺服器多工結論.....	87

4.2	RTSP 訊令之傳輸以及辨認.....	88
4.3	RTSP 進入 RTP 傳送之機制.....	92
4.4	檔案資料內容之解讀及訊框之取出.....	94
4.4.1	開檔與讀檔.....	95
	建立以 MPEG4 為主的影像流訊框產生器	
4.4.2	(MPEG4VideoStreamFrame).....	97
4.4.2.1	MPEG4 概要.....	97
4.4.2.2	MPEG-4 start codes.....	98
4.4.2.3	MPEG4 video stream parsing flow.....	100
4.5	建立以及傳送 RTP 封包.....	107
4.5.1	RFC3016 封裝演算法分析.....	107
4.5.2	Live555 封裝演算法分析.....	110
五、	串流程式開發手扎.....	115
六、	結論.....	117
參考文獻	118
附錄一	How to configure and build the code on Windows/Linux.....	119
附錄二	Overview of Live555 Server's Module、Architecture and Performance	122
自傳	124

圖目錄

圖 2.1(a) Live555 Streaming Library.....	5
圖 2.1(b) Live555 Streaming Server.....	5
圖 2.2 Overview of Streaming Server.....	5
圖 2.3 TCP socket API flowchart.....	7
圖 2.4 UDP socket API flowchart.....	8
圖 2.5 Basic Streaming Server Protocol Stack.....	8
圖 2.6 Socket Data Structure.....	9
圖 2.7 Socket Descriptor Table.....	9
圖 2.8 Socket Queue.....	12
圖 2.9(a) Listening/Connected Socket..	13
圖 2.9(b) Streaming Server flowchart.....	14
圖 2.9(c) Basic Server flowchart.....	14
圖 2.10 Capturing Streaming Packet.....	15
圖 2.11 Live555 support RTSP.....	21
圖 2.12 RTSP negotiation.....	22
圖 2.13 RTSP DESCRIBE.....	23
圖 2.14 SDP content.....	24
圖 2.15 RTSP Unauthorized.....	27
圖 2.16 server RTSP OPTIONS response.....	35
圖 2.17 client RTSP DESCRIBE request.....	36
圖 2.18 server RTSP DESCRIBE response.....	37
圖 2.19 client RTSP SETUP request.....	38
圖 2.20 server RTSP SETUP response.....	38
圖 2.21 client RTSP PLAY request.....	39

圖 2.22 server RTSP PLAY response.....	40
圖 2.23 client RTSP PAUSE request.....	40
圖 2.24 server RTSP PAUSE response.....	41
圖 2.25 client RTSP TEARDOWN request.....	41
圖 2.26 server RTSP TEARDOWN response.....	42
圖 3.1 Create RTP/RTCP socket after RTSP PLAY.....	49
圖 3.2(a) OSI 7-layer model.....	57
圖 3.2(b) IP fragmentation.....	58
圖 3.3 A RTP/RTSP packet.....	59
圖 3.4 RTCP RR.....	61
圖 3.5 IP packet consistence.....	66
圖 3.6 RTP header.....	72
圖 4.1 Multi-Process, Concurrent Servers.....	79
圖 4.2(a) Single-Process (Event-Based) , Concurrent Servers.....	81
圖 4.2(b) Single-Process with select flowchart.....	82
圖 4.3 RTSP Parser chart.....	88
圖 4.4 Results RTSP command name.....	91
圖 4.5 RTSP State Machine.....	92
圖 4.6 『PLAY』 - Send RTP packet.....	93
圖 4.7(a) fopen/fread.....	94
圖 4.7(b) MPEG4VideoStreamFramer function-call	96
圖 4.8 MPEG4 video bitstream logical structure	98
圖 4.9(a) m4v bitstream example.....	102
圖 4.9(b) Live555 visual bitstream start-code.....	102
圖 4.10(a) VOP header and I/P/B frame.....	105

圖 4.10(b) mpeg-4 video stream example.....105

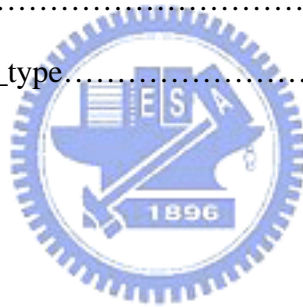
圖 4.11 mpeg-4 video stream in RTP payload108

圖 4.12 Examples of RTP packetized MPEG-4 Visual bitstream.....110



表目錄

表 2.1 payload type table	25
表 2.2 RTSP Method table.....	33
表 2.3 RTSP status code table.....	45
表 3.1 MTU limited table.....	58
表 4.1 Socket API 的作業模式比較.....	86
表 4.2 fork 與 select 分析比較表.....	86
表 4.3 Implemented RTSP method of Live555	93
表 4.4 MPEG-4 start codes.....	100
表4.5 Meaning of visual_object_verid.....	103
表 4.6 visual object type.....	103
表 4.7 Meaning of vop_coding_type.....	104



第一章 緒論

1.1 研究背景與動機

一個串流伺服器的組成即代表著必須克服我們在摘要中所提到的三個問題，如果無法克服的話表示我們將不會有一個快速、穩定、效率高的串流伺服器。因此，一個串流伺服器的實作上就是要設計一些機制來克服上述問題。

以第一個問題『連線建立的標準程序』來說，現今的網路程式都依循著 BSD(Berkeley Software Distribution) socket 的實現標準，所以我們只要按照 BSD socket API 的建立程序便可以建構出一個標準的串流連線架構，但光靠 BSD socket API 仍無法滿足串流媒體建構所需(比如何服器的多工、RTSP Signaling)，因此衍生出的配套措施便是我們要去研究的主題之一。



第二個問題是串流伺服器要透過何種方式來服務用戶端的『播放需求』，而這個問題 IETF(Internet Engineering Task Force)也特別制訂了一個標準來因應，此標準為 RTSP (Real Time Streaming Protocol，RFC2326)[1]。該標準主要以一些 Text-Based (也就是 SDP，Session Description Protocol，會談描述協定，RFC 2327[2]) 文字為主的協定來傳遞伺服端與用戶端間的『溝通協調』需求。

最後一個是『封包包裝及傳送』的問題，當串流將媒體檔案分成許多小封包在網路上進行傳輸時會碰到各種不可預期的情形，所以我們必須有一個封包包裝以及傳送的標準來因應。在封包傳送的標準上 IETF 制訂了 RTP(Real-time Transport Protocol,RFC1889,

現已更新為 RFC3550) ;而因為各種多媒體檔案來源它們的壓縮方式都有其特別之處，因此就必須替它們量身定作不同的封包包裝方式，我們稱之為封包的封裝演算法 (Packetization Algorithm.)。

封裝演算法最主要的目的在於根據串流伺服器的網路使用環境來選擇適當的封裝演算法，藉此達到封包使用率以及封包錯誤恢復能力(error-resilience)間的最佳化關係。因此有線、無線、傳輸速率高以及傳輸速率低的網路串流環境就各有適當的封裝演算法，在本論文中我們將以 IETF 制訂的 RTP Payload Format for MPEG-4 Audio/Visual Streams (RFC 3016)標準來進行解說。而最後把上述三個問題的答案集合起來便是一個基本串流伺服器的核心組成。

目前串流媒體的三大主流分別是RealNetworks.com 的RealSystem、Microsoft 的 Media Services以及蘋果電腦的Quick Time Server，但因商用軟體競爭激烈，導致了特定的串流程式只能對應其特定的編碼/解碼格式來進行播放，這也造成了很多使用上的不便。所幸在自由軟體的發展上出現了一套重量級的應用程式，稱之為VLC (VideoLanClient media player)[3]，VLC的特點如下所示：

- 屬於自由軟體發表於 GNU 通用大眾許可證(GPL) 之下。
- 跨平台的軟體 (Windows , Linux...etc)。
- 可作為媒體播放器使用 (MPEG-1/2/4,DivX,mp3,Ogg,DVDs....etc)。
- 可作為轉碼(Transcoding)程式使用。
- 作為串流影音輸出的伺服端，例如 VOD (Video On Demand，隨選視訊)的應用。

- 可作為串流影音接收的用戶端使用，並且因為內建相當多的影音解碼(decoder)程式，故大部分的影音格式它都可以解，例如時下最流行的 DivX 亦可。
- 可支援各種不同的串流型式來傳輸，例如網路上的點播(unicast)、群播(multicast)、http 等。

上述特點幾乎具備了所有串流程式應有的功能，最重要的是在它的核心程式模組中關於串流傳輸協定的部分是直接套用了 Live555.com[4]中的『liveMedia.hh』library，『liveMedia.hh』裡面主要實作了關於串流傳輸協定 RTSP/RTP/RTCP 的部分，因此我們便從而由 Live555 這個 open source 開始研讀它所提供的程式碼，藉以瞭解串流伺服器的核心組成為何。



1.2 論文架構

瞭解上述三個問題點之後，我們便依序在論文中針對這些問題點進行剖析，首先在第二章介紹串流伺服器的連線建立與 RTSP 信令溝通，內容包含 Live555 資源的簡介、BSD socket、RTSP 以及 SDP。第三章是封包包裝及傳送，內容包含 MPEG4 elementary stream 的解析以及 RTP 封包的包裝。第四章是串流伺服器的設計討論，內容主要是針對串流伺服器實作上的一些關鍵點深入探討，比如說串流伺服器的多工、RTSP 信令的辨識以及媒體檔案訊框的取得。第五章則是串流程式開發手扎，內容主要是對於 Live 的研究心得。最後第六章是結論。

第二章 連線建立與 RTSP 信令溝通

在瞭解基本串流伺服器的組成後，我們接下來開始論述 Live555 串流伺服器是如何克服上述三個問題以及它的實作方式，在開始之前，我們必須先對 Live 所提供的資源做一個簡要的介紹。

2.1 Live555 串流伺服器組成

Live555 是一套提供 GPL 開放原始碼 (open source) 多媒體串流的 C++ 函數庫，它提供的函數庫可作為影音串流的共通平台~此共通平台定義同樣的信令協定以建立兩端的連線，定義同樣的封包傳輸格式以及同樣的編解碼方式。這個函數庫包含四個小函數庫，說明如下：



- #1 『UsageEnvironment』：一些程式的初始化功能。
- #2 『Groupsock』：允許呼叫 BSD Socket API。
- #3 『Medium』：主要提供了 RTP/ RTSP/ Container format 等功能。
- #4 『BasicUsageEnvironment』：提供整合的初始化程式。

透過前三個 Libraries 的互動，我們便能夠建構出一個串流伺服器的基本功能了（如圖 2.1(a) 所示）。此外，如果串流伺服器想提供影像即時擷取傳送功能的話 (Live Capture)，我們會在程式中外掛一個 Encoder（如圖 2.1(b) 所示），目的是把影像擷取裝置

所擷取到的raw data再經編碼壓縮處理，藉此得到更低的網路資料傳輸量。

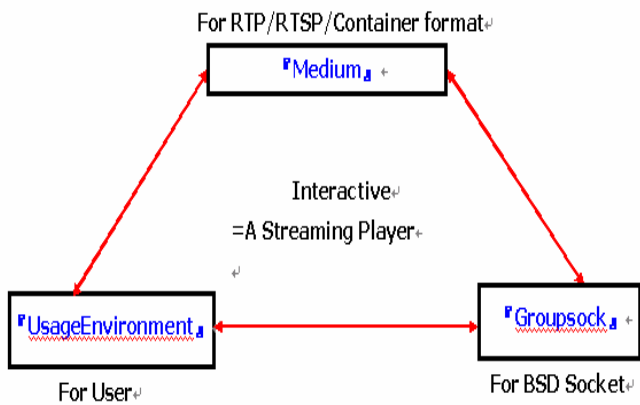


圖2.1(a) Live555 Streaming Library

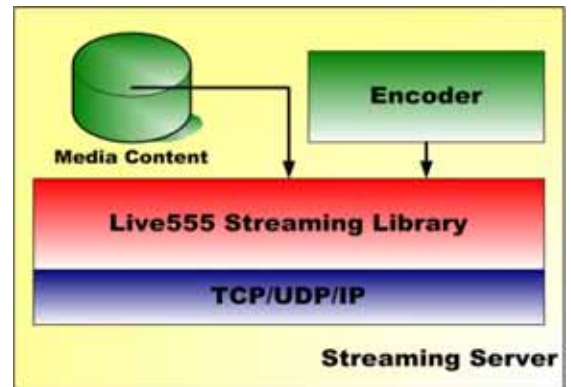


圖2.1(b) Live555 Streaming Server

下圖 2.2 所示為一個串流伺服器與用戶端的示意圖，我們可以看到不僅是個人電腦，在一些嵌入式系統中也可以結合串流伺服器成為一個很好的應用(比如說監控系統)。因此，討論串流伺服器的核心組成便是一個很好的研究主題，唯有瞭解其核心組成，我們才有能力再去結合更多的應用，比如在影像處理方面，我們可以結合人臉辨識成為一套很好的防盜監控系統，或是水庫水位自動監測警報系統等等。

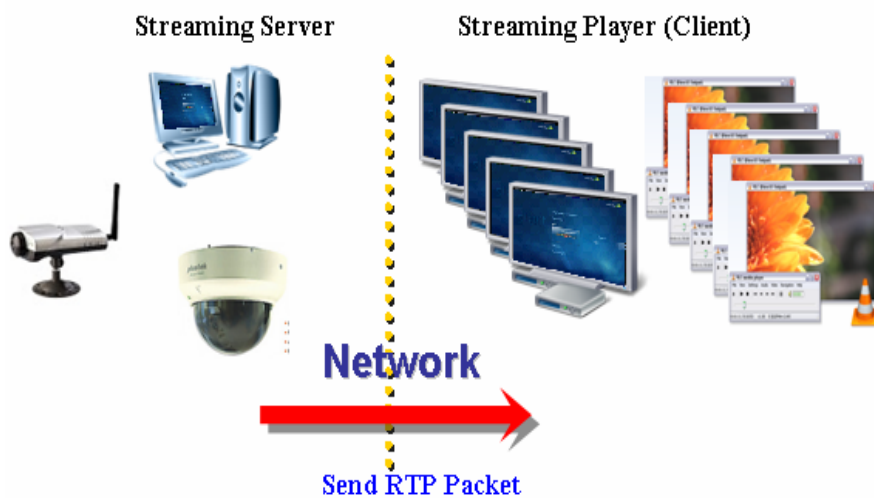


圖 2.2 Overview of Streaming Server

2.2 連線建立的標準程序

接下來我們開始由 Live555 的串流伺服器程式開始追蹤，藉以瞭解其核心組成為何，程式碼請自行參考 testOnDemandRTSPServer.cpp，首先我們看到的程式碼如#1 以及 #2 所示，其功能敘述於程式碼下方。

```
#1.TaskScheduler* scheduler = BasicTaskScheduler::createNew();  
  
#2.env = BasicUsageEnvironment::createNew(*scheduler);
```

#1- BasicTaskScheduler 建構子中最重要的功能便是完成串流伺服器的 Socket 多工配置，此部分我們會在第四章中詳述之。

#2- BasicUsageEnvironment 建構子主要是執行 WSASStartup() 函數，因為只要是在微軟作業系統下所執行的網路程式都必須呼叫該函數來完成初始化的動作。



而因為上述 BasicTaskScheduler 以及 BasicUsageEnvironment 類別(class)中尚實作了很多函數我們無法一一列舉說明，故在此僅將#1 以及#2 所共同完成的功能列舉如下：

1. Scheduling deferred events(socket 資料的接收與傳送)。
2. For assigning handlers for asynchronous read events
(針對讀取到的事件指派 handler 去處理之)。
3. For outputting error/warning messages(程式錯誤/警告訊息的輸出)。

接下來執行的程式如下紅框內所示，該行程式即是 Live555 Streaming Server 開始建立 socket 標準連線程序的起點，也是我們核心組成的第一個要件。

```
# 3 RTSPServer* rtpServer = RTSPServer::createNew(*env, 8554, authDB);
```

3 程式將會開始應用 BSD socket API，如下圖 2.3 所示為整個 RTSPServer.cpp 所應用到的 TCP socket API（傳送 RTSP 命令）；圖 2.4 則為 RTSPServer.cpp 所應用到的 UDP socket API（傳送 RTP 封包），而串流伺服器一開始會建立一個 TCP 模式 RTSP socket，等到用戶端 connect() 連線成功後即進入 RTSP 信令的辨識以及執行，等到收到用戶端發送 RTSP PLAY 信令後便以 UDP 模式建立一個 RTP socket 來完成封包的傳送，關於 RTSP 的部分我們在後面會解說。

(註) TCP 跟 UDP 的 socket 建立過程差別在於 UDP socket 不需要使用 listen() 以及 accept()。

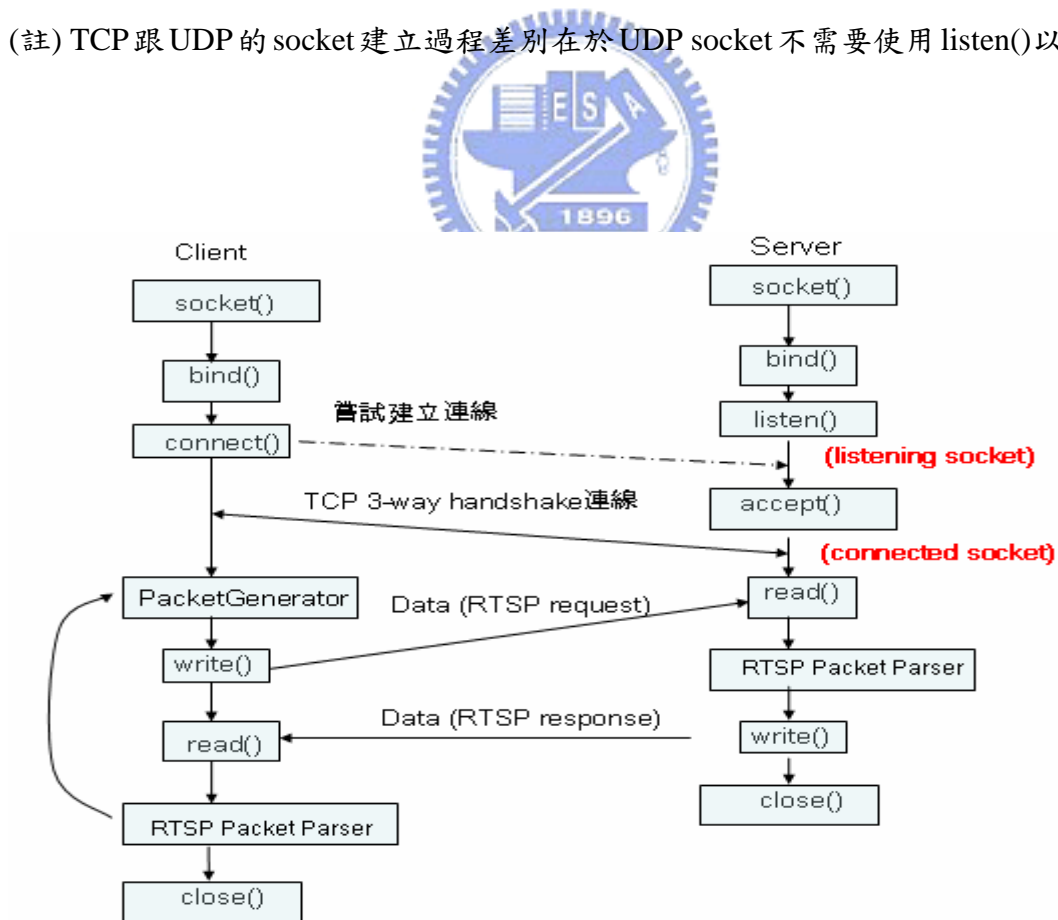


圖 2.3 TCP socket API flowchart

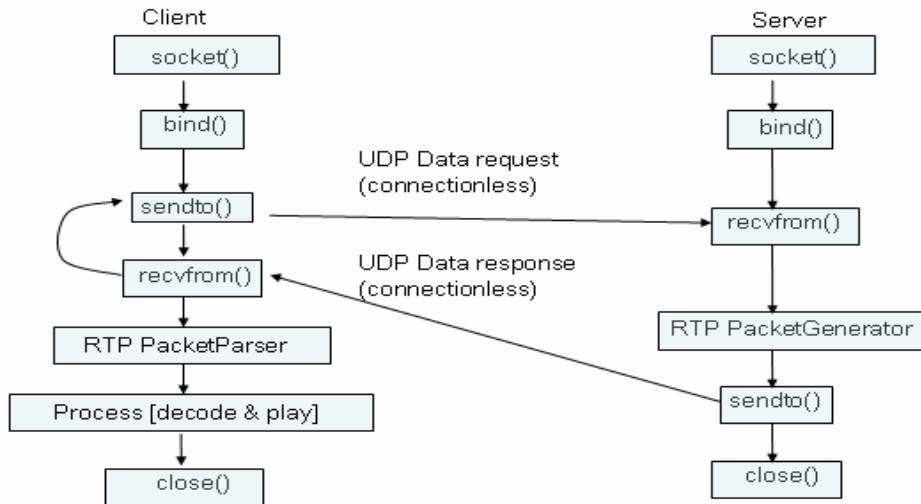


圖 2.4 UDP socket API flowchart

下圖 2.5 為 Live555 串流伺服器使用的通訊協定堆疊圖，目前上述 socket API 均已整合在作業系統內，因此我們只要透過 socket API 的 system call，OS Kernel 便會幫我們完成 TCP/UDP/IP 通訊協定的設定，簡化網路程式的設計。

(註.微軟作業系統為 #include<winsock.h>，一般的 C 則為#include<sys/socket.h>)

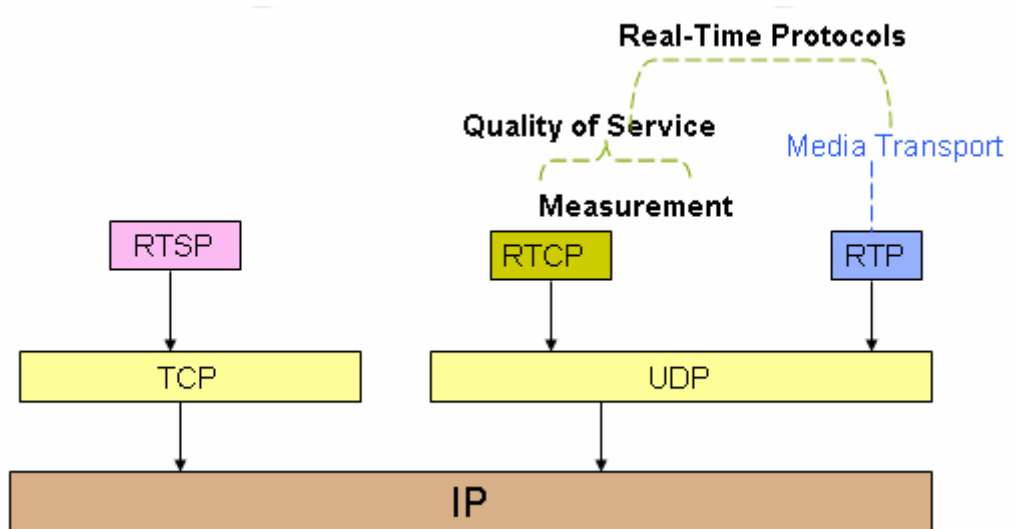


圖 2.5 Basic Streaming Server Protocol Stack

瞭解了串流伺服器的通訊協定堆疊圖後，我們接下來針對『連線建立的標準程序』所需使用的 socket API 函數以及流程作簡單的介紹。

2.2.1 Socket

要進行網路 I/O，程式中至少要有一個 process 來呼叫 socket() 函式，藉以指定所需的通訊協定類型（EX: 使用 IPv4 的 TCP、使用 IPv6 的 UDP 等），socket() 會傳回一個整數型態的 socket file descriptor，稱為承接口描述子(以下簡稱 socketfd)。其意義類似開啟檔案時所需要用到的檔案指標(file descriptor)，socketfd 目的是用來索引(index)一個『socket』的資料結構，如下圖 2.6 所示，而 server/client 要能彼此通連即代表它們 socket 的資料結構內容都必須相同。

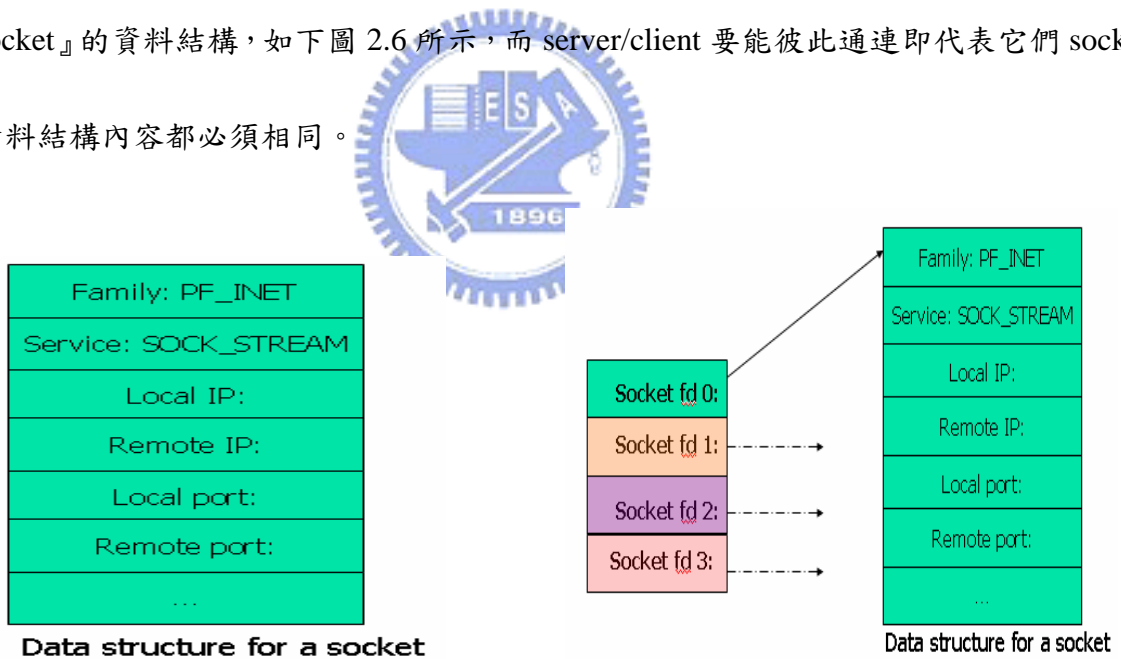


圖 2.6 Socket Data Structure

圖 2.7 Socket Descriptor Table

此外，每一個行程在 OS kernel 中都會擁有一個 socket 描述子表格 (descriptor table)，而表格中的 socketfd 就是指向(index)socket()在系統中所建立的 socket 資料結

構，如上圖 2.7 所示。

而 TCP socket 與 UDP socket 建立的差別如下所示，重點在於 socket() 中的第二個引數，如果要建立 TCP socket 的話第二個引數要填入 SOCK_STREAM; 要建立 UDP socket 的話第二個引數要填入 SOCK_DGRAM，以下摘錄 RTSPServer.cpp 中分別呼叫 setupStreamSocket() 以及 setupDatagramSocket() 函數的程式碼。

(範例以及函數說明)

```
*****/
/* TCP Socket()*/
int newSocket = socket(AF_INET, SOCK_STREAM, 0);
if (newSocket < 0) {
    socketErr(env, "unable to create stream socket: ");
    return newSocket;
} //end if

*****/
/* UDP Socket()*/
//SOCK_DGRAM表示傳輸模式是 使用UDP
int newSocket = socket(AF_INET, SOCK_DGRAM, 0);
if (newSocket < 0) {
    socketErr(env, "unable to create datagram socket: ");
    return newSocket;
}
```

```
SOCKET PASCAL FAR socket (int af , /*protocol suite*/
int type , /*protocol type */
int protocol ) ; /*protocol name*/
```

伺服器端的程式在呼叫 socket() 完成一個 socket 的資料結構之後，必須為這個 socket 的資料結構『命名』，也就是設定三個參數，分別是連線協定、連結埠號碼(port number) 以及位址(IP address)，而用戶端便是以輸入伺服器的埠號以及 IP 位址來發送 connect 的訊息。

```

//初始化結構地址變數name的內容
name.sin_family = AF_INET;//assign to Internet
name.sin_port = port.num();//setup port

//設定 巢狀結構內的s_addr(IP位址)
name.sin_addr.s_addr = ReceivingInterfaceAddr;
//Setup Server can communicate with hosts from any IP address

```

在 socket 『命名』 的這個步驟要注意的是 port number 的設置，RFC 2326 中規定要透過 RTSP 建立 『溝通協調』 管道的話該 socket 的阜號(port number)預設為 554，這邊要注意的是盡量避免和一些網路上 well known 的阜號重複，比如說最常用的 http 協定其阜號是 80，我們便不應該把我們 RTSP 的阜號也設為 80。

(註)socket 函數所產生的 socket 預設值為 Blocking socket。

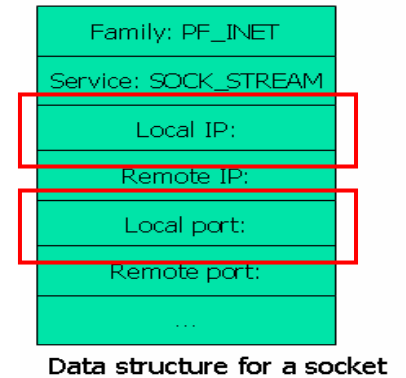


2.2.2 Bind

設定初始值後，必須呼叫 bind()函數來完成

socket 的命名工作，也就是把命名所設定的資料(port number 以及 IP address)和 sockfd 繫結在一起，

示意圖如右圖紅框所示。



(範例以及函數說明)

```

if (bind(newSocket, (struct sockaddr*)&name, sizeof name) != 0) {
    char tmpBuffer[100];
    sprintf(tmpBuffer, "bind() error (port number: %d): ",
            ntohs(port.num()));
}

```

```

int PASCAL FAR bind (SOCKET s, /*an unbound socket*/
struct sockaddr FAR *addr, /*local port and IP addr*/
int namelen); /*addr structure length*/

```


2.2.3 Listen

當應用程式一旦 `listen()` 成功之後，此 socket 便維持在等待連接狀態，我們稱為 listening socket; 一個 socket 在呼叫 `listen()` 成為 listening socket 之後，核心便會配置兩個佇列如下圖 2.8 所示。

Listening socket 所接收的 client 連線我們稱之為『未完成連線佇列』；這些未完成連線的佇列要直到完成 3WS 後在 OS 核心內才會變成『已完成連線佇列』；這些已完成的連線佇列接下來要呼叫 `accept()` 才會變成一個 connected socket，圖 2.8 虛線中的 socket 總數不得超過 `listen()` 函數中，第二個整數型態的引數數目 [15]。

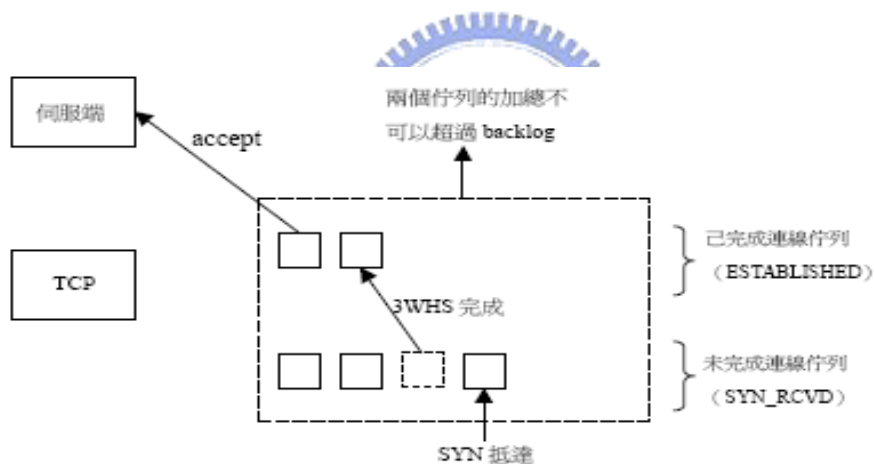


圖 2.8 Socket Queue

(範例以及函數說明)

```
if (listen(ourSocket, LISTEN_BACKLOG_SIZE) < 0) {  
    env.setResultErrMsg("listen() failed: ");  
    break;  
}
```

```
int PASCAL FAR listen (SOCKET s, /*a named unconnected socket*/  
int backlog); /*pending connect queue length*/
```

2.2.4 Accept

當 Server 收到用戶端連線要求並完成 3WHS 後，則使用 `accept()` 來開啟另一個新的 socket 來和用戶端進行連線，注意新的 socket 稱之為 `connected socket`，程式中我們會以 `accept()` 所傳回的一個整數型態變數來代表之，我們稱之為 `connected sockfd` (以後簡稱 `connfd`)。原來的 `listening socket` 則持續監聽有無新的 client 的連線要求，這就是一個串流伺服器可以同時服務多個 client 的關鍵。

如下圖 2.9 所示，「`listening socket`」是負責接收 client 端透過 `connect()` 函數所發給 server 建立連線的請求，在 server 結束執行或被中止之前，此 `listening socket` 皆會存在；而核心會針對每個已完成 `accept()` 的 client 連線建立一個 `connected socket` 作為 server 與 client 資料 I/O 的介面，而當 server 服務結束或是 client 主動要求斷線時，此 `connected socket` 便會被關閉。

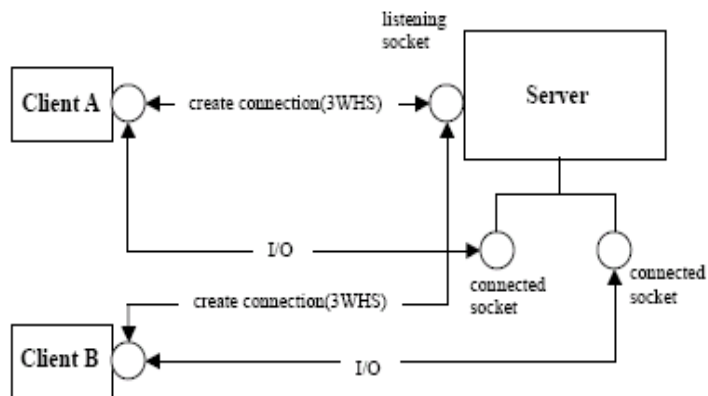


圖 2.9(a) Listening/Connected Socket

(範例以及函數說明)

```
int clientSocket = accept(fServerSocket, (struct sockaddr*)&clientAddr,  
                          &clientAddrLen);  
SOCKET PASCAL FAR accept(SOCKET s, /*a listening socket*/  
struct sockaddr FAR *addr, /*name of incoming socket*/  
int FAR *addrlen); /*length of sockaddr*/
```

如下圖 2.9(b)所示，經由 accept()所產生的 connected socket 便是我們以後多媒體串流通訊的 entry point，通常一個連線會需要三個 socket，分別作為 RTSP、RTP 以及 RTCP 通訊之用。而由下圖 2.9(c)所示，我們可以看到由 BSD socket API 所建立的基本伺服器與我們的串流伺服器(圖 2.9(b))有著些微的差異。

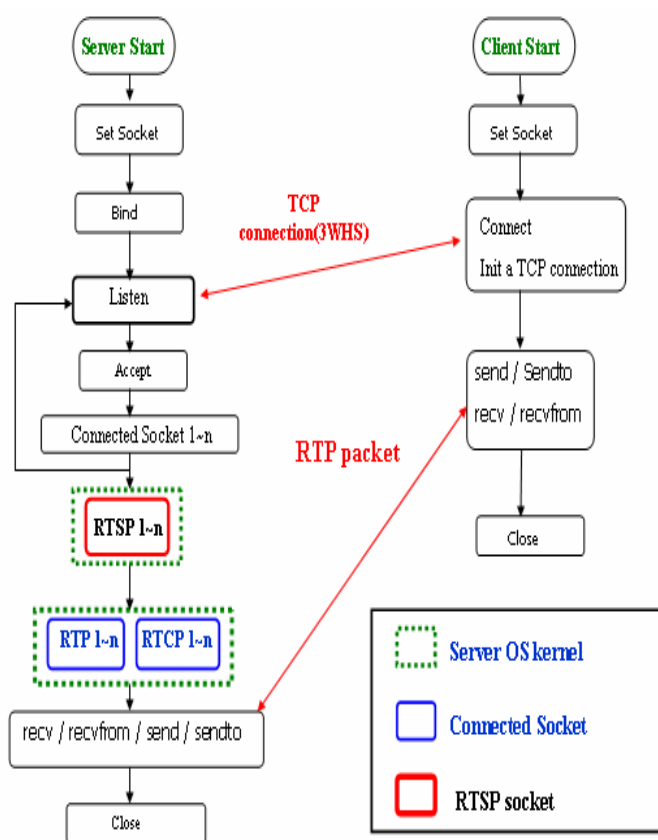


圖 2.9(b) Streaming Server flowchart

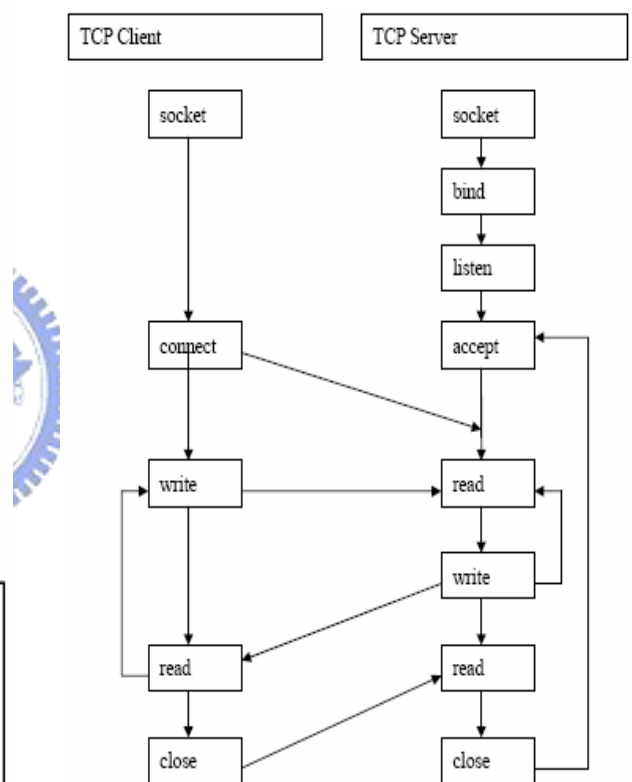


圖 2.9(c) Basic Server flowchart

這中間的主要差異便是在於 RTSP 以及 RTP socket 的設置，RTSP 以及 RTP 的部分分別在第二章後半段及第三章會予以說明，而串流伺服器的多工(Socket I/O Multiplex)細節在四章會詳細討論之。

2.2.5 Connect

用戶端會呼叫 connect() 函式與伺服器端的 socket 完成連接的動作，也就是進行 TCP 3-way handshaking 以得知彼此的 IP 地址以及位址，如下圖 2.10 所示當 client 呼叫 connect() 後即開始了 ARP 以及 TCP 3-way handshaking 的動作，在建立一個 TCP 的 oriented-connection 後，我們即可以在此連線之上傳送重要的 RTSP 控制訊息（註：140.113.13.87 為 client；140.113.13.97 為 server）。

No.	Time	Source	Destination	Protocol	Info.
128	22:52:54.005200	00000000.0010c6	00000000.ffffff	BROADCAST	Host Announcement CHANG_DB, Workstation, Server, Print Queue Server, NT Work
130	22:52:54.014884	140.113.13.87	Broadcast	ARP	Who has 140.113.13.97? Tell 140.113.13.87
131	22:52:54.015181	140.113.13.97	140.113.13.87	ARP	140.113.13.97 is at 00:04:23:b7:e9:52
132	22:52:54.015240	140.113.13.87	140.113.13.97	TCP	1056 > 8554 [SYN] Seq=0 Ack=0 Win=65535 Len=0 MSS=1460
133	22:52:54.015464	140.113.13.87	140.113.13.97	TCP	8554 > 1056 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
134	22:52:54.015518	140.113.13.87	140.113.13.97	TCP	1056 > 8554 [ACK] Seq=1 Ack=1 Win=65535 Len=0
135	22:52:54.015622	140.113.13.87	140.113.13.97	RTSP	OPTIONS rtsp://140.113.13.97:8554/mpeg1or2AudioVideoTest RTSP/1.0
136	22:52:54.015973	140.113.13.97	140.113.13.87	RTSP	Reply: RTSP/1.0 200 OK
137	22:52:54.017597	140.113.13.87	140.113.13.97	RTSP	DESCRIBE rtsp://140.113.13.97:8554/mpeg1or2AudioVideoTest RTSP/1.0
138	22:52:54.018031	140.113.13.97	140.113.13.87	RTSP/SD	Reply: RTSP/1.0 200 OK, with session description
139	22:52:54.028104	140.113.13.87	224.0.0.22	IGMP	V3 Membership Report
140	22:52:54.028199	140.113.13.87	228.67.43.91	UDP	Source port: 15947 Destination port: 15947
141	22:52:54.028356	140.113.13.87	224.0.0.22	IGMP	V3 Membership Report
142	22:52:54.058508	140.113.13.87	140.113.13.97	RTSP	SETUP rtsp://140.113.13.97:8554/mpeg1or2AudioVideoTest/track1 RTSP/1.0
143	22:52:54.062153	140.113.13.97	140.113.13.87	RTSP	Reply: RTSP/1.0 200 OK
144	22:52:54.071631	140.113.13.87	140.113.13.97	RTSP	SETUP rtsp://140.113.13.97:8554/mpeg1or2AudioVideoTest/track2 RTSP/1.0
145	22:52:54.074247	140.113.13.97	140.113.13.87	RTSP	Reply: RTSP/1.0 200 OK
146	22:52:54.075850	140.113.13.87	140.113.13.97	RTSP	PLAY rtsp://140.113.13.97:8554/mpeg1or2AudioVideoTest RTSP/1.0
147	22:52:54.076473	140.113.13.97	140.113.13.87	RTSP	Reply: RTSP/1.0 200 OK
148	22:52:54.076683	140.113.13.97	140.113.13.87	MPEG-1	MPEG-1 message
149	22:52:54.076835	140.113.13.97	140.113.13.87	MPEG-1	MPEG-1 message
150	22:52:54.076872	140.113.13.97	140.113.13.87	MPEG-1	MPEG-1 message
151	22:52:54.077037	140.113.13.97	140.113.13.87	MPEG-1	MPEG-1 message
152	22:52:54.077074	140.113.13.97	140.113.13.87	MPEG-1	MPEG-1 message
153	22:52:54.077266	140.113.13.97	140.113.13.87	MPEG-1	MPEG-1 message

圖 2.10 Capturing Streaming Packet

(範例以及函數說明)

```
if (connect(fInputSocketNum, (struct sockaddr*)&remoteName, sizeof remoteName)
    != 0) {
    envir().setResultErrMsg("connect() failed: ");
    break;
}
```

```
int PASCAL FAR connect(SOCKET s, /*an unconnected socket*/
struct sockaddr FAR *addr, /*remote port and IP address*/
int namelen); /*address structure length*/
```

2.2.6 Recv/Recvfrom

Server 端與 Client 端間已建立了連線之後，Recv/Recvfrom 目的便是讀取網路傳到 socket 介面的封包資料，通常 UDP socket 利用 recvfrom() 來接收 RTP 封包資料，並且可同時得知來源的 address; 而 TCP socket 則使用 recv() 來接收 RTSP 信令封包即可，因為 TCP 已經是一個 connection-oriented 的連線了。

(範例以及函數說明)

```
int bytesRead
= recv(sock, &((char*)buf)[totBytesRead], numChars - totBytesRead, 0);
```

```
int PASCAL FAR recv(SOCKET s, /*associated socket*/
char FAR *buf, /*buffer for incoming data*/
int len, /*length of buf*/
int flags); /*option flags*/
```

(範例以及函數說明)

```
bytesRead = recvfrom(socket, (char*)buffer, bufferSize, 0,
(struct sockaddr*)&fromAddress,
&addressSize);
```

```
int PASCAL FAR recvfrom(SOCKET s, /*associated socket*/
char FAR *buf, /*buffer for incoming data*/
```

```

int len, /*length of buf*/
int flags, /*option flags*/
struct sockaddr FAR *from, /*remote socket name*/
int fromlen); /*length of sockaddr*/

```

2.2.7 Send/Sendto

同樣是在 Server 端與 Client 端間已建立了連線之後，我們便利用 Send/Sendto 來進行資料的傳送，而 UDP socket 利用 Sendto()來傳送 RTP 封包資料;而 TCP socket 則使用 Send()來傳送 RTSP 信令封包。

(範例以及函數說明)

```

send(fClientSocket, okResponse, strlen(okResponse), 0);

```

```

int PASCAL FAR send(SOCKET s, /*associated socket*/
const char FAR *buf, /*buffer with outgoing data*/
int len, /*bytes to send*/
int flags); /*option flags*/

```

(範例以及函數說明)

```

int bytesSent = sendto(socket, (char*)buffer, bufferSize, 0,
(struct sockaddr*)&dest, sizeof dest);

```

```

int PASCAL FAR sendto(SOCKET s, /*a valid socket*/
const char FAR *buf, /*buffer with outgoing data*/
int len, /*bytes to send*/
int flags, /*option flags*/
struct sockaddr FAR *to, /*remote socket name*/
int tolen); /*length of sockaddr*/

```

2.2.8 Close

主動關閉 socket 連線。

```
if (sock >= 0) {  
    socketLeaveGroup(env, sock, testAddr.s_addr);  
    _close(sock);  
}
```

2.3 串流伺服器的溝通協調

在建立connected socket後，server和client雙方便可以傳遞『溝通協調』的訊息，也就是client會透過RTSP協定來告知server它所要的媒體檔案資訊，在確認相關RTSP控制訊息無誤後，server便開始透過UDP socket傳送RTP封包媒體資料給client的socket介面接收;由上圖2.10我們可以看到RTP payload為裝載著MPEG-1的影像資料，而MPEG-1影像資料的RTP裝載方法規定於RFC 2250[5]之中。

Server傳送RTP封包的動作還是必須經由BSD socket API，所以我們接下來要繼續談到recv()/recvfrom()以及send()/sendto()函數，同時也會配合Live555的範例程式來進行說明(testOnDemandRTSPServer.cpp)。

2.3.1 Live555 RTSPClientSession

Live555 server的recv()以及send()是以RTSPClientSession建構子方式執行，該建構子第一個執行的動作便是呼叫turnOnBackgroundReadHandling()。

```
//打開背景讀取處理的函數

envir().taskScheduler().turnOnBackgroundReadHandling(fClientSocket,

(TaskScheduler::BackgroundHandlerProc*)&incomingRequestHandler, this);
```

turnOnBackgroundReadHandling()函數主要負責串流伺服器的多工設置，Live555 的串流伺服器基本上是屬於iterative的形式，也就是一個程式中只有一個行程同時負責連線的『接待』以及『服務』，此種型態主要需配合select()函數方能完成之，關於select()我們在第四章的串流伺服器實作細節再行討論。

當串流伺服器建立了 connected socket 後，我們便可以使用 BSD socket API 的 recv()/recvfrom()來接收 client 呼叫 send()/sendto()所傳送的封包訊息，也就是開始進入 RTSP 信令的傳送、接收、解析以及執行的實現階段，而要完成這個工作我們首先必須在串流伺服器實做一個 RTSP 解析器 (RTSP parser)，目的是辨識出 RTSP 封包中的文字訊息，在訊息中有四個比較重要的訊息，說明請參考下面紅框內的註釋。

```

/*****
RTSP parser:
由buffer中把RTSP command 給解析出來
接下來用strcmp比對出要執行的RTSP method為何
*****/
//先定義一些字串名稱
// Parse the request string into command name and 'CSeq',
// then handle the command:
char cmdName[PARAM_STRING_MAX]; //RTSP command name

char urlPreSuffix[PARAM_STRING_MAX];
/*urlPreSuffix :URL前字串
EX:"mpeg1or2AudioVideoTest" (欲串流URL的 sessionName)
也就是連線的名稱(=presentation)
*/

char urlSuffix[PARAM_STRING_MAX];
/*URL後字串
urlSuffix EX:"track1"=video;urlSuffix EX:"track2"=Audio
*/

char cseq[PARAM_STRING_MAX];
//command sequence

```


而 RTSP parser 解析後要如何決定執行哪一個 RTSP 命令呢？所以我們需一個函數來實作這個『判斷機制』的功能，也就是我們下面看到的 parseRequestString()函數，該函數主要負責解析讀取來自 buffer 內的資料。

```
/*parseRequestString()函數功能:  
1.負責讀取fbuffer中的資料  
2.RTSP命令的解析(解析後會有對應的RTSP handleCmd()函數去執行)  
*/  
//parseRequestString()的引數即為fBuffer[10000]讀進的資料  
//parseRequestString()詳細的內容在下面  
if (!parseRequestString((char*)fBuffer, totalBytes,  
                        cmdName, sizeof cmdName,  
                        urlPreSuffix, sizeof urlPreSuffix,  
                        urlSuffix, sizeof urlSuffix,  
                        cseq, sizeof cseq)) {
```



Live555 的 parseRequestString()函數最重要的便是要能正確解析 **CmdName (RTSP method)**，關於解析的實作技巧我們留至第四章節(RTSP 訊令之傳輸以及辨認)討論之。

如下紅框內所示，當 **CmdName** 解析出來後，Live555 接下來便以 strcmp()函數來判斷接收到的 **cmdName** 字串是否為串流伺服器中所設置的 RTSP 信令之一，相同的話便去執行相對應的信令執行函數，比如說收到 RTSP "OPTIONS"命令之後便執行該 block 內的『handleCmd_OPTIONS(cseq);』函數。

```
/*  
*****  
用 strcmp比較字串cmdName與RTSP的命令  
相同的話就執行該命令  
*****  
*/  
if (strcmp(cmdName, "OPTIONS") == 0) {  
    handleCmd_OPTIONS(cseq);  
} else if (strcmp(cmdName, "DESCRIBE") == 0) {  
    handleCmd_DESCRIBE(cseq, urlSuffix, (char const*)fBuffer);  
} else if (strcmp(cmdName, "SETUP") == 0) {  
    handleCmd_SETUP(cseq, urlPreSuffix, urlSuffix, (char const*)fBuffer);  
} else if (strcmp(cmdName, "TEARDOWN") == 0  
            || strcmp(cmdName, "PLAY") == 0  
            || strcmp(cmdName, "PAUSE") == 0) {  
    handleCmd_withinSession(cmdName, urlPreSuffix, urlSuffix, cseq,  
                            (char const*)fBuffer);  
} else {  
    handleCmd_notSupported(cseq);  
}  
} ? end else ?
```

2.3.2 Live555 RTSP 連線建立流程

下圖 2.11 所示的是 Live555 server 所支援的 RTSP method，共有『OPTIONS』、『DESCRIBE』、『SETUP』、『TEARDOWN』、『PLAY』、『PAUSE』等六種，可以看到目前 Live555 server 尚未支援如『RECORD』的功能。

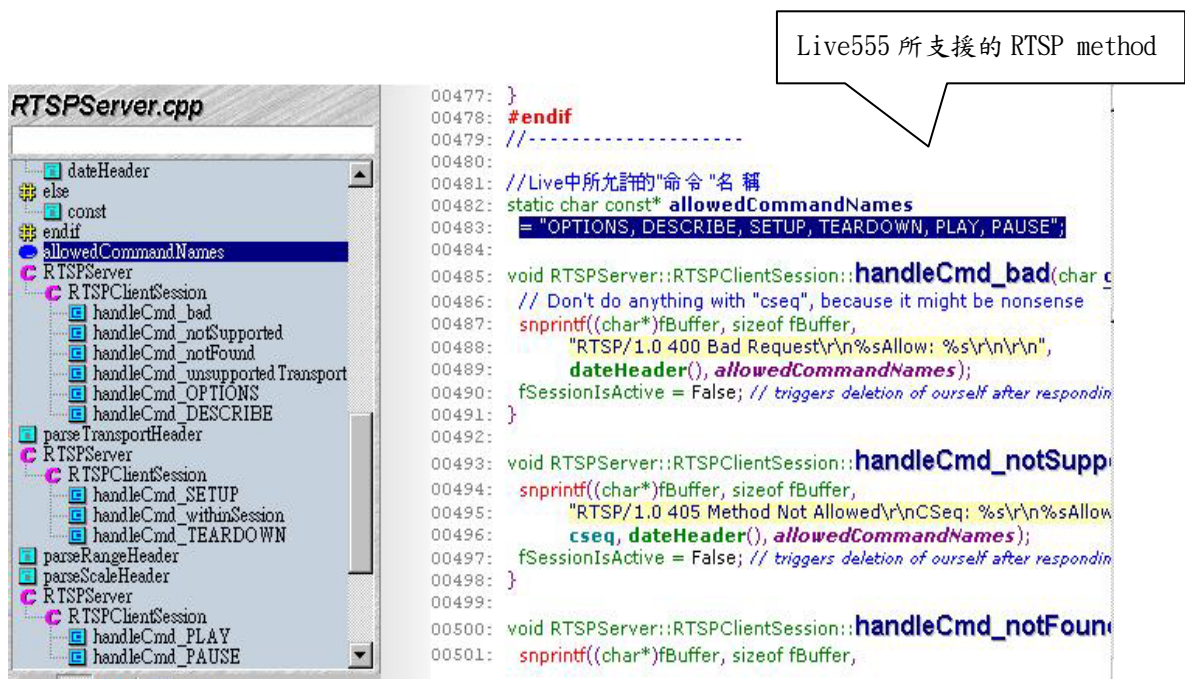


圖 2.11 Live555 support RTSP

到目前為止，串流伺服器的『標準連線建立』以及『溝通協調』部分我們已經概略論述完畢，故在此我們先做一個小結：

『當 TCP socket 建立完成之後，串流伺服器利用 recv()接收 client 的 RTSP 要求並回應之，也就是經由下圖 2.12 的 RTSP 命令傳送順序：OPTIONS->DESCRIBE->SETUP->PLAY->PAUSE->TEARDOWN，『PLAY』代表串流伺服器開始發送 RTP 封包的過程，

而這過程持續到收到『TEARDOWN』訊息才結束。

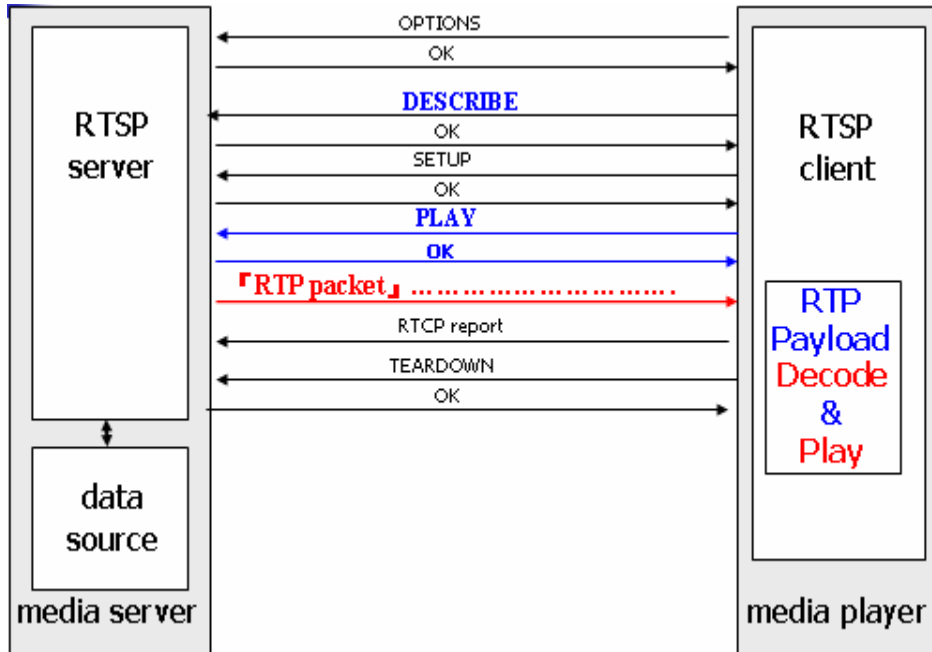


圖 2.12 RTSP negotiation

上圖 2.12 中我們必須深入解說的部分為『DESCRIBE』，因為這『DESCRIBE』是 client 能與 server 『溝通協調』的關鍵所在，所以我們必須對它的實現方式詳加論述;而『PLAY』因為牽涉到『RTP 封包包裝以及傳送』的部分，所以我們留到第三章再予以論述。

2.3.3 RTSP DESCRIBE

client 利用 DESCRIBE 信令向 server 要求媒體檔案的詳細描述資訊，server 端接受 DESCRIBE 後便把該檔案的一些重要資訊以 SDP[2] 協定回應給 client 端 (SDP 稍後會於 2.3.4 SDP 中解釋)，client 端收到這份關於媒體描述的 SDP 後，即檢查 SDP 內容中的影音編碼格式是否是 client 可以播放的，如果可以的話 client 接下來便發出 RTSP SETUP 信令給 server，當 client 收到 server SETUP 信令的 OK 回應後，便發送 RTSP PLAY 的請求給 server，在 server 回應 PLAY OK 給 client 後，便會開始傳送 RTP 封包，詳細的 RTSP 過程我們後面會加以敘述。下圖 2.13 便是 DESCRIBE 信令範例，我們可以看到 client 發送給 server 的內容中包含了一個 Accept header，目的是告知 server 可以用什麼協定來進行溝通，而虛線框中便是一個 SDP 的範例內容。



圖 2.13 RTSP DESCRIBE

2.3.4 SDP (Session Description Protocol , RFC2327[2])

SDP 是一種以文字格式 (Text-based) 描述多媒體會議(session)的通訊協定。SDP 可以被包含在SIP、RTSP、SAP、HTTP 甚至E-Mail 等各種協定中，其內容包含各種有關session的描述，比如說媒體種類(Video/Audio)、媒體傳輸協定(RTP/UDP/IP 等)、媒體格式(H.261 video/MPEG video/G.723.1 audio)等。

下圖2.14是一個用SDP 描述RTSP “DESCRIBE” method的範例，紅色框框內的就是
一個SDP格式的內容，我們實際擷取RTSP DESCRIBE的封包來分析說明：

```
▽ Session Description Protocol
  Session Description Protocol Version (v): 0
  ▶ Owner/Creator, Session Id (o): - 1156865095280000 1 IN IP4 140.113.13.97
  Session Name (s): Session streamed by "testOnDemandRTSPServer"
  Session Information (i): mpeg1or2AudioVideoTest
  ▶ Time Description, active time (t): 0 0
  ▶ Session Attribute (a): tool:LIVE555 Streaming Media v2006.08.06
  ▶ Session Attribute (a): type:broadcast
  ▶ Session Attribute (a): control:*
  ▶ Session Attribute (a): range:npt=0-86.479
  ▶ Session Attribute (a): x-qt-text-nam:Session streamed by "testOnDemandRTSPServer"
  ▶ Session Attribute (a): x-qt-text-inf:mpeg1or2AudioVideoTest
  ▽ Media Description, name and address (m): video 0 RTP/AVP 32
    Media Type: video
    Media Port: 0
    Media Proto: RTP/AVP
    Media Format: MPEG-I/II Video
  ▶ Connection Information (c): IN IP4 0.0.0.0
  ▶ Media Attribute (a): control:track1
  ▶ Media Description, name and address (m): audio 0 RTP/AVP 14
  ▶ Connection Information (c): IN IP4 0.0.0.0
  ▶ Media Attribute (a): control:track2
```

圖2.14 SDP content

1. v=0，表示 SDP version number 為 0
2. o表示 owner (session creator)連線建立者的相關資訊：
一開始為其 session id ,IN：Internet Network type 為 Internet
版本為 IPv4, server 的 address 為 140.113.13.97
3. session name：執行程式：testOnDemandRTSPServer，連線名稱：mpeg1or2A/Vtest

4. session information : 連線相關的資訊為mpeg1or2AudioVideoTest

5. Time Description : 對於時間的描述


6. Session Attribute: 連線的一些屬性資訊, 比如說:

tool(連線所使用的工具版本):Live555的版本為V2006.08.06

type (傳送方式) : broadcast , range (欲播放的範圍) : Normal Play Time(NPT)

7. Media Description (關於媒體的種類以及格式) :

在解釋之前我們必須要先介紹一下 RTP payload type (7 bits), 其定義在 RFC1890[14] 中。如下表 2.1 所示, Payload Type 提供了 128 種可能的 encoding 的方法, 此欄位表示了 RTP 的 payload 為何種格式, 故用戶端可以藉此明白該用什麼 decoder 來解碼。



PT	encoding name	audio/video (A/V)	clock rate (Hz)	channels (audio)
0	PCMU	A	8000	1
1	1016	A	8000	1
2	G721	A	8000	1
3	GSM	A	8000	1
4	unassigned	A	8000	1
5	DVI4	A	8000	1
6	DVI4	A	16000	1
7	LPC	A	8000	1
8	PCMA	A	8000	1
9	G722	A	8000	1
10	L16	A	44100	2
11	L16	A	44100	1
12	unassigned	A		
13	unassigned	A		
14	MPA	A	90000	(see text)
15	G728	A	8000	1
16--23	unassigned	A		
24	unassigned	V		
25	CelB	V	90000	
26	JPEG	V	90000	
27	unassigned	V		
28	nv	V	90000	
29	unassigned	V		
30	unassigned	V		
31	H261	V	90000	
32	MPV	V	90000	
33	MP2T	AV	90000	
34--71	unassigned	?		
72--76	reserved	N/A	N/A	N/A
77--95	unassigned	?		
96--127	dynamic	?		

表2.1 payload type table

故一個SDP其內容 (content) 主要可歸納成三部分：

第一部份包含了session名稱和目的以及該會議活動的時間：

- Session name and purpose -- session 的名稱及目的
- Time of the session is active -- session 開始的時間 (for conference)
- Information to receive those media -- 跟 session 相關的資料, 比如 address, ports, formats 等等.
- Information about the bandwidth to be used by the conference -- 可用以限制 session 耗用的流量
- Contact information for the person responsible for the session -- 跟 session 的負責人聯絡的資訊

第二部分是組成該會議的媒體種類以及接收這些媒體的控制信息：

- The type of media -- video 、audio或是 white board
- The transport protocol -- RTP/UDP/IP , H.320...etc.
- The format of the media -- H.261 video 、MPEG video... etc

第三部分是控制信息相關的：

IP multicast (群播) 以及 unicast (點播) 等。

- Multicast address and transport port for media
- Contact address and transport port for media

要特別注意的是因為RTSP是一個文字為主(Text-Based)的協定，故我們在撰寫RTSP這邊的程式時要特別注意其文字格式要完全依照RFC 2327 SDP所規定的格式來coding，如果未依照標準格式coding的話，則會發生server RTSP parser無法解析該字串訊息的情形發生。

當有這種情形發生時，代表雙方在『DESCRIBE』SDP時無法傳送或解析，這也意味著其後的『SETUP』與『PLAY』無法繼續下去，如下圖2.15所示『RTSP Unauthorized』的部分。

```

33 2.009147 140.113.13.87 Broadcast ARP Who has 140.113.13.97? Tell 140.113.13.87
34 2.009335 140.113.13.97 140.113.13.87 ARP 140.113.13.97 is at 00:04:23:b7:e9:52
35 2.009398 140.113.13.87 140.113.13.97 TCP 1266 > 554 [SYN] Seq=0 Ack=0 Win=65535 Len=0 MSS=1460
36 2.009606 140.113.13.97 140.113.13.87 TCP 554 > 1266 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
37 2.009665 140.113.13.87 140.113.13.97 TCP 1266 > 554 [ACK] Seq=1 Ack=1 Win=65535 Len=0
38 2.009775 140.113.13.87 140.113.13.97 RTSP OPTIONS rtsp://140.113.13.97/1.mp4 RTSP/1.0
39 2.048546 140.113.13.213 140.113.13.255 BROWSER Browser Election Request
40 2.060247 140.113.13.97 140.113.13.87 RTSP Reply: RTSP/1.0 200 OK
41 2.173084 140.113.13.87 140.113.13.97 TCP 1266 > 554 [ACK] Seq=126 Ack=257 Win=65279 Len=0
42 2.173292 140.113.13.97 140.113.13.87 RTSP OPTIONS * RTSP/1.0
43 2.177353 140.113.13.87 140.113.13.97 RTSP DESCRIBE rtsp://140.113.13.97/1.mp4 RTSP/1.0
44 2.285055 140.113.13.97 140.113.13.87 RTSP Reply: RTSP/1.0 401 Unauthorized

```

圖 2.15 RTSP Unauthorized

瞭解了 RTSP 協定的原理之後，我們接下來看 Live555 是如何實現『DESCRIBE』SDP 的功能。在收到 client 傳過來的『DESCRIBE』訊息後，server 端即進入『handleCmd_DESCRIBE{}』的 block 中，在該 block 主要利用 * session(會談連線指標)

來指向一個 ServerMediaSession(伺服器媒體會談連線)物件，ServerMediaSession 是 Live555 實現的一種表示『連線會談』的資料結構。

```
void RTSPServer::RTSPClientSession
::handleCmd_DESCRIBE(char const* cseq, char const* urlSuffix,
                    char const* fullRequestStr) {
    char* sdpDescription = NULL;
    char* rtspURL = NULL;
    do {
        if (!authenticationOK("DESCRIBE", cseq, fullRequestStr)) break;

        // Begin by looking up the "ServerMediaSession" object for the
        // specified "urlSuffix":
        ServerMediaSession* session
            = (ServerMediaSession*)(fOurServer.fServerMediaSessions->Lookup(urlSuffix));
        if (session == NULL) {
            handleCmd_notFound(cseq);
            break;
        }
    }
}
```

如下紅框所示，ServerMediaSession 建構子也包含了設置串流名稱以及顯示 Live 目前的版本資訊的串流初始工作。

```
/*ServerMediaSession建構子以及預設值
ServerMediaSession建構子主要目的為建立'伺服器媒體連線會談'物件初值
初值包含
1.設定 env物件
2.streamName串流名稱
3.info = Live版本的資訊
4.description = "LIVE.COM Streaming Media v";
5.gettimeofday(&fCreationTime, NULL);
*/ServerMediaSession::ServerMediaSession(UsageEnvironment& env,
                                           char const* streamName,
                                           char const* info,
                                           char const* description,
                                           Boolean isSSM, char const* miscSDPLines)
: Medium(env), fIsSSM(isSSM), fSubsessionsHead(NULL),
  fSubsessionsTail(NULL), fSubsessionCounter(0),
  fReferenceCount(0), fDeleteWhenUnreferenced(False)
{
    fStreamName = strDup(streamName == NULL ? "" : streamName);
    //Info為Live版本的資訊
    fInfoSDPString = strDup(info == NULL ? libNameStr : info);
    fDescriptionSDPString
        = strDup(description == NULL ? libNameStr : description);
    fMiscSDPLines = strDup(miscSDPLines == NULL ? "" : miscSDPLines);
    gettimeofday(&fCreationTime, NULL);
}
```

Live555 在使用 ServerMediaSession 建構子完成目前串流連線的一些基本資訊後，server 便呼叫 generateSDPDescription()產生一個以 SDP 文字為主的封包給 client 接收。

如下程式碼所示，generateSDPDescription()函數的執行的內容即是產生一個符合標準 SDP 格式的字串檔案，並且把這些 SDP 字串檔案封包再經由 socket API(send())傳送給 client 去接收，client 再依這個 SDP 協定內容(DESCRIBE response)去做設置 SETUP 的動作，故下一小節 2.4 中我們會以 client 為主來觀察其 RTSP 信令的互動情形。

```

//generateSDPDescription() 產生一個SDP 純文字的format的函數
//目的是由server向client傳回更多關於URL的訊息

// Then, assemble a SDP description for this session:
sdpDescription = session->generateSDPDescription();
//sdpDescription是字元型態的指標
//session是 ServerMediaSession類別型態的指標

//表示client要求播放的"檔案名稱"(file name)不存在
if (sdpDescription == NULL) {
// This usually means that a file name that was specified for a
// "ServerMediaSubsession" does not exist.
snprintf((char*)fBuffer, sizeof fBuffer,
"RTSP/1.0 404 File Not Found, Or In Incorrect Format\r\n"
"CSeq: %s\r\n"
"%s\r\n",
cseq,
dateHeader());
break;
}

```

```

// (產生一個SDP 純文字的format)
char const* const sdpPrefixFmt =
"v=0\r\n"
"o=- %ld%06ld %d IN IP4 %s\r\n"
"s=%s\r\n"
"i=%s\r\n"
"t=0 0\r\n"
"a=tool:%s\r\n"
"a=type:broadcast\r\n"
"a=control:*\r\n"
"%s"
"%s"
"a=x-qt-text-nam:%s\r\n"
"a=x-qt-text-inf:%s\r\n"
"%s";
sdpLength += strlen(sdpPrefixFmt)
+ 20 + 6 + 20 + ourIPAddressStrSize
+ strlen(fDescriptionSDPString)
+ strlen(fInfoSDPString)
+ strlen(libNameStr) + strlen(libVersionStr)
+ strlen(sourceFilterLine)
+ strlen(rangeLine)
+ strlen(fDescriptionSDPString)
+ strlen(fInfoSDPString)
+ strlen(fMiscSDPLines);
sdp = new char[sdpLength];
if (sdp == NULL) break;

// Generate the SDP prefix (session-level lines):
//傳回 SDP格式化字串

```

2.4 Server 與 Client 互動之流程

Client 端的標準連線程序與 server 大致相同，也就是利用 BSD socket API 的 socket() 以及 bind()，其後便呼叫 connect() 來啟動 TCP 3WHS。Live555 此部分可以參考 RTSPClient.cpp，該程式中的 RTSPClient 建構子便是負責 socket()、bind() 以及 connect() 等函數。

2.4.1 RTSPClient 動作流程

一開始 RTSPClient 建構子中會呼叫 describeURL () 來取得使用者由播放器輸入 URL 中的 IP 以及埠號的資訊。

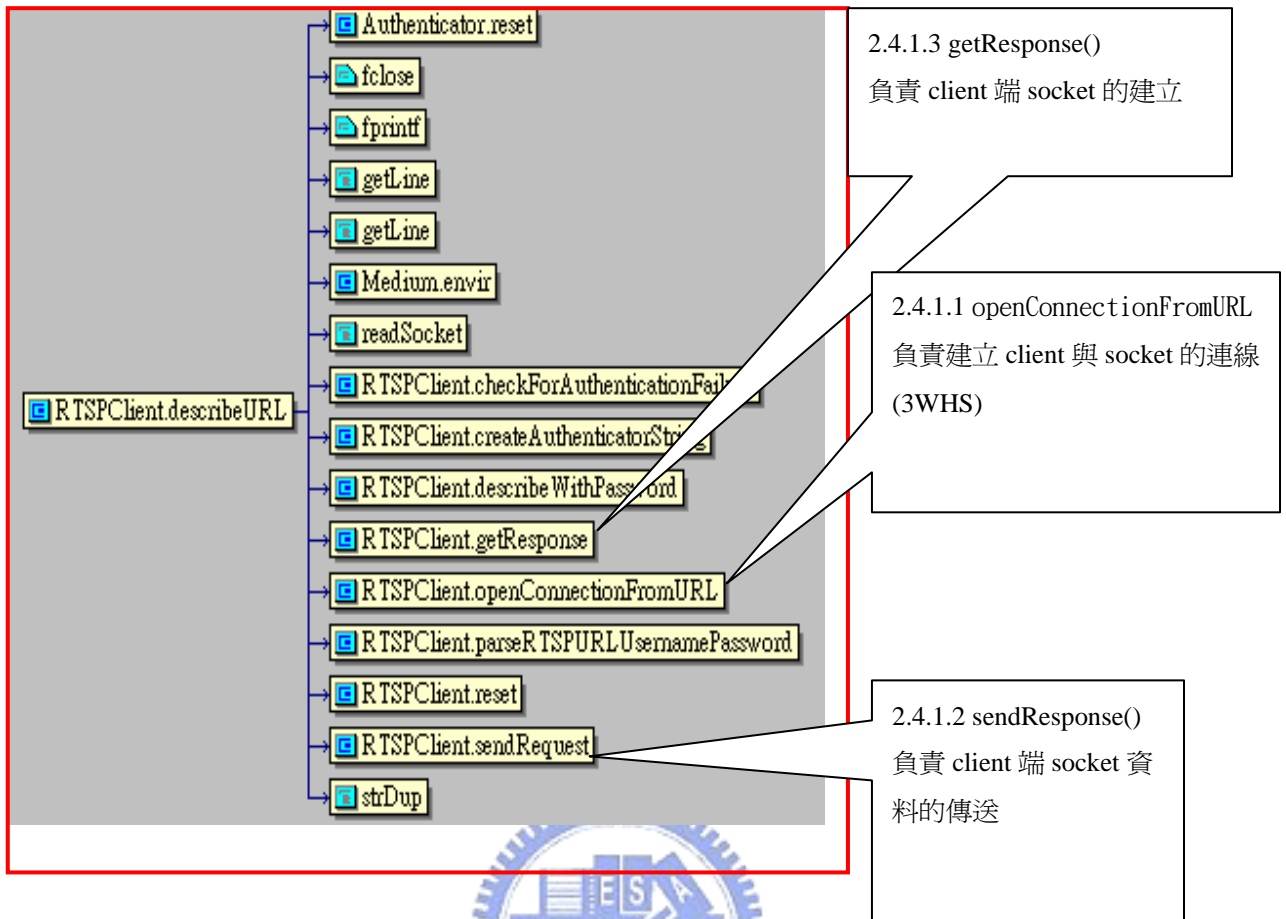


```
char* RTSPClient::describeURL(char const* url, Authenticator* authenticator,
                             Boolean allowKasennaProtocol) {
//cmd=command,字元型態的指標cmd
char* cmd = NULL;

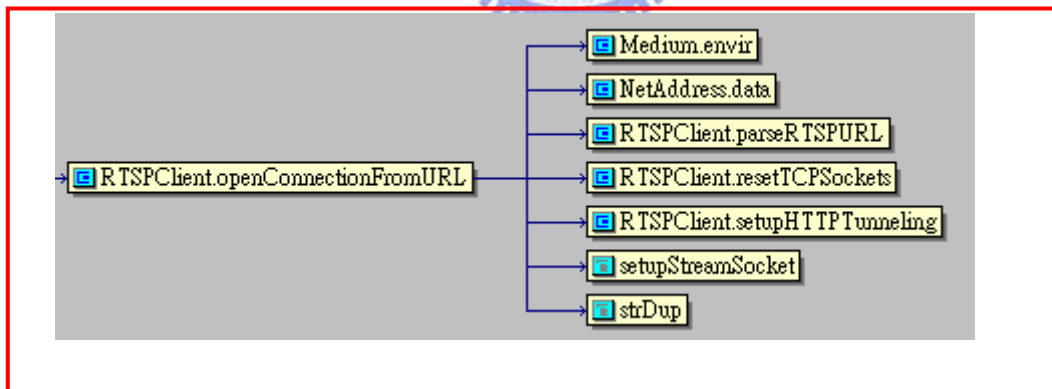
// fDescribeStatusCode(unsigned)
//0: OK; 1: connection failed; 2: stream unavailable
fDescribeStatusCode = 0;
do {
// First, check whether "url" contains a username:password to be used:
char* username; char* password;

/*如果認證用指標指向 NULL以及URL字串中的使用者名稱密碼傳回值為TRUE的話
則繼續向下執行openConnectionFromURL(url)開始建立一個client端的socket
*/
if (authenticator == NULL
    && parseRTSPURLUsernamePassword(url, username, password)) {
```

接下來 client 呼叫了下面紅框內的函數，但我們接下來只論述用戶端串流程式的關鍵部分，也就是 openConnectionFromURL(url) 、getResponse () 以及 sendResponse () 這三個函數。



2.4.1.1 openConnectionFromURL(url)



如上面的程式方塊圖所示，openConnectionFromURL(url)主要接收 url 字串中的 IP 地址以及埠號，接下來 openConnectionFromURL()會呼叫 setupStreamSocket()。setupStreamSocket() 就如同我們在 server 端所提過的就是呼叫 socket()以及 bind();bind()之後便是呼叫 connect ()，所以在 openConnectionFromURL(url)已經完成了 3WHS 的

連線並完成了一個 connected socket 的建立，也就是 OS kernel 會配置一個專屬的 connected socket 給目前的 client 以及 server 去傳輸資料。

2.4.1.2 sendRequest

建立 connected socket 後，client 便可以向 server 發送資料以進行串流連線的『溝通協調』，也就是呼叫 BSD API 的 send()來把 requestString 給傳送到 server 端。

```

/*****
Client開始傳送Request給Server,用BSD的send()
*****/
Boolean RTSPClient::sendRequest(char const* requestString, char const* tag,
                               Boolean base64EncodeIfOverHTTP) {
    if (fVerbosityLevel >= 1) {
        envir() << "Sending request: " << requestString << "\n";
    }
    char* newRequestString = NULL;
    //base64Encode~基本64字元編碼的函數
    if (fTunnelOverHTTPPortNum != 0 && base64EncodeIfOverHTTP) {
        requestString = newRequestString = base64Encode(requestString);
        if (fVerbosityLevel >= 1) {
            envir() << "\tThe request was base-64 encoded to: " << requestString << "\n\n";
        }
    }
}

/*****
C3.BSD send()
*****/
接下來參考getResponse1會去呼叫BSD的readSocket()
(Client開始接收Server傳回的資訊)
*****/
Boolean result
    = send(fOutputSocketNum, requestString, strlen(requestString), 0) >= 0;
delete[] newRequestString;

```

在我們實際以 Live555 RTSPServer 為串流伺服器，VLC player 為用戶端的連線情形下，我們看到一開始 VLC player 會傳送 RTSP『OPTIONS』給 Live RTSPserver，接下來便是『DESCRIBE』、『SETUP』、『PLAY』、『PAUSE』以及『TEARDOWN』，說明如下 2.4.2 章節。

2.4.2 Live555 RTSP 動作流程

接下來我們要介紹 Live555 對 RFC2326 的實現，依序說明於 2.4.2.1~2.4.2.6 小節，

下表為各小節的簡要說明表：

章節	RTSP 信令	簡要說明
2.4.2.1	OPTIONS	用戶端向伺服器端詢問支援哪些 RTSP 信令;伺服器端收到此信令後必須把它所支援的 RTSP 信令用封包回傳告知用戶端
2.4.2.2	DESCRIBE	用戶端利用 DESCRIBE 這個信令向伺服器端要求有關影音的相關資料; 伺服器端收到此信令後便先行解析該檔案，找出該檔案的影音編碼資訊並以 SDP 協定回傳之
2.4.2.3	SETUP	SETUP 是用戶端把它設定的傳送機制(machanism)傳給伺服器端，以便伺服器端依此機制開啟一個新的串流連線，倘若此機制伺服器端不支援，連線便結束之
2.4.2.4	PLAY	用戶端經由 PLAY 告知伺服器端可以開始傳送 RTP 封包，而伺服器端便依照 SETUP 時所設定的串流機制來傳送封包
2.4.2.5	PAUSE	用戶端要求伺服器端暫時停止串流的動作，但伺服器端並不釋放目前所有的串流資源
2.4.2.6	TEARDOWN	用戶端要求伺服器端釋放掉串流佔用的資源，也就是 TCP/UDP socket 的關閉動作

表 2.2 RTSP Method table

2.4.2.1 RTSP OPTIONS

如下紅框內所示，client 端呼叫 RTSPClient.cpp 程式內的 sendOptionsCmd()來發送 RTSP『OPTIONS』給 server 接收，最後便是要再呼叫 sendRequest()也就是 BSD 的 send()來把這個溝通協調的 RTSP 訊息傳給 server 去接收處理，OPTIONS 程式碼實作部分如下紅色框框內所示。

(client 端傳送 RTSP OPTIONS 的程式碼)

```
/*認證使用者資訊過後,由C->S一個OPTIONS RTSP command
  必須注意的是 每一個RTSP都會用sendRequest來傳送
  接下來用getResponse()來接收Server回應訊息
*/
char* RTSPClient::sendOptionsCmd(char const* url) {
    char* result = NULL;
    char* cmd = NULL;
    do {
        if (!openConnectionFromURL(url)) break;

        // Send the OPTIONS command:
        char* const cmdFmt =
            "OPTIONS %s RTSP/1.0\r\n"
            "CSeq: %d\r\n"
            "%s"
#ifdef SUPPORT_REAL_RTSP
            REAL_OPTIONS_HEADERS
#endif
            "\r\n";
        unsigned cmdSize = strlen(cmdFmt)
            + strlen(url)
            + 20 /* max int len */
            + fUserAgentHeaderStrSize;
        cmd = new char[cmdSize];
        sprintf(cmd, cmdFmt,
            url,
            ++fCSeq,
            fUserAgentHeaderStr);
        if (!sendRequest(cmd, "OPTIONS")) break;
    } while (0);
}
```

openConnectionFromURL()完成了 socket()、bind()以及 connect () 的動作。

以 SDP 文字協定來傳送『OPTIONS』訊息。

最後呼叫 send ()把 RTSP『溝通協調』命令經網路傳送至 server 端。

Server利用recvfrom()接收到這個RTSP OPTIONS 封包後，便透過一個RTSP Parser去比對所接收到的字串，然後去執行該RTSP 信令相對應的執行函數。以此例來說，server即是把它所支援的RTSP methods：『OPTIONS、DESCRIBE、SETUP、

TEARDOWN、PLAY、PAUSE』字串作為一個RTSP response封包回傳給client接收，如下圖2.16紅框中所示。

註:1.140.113.13.87 為 VLC client

2.140.113.13.97 為 Live555 RTSP Server。

(Live RTSPserver 端回送 RTSP OPTIONS OK 的封包內容)

No.	Time	Source	Destination	Protocol	Info
79	14.623684	140.113.13.87	140.113.13.97	TCP	2554 > 8554 [ACK] Seq=1 Ack=1 Win=65535 Len=0
80	14.623814	140.113.13.87	140.113.13.97	RTSP	OPTIONS rtsp://140.113.13.97:8554/mpeg1or2AudiovideoTest RTSP/
81	14.638781	140.113.13.97	140.113.13.87	RTSP	Reply: RTSP/1.0 200 OK
82	14.640585	140.113.13.87	140.113.13.97	RTSP	DESCRIBE rtsp://140.113.13.97:8554/mpeg1or2AudiovideoTest RTSP/
83	14.645241	140.113.13.97	140.113.13.87	RTSP/5	Reply: RTSP/1.0 200 OK, with session description


```

Frame 81 (176 bytes on wire, 176 bytes captured)
  Ethernet II, Src: 00:04:23:b7:e9:52, Dst: 00:02:44:63:fe:4e
  Internet Protocol, Src Addr: 140.113.13.97 (140.113.13.97), Dst Addr: 140.113.13.87 (140.113.13.87)
  Transmission Control Protocol, Src Port: 8554 (8554), Dst Port: 2554 (2554), Seq: 1, Ack: 148, Len: 122
  Real Time Streaming Protocol
    RTSP/1.0 200 OK\r\n
    CSeq: 1\r\n
    Date: Tue, Aug 29 2006 15:26:09 GMT\r\n
    Public: OPTIONS, DESCRIBE, SETUP, TEARDOWN, PLAY, PAUSE\r\n
  \r\n
  
```

圖 2.16 server RTSP OPTIONS response

2.4.2.2 RTSP DESCRIBE

Client利用DESCRIBE這個信令向server要求有關於媒體檔案的相關資料，如下圖

2.17所示，DESCRIBE訊息內容則包括了所需要多媒體連線的名稱，發出訊息的目的是為了要取得多媒體檔案的描述資訊。重點是發送的命令格式一定要符合SDP的格式（紅框內所示）。

(VLC client 端發送 RTSP DESCRIBE 的程式碼範例)

```

00328:
00329: /*定義char* authenticatorStr認證用字串型態的指標,如果有認證使用者ID以及PassWord
00330: 的相關資訊,會一起在DESCRIBE命令中傳遞
00331: */
00332: char* authenticatorStr
00333: = createAuthenticatorString(authenticator, "DESCRIBE", url);
00334:
00335: char const* acceptStr = allowKasennaProtocol
00336: ? "Accept: application/x-rtp-mh, application/sdp\r\n"
00337: : "Accept: application/sdp\r\n";
00338:
00339: // (Later implement more, as specified in the RTSP spec, sec D.1 ####)
00340:
00341: //定義一個char* cmdFmt字元型態的指標,內容即是 RTSP命令的格式
00342: char* const cmdFmt =
00343: "DESCRIBE %s RTSP/1.0\r\n"
00344: "CSeq: %d\r\n"
00345: "%s"
00346: "%s"
00347: "%s"

```

(VLC client 發送 DESCRIBE 的封包內容)

```

82 14.640585 140.113.13.87 140.113.13.97 RTSP DESCRIBE rtsp://140.113.13.97:8554/mpeg1or2AudiovideoTest RTSP/1.0
83 14.645241 140.113.13.97 140.113.13.87 RTSP/S Reply: RTSP/1.0 200 OK, with session description
84 14.660344 140.113.13.87 224.0.0.22 IGMP v3 Membership Report
85 14.660456 140.113.13.87 228.67.43.91 UDP Source port: 15947 Destination port: 15947
.....
> Frame 82 (227 bytes on wire, 227 bytes captured)
> Ethernet II, Src: 00:02:44:63:fe:4e, Dst: 00:04:23:b7:e9:52
> Internet Protocol, Src Addr: 140.113.13.87 (140.113.13.87), Dst Addr: 140.113.13.97 (140.113.13.97)
> Transmission Control Protocol, Src Port: 2554 (2554), Dst Port: 8554 (8554), Seq: 148, Ack: 123, Len: 173
> Real Time Streaming Protocol
  > DESCRIBE rtsp://140.113.13.97:8554/mpeg1or2AudiovideoTest RTSP/1.0\r\n
    Method: DESCRIBE
    URL: rtsp://140.113.13.97:8554/mpeg1or2AudiovideoTest
    CSeq: 2\r\n
    Accept: application/sdp\r\n
    User-Agent: VLC Media Player (LIVE.COM Streaming Media v2004.11.11)\r\n
    \r\n

```

圖 2.17 client RTSP DESCRIBE request

如果此多媒體連線名稱對 server 而言是有效檔案的話(因為 Live RTSPServer 會去看連線名稱來決定所要播放的檔案是哪一個), 則 server 將此檔案的描述資訊以 SDP 協定回傳給 client。如下圖 2.18 所示, 內容將包括了多媒體檔案的時間長度, 影音的磁軌(track)

資訊，多媒體解壓縮資訊等。

(LiveRTSPserver 回應 RTSP DESCRIBE 的封包內容)

```
83 14.645241 140.113.13.97 140.113.13.87 RTSP/S Reply: RTSP/1.0 200 OK, with session description
84 14.660344 140.113.13.87 224.0.0.22 IGMP V3 Membership Report
.....
▶ Frame 83 (682 bytes on wire, 682 bytes captured)
▶ Ethernet II, Src: 00:04:23:b7:e9:52, Dst: 00:02:44:63:fe:4e
▶ Internet Protocol, Src Addr: 140.113.13.97 (140.113.13.97), Dst Addr: 140.113.13.87 (140.113.13.87)
▶ Transmission Control Protocol, Src Port: 8554 (8554), Dst Port: 2554 (2554), Seq: 123, Ack: 321, Len: 628
▼ Real Time Streaming Protocol
  ▼ RTSP/1.0 200 OK\r\n
    Status: 200
    CSeq: 2\r\n
    Date: Tue, Aug 29 2006 15:26:09 GMT\r\n
    Content-Base: rtsp://140.113.13.97:8554/mpeg1or2AudiovideoTest/\r\n
    Content-Type: application/sdp\r\n
    Content-Length: 446\r\n
    \r\n
  ▶ Session Description Protocol
```

圖 2.18 server RTSP DESCRIBE response

2.4.2.3 RTSP SETUP

Client端利用**SETUP**信令告知server目前用戶端的埠號以及所使用的RTP通訊協定。如圖2.19所示，這些資訊都包含在『Transport Header』（程式碼請參考RTSPClient.cpp）。

(VLC client 端發送 RTSP SETUP 的封包內容)

```
87 14.669769 140.113.13.87 140.113.13.97 RTSP SETUP rtsp://140.113.13.97:8554/mpeg1or2AudiovideoTest/track1 RTSP/1.0
88 14.672865 140.113.13.97 140.113.13.87 RTSP Reply: RTSP/1.0 200 OK
89 14.681223 140.113.13.87 140.113.13.97 RTSP SETUP rtsp://140.113.13.97:8554/mpeg1or2AudiovideoTest/track1 RTSP/1.0
90 14.684104 140.113.13.97 140.113.13.87 RTSP Reply: RTSP/1.0 200 OK

Frame 87 (256 bytes on wire, 256 bytes captured)
Ethernet II, Src: 00:02:44:63:fe:4e, Dst: 00:04:23:b7:e9:52
Internet Protocol, Src Addr: 140.113.13.87 (140.113.13.87), Dst Addr: 140.113.13.97 (140.113.13.97)
Transmission Control Protocol, Src Port: 2554 (2554), Dst Port: 8554 (8554), Seq: 321, Ack: 751, Len: 202
Real Time Streaming Protocol
  SETUP rtsp://140.113.13.97:8554/mpeg1or2AudiovideoTest/track1 RTSP/1.0\r\n
    Method: SETUP
    URL: rtsp://140.113.13.97:8554/mpeg1or2AudiovideoTest/track1
    CSeq: 3\r\n
    Transport: RTP/AVP;unicast;client_port=2556-2557\r\n
    User-Agent: VLC Media Player (LIVE.COM Streaming Media v2004.11.11)\r\n
  \r\n
```

圖 2.19 client RTSP SETUP request

而接下來 server 便會遵照 client 所要求的播放方式來進行串流的動作(如上圖 2.19 紅框中的串流傳輸方式)，也就是 Live RTSPserver 會依照 VLC player 傳過來的 SETUP 信令來設置封包的傳輸方式，比如說在 Transport header 中我們可以看到這個連線將會以 unicast 的方式來播放，而 VLC player 所使用的 RTP/RTCP 埠號也會在此告知，而 Live RTSPServer 其回應訊息如下圖 2.20 所示。

(Live555 RTSPServer 回應給 VLC client 的封包內容)

```
86 14.660615 140.113.13.87 224.0.0.22 IGMP v3 Membership Report
87 14.669769 140.113.13.87 140.113.13.97 RTSP SETUP rtsp://140.113.13.97:8554/mpeg1or2Audiov
88 14.672865 140.113.13.97 140.113.13.87 RTSP Reply: RTSP/1.0 200 OK
89 14.681223 140.113.13.87 140.113.13.97 RTSP SETUP rtsp://140.113.13.97:8554/mpeg1or2Audiov
90 14.684104 140.113.13.97 140.113.13.87 RTSP Reply: RTSP/1.0 200 OK

Frame 88 (229 bytes on wire, 229 bytes captured)
Ethernet II, Src: 00:04:23:b7:e9:52, Dst: 00:02:44:63:fe:4e
Internet Protocol, Src Addr: 140.113.13.97 (140.113.13.97), Dst Addr: 140.113.13.87 (140.113.13.87)
Transmission Control Protocol, Src Port: 8554 (8554), Dst Port: 2554 (2554), Seq: 751, Ack: 523, Len: 175
Real Time Streaming Protocol
  RTSP/1.0 200 OK\r\n
  CSeq: 3\r\n
  Date: Tue, Aug 29 2006 15:26:00 GMT\r\n
  Transport: RTP/AVP;unicast;destination=140.113.13.87;client_port=2556-2557;server_port=6970-6971\r\n
  Session: 1\r\n
  \r\n
```

圖 2.20 server RTSP SETUP response

2.4.2.4 RTSP PLAY

Client 接收 server 回傳的 SETUP 訊息後便會知道 server 端所使用的 RTP/RTCP 埠號資訊，故 VLC player(Client)在完成用戶端接收模組的初始化動作後，便發出 **PLAY** 訊息告知伺服器可開始傳輸 RTP 封包。VLC player 發送的 PLAY 信令內容包括了多媒體檔案的起始時間、終止時間等資訊，如下圖 2.21 所示，起始時間為 0.000 即是片頭一開始，而終止時間未著名表示請 Live RTSPserver 播放到最後。

(VLC client 發送 PLAY 訊息給 Live RTSPServer，請求傳送 RTP 封包資料)

```
91 14.685751 140.113.13.87 140.113.13.97 RTSP PLAY rtsp://140.113.13.97:8554/mpeg1or2AudioVideoTest RTSP/1.0
92 14.686668 140.113.13.97 140.113.13.87 RTSP Reply: RTSP/1.0 200 OK
02 14.686874 140.113.13.97 140.113.13.87 MPEG-1 MPEG-1 message
.....
▶ Frame 91 (229 bytes on wire, 229 bytes captured)
▶ Ethernet II, Src: 00:02:44:63:fe:4e, Dst: 00:04:23:b7:e9:52
▶ Internet Protocol, Src Addr: 140.113.13.87 (140.113.13.87), Dst Addr: 140.113.13.97 (140.113.13.97)
▶ Transmission Control Protocol, Src Port: 2554 (2554), Dst Port: 8554 (8554), Seq: 737, Ack: 1101, Len: 175
▼ Real Time Streaming Protocol
  ▼ PLAY rtsp://140.113.13.97:8554/mpeg1or2AudioVideoTest RTSP/1.0\r\n
    Method: PLAY
    URL: rtsp://140.113.13.97:8554/mpeg1or2AudioVideoTest
    CSeq: 5\r\n
    Session: 1\r\n
    Session: 1
    Range: npt=0.000-\r\n
    User-Agent: VLC Media Player (LIVE.COM Streaming Media v2004.11.11)\r\n
    \r\n
```

圖 2.21 client RTSP PLAY request

下圖2.22所示的是Live RTSPserver的PLAY回應封包內容，伺服器如果完成串流傳輸的準備，則將下一個RTP封包的序號(sequence number)先傳給client做為確認之用。

(Live RTSPserver 的 PLAY 封包內容，傳回給 VLC client player)

```


92 14.686668 140.113.13.97 140.113.13.87 RTSP Reply: RTSP/1.0 200 OK
02 14.686674 140.113.13.97 140.113.13.87 MPEG-1 MPEG-1 message
.....
▶ Frame 92 (338 bytes on wire, 338 bytes captured)
▶ Ethernet II, Src: 00:04:23:b7:e9:52, Dst: 00:02:44:63:fe:4e
▶ Internet Protocol, Src Addr: 140.113.13.97 (140.113.13.97), Dst Addr: 140.113.13.87 (140.113.13.87)
▶ Transmission Control Protocol, Src Port: 8554 (8554), Dst Port: 2554 (2554), Seq: 1101, Ack: 912, Len: 284
▼ Real Time Streaming Protocol
  ▼ RTSP/1.0 200 OK\r\n
    Status: 200
    CSeq: 5\r\n
    Date: Tue, Aug 29 2006 15:26:09 GMT\r\n
    Range: npt=0.000-\r\n
  ▼ Session: 1\r\n
    Session: 1
  RTP-Info: url=rtsp://140.113.13.97:8554/mpeg1or2AudiovideoTest/track1;seq=50878;rtptime=384912574,url=rtsp://1
  \r\n

```

圖 2.22 server RTSP PLAY response

2.4.2.5 client RTSP PAUSE request

Client 要求暫停 session 傳送但不釋放伺服器端資源，如下圖 2.23 所示。



No.	Time	Source	Destination	Protocol	Info
2683	20.532088	140.113.13.97	140.113.13.87	MPEG-1	MPEG-1 message
2684	20.532276	140.113.13.97	140.113.13.87	MPEG-1	MPEG-1 message
2685	20.532392	140.113.13.97	140.113.13.87	RTP	Payload type=MPEG-I/II Audio, SSRC=4278282256, Seq=65035,
2686	20.532557	140.113.13.97	140.113.13.87	RTP	Payload type=MPEG-I/II Audio, SSRC=4278282256, Seq=65036,
2687	20.540983	140.113.13.87	140.113.13.97	RTSP	PAUSE rtsp://140.113.13.97:8554/mpeg1or2AudiovideoTest/tr
2688	20.541409	140.113.13.97	140.113.13.87	RTSP	Reply: RTSP/1.0 200 OK
2689	20.542237	140.113.13.87	140.113.13.97	RTSP	PAUSE rtsp://140.113.13.97:8554/mpeg1or2AudiovideoTest/tr
2690	20.542595	140.113.13.97	140.113.13.87	RTSP	Reply: RTSP/1.0 200 OK
2691	20.699523	140.113.13.87	140.113.13.97	TCP	2554 > 8554 [ACK] Seq=1240 Ack=1539 win=65458 Len=0
2692	20.770718	140.113.13.221	239.255.255.2	SSDP	NOTIFY * HTTP/1.1

```

.....
▶ Frame 2687 (218 bytes on wire, 218 bytes captured)
▶ Ethernet II, Src: 00:02:44:63:fe:4e, Dst: 00:04:23:b7:e9:52
▶ Internet Protocol, Src Addr: 140.113.13.87 (140.113.13.87), Dst Addr: 140.113.13.97 (140.113.13.97)
▶ Transmission Control Protocol, Src Port: 2554 (2554), Dst Port: 8554 (8554), Seq: 912, Ack: 1385, Len: 164
▼ Real Time Streaming Protocol
  ▼ PAUSE rtsp://140.113.13.97:8554/mpeg1or2AudiovideoTest/track1 RTSP/1.0\r\n
    Method: PAUSE
    URL: rtsp://140.113.13.97:8554/mpeg1or2AudiovideoTest/track1
    CSeq: 6\r\n
  ▼ Session: 1\r\n
    Session: 1
    User-Agent: VLC Media Player (LIVE.COM Streaming Media v2004.11.11)\r\n
  \r\n

```

圖 2.23 client RTSP PAUSE request

如果Server接受暫停指令，則回傳一個封包給client確認之，並暫停串流資料的傳輸，server針對RTSP PAUSE回應內容如下圖2.24所示。

(Live RTSPServer告知VLC player已暫停封包的傳送)

```

2688 20.541409 140.113.13.97 140.113.13.87 RTSP Reply: RTSP/1.0 200 OK
2689 20.542237 140.113.13.87 140.113.13.97 RTSP PAUSE rtsp://140.113.13.97:8554/mpeg1or2AudiovideoTest/trac
2690 20.542595 140.113.13.97 140.113.13.87 RTSP Reply: RTSP/1.0 200 OK
2691 20.699523 140.113.13.87 140.113.13.97 TCP 2554 > 8554 [ACK] Seq=1240 Ack=1539 win=65458 Len=0
2692 20.770718 140.113.13.221 239.255.255.2 SSDP NOTIFY * HTTP/1.1
.....
|> Frame 2688 (131 bytes on wire, 131 bytes captured)
|> Ethernet II, Src: 00:04:23:b7:e9:52, Dst: 00:02:44:63:fe:4e
|> Internet Protocol, Src Addr: 140.113.13.97 (140.113.13.97), Dst Addr: 140.113.13.87 (140.113.13.87)
|> Transmission Control Protocol, Src Port: 8554 (8554), Dst Port: 2554 (2554), Seq: 1385, Ack: 1076, Len: 77
|> Real Time Streaming Protocol
|> RTSP/1.0 200 OK\r\n
|> Status: 200
|> CSeq: 6\r\n
|> Date: Tue, Aug 29 2006 15:26:15 GMT\r\n
|> Session: 1\r\n
|> Session: 1
|> \r\n

```

圖 2.24 server RTSP PAUSE response

2.4.2.6 client RTSP TEARDOWN request

Client 要求將整個串流傳輸停止，如下圖 2.25 所示。

No.	Time	Source	Destination	Protocol	Info
4415	27.779912	140.113.13.97	140.113.13.87	MPEG-1	MPEG-1 message
4416	27.780149	140.113.13.97	140.113.13.87	MPEG-1	MPEG-1 message
4417	27.780249	140.113.13.87	140.113.13.97	RTSP	TEARDOWN rtsp://140.113.13.97:8554/mpeg1or2AudiovideoTest
4418	27.780373	140.113.13.97	140.113.13.87	MPEG-1	MPEG-1 message
4419	27.780463	140.113.13.97	140.113.13.87	MPEG-1	MPEG-1 message
4420	27.780543	140.113.13.97	140.113.13.87	MPEG-1	MPEG-1 message

```

.....
|> Frame 4417 (215 bytes on wire, 215 bytes captured)
|> Ethernet II, Src: 00:02:44:63:fe:4e, Dst: 00:04:23:b7:e9:52
|> Internet Protocol, Src Addr: 140.113.13.87 (140.113.13.87), Dst Addr: 140.113.13.97 (140.113.13.97)
|> Transmission Control Protocol, Src Port: 2554 (2554), Dst Port: 8554 (8554), Seq: 1604, Ack: 1930, Len: 161
|> Real Time Streaming Protocol
|> TEARDOWN rtsp://140.113.13.97:8554/mpeg1or2AudiovideoTest RTSP/1.0\r\n
|> Method: TEARDOWN
|> URL: rtsp://140.113.13.97:8554/mpeg1or2AudiovideoTest
|> CSeq: 10\r\n
|> Session: 1\r\n
|> Session: 1
|> User-Agent: VLC Media Player (LIVE.COM Streaming Media v2004.11.11)\r\n
|> \r\n

```

圖 2.25 client RTSP TEARDOWN request

當伺服器接收到此訊息後便關閉此串流傳輸，釋放連線的所有資源，並回應確認訊息給使用者，Server針對RTSP TEARDOWN回應內容如下圖2.26所示，可以看到Live RTSPServer傳回確認訊息給VLC player後，便是TCP連線結束的handshaking。

No.	Time	Source	Destination	Protocol	Info
4424	27.780821	140.113.13.97	140.113.13.87	MPEG-1	MPEG-1 message
4425	27.780854	140.113.13.87	140.113.13.97	RTSP	Reply: RTSP/1.0 200 OK
4426	27.780892	140.113.13.97	140.113.13.87	TCP	8554 > 2554 [FIN, ACK] Seq=1996 Ack=1765 Win=65374 Len=0
4427	27.780953	140.113.13.87	140.113.13.97	TCP	2554 > 8554 [ACK] Seq=1765 Ack=1997 Win=65001 Len=0
4428	27.781653	3comEuro_ab:88	Spanning-tree	STP	RST. Root = 32768/00:00:a2:87:2c:00 Cost = 18 Port = 0x800a
4429	27.782040	140.113.13.87	140.113.13.97	RTCP	Receiver Report
4430	27.782225	140.113.13.87	140.113.13.97	RTCP	Receiver Report
4431	27.782390	140.113.13.87	140.113.13.97	TCP	2554 > 8554 [FIN, ACK] Seq=1765 Ack=1997 Win=65001 Len=0
4432	27.782628	140.113.13.97	140.113.13.87	TCP	8554 > 2554 [ACK] Seq=1997 Ack=1766 Win=65374 Len=0


```

Frame 4425 (120 bytes on wire, 120 bytes captured)
  Ethernet II, Src: 00:04:23:b7:e9:52, Dst: 00:02:44:63:1
  Internet Protocol, Src Addr: 140.113.13.97 (140.113.13.97),
  Transmission Control Protocol, Src Port: 8554 (8554),
  Real Time Streaming Protocol
    RTSP/1.0 200 OK\r\n
      Status: 200
      CSeq: 10\r\n
      Date: Tue, Aug 29 2006 15:26:22 GMT\r\n
      \r\n
  
```

連線結束 TCP handshaking

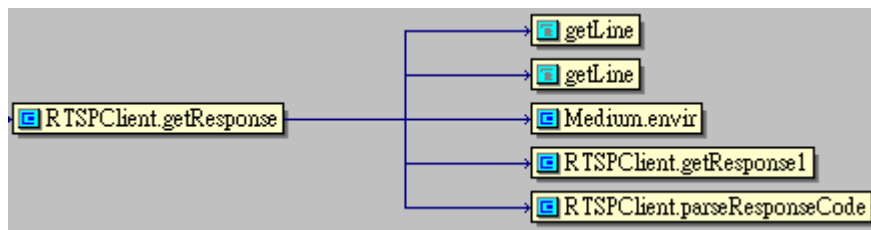
圖 2.26 server RTSP TEARDOWN response

要注意的是 client 端的程式設計在完成 SDP 格式的 coding 後，最後記得都要呼叫 sendRequest()將封包傳送之(也就是 BSD API 的 send())。

2.4.3 client getResponse ()

在 client 端呼叫 BSD send () 來發送 RTSP 命令後，server 端即回應相對應的 RTSP response 給 client，這時候 client 必須呼叫 getResponse() 來接收 server 端所回應的 RTSP 訊息。

如下面的 getResponse () 函數呼叫圖所示，當 client 接收到 server 的 RTSP response 後，需再呼叫五個函數，功能如下所述：



a. **getLine()** 是一行一行地去讀取 SDP 的文字內容。

b. **envir()** 函數主要是傳回一個 UsageEnvironment 的地址參照，如我們前面所介紹過的，

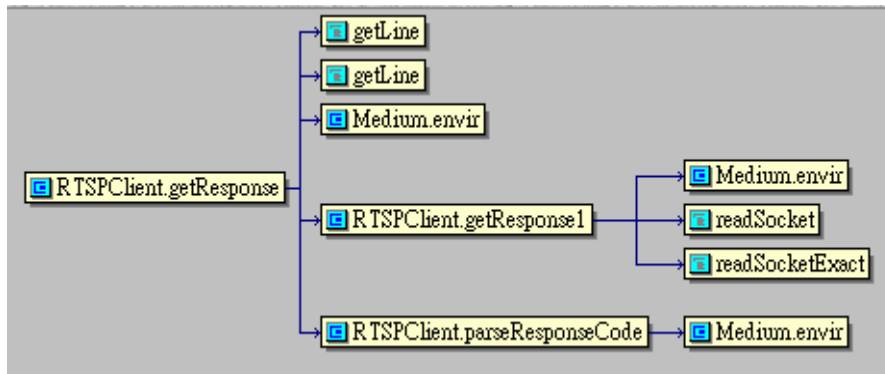
UsageEnvironment 這個類別的功用是負責接收串流程式內部所產生的訊息，並且把這些訊息傳達給使用者知道。

```
Boolean RTSPClient::getResponse(char const* tag,
                                unsigned& bytesRead, unsigned& responseCode,
                                char*& firstLine, char*& nextLineStart,
                                Boolean checkFor200Response) {
do {
    char* readBuf = fResponseBuffer;
    bytesRead = getResponse1(readBuf, fResponseBufferSize);
    if (bytesRead == 0) {
        envir().setResultErrMsg("Failed to read response: ");
        break;
    }
}
```

c. **getResponse1()**

由下面程式方塊圖可知 getResponse1() 函數主要呼叫 readSocket(); 而 readSocket() 會

再去呼叫 BSD 的 `recvfrom()` 來接收 server 所回應的 RTSP 訊息。



BSD socket API `recvfrom()` 如下所標示部分。

GroupsockHelper.cpp

- arg
- if ioctsocket(newSocket...
- elif defined(VX WORKS)
- else
- endif
- ifndef IMN_PIM
- blockUntilReadable
- else
- blockUntilReadable
- endif
- readSocket
- readSocketExact
- writeSocket
- getBufferSize
- getSendBufferSize
- getReceiveBufferSize
- setBufferTo
- setSendBufferTo
- setReceiveBufferTo
- increaseBufferTo

```

00548: #endif
00549:
00550: //讀取Socket內的data,
00551: //buffer= 存放接收到的資料的暫存區
00552: int readSocket(UsageEnvironment& env,
00553:               int socket, unsigned char* buffer, unsigned bufferSize,
00554:               struct sockaddr_in& fromAddress,
00555:               struct timeval* timeout) {
00556:     int bytesRead = -1;
00557:     do {
00558:
00559: //blockUntilReadable()直到socket可讀取前,先封鎖socket資料的傳送
00560: /*另一種非 攔阻式non-block說明如下
00561: */
00562:     int result = blockUntilReadable(env, socket, timeout);
00563:     if (timeout != NULL && result == 0) {
00564:         bytesRead = 0;
00565:         break;
00566:     } else if (result <= 0) {
00567:         break;
00568:     }
00569:
00570:     SOCKLEN_T addressSize = sizeof fromAddress;
00571:
00572: /*用標準的recvfrom()函數接收來自client的資料
00573: recvfrom(): 經由clientsocket(connected socket)讀取資料,
00574: 並儲存資料來源(對方)的地址。
00575: */
00576:
00577:     bytesRead = recvfrom(socket, (char*)buffer, bufferSize, 0,
00578:                        (struct sockaddr*)&fromAddress,

```

`recvfrom()` :

經由 `clientsocket` (connected socket) 讀取資料, 並儲存資料來源(對方)的地址。

d.parseResponseCode()

這邊開始讀取 `responseCode` 的資料, 解析 Server 傳回的狀態碼, 比如說狀態碼 200 是 OK 的意思, RTSP status code 的意思如下表 2.3 所示。

```

Boolean RTSPClient::parseResponseCode(char const* line,
                                     unsigned& responseCode) {
    if (sscanf(line, "%*s%u", &responseCode) != 1) {
        envir().setResultMsg("no response code in line: \\", line, "\\");
        return False;
    }
    return True;
}

```

Status-Code	=	"100"	; Continue
		"200"	; OK
		"201"	; Created
		"250"	; Low on Storage Space
		"300"	; Multiple Choices
		"301"	; Moved Permanently
		"302"	; Moved Temporarily
		"303"	; See Other
		"304"	; Not Modified
		"305"	; Use Proxy
		"400"	; Bad Request
		"401"	; Unauthorized
		"402"	; Payment Required
		"403"	; Forbidden
		"404"	; Not Found
		"405"	; Method Not Allowed
		"406"	; Not Acceptable
		"407"	; Proxy Authentication Required
		"408"	; Request Timeout

表 2.3 RTSP status code table

在第二章最後，我們用以下 8 個步驟來說明 Live555 Client-Server 間傳送接收 RTSP

的『溝通協調』命令順序：

- 1.Client 發送 RTSP 命令給 Server。
- 2.Server 接受、解析、判斷、執行 RTSP，並產生適當的回應訊息。
- 3.Client 接收到 Server 回應訊息，並解析其回應碼(Response code)。

4.Client 依回應碼決定相對應的處理函數。

5.Client 在處理函數中更進一步地去解析重要的串流資訊：

5.1 Transport Header (During RTSP 『SETUP』) 。

5.2 RTP 表頭資訊 (During RTSP 『PLAY』) 。

5.3 Scale Header 播放範圍表頭 (During RTSP 『PLAY』) 。

6.Client 把 RTP payload 的內容傳給上 decoder 應用程式去執行播放

(During PLAY) 。



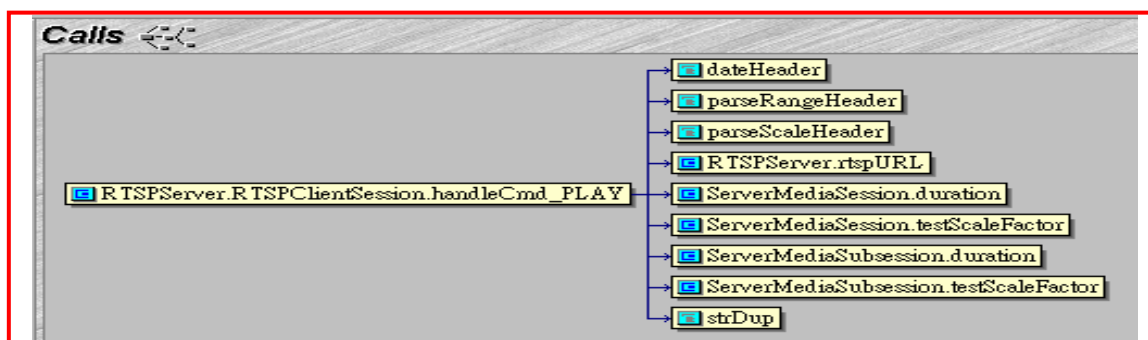
第三章 封包包裝及傳送

到目前為止我們已經把一個串流伺服器的『連線建立的標準程序』以及『溝通協調』部分論述完畢，在第三章要更去探討的地方有下面幾點：

- 1.在client發送RTSP『PLAY』給server接收後，server將會採取什麼動作？
- 2.在server開始傳送RTP封包之前，這些媒體封包的來源訊框(frame)是如何由一個媒體檔案的資料流中取得的？
- 3.在分離媒體檔案資料流中的表頭(header)以及訊框後，我們便把訊框打包進一個封包之中，而當然一個封包不可能裝進所有的訊框資料，因此衍生出來的切割封裝的演算法與所處網路環境的對應關係也是值得去瞭解的問題，因為這些問題牽動著串流媒體的效能表現。（註.詳細的封裝演算法在第四章才會討論之，第三章主要以瞭解Live555的程式流程為主）。

而在第二章中我們提到了Live555 server在收到RTSP『PLAY』後便會去執行handleCmd_PLAY()，所以我們接下來便由該行程式出發，以求瞭解在這個階段串流伺服器彙整了哪些功能來完成影音的串流傳輸。

3.1 RTSP PLAY信令的執行-handleCmd_PLAY()



如上紅框內所示，handleCmd_PLAY () 呼叫了九個函數，以下先對這九個函數做一個簡單介紹：

1.dateHeader() :

傳回系統儲存時間的相關資訊。

2.parseRangeHeader():

RangeHeader 是 client 用來告知 server 它所需的 data 範圍，也就是 Range header 會將 streaming media 的起始及結束的範圍告知 Server，所以 server 端需用這個函數來解析 client 所傳過來的 RTSP 『PLAY』 中的表頭資訊。

3.parseScaleHeader() :

ScaleHeader 是 client 用來告知 server 它所需的影片播放速率，所以 server 端需用這個函數來解析 client 所傳過來的 RTSP 『PLAY』 中的表頭資訊。



4.RTSPServer.rtspURL()

依照 URL 的 IP 位址去建立一個 UDP socket 以傳送 RTP 封包資料。

5.ServerMediaSession.duration() : 媒體連線會談持續的時間。

6.ServerMediaSession.testScaleFactor() : 測試媒體連線會談的影片播放率。

7.ServerMediaSubSession.duration() : 媒體子連線會談持續的時間。

8.ServerMediaSubSession.testScaleFactor() : 測試媒體子連線會談的影片播放率。

9.strDup(): 主要用途在 Buffer 中 string 的複製 (scaleHeader = strDup(buf)) 。

上述函數中我們針對 RTSPServer.rtspURL() 函數做特別說明，該函數會先呼叫

ourSourceAddressForMulticast(envir()), 最後會再去呼叫下圖紅框內的十個函數, 而我們

在此只論述到比較重要的三個函數, 說明如下:

1. setupDatagramSocket(): 建立一個 UDP Socket, 如下圖 3.1 所示。

2. readSocket(): server 讀取 client 端所傳輸的資料, 比如說 RTCP/UDP。

3. writeSocket(): server 傳送資料給 client 接收, 比如說 RTP/UDP。

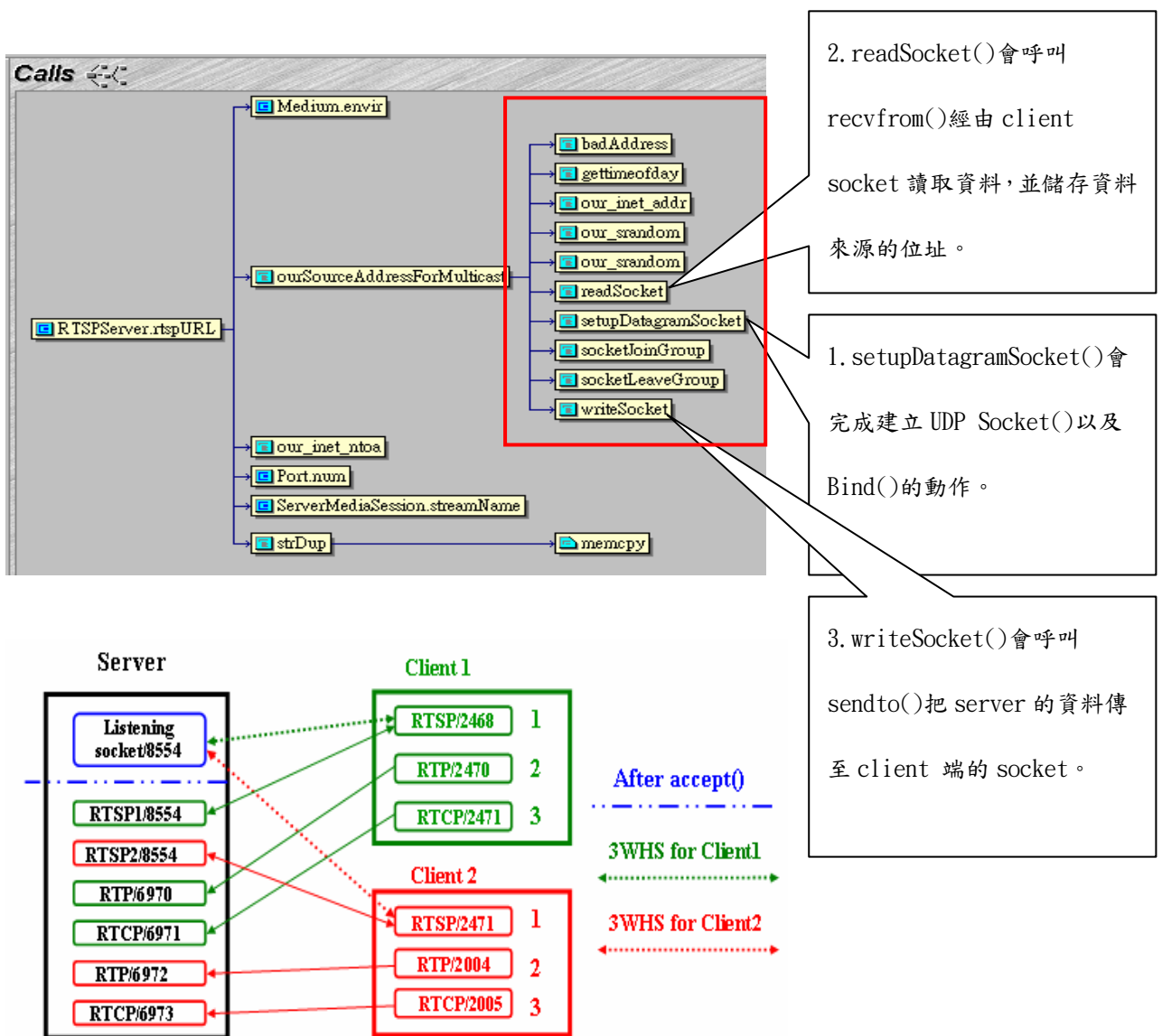


圖 3.1 Create RTP/RTCP socket after RTSP PLAY

而 UDP socket 和 TCP socket 的使用差別就是因為 UDP socket 是非連接導向 (Connectionless)，故不需要 connect () 函數，其餘的函數使用方法大致上同於我們第二章的介紹。（註.UDP socket API 函數流程圖可以參考圖 2.4）

3.1.1 伺服器媒體連線會談 (ServerMediaSession)

而建立好 UDP socket 之後，串流伺服器便具有能力把 RTP 封包傳送到網路上，但是 RTP 封包資料從哪裡來呢？所以我們接下來繼續看 Live555 的下一行程式碼。

```
# 4 ServerMediaSession* sms = ServerMediaSession::createNew(  
*env, streamName, streamName,descriptionString);
```

在前面 2.2.4 章節中 (SDP) 中我們約略介紹到 Live555 所提供『ServerMediaSession』類別的用途，它是 Live555 用來表示一個『伺服器連線會談的資料結構』，而 createNew() 會產生一個 ServerMediaSession 的建構子，該建構子用來建立以及初始化一個多媒體『連線會談』的相關資訊，比如說有：

- 1.streamName(串流名稱)~此即為 client 端輸入 URL suffix 的連線代號，需特別注意此變數的處理，因為 Live555 是以連線名稱來決定所要播放的檔案。
- 2.info (Live 版本的資訊)。
- 3.description= "LIVE.COM Streaming Media v" (軟體開發公司的 title)。
- 4.gettimeofday(取得系統的時間資訊)

而這也就是我們下面所看到的 User-Agent 欄位資訊。

```
80 14.623814 140.113.13.87 140.113.13.97 RTSP OPTIONS rtsp://140.113.13.97:8554/mpeg1or2AudioVideoTest
81 14.638781 140.113.13.97 140.113.13.87 RTSP Reply: RTSP/1.0 200 OK
82 14.640585 140.113.13.87 140.113.13.97 RTSP DESCRIBE rtsp://140.113.13.97:8554/mpeg1or2AudioVideoTest
83 14.645241 140.113.13.97 140.113.13.87 RTSP/S Reply: RTSP/1.0 200 OK, with session description
84 14.660344 140.113.13.87 224.0.0.22 IGMP V3 Membership Report
85 14.660456 140.113.13.87 224.0.0.22 UDP Source port: 15047 Destination port: 15047
.....
Frame 80 (201 bytes on wire, 201 bytes captured)
Ethernet II, Src: 00:02:44:63:fe:4e, Dst: 00:04:23:b7:e9:52
Internet Protocol, Src Addr: 140.113.13.87 (140.113.13.87), Dst Addr: 140.113.13.97 (140.113.13.97)
Transmission Control Protocol, Src Port: 2554 (2554), Dst Port: 8554 (8554), Seq: 1, Ack: 1, Len: 147
Real Time Streaming Protocol
  OPTIONS rtsp://140.113.13.97:8554/mpeg1or2AudioVideoTest RTSP/1.0\r\n
  CSeq: 1\r\n
  User-Agent: VLC Media Player (LIVE.COM Streaming Media v2004.11.11)\r\n
  \r\n
```

由 ServerMediaSession 建構子所完成的 User-Agent 資訊，包含串流名稱以及軟體版本資訊



3.1.2 伺服器的媒體底層連線會談 (ServerMediaSubSession)

ServerMediaSession (伺服器媒體連線會談) 類別的功用是幫助伺服器建立一些基本的訊息;而接下來的程式如 #5 所示:

```
# 5 sms->addSubsession(MPEG4VideoFileServerMediaSubsession::createNew
(*env, inputFileName, reuseFirstSource));
```

上述 #5 程式開始便進入媒體資料切割打包的部分，而

『MPEG4VideoFileServerMediaSubsession』這部分是我們所要串流的影音格式是什麼，即套用該影音格式的對應建構子，舉例來說:

『MPEG4VideoFile』 + 『ServerMediaSubsession』 = 『MPEG4VideoFileServerMediaSubsession』,或是

『MP3AudioFile』 + 『ServerMediaSubsession』 = 『MP3AudioFileServerMediaSubsession』

，也就是套用特定影音格式的 createNew 部分，這是因為每一種影音格式都有特定的 RTP 包裝方式，並且規定於不同的 RFC 文件中。因此，Live555 目前支援影音格式共有九種，每一種建構子都會對應到特定影音格式的解析以及封包演算法，以下列舉的是該九種建構子的程式碼。

1.A MPEG-4 video elementary stream: :

```
sms->addSubsession(MPEG4VideoFileServerMediaSubsession  
::createNew(*env, inputFileName, reuseFirstSource));
```

2. MPEG-1 or 2 audio+video program stream (elementary stream):

```
MPEG1or2FileServerDemux* demux = MPEG1or2FileServerDemux::createNew(*env,  
inputFileName, reuseFirstSource);  
  
sms->addSubsession(demux->newVideoServerMediaSubsession(iFramesOnly));  
  
sms->addSubsession(demux->newAudioServerMediaSubsession());
```

3.mp3 :

```
sms->addSubsession(MP3AudioFileServerMediaSubsession::createNew  
(*env, inputFileName, reuseFirstSource, useADUs, interleaving));
```

4.wav :

```
sms->addSubsession(WAVAudioFileServerMediaSubsession::createNew
```

```
(*env, inputFileName, reuseFirstSource, convertToULaw));
```

5.amr :

```
sms->addSubsession(AMRAudioFileServerMediaSubsession::createNew
```

```
(*env, inputFileName, reuseFirstSource));
```

6.vob :

```
sms->addSubsession(demux->newVideoServerMediaSubsession(iFramesOnly));
```

```
sms->addSubsession(demux->newAC3AudioServerMediaSubsession());
```

7.ts :

```
sms->addSubsession(MPEG2TransportFileServerMediaSubsession::createNew
```

```
(*env, inputFileName, reuseFirstSource));
```

8.m4v :

```
sms->addSubsession(MPEG4VideoFileServerMediaSubsession::createNew
```

```
(*env, inputFileName, reuseFirstSource));
```

9.aac

```
sms->addSubsession(ADTSAudioFileServerMediaSubsession
```

```
::createNew(*env, inputFileName, reuseFirstSource));
```

所以在 Live555 的『**testOnDemandRTSPServer**』這個程式所實作的串流伺服器具有在『同一時間』串流『九種媒體格式』的能力，也就是說在最大 connected socket 限制數目範圍內，串流伺服器可以隨意傳送這九種媒體格式的媒體檔案給用戶端接收。

*在此特別注意 *inputFileName* 是一個指向媒體檔案名稱的指標，預設會播放同一目錄下所設定的檔案(如 test.mpg)。如果想建置一個 Live Capture 即時影像傳送的串流伺服器，只要把即時抓取所儲存的檔案名稱設置成 test.mpg 即可，但注意即時抓取影像所儲存的影音格式要和支援的格式相符。

接下來我們便以 MPEG4VideoFileServerMediaSubsession::createNew () 來做說明。

如下紅框內所示，執行 createNew 之後便會傳回一個

MPEG4VideoFileServerMediaSubsession 的建構子。

```
MPEG4VideoFileServerMediaSubsession*
MPEG4VideoFileServerMediaSubsession::createNew(UsageEnvironment& env,
        char const* fileName,
        Boolean reuseFirstSource) {
    return new MPEG4VideoFileServerMediaSubsession(env, fileName, reuseFirstSource);
}
```

而該 MPEG4VideoFileServerMediaSubsession 的建構子主要會去做

FileServerMediaSubsession 的預設值設定。

```
: MPEG4VideoFileServerMediaSubsession
: ::MPEG4VideoFileServerMediaSubsession(UsageEnvironment& env,
:         char const* fileName, Boolean reuseFirstSource)
: : FileServerMediaSubsession(env, fileName, reuseFirstSource),
:   fDoneFlag(0) {
: }
:
```

而在 FileServerMediaSubsession 的建構子中會去做 OnDemandServerMediaSubsession 的預設值設定。

```
FileServerMediaSubsession
::FileServerMediaSubsession(UsageEnvironment& env, char const* fileName,
        Boolean reuseFirstSource)
: OnDemandServerMediaSubsession(env, reuseFirstSource),
  fFileSize(0) {
  fileName = strDup(fileName);
}
```

OnDemandServerMediaSubsession.cpp 這一個程式主要功能便是由 sdpLines() 函數

去呼叫『createNewStreamSource()』以及『createNewRTPSink()』，如下所示：

```
char const*
OnDemandServerMediaSubsession::sdpLines(ServerMediaSession& parentSession) {
    if (fSDPLines == NULL) {
        // We need to construct a set of SDP lines that describe this
        // subsession (as a unicast stream). To do so, we first create
        // dummy (unused) source and "RTPSink" objects,
        // whose parameters we use for the SDP lines:
        unsigned estBitrate; // unused
        FramedSource* inputSource = createNewStreamSource(0, estBitrate);
        if (inputSource == NULL) return NULL; // file not found

        struct in_addr dummyAddr;
        dummyAddr.s_addr = 0;
        Groupsock dummyGroupsock(envir(), dummyAddr, 0, 0);
        unsigned char rtpPayloadType = 96 + trackNumber() - 1; // if dynamic
        RTPSink* dummyRTPSink
            = createNewRTPSink(&dummyGroupsock, rtpPayloadType, inputSource);
    }
    return fSDPLines;
}
```

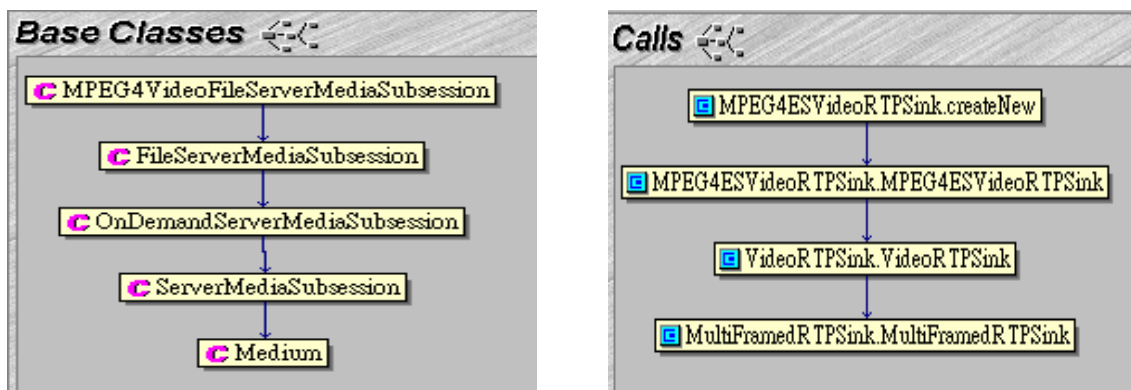
createNewStreamSource()函數功用主要負責開啟串流檔案，並從該檔案取得訊框

(frame)，這部分我們會在接下來的 3.2 章節（開啟串流檔案來源，

MPEG4VideoFileServerMediaSubsession.cpp）中論述。

而 createNewRTPSink()主要負責對這些訊框做打包(pack)的動作，並且在這些封包之中加入 RTP header 的資訊，這部分我們會在接下來的 3.4.3 章節（影像訊框的切割與

包裝，MultiFramedRTPSink.cpp）中論述。



(MPEG4VideoFileServerMediaSubsession 類別的繼承關係圖與函數呼叫圖)

3.2 訊框的切割、包裝以及傳送

在論述 Live555 訊框(frame)的切割、包裝以及傳送的方式之前，我們在此先對 3.4 章節的六個小節做個預覽說明，以便接下來能夠更有條理地分析 Live555 的 MPEG4VideoFileServerMediaSubsession.cpp 程式的功能模組。

小節	標題 (主要函數)	簡要說明
3.2.1	一些基本的網路資料傳輸觀念 (無)	如 IP、Fragment and MTU 等。
3.2.2	建立影像來源 (createNewStreamSource())	串流伺服器須能『看的懂』欲串流的檔案格式。
3.2.3	影像訊框的組成 (createNewRTPSink())	把解析出來的影音 frame 加上 RTP header 的相關資訊。
3.2.4	影像封包的切割與包裝 (MultiFramedRTPSink())	把訊框盡量打包進一個 MTU 封包中，並以 UDP 協定傳送之。
3.2.5	RTP Payload format for MPEG4 (createNewRTPSink())	主要是對於 RTP 協定以及 RFC3016 的說明。
3.2.6	取得各種影音格式的訊框: (doGetNextFrame())	特定媒體格式訊框的解析。

3.2.1 IP (Internet Protocol) 、Fragment and RTCP

IP 通訊協定定義於 RFC 791，位於下圖 3.2(a)OSI model 的第三層，是網際網路所使用的網路層通訊協定。IP 負責傳送資料到指定位址，但並不確認資料是否正確傳達，是一種無連結(Connectionless)的通訊協定，而 IP 主要負責以下三點：

- 1.Packet 路徑選擇(Routing)
- 2.Packet 分割(Fragmentation)
- 3.Packet 重組 (Re-assembly)

- by Packet's Fragment ID。

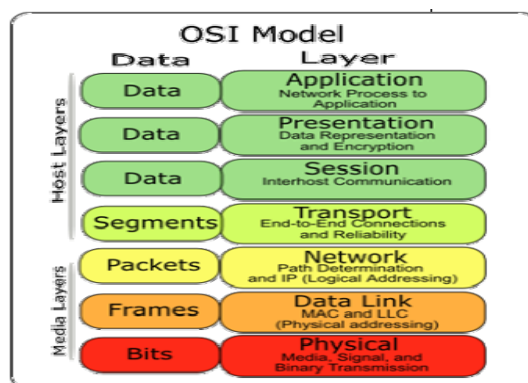


圖 3.2(a) OSI 7-layer model

通常網路上傳輸的封包大小都有限制(如下表 3.1 所示)，比如說在 Ethernet 上 MTU 的限制為 1500 Bytes，而 1500 Bytes 便是 MTU (Maximum Transfer Unit，最大傳輸單位) ;MTU 也就是 Data-link layer 中對資料(payload)傳輸的最大限制，而 Network layer 會因 Data-link layer 的 MTU 而對 frame 做切割的動作。

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

表 3.1 MTU limited table

以下為兩個 IP fragmentation 的例子：

範例一：假設原始 IP 封包大小為 4520Bytes，IP 表頭為 20Bytes，故 IP Payload 為 4500 Bytes；

若必須經由乙太網路傳送,在切割後每個 IPfragment 最大長度為 1500 Bytes，其中 IP 表

頭的長度為 20 Bytes，每一個 IP Payload 最大為 1480 Bytes，因此，會產生如下圖 3.2(b)

的 4 個新 IP fragment：

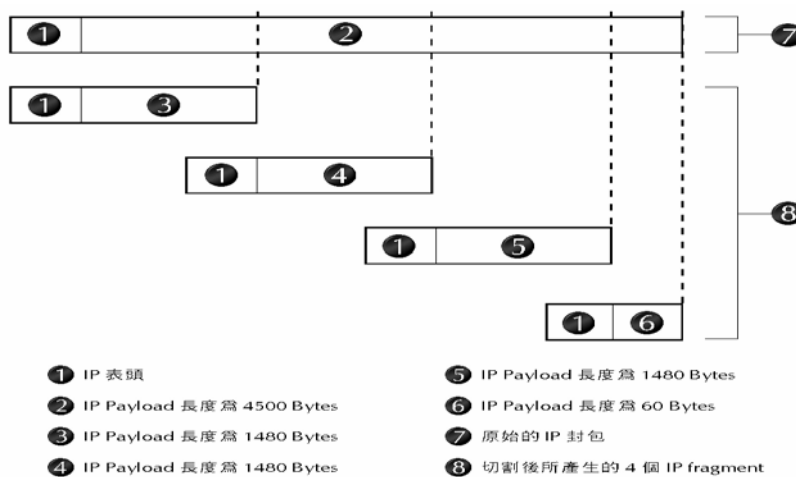


圖 3.2(b) IP fragmentation

下圖 3.3 為串流伺服器傳送 RTP 以及 RTSP 封包的內容資訊，因此以 MTU 為 1500Bytes 來說，最大的 RTP 封包 payload 便只能是 1460Bytes，如果串流伺服器故意設定 payload 長度為 1470Bytes 將會造成系統額外的負擔，因為 Network layer 會把一個 1470Bytes 的訊框切割成二個封包，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；而一般來說 RTSP 封包 payload 約在 100~200 Bytes 左右，故原則上是不會遇到一個 RTSP 封包因過大而要切割成兩個封包來傳送的問題。

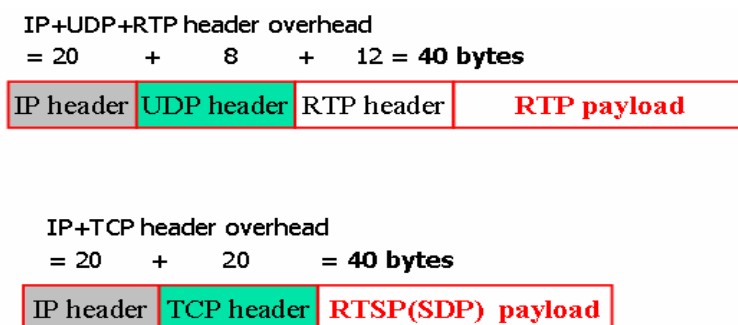


圖 3.3 A RTP/RTSP packet

範例二：假設我們目前欲傳送一段 DVD 畫質的影片，畫質要求如下

30 frames/sec，720x480 resolution，3 bytes per pixel

故每秒的資料量為=720x480x3x30=31104000 Bytes/sec(raw data)

假設可壓縮 100 倍，則每秒的資料量=311040 Bytes/s (MPEG-2 compressed data)

接下來我們分別考慮在 A 以及 B 環境下傳輸所需要的總封包資料量：

A 環境下，假設 MTU = 1500 Bytes，則每秒的總封包資料量將為 **319500 Bytes/sec**：

■ Header Overhead = 40 Bytes，Payload=1460 Bytes

$$311040/1460 = 213 \text{ packets/sec}$$

故(封包數)x(MTU) = 213x1500 = 319500 Bytes/sec

B 環境下，假設 MTU = 100 Bytes，每秒的總封包資料量為 **518400 Bytes/sec**：

■ Header Overhead = 40 Bytes，Payload=60 Bytes

$$311,040/60 = 5184 \text{ packets/sec}$$

故(封包數)x(MTU) = 5184x100 = 518400 Bytes/sec

由 A.B 的例子我們得到一個簡單的結論～

『MTU 以及 Header overhead 成反比關係』，故原則上我們希望 MTU 越大越好;但是過大的 MTU 也表示著當封包遺失時，使用者將會越容易發覺畫面上的掉格情形，影響收訊的品質。

因此，理想串流伺服器所傳送的封包大小應該能夠隨著『網路即時的頻寬狀況』來改變之。目前市面上 RealPlayer（固定兩種封包大小）是屬於固定封包大小的串流軟體；而像微軟 NetMeeting 封包則不固定大小(隨網路頻寬即時狀況而改變傳送封包的大小)，可以盡量在封包使用率、封包錯誤恢復能力以及收訊品質間達到比較佳的平衡點。

要達到動態封包大小的重要機制就是要取得 RTCP (RTP control protocol, RTP 控制協定, RFC3550[6]) 的 RR(Receiver Report)，其功能如下圖 3.4 所示。RTCP 主要功能便是用戶端每隔一段時間會把關於資料傳輸的 RR 以 RTCP 封包傳送給串流伺服器，伺服器端便可以由 RTCP 封包內的統計數字來動態地改變傳輸的速率來完成流量以及壅塞控制。此外，RTCP 在設計上要注意的是其發送的封包數目不可超過總傳輸封包數的 5%。

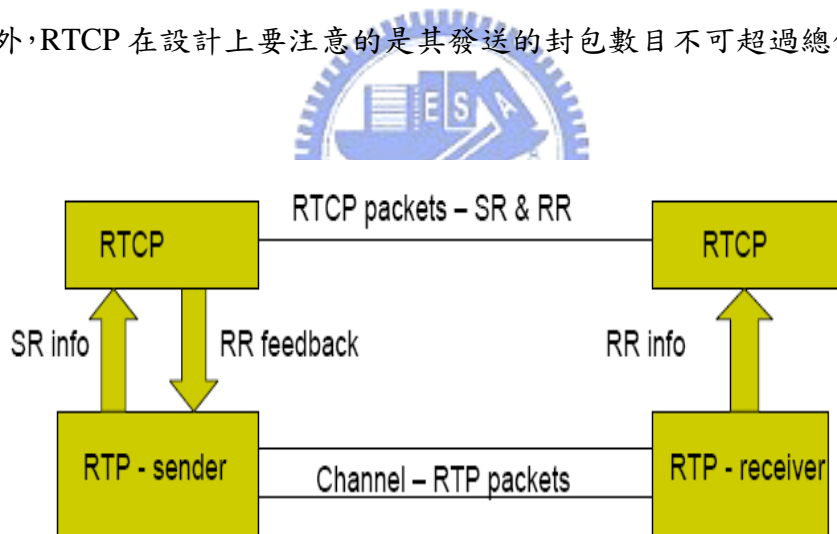


圖 3.4 RTCP RR

對上述的兩個範例有基本認識後，接下來我們便開始去探討串流伺服器在這個地方上是如何實作，首先我們回顧一下在前面 3.3 章節（伺服器的媒體底層連線會談）中，在 MPEG4VideoFileServerMediaSubsession.cpp 程式中看到的 createNewStreamSource() 函數以及 createNewRTPSink() 函數，其程式碼如下面兩個紅框所示，我們接下來分別敘述

于 3.2.2 以及 3.2.3 章節：

```
FramedSource* MPEG4VideoFileServerMediaSubsession
::createNewStreamSource(unsigned /*clientId*/, unsigned& estBitrate) {
    estBitrate = 500; // kbps, estimate

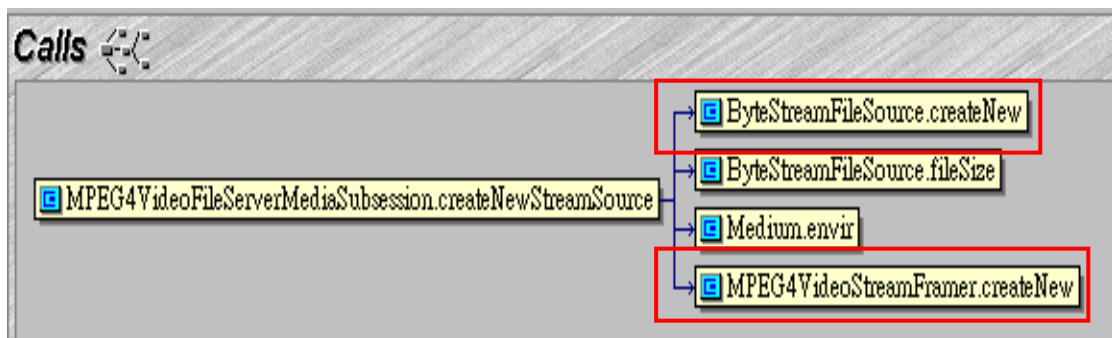
    // Create the video source:
    ByteStreamFileSource* fileSource
    = ByteStreamFileSource::createNew(envir(), fileName);
    if (fileSource == NULL) return NULL;
    fileSize = fileSource->fileSize();

    // Create a framer for the Video Elementary Stream:
    return MPEG4VideoStreamFramer::createNew(envir(), fileSource);
}

RTPSink* MPEG4VideoFileServerMediaSubsession
::createNewRTPSink(Groupsock* rtpGroupsock,
    unsigned char rtpPayloadTypeIfDynamic,
    FramedSource* /*inputSource*/) {
    return MPEG4ESVideoRTPSink::createNew(envir(), rtpGroupsock,
        rtpPayloadTypeIfDynamic);
}
```

3.2.2 建立影像來源 (createNewStreamSource())

如下函數呼叫圖所示，createNewStreamSource()主要負責建立影像來源，並設置一個框架產生器(framer)，負責由 elementary stream 中把訊框(frame)以及表頭(header)給分離出來。我們接下來要介紹的是比較重要的兩個函數，如下紅框所示，分別敘述於 3.2.2.1 小節 (ByteStreamFileSource.createNew)以及 3.2.2.2 小節 (MPEG4VideoStreamFramer.createNew)。



3.2.2.1 開啟檔案並讀取檔案內容 (ByteStreamFileSource.createNew())

1. 『FILE』 structure Stores information about current state of stream; used in all stream I/O operations. @ STDIO.H

2. OpenOutputFile ()
傳回欲開啟檔案的地址給 fid 檔案指標。

```
//      /// ByteStreamFileSource //////////  
ByteStreamFileSource*  
ByteStreamFileSource::createNew(UsageEnvironment& env, char const* fileName,  
    unsigned preferredFrameSize,  
    unsigned playTimePerFrame) {  
    FILE* fid = OpenInputFile(env, fileName);  
    if (fid == NULL) return NULL;  
  
    ByteStreamFileSource* newSource  
        = new ByteStreamFileSource(env, fid, preferredFrameSize, playTimePerFrame);  
    newSource->fFileSize = GetFileSize(fileName, fid);  
  
    return newSource;  
}
```

如上程式碼所示，我們利用『File』型態定義的檔案指標(fid)來指向所欲開啟的檔案，接下來 OpenInputFile()會開啟檔案並傳回檔案指標。注意第二個引數 fileName 為所要開啟檔案的名稱，而 OpenInputFile()函數主要用到了 C 所提供的 fopen()，如下所示。

```
FILE* OpenInputFile(UsageEnvironment& env, char const* fileName) {  
    FILE* fid;  
  
    // Check for a special case file name: "stdin"  
    if (strcmp(fileName, "stdin") == 0) {  
        fid = stdin;  
    #if defined(__WIN32__) || defined(_WIN32)  
        _setmode(_fileno(stdin), _O_BINARY); // convert to binary mode  
    #endif  
    } else {  
        fid = fopen(fileName, "rb");  
        if (fid == NULL) {  
            env.setResultMsg("unable to open file \"", fileName, "\"");  
        }  
    }  
}
```

Fopen()函數-
讀取媒體 binary 檔案
資料，fid 指標指向檔案
的開頭地址。

開啟檔案之後，便要由該檔案取得位元組資料，這部分的功能便是由接下來所呼叫的 ByteStreamFileSource 建構子完成，如下紅框內程式碼所示：

```

/*ByteStreamFileSource建構子負責建立一個訊框讀取的物件
並且這個物件訊框的讀取大小可以由我們自己設定
包含每個訊框的播放時間亦可在這設定
*/
ByteStreamFileSource::ByteStreamFileSource(UsageEnvironment& env, FILE* fid,
        unsigned preferredFrameSize,
        unsigned playTimePerFrame)
: FramedFileSource(env, fid), fPreferredFrameSize(preferredFrameSize),
  fPlayTimePerFrame(playTimePerFrame), fLastPlayTime(0), fFileSize(0) {
}

```

由下圖所示，在 ByteStreamFileSource.cpp 程式中的 ByteStreamFileSource 建構子中可以看到有一個 doGetNextFrame() 函數，該函數目的便是不斷地從已開啟的媒體檔案中讀取資料，而該函數主要用到了 C 所提供的 fread()，要注意的是不同的影音格式需要實現不同的 doGetNextFrame() 函數，此部分留至 3.2.6 章節再論述。

```

void ByteStreamFileSource::doGetNextFrame() {
    if (feof(fFid) || ferror(fFid)) {
        handleClosure(this);
        return;
    }
    // Try to read as many bytes as will fit in the buffer provided
    // (or "fPreferredFrameSize" if less)
    if (fPreferredFrameSize > 0 && fPreferredFrameSize < fMaxSize) {
        fMaxSize = fPreferredFrameSize;
    }
    fFrameSize = fread(fTo, 1, fMaxSize, fFid);
}

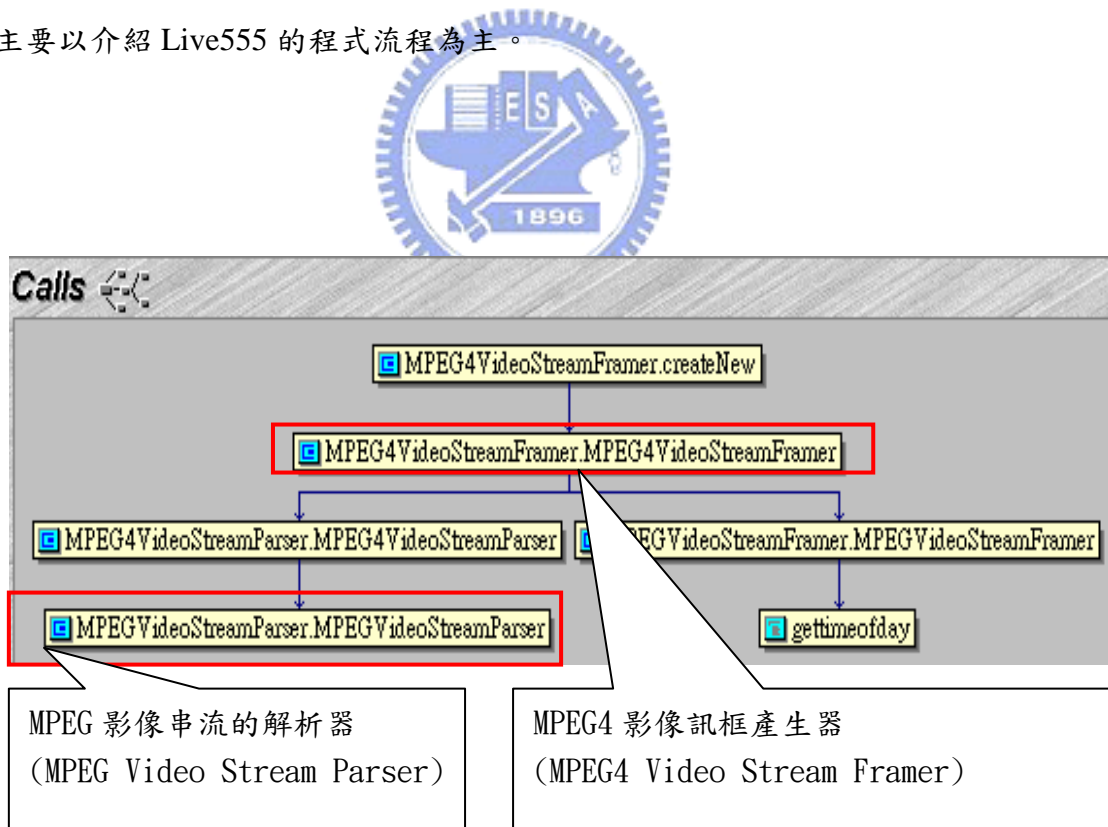
```

fread() 由 fFid 指標所指的地址每次讀取 1 個 record(1 Byte)，共需讀取『fMaxSize』次(fMaxSize 為 record 的個數)，然後把媒體資料讀到 fTo 指標所指向的記憶體位置，並且 fread() 會回傳所讀到的位元組數目給 fFrameSize。

3.2.2.2 位元流解析器/訊框產生器(MPEG4VideoStreamFramer.createNew)

在 3.2.2.1 小節中，我們呼叫了 `ByteStreamFileSource.createNew()` 來完成 MPEG4 檔案的開啟以及讀取，接下來必須由 MPEG4 elementary stream 中取得訊框資料，Live555 這部分的功能是由 `MPEG4VideoStreamFramer::createNew()` 函數來完成的。

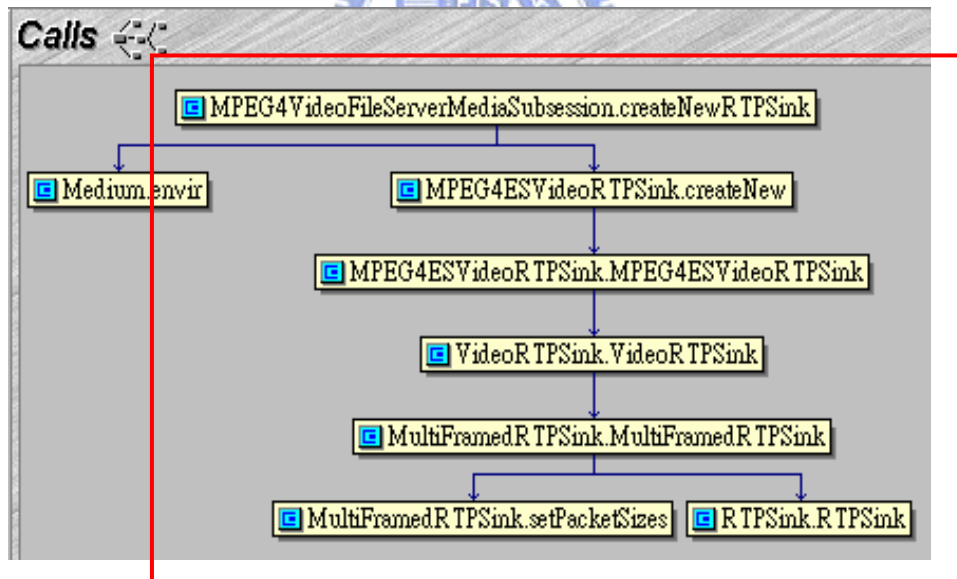
`MPEG4VideoStreamFramer` 的功用就像是一個 filter，功用是從 elementary stream 中找出 Visual Object Sequence (VS) Header + Visual Object (VO) Header + Video Object Layer (VOL) Header、Group of VOP (GOV) Header 以及 VOP frame 的資訊(利用 `MPEG4VideoStreamParser`)，以便可以從中取得我們要的訊框資料，其函數呼叫關係圖如下所示，而關於 `MPEG4VideoStreamParser` 的解析細節我們在第四章會詳細說明，在此主要以介紹 Live555 的程式流程為主。



3.2.3 影像訊框的組成：createNewRTPSink()

在讀取MPEG-4 elementary stream取得訊框之後，接下來便進入封裝演算法(Packetization Algorithm)的部分。串流伺服器的封裝演算法必須能夠依照RFC的標準來進行封包的切割與包裝。在切割方面以MPEG-1/2來說便要參考RFC2250，MPEG-4來說便是RFC3016;而在Payload的包裝上便是替Payload加入RTP表頭(RFC3550)以組成一個RTP封包。(註.RTP表頭的功用會在3.2.5.1小節說明)

因此接下來在createNewRTPSink()函數中，我們主要便會看到關於封包演算法以及RTP表頭實作的程式碼部分，如下面的函數呼叫圖所示，紅框中所包含的函數均是屬於封包演算法的一部份。



(註.Sink 以及 Source 的涵意，在 LIVE555 程式中看到類別/函數名稱掛上『Source』字樣時，代表該類別/函數有負責資料的過濾(filter)，『Sink』則是最後這個資料過濾的接收端，資料傳輸鍊(Data pass chain)示意圖如下所示。

```
'source1' -> 'source2' (a filter) -> 'source3' (a filter) -> 'sink'
```

而有了 MPEG4 的訊框之後，我們便要把若干個訊框打包在一個封包之中，要注意的是不能超過 MTU 的限制。而在這些封包的 payload 之前，我們還需替每一個 RTP packet 加入 RTP 表頭（如下圖 3.5 所示），這些事情都是由上圖紅框內的函數共同完成的，而關於封裝演算法的核心程式部分則是在 MultiFramedRTPSink.cpp 中可以看到。

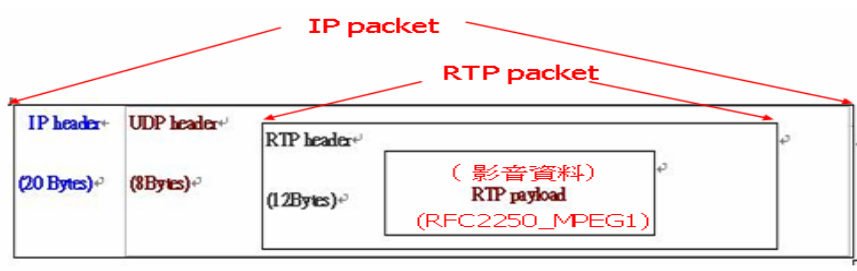
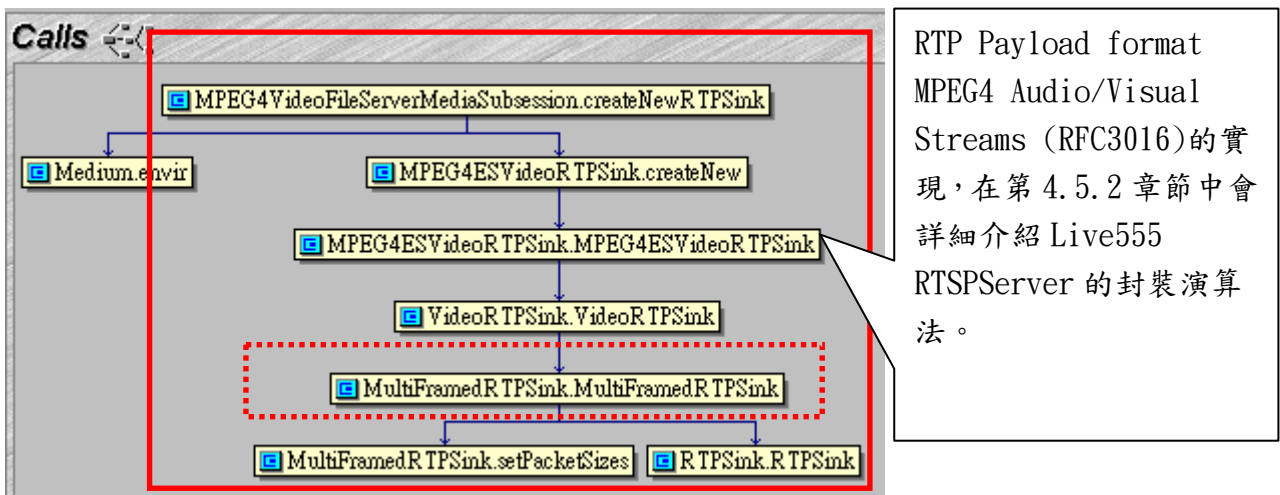


圖 3.5 IP packet consistence

3.2.4 影像封包的切割與包裝：MultiFramedRTPSink()

接下來要介紹的是 MultiFramedRTPSink.cpp 程式，其程式主要負責的是進行訊框的打包、溢位訊框資料的處理、加入 RTP 表頭...等等，歸納起來就是一個 RTP 封包的建立以及傳送均在此程式中完成。



本程式的主要目的概述如下面六點：

1. 設定封包大小(setPacketSizes)。

2. 取得下一個訊框 (frame)。

3. 訊框打包工作(pack)。

4. 設定 RTP header 內的欄位資訊：

RTP version、Marker bit、Payload type、Sequence Number、TimeStamp 和 SSRC 欄位。

5. 在 payload 之前加入 RTP header:

組成一個『RTP packet』。

6. 呼叫 UDP socket：

目的是把 RTP packet 遞交給作業系統內部的 UDP/IP 協定核心 (Transport/Network layer) 以便在 RTP packet 之前加入 UDP 以及 IP header，組成一個可在網路上傳送的 IP packet。



接下來我們簡單由函數呼叫流程圖來進行 MultiFramedRTPSink.cpp 程式的解說，因為程式是由 MultiFramedRTPSink 建構子開始執行的，故接下來便由該建構子開始解說：

MultiFramedRTPSink 建構子目的~

1. 預設封包大小為 1448Bytes(但允許使用者 setPacketSizes()改變之)。

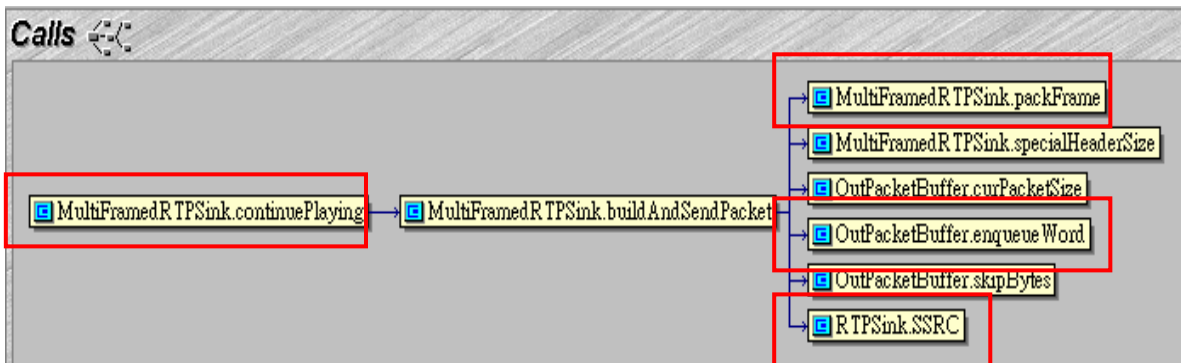
2. 設定一個 OutPacketBuffer 建構子(輸出封包的緩衝區，預設為 60000 Bytes)。

建構子執行完後，程式會由 continuePlaying()開始向下呼叫 (紅框部分所示)，一

直呼叫到最後的 `setupDatagramSocket()` 函數，過程如下面三個步驟所示。

步驟一：開始建立封包以及打包訊框

`continuePlaying()` -> `buildAndSendPacket()` -> `packFrame()`



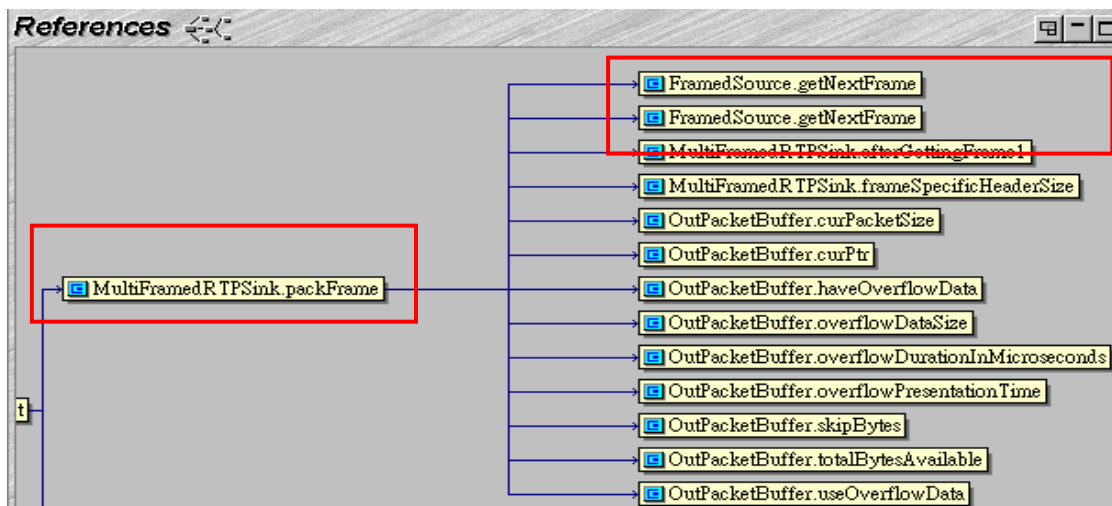
1.1 `packFrame()`：負責把訊框打包進一個封包之中。

1.2 `enqueueWord()`：用 `enqueue()` 把 RTP 標頭資料加入在封包的前端

(參考 `MediaSink.cpp` 的 `OutPacketBuffer` 資料結構)。

1.3 `SSRC()`：Synchronization Source (SSRC) ID 設置多媒體資料的來源值。

1.4 `packFrame()`：讀取下一個訊框來源並打包，同時檢查封包是否已經溢位。



步驟二：packFrame()->afterGettingFrame1()

1.先記錄系統目前的時間

2.確認封包是否仍可塞進訊框(isTooBigForAPacket())

2.1 封包大小如果未超過 MTU 的話,則程式繼續執行打包工作(packframe())。

2.2 封包大小如果超過 MTU 的話,則進入下面流程:

2.2.1 塞滿訊框的封包呼叫 sendPacketIfNecessary(), 進行封包傳送の確認, 在這邊會

有一個加速封包溢位資料處理的流程。

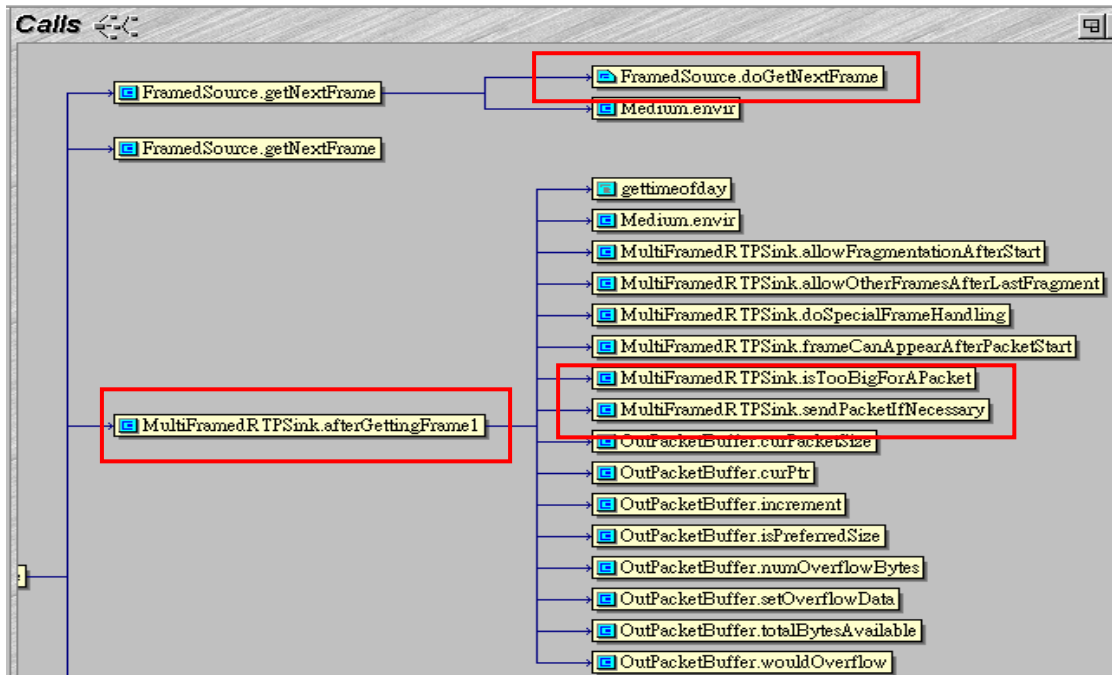
2.2.2 MultiFramedRTPSink::isTooBigForAPacket()~

把超過 MTU 限制的 frame 部分給截斷, 並把封包溢位的長度、每一個封包的起始地址以及每一個封包的偏移量資訊都給記錄下來, 因為 Live555 主要是以指標來指向下一個封包要由 frame 的哪一個位元組開始打包, 所以上述三個變數是很重要的指標位移資訊。而在 MTU 限制下的封包便可呼叫 UDP socket API 傳送之。

2.2.3 MultiFramedRTPSink::sendPacketIfNecessary()~

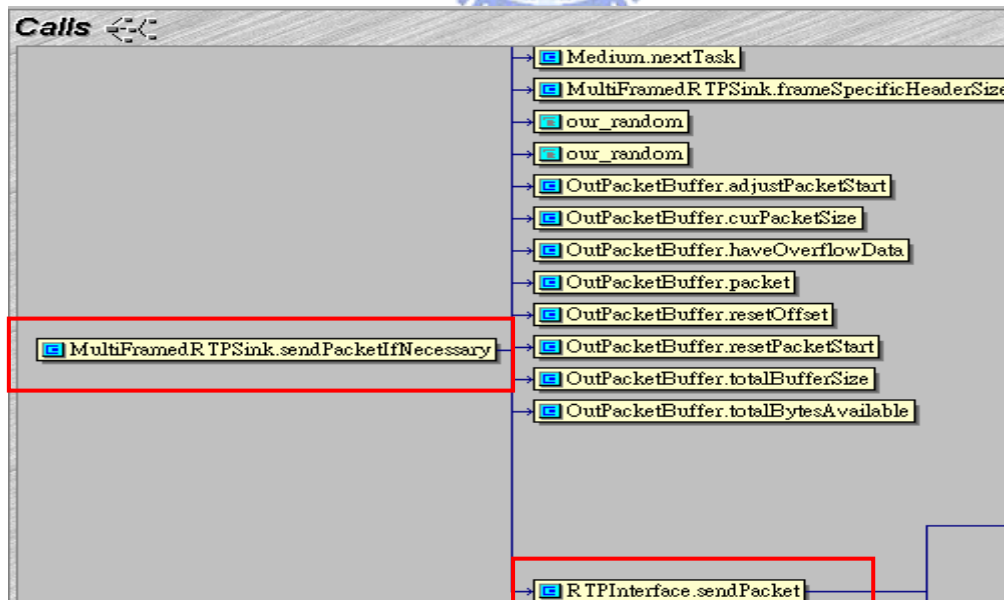
截斷的(frame)部分設置給下一個封包, 這意思便是把下一個封包的起始點指標指向 frame 被截斷的位元組, 同樣由上述步驟 2.1 開始執行, 但如果已經讀取到一個 VOP 的結束(VOP header 中的 Market bit 會標示為 1), 即使目前封包裡面的 frame 未超過 MTU, 程式會馬上呼叫 UDP socket API 傳送之, 而不會把下一個 VOP 的資料繼續打包進來, 這便是整個 Live555 封裝演算法的簡要說明。

(註. Live555 封裝演算法的程式碼解說在第四章會更詳細地介紹說明)



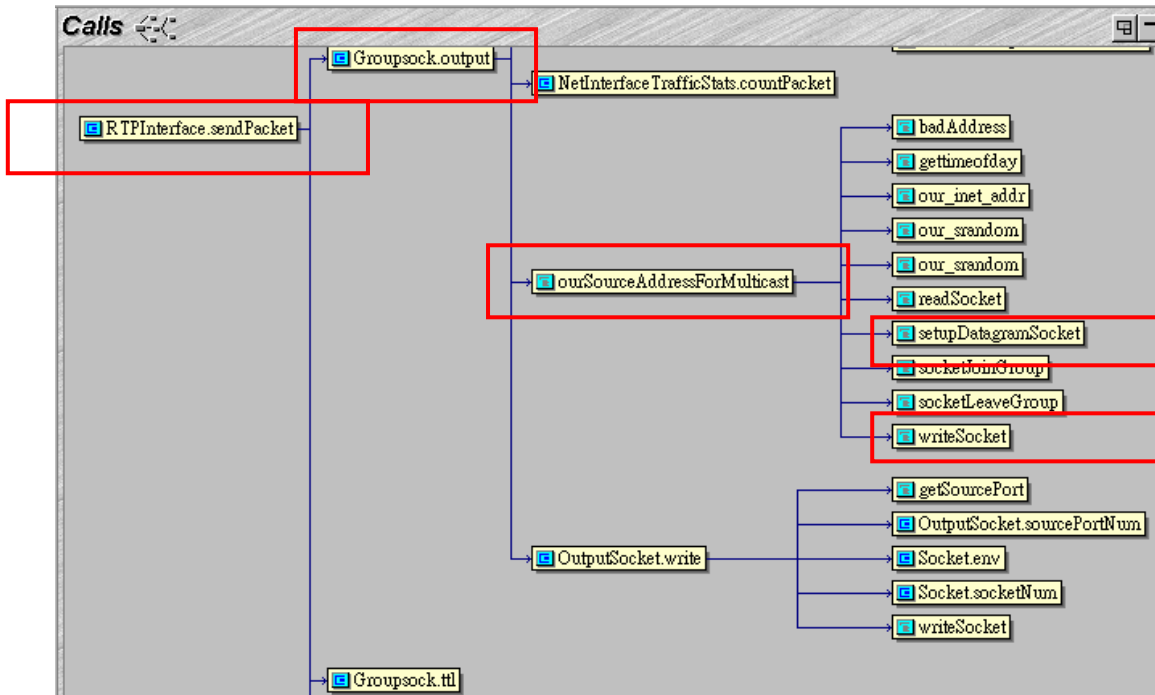
步驟三：當封包已裝完訊框並符合 2.2.2~2.2.3 的要求，則把此封包給傳送出去。

sendPacketIfNecessary()->sendPacket() :



封包的傳送是呼叫 UDP socket API，開始進行 UDP/RTP 資料的傳送。

sendPacket()->output()->ourSourceAddressForMulticast()->setupDatagramSocket()。



經 DatagramSocket()建立一個 UDP socket 之後，便可以呼叫 writeSocket()也就是 sendto()把封包給傳送出去。



3.2.5 RTP Payload format for MPEG4 (createNewRTPSink())

接下來要討論的是RFC 3016所提出了MPEG-4 的影音封包與RTP的封裝方式，首先在3.2.5.1小節會簡單介紹一下RTP協定；在3.2.5.2小節則會介紹為什麼還需要有RTP MPEG4 payload format的存在。

3.2.5.1 RTP (Real Time Protocol) [6]

因為多媒體串流傳輸要求『即時性』的特性，故我們捨棄了 TCP 而改採用 UDP(User

datagram protocol)協定，這在我們前面第二章中已經提過了。而當封包遺失或未依序抵達時，UDP 將不採取任何動作，因此為了解決這個問題，我們必須在 UDP 之上再加入 RTP 協定。

RTP雖無法保證QoS或是提供更可靠的傳輸，但是提供了大部份即時資料傳輸的基本功能。如下圖3.6所示，RTP協定主要的特點是提供**封包順序號碼 (Sequence Number)**來記錄傳送封包的順序，封包序號可用在接收端偵測封包遺失的情況(配合RTCP的話便可以提供一些QoS的參數)，並且在接收端的封包緩衝區(buffer)便可以用封包序號對未依序的抵達封包做重排(reorder)的動作，以確保影音順序的正確性。

而**時間戳記 (Timestamp)**可以量得從傳送端到接收端的延遲時間，並且透過此時間戳記來使MPEG-4視訊和其他多媒體資料流作同步的動作；另外有**marker bit**主要用來指出資料流的邊界，對視訊來說即為一張畫面frame的結束，而當串流伺服器由MPEG4影像流中的VOP header解析到Marker bit為1時，便知道目前的VOP frame已經結束。

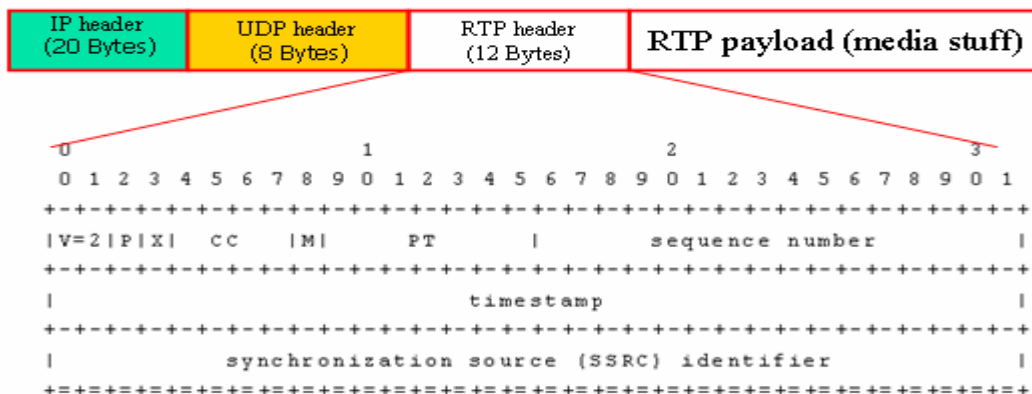


圖 3.6 RTP header

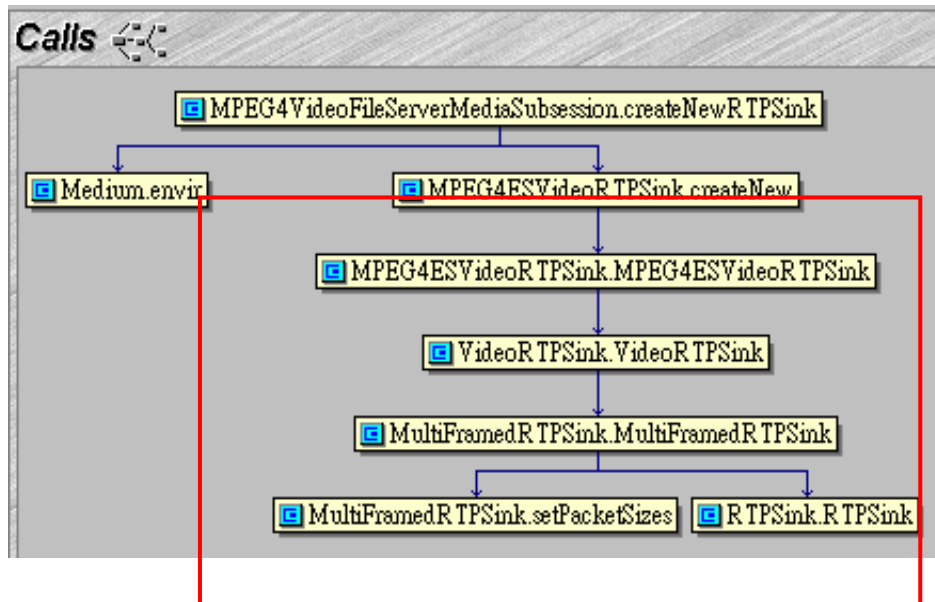
- Version (V)，代表 RTP 的版本，2bits。
- Padding(P)，在加密時需要特定長度時使用，表示總共有多少位元組的 padding，1bit。
- Extension(X)，定義 RTP 的延伸標頭是否存在，通常會有額外文件定義，1bit。
- CSRC count(CC)，代表 CSRC 的個數，4bits。
- Mark(M)，設為 1 時指出資料流的邊界，1 bit。
- Payload type(PT)，此部份規範在 RFC2327 的文件中，主要是說明封包中的資料格式(Payload Format)，每種不同資料格式都有對應的文件說明如何使用 RTP 來傳遞，7bits。
- Sequence Number，每一個 RTP 封包都會有一個 sequence number，每傳送一個封包就會累加一，可以計算遺失封包數、調整封包順序等作用，16bits。
- Timestamp，這部份指出 RTP 封包內 VOP 的取樣時間(Sampling Instance)，同一個 VOP 的數個封包此值均會相同，主要作為同步的使用。在安全加密方面也是提供時間戳記(Timestamp) 隨機亂數的初始值設定，32bits。
- SSRC，識別同步來源(Synchronization Source)，而同步來源是指一個 RTP 資料來源，不同的資料流會有個別的 SSRC 值，此值是以隨機亂數設定，32bits。

3.2.5.2 RTP payload format for MPEG4 (MPEG4ESVideoRTPSink.cpp)

在前面3.4.3小節中我們提到createNewRTPSink()函數負責對影像封包的訊框做『切割與包裝』的任務，而在封包包裝的任務中，因為必須考慮未來MPEG-4影像封包在網路上可能會遺失進而導致接收端無法順利解碼的情形發生(如果該遺失封包中包含了重要的解碼VOP header資訊的話)，因此在封包包裝的任務中就必須先經過RFC 3016的處

理，進而在不同的網路使用環境中取得封包使用率以及錯誤恢復能力的最佳化。

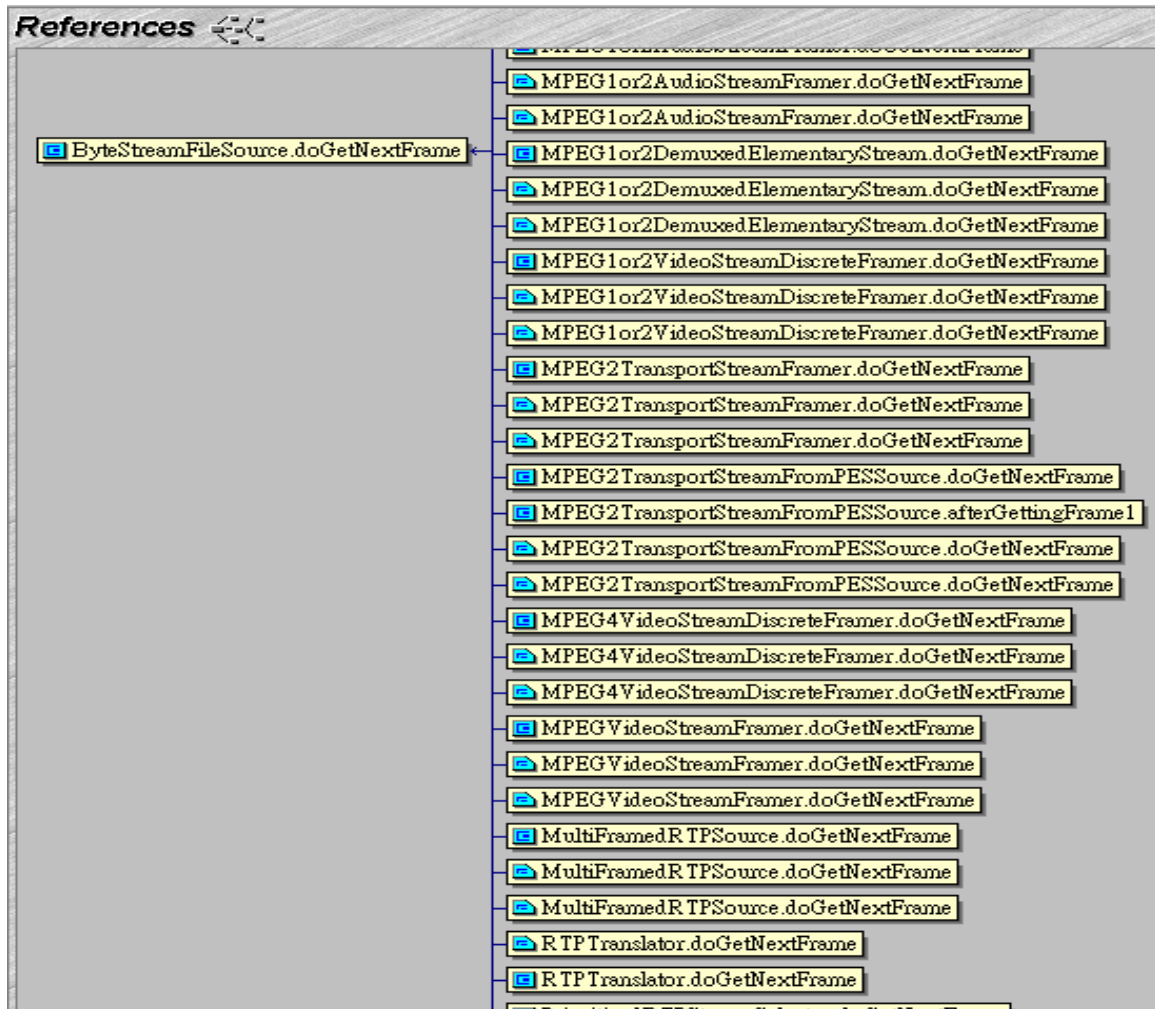
因此當createNewStreamSource()函數完成了MPEG-4 基本流的解析以及訊框組成後，在進行訊框打包之前，MPEG4VideoFileServerMediaSubsession建構子還會呼叫MPEG4ESVideoRTPSink.createNew函數來動態配置一個MPEG4ESVideoRTPSink建構子，該建構子會再去呼叫如VideoRTPSink建構子以及MultiFramedRTPSink建構子來完成RTP MPEG-4 的封裝演算法。



3.2.6 取得各種影音格式的訊框(doGetNextFrame)

如下面的程式呼叫方塊圖所示，各種影音格式在打包(pack)的時候均需使用doGetNextFrame()來取得『特定』影音格式的訊框資料，如我們在前面 3.2.2.1 章節(ByteStreamFileSource.createNew)中所提過的，doGetNextFrame()主要會呼叫 fread()來讀

取『特定』媒體資料流，也就是進行特定媒體格式流的解析(parser)。



如上所述，訊框來源的取得 (doGetNextFrame()) 會依照媒體影音格式的不同而有不同的函數實現方式，舉例來說下面紅框內的 AC3 聲音格式要取得其 audio 的訊框便需先呼叫 AC3AudioStreamFramer.cpp(AC3 訊框產生器)特定的 doGetNextFrame()函數;同樣地，串流伺服器欲串流 AC3 檔案也會需要一個 AC3 解析器(AC3 Bitstream Parser)方可讀取 AC3 格式的 Bitstream，如下紅框所示。

```
void AC3AudioStreamFramer::doGetNextFrame() {  
    fParser->registerReadInterest(fTo, fMaxSize);  
    parseNextFrame();  
}
```


所以原則上 AC3 檔案或是其他檔案格式媒體的串流過程均同於我們上述的 MPEG4

串流過程，也就是依下列六個主要的流程步驟：

Step1.開啟並讀取檔案: fopen()、fread()。

Step2.讀取特定檔案格式的資料流: specific 『Parser』。

Step3.把資料流組成訊框: specific 『Framer』。

Step4.把若干個訊框打包進單一封包內: packframe()。

Step5.在每個封包內的 payload 前插入 RTP header: by enqueue()/insert()。

Step6.經 UDP socket 傳送 RTP 封包: by sendPacketIfNecessary()。



到目前為止我們已經把一個串流伺服器的『連線建立的標準程序』、『溝通協調』以及『封包包裝及傳送』以 Live555 的程式流程論述完畢，最後在這章節我們以這五行程式的功能解說作為結尾。

1. // Begin by setting up our usage environment:

```
TaskScheduler* scheduler = BasicTaskScheduler::createNew();
```

#2. // Begin by setting up our usage environment:

```
env = BasicUsageEnvironment::createNew(*scheduler);
```

3. //Create the RTSP server:

```
RTSPServer* rtspServer = RTSPServer::createNew(*env, 8554, authDB, 45);
```

4 /*Set up each of the possible streams that can be served by the RTSP server. Each

such stream is implemented using a "ServerMediaSession" object, plus one or more

"ServerMediaSubsession" objects for each audio/video substream.*/

```
ServerMediaSession* sms = ServerMediaSession::createNew(*env, streamName,  
streamName,descriptionString);
```

5. /*from 『 createNew(*env, **inputFileName**, reuseFirstSource); 』 , the 5 tasks will be

done as below ~

1.fopen()- 『 inputFileName 』 (point to the Media) is being opened

2.parser() -by recognizing its bitstream and specific Marker

3.framer()-A filter that breaks up an MPEG-4 video elementary stream into

frames

4.pack() -with different package(ex: RFC3016 for MPEG-4)

5.sendPacketIfNecessary() -call UDP socket to send the RTP packet

*/

```
sms->addSubsession(MPEG4VideoFileServerMediaSubsession::createNew
```

```
(*env, inputFileName, reuseFirstSource));
```

第四章 串流伺服器實作細節

在前面三章我們已經以Live555為例把串流伺服器的『連線建立的標準程序』、『溝通協調』以及『封包包裝及傳送』三個部分的程式流程解說完畢，接下來我們要考慮的是幾個串流伺服器實作上需要特別注意的細節部分。

4.1 串流伺服器的多工

串流伺服器的多工是指一個串流伺服器可以在同一時間內服務多個用戶端的連線需求，也就是可以同時進行多個影音媒體的串流。要達成多工我們有下列三種方式去實現，第一種是使用 fork 函數，第二種則是使用 select，第三種是 multi-thread。在此我們針對自由開放軟體 ffmpeg 以及 Live RTSPServer 來進行第一種與第二種的差異討論。

4.1.1 Multi-Process, Concurrent Servers

如下圖 4.1 所示，**process-based** 的伺服器多工是指主行程(master process)只負責接待但不負責服務，服務的工作是交由子行程(slave process)來負責，ffmpeg 的 ffmpeg 便是屬於此種架構。原則上這種類型的串流伺服器會有一個主行程專責監聽主機上的 socket 有無 client 的連線需求，有的話主行程會替每個連線建立子行程來負責後續的封包傳送工作，關鍵點是使用 fork() 函數來完成多工伺服器的建立(請參考下方 fork 的虛擬碼範例)。

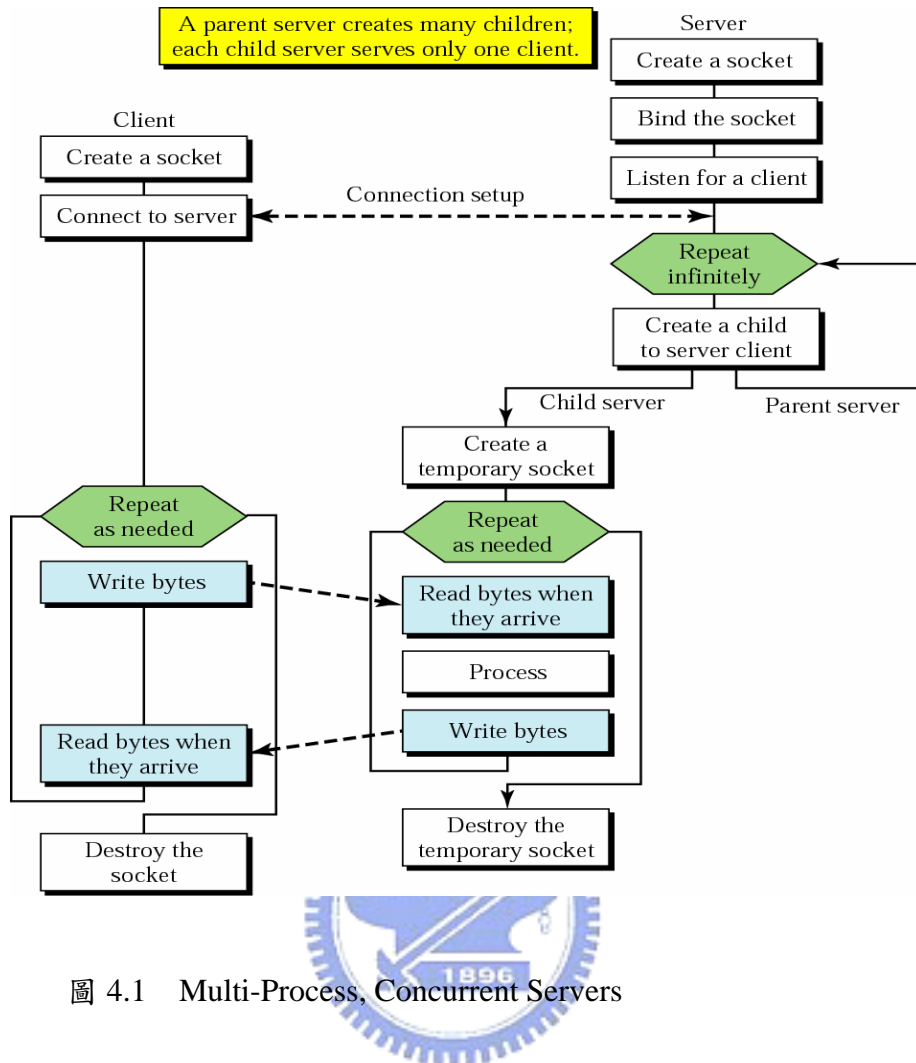


圖 4.1 Multi-Process, Concurrent Servers

4.1.1.1 Fork()說明

fork()通常是配合 Blocking socket 來使用，fork()會在程式中產生一個新的子行程，並複製主行程的資料與堆疊空間，繼承主行程的 UID、GID、環境變數、已開啟的檔案代碼、工作目錄和資源限制等。如果 fork 成功則在主行程會傳回新建立的子行程代碼 (PID);若子行程成功會傳回 0，若失敗則傳回-1，以下簡要列出如何使用 fork()的虛擬碼(pseudo code)。

4.1.1.2 Fork() 虛擬碼範例

```
signal(SIGCHLD, SIG_IGN); //zombie child reaper

int sockfd = socket(...);

bind(sockfd, ...);

listen(sockfd, ...);

while(1){

connfd = accept(sockfd, ...);

/*master process*/

if (fork() >0){

    /*關閉主行程的 connected sockfd，只留下 listening sockfd */

    close(connfd);

    continue;    //fork ok -still in while loop }

/*slave process*/

    /*關掉子行程的 listening sockfd，只留下 connected sockfd*/

    close(sockfd, ...)

    /*data processing*/

    <TODO : RTSP negotiation and send RTP packet>

    /*ex: recvfrom(connfd, buf, sizeof(buf)); */

    /*收到用戶端要求結束連線訊息，伺服器便結束子行程的 socket */

    close(connfd); exit(0); } }
```



4.1.2 Single-Process (Event-Based)， Concurrent Servers

如下圖 4.2 所示，Single-Process 的意思是主行程(master process)需同時擔任接待以及服務的工作，因為我們只有一個行程(Single process)。Live 的 RTSPServer 便是屬於此

種串流多工架構，這種串流伺服器會由這個單一的行程來控制多個 sockets 的 I/O。如下

圖 4.2 所示，listening socket 接收來自 client 端的連線並以 accept() 建立了 connected socket

後，負責連線的後續工作。

通常這種 Non-blocking socket 會搭配 select() 函數來使用，select() 目的是事先詢問各

個 socket 的狀態，判斷目前 socket 是否已經可以讀取，確定有之後我們再去讀取，避免

類似 polling I/O 浪費 CPU 資源。

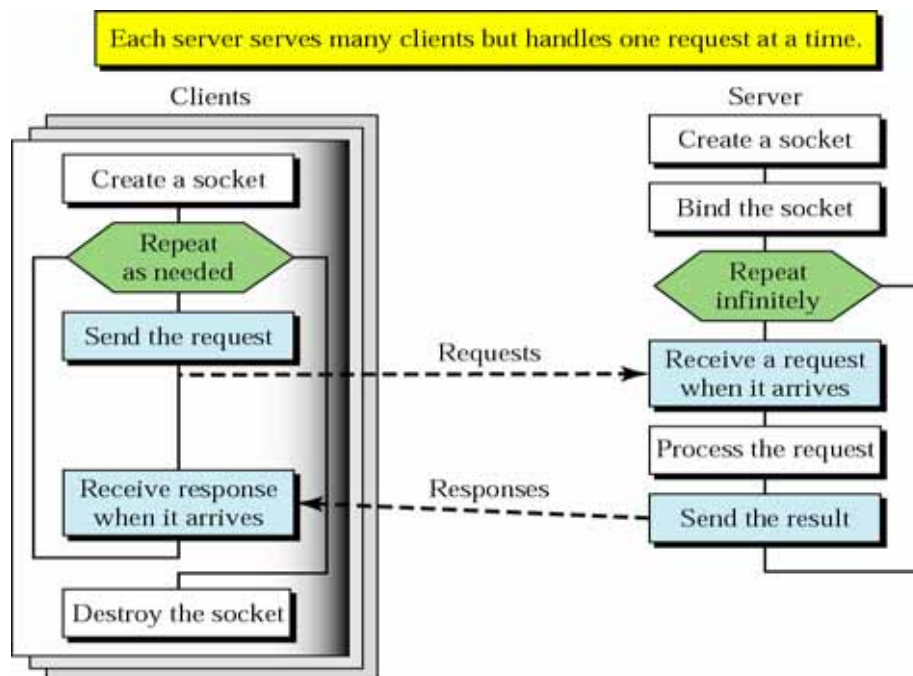


圖 4.2(a) Single-Process (Event-Triggered) , Concurrent Servers

(註)欲讓原本預設為 blocking 的 socket 變成 Non-blocking socket 的設定如下所示:

```
int curFlags = fcntl(newSocket, F_GETFL, 0);
if (fcntl(newSocket, F_SETFL, curFlags|O_NONBLOCK) < 0) {
```

4.1.2.1 Select()說明

select 這個 system call 可以告知 kernel 它想監控的 sockfd，當 sockfd 所 index 的 socket 有事件發生時，kernel 會叫醒該程式來處理之，可以監控的事件有三種，分別是『讀取』、『寫入』以及『例外』，select 的敘述以及 Single-Process (Event-Triggered) 流程圖如下所示：

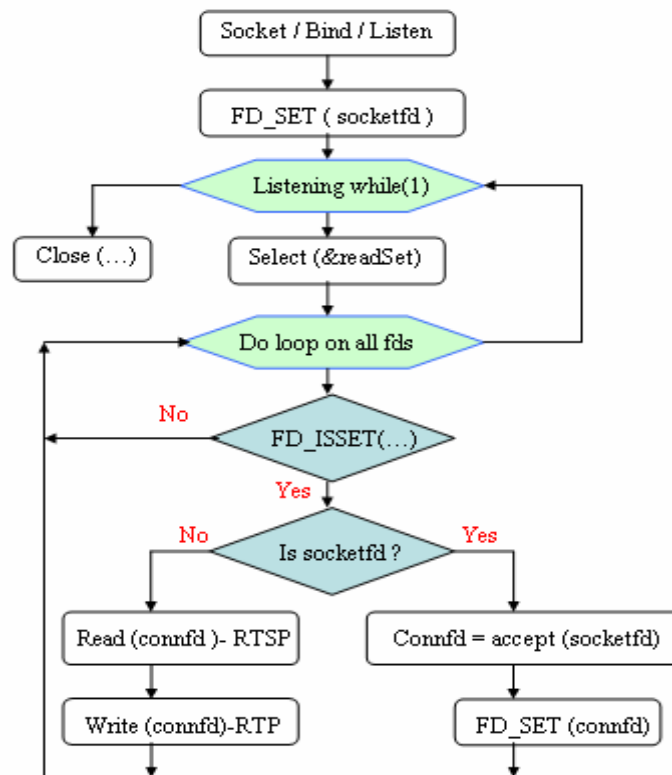


圖 4.2(b) Single-Process with select flowchart

(使用 select 來達成單一行程的多工步驟如下)

1. 針對欲監控的 fd_set 宣告結構變數，三種情形：讀、寫、例外的集合視需要宣告之。
2. 宣告 readSet 以及 freadSet，並使用 FZ_ZERO 把結構的 bit array 內容清為 0。

3.使用 FD_SET 把我們的 Listen sockfd 加入 fReadSet 的集合中，當進入迴圈之後，把 fReadSet 的值設定給 readSet。

4.呼叫 select()，使行程 block 住直到所監控的 sockfd 有動靜(由 kernel 主動喚醒程式)，程式碼如下所示。

```
int selectResult = select(fMaxNumSockets, &readSet, NULL, NULL,
                        &tv_timeToDelay);
```

Live RTSPServer 只把先前宣告好的 readSet 放到 select 的第二個參數中，表示我們只等待『讀取』的事件，因此我們把參數三及參數四 *writeSet、*exceptSet 都設為 NULL；第五個參數在 Live RTSPServer 的程式中是設為一百萬秒(11.5 天)，原因是 select 所 block 的時間如果超過一百萬秒(此段時間沒有用戶連線)，則 select 函數會失效。

故最佳的設定便是設成一百萬秒，如果這段時間都沒有用戶連線，Select()便會回傳一次結果，如此 Timer 便可以重新計數而不會導致 Select()失效；同時 Live555 也替每一個 connected socket 設定了一個『通關』的函數 timeToNextAlarm()，目的是使伺服器可以同時接收新的用戶連線並且對舊有的用戶連線進行封包的傳送，此地方的設計需要特別注意。

5.直到所監控的 sockfd 有動靜，行程就會從 select loop 中跳出，因 select 回傳的是符合監控狀況的 sockfd 數目，因此尚須由 FZ_ISSET 巨集來判斷是哪個 sockfd 有動靜。

6. 使用迴圈並配合 FZ_ISSET 來檢查是 readSet 中的那個 connected sockfd 有反應，這

表示 kernel 所監控的 readSet 有讀取事件發生，也就是有封包抵達 NIC(Network InterfaceCard，網路介面卡)。

7.當完成 TCP 3WHS 建立完 RTSP 的 connected socket 後，我們便可以使用 BSD socket API 的 `recv()` 來接收送到 Server socket 上的 TCP/RTSP 封包;當 RTSP connected socket 收到 RTSP PLAY 信令後，Server 便會建立 UDP socket 以作為 RTP 封包傳送之用。

4.1.2.2 Select() 虛擬碼範例

```
int sockfd = socket(...);
bind(sockfd, ...);
int curFlags = fcntl(listen_fd, F_GETFL, 0); //(make Socket Non-Blocking )
fcntl(newSocket, F_SETFL, curFlags|O_NONBLOCK);
listen(sockfd, ...);
fd_set fReadSet;

FD_ZERO(&fReadSet);
FD_SET(sockfd, & fReadSet);

int max_fd =socketfd;

while(1) {
fd_set readSet=fReadSet;

/*只等待 read 的事件，故*writefds，*exceptfds 都設為 NULL*/

int selectResult = select(max_fd + 1, &readfds, NULL, NULL,
&tv_timeToDelay);
```

```

/*Do loop on all fds to look for RTSP connected socket or accept a new connection request.*/

for (int i = 0; i <= max_fd; i++) {
    if (FD_ISSET(i, &readSet) == 1)
    {
        if (i == socketfd) //new connection
        {
            connfd = accept(socketfd...);
            FD_SET(connfd, &readSet);
            if (connfd > max_fd)
                max_fd = connfd;
        }
        else //RTSP connected socket
        {
            /*data processing*/
            /*ex: recvfrom(connfd, buf, sizeof(buf)); */
            <TODO : RTSP negotiation and creates UDP socket for RTP > }
        }
    }
}

//end if FD_ISSET
}

//end loop on all fds

//For each UDP socket
<TODO : send RTP packet by calling BSD socket API sendto() >
.....
}

} // end while-loop

```



4.1.3 多工串流方式的效能比較 4.1.3.1 Socket API 的作業模式比較

	優點	缺點	附註
阻攔模式 (Blocking)	程式設計最簡單， 適用於程式功能不	生產量(throughput) /效率較差	為達多工通常需配 合fork使用

	多的情形下		
非阻攔模式 (Non-Blocking)	不會浪費無謂的等待時間，生產量/效率較佳	程式設計困難，因為必須決定等待的時間。等待時間太長，程式會沒有效率；太短則類似輪詢式I/O會吸乾CPU資源	為克服其缺點，故通常搭配select使用，以簡化程式設計
非同步模式 (WSAAsyncSelect)	效率高且方便使用	與BSD socket不相容	僅限於Winsock使用

表4.1 Socket API的作業模式比較

4.1.3.2 fork 與 select 分析比較

項目	特色	需小心的地方
fork	程式執行模式清楚直接	 <ol style="list-style-type: none"> 1. IPC 問題(IPC, InterProcess Communication), 比如說行程間的記憶體共享以及 semaphores 的使用。 2. IPC 在各種作業系統平台上並沒有一致性的標準，故可能導致可移植性降低。 2. 增加行程額外的控制負擔(如果行程間要溝通的話) 3. 子行程的記憶體回收問題(zombie children)
select	<ol style="list-style-type: none"> 1. 單一程式邏輯控制流程 2. 程式除錯上比較方便 3. 系統可移植性較佳，因為 UNIX 以及 POSIX 皆採用此種方法 	程式邏輯控制流程的設計複雜度比 fork 來的困難

表4.2 fork與select分析比較表

4.1.4 串流伺服器多工結論

在 select 單一行程的方式中，串流伺服器利用陣列(fd_set 所宣告的結構變數)來儲存各個 socketfd 的監控情形，而儲存每個 socket 監控狀況所需的記憶體約為 26 Bytes，故 1KB 的記憶體估算下來可以應付約四十個 Client 端的連線所需;而用 fork()分叉出一個子行程來處理一個 Client 的連線所消耗的記憶體即不只 1KB 了，並且 fork()還有 context switch 的 overhead。

因此，雖然我們在 desktop PC 上開發串流伺服器程式可以選擇 select 或是 fork 的伺服器多工型式，但如果考慮以後要把串流伺服器程式移植在嵌入式系統(Embedded system)上如 DVR(digital video recorder)、Home surveillance (居家安全) ..等應用上，考慮到**功耗**以及記憶體等系統資源的應用，則應選擇 select()架構來發展較佳，這也是在高效能的串流伺服器中常常會見到 select 的原因。

然而 select 的使用也是有一個上限，當連線資料高達幾千筆的時候(也就是說同一時間查詢幾千個 socket 的狀況時)，伺服器有可能因而停止運作，實際的上限值數目範圍須由實際測試結果得知。

4.2 RTSP 訊令之傳輸以及辨認

在前面 2.1 章節(連線建立的標準程序)中我們已經瞭解到 RTSP 是用來在 IP 網路上有效率的傳送連續性媒體串流的協定，並且必須基於連接導向的 TCP 協定之上來傳送之，如下圖 4.3 所示之。

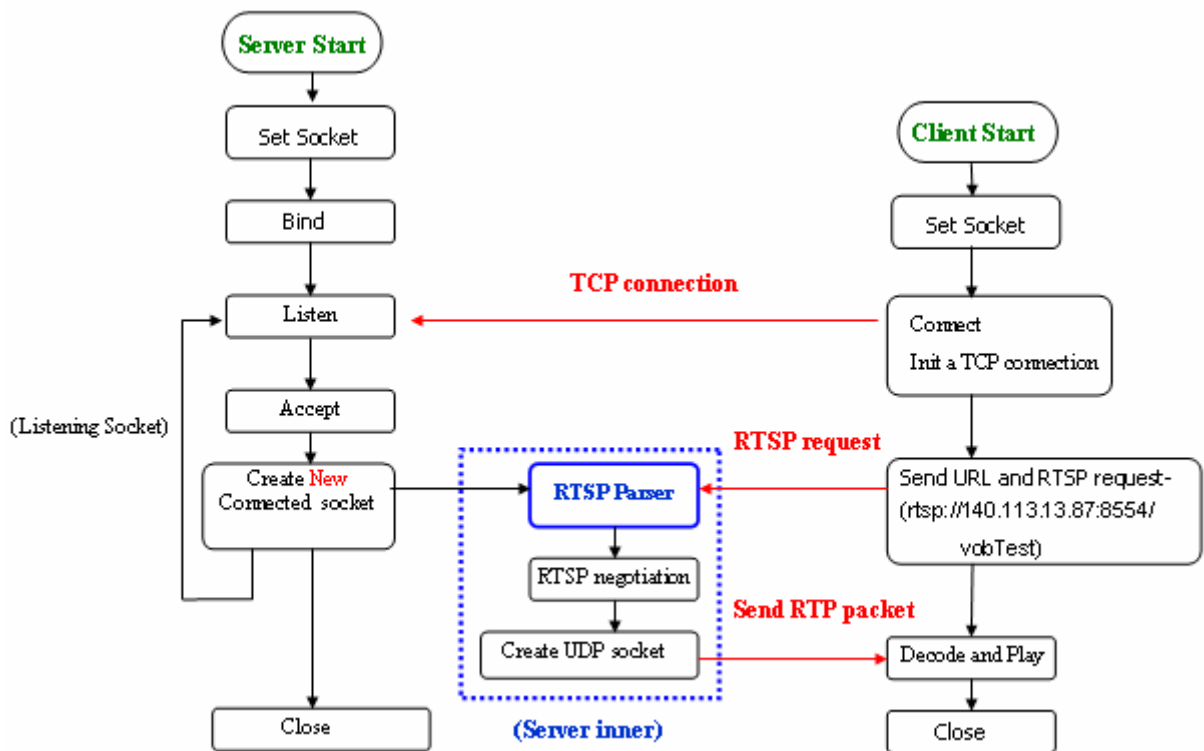
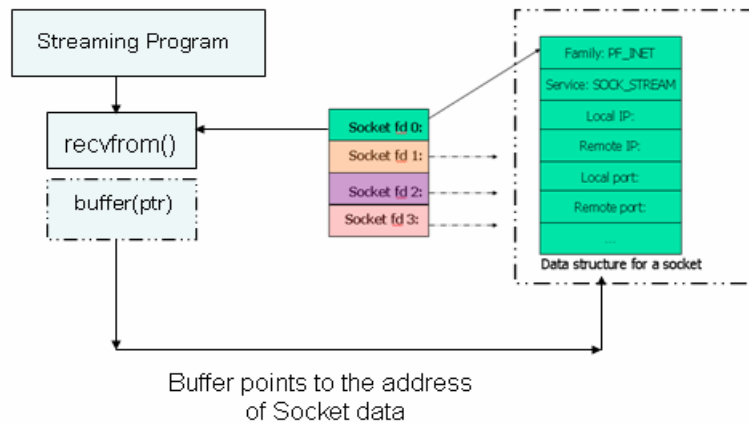


圖 4.3 RTSP Parser chart

而 RTSP Parser 的資料來源如下紅框所示，如前面 2.1.1 章節(socket)提到的，socketfd 會索引到一個『socket』的資料結構，而 recvfrom()函數首先需要透過 socket(socketfd) 得到『socket』的資料結構，並且每當有封包傳進主機上的網路卡時，kernel 會把封包指引到這個『socket』資料結構中，這也就是 recvfrom()的第一個引數的功用。

recvfrom()的第二個引數 buffer 便是一個指標，recvfrom()會把 socket 所接收到的資料複製一份到 buffer 所指的地址。

```
bytesRead = recvfrom(socket, (char*)buffer, bufferSize, 0,
                    (struct sockaddr*)&fromAddress,
                    &addressSize);
```



接下來主要用 Call by address 把從 socket 得到的封包資料傳到 unsigned char fBuffer[10000]之中。

```
int bytesRead = readSocket(envir(), fClientSocket, ptr, bytesLeft, dummy);
unsigned char* ptr = fBuffer;
```

接下來我們來看要如何實際解析用戶端所傳過來的 RTSP 訊令，以下面的標準 RTSP 訊令為例，一開始由 socket 接收而傳送到 fBuffer 到的文字字串便會是「OPTIONS」，而程式此時便會呼叫 parseRequestString()函數來處理之。

```
Real Time Streaming Protocol
  OPTIONS rtsp://140.113.13.97:8554/mpeg1or2AudiovideoTest RTSP/1.0\r\n
    Method: OPTIONS
    URL: rtsp://140.113.13.97:8554/mpeg1or2AudiovideoTest
    CSeq: 1\r\n
    User-Agent: VLC Media Player (LIVE.COM Streaming Media v2004.11.11)\r\n
    \r\n
```

```

RTSPClientSession
~RTSPClientSession
reclaimStreamStates
incomingRequestHandler
incomingRequestHandler1
#ifdef WIN32_WCE
dateHeader
#else
#endif
const
00650:
00651: //513 # unsigned char* ptr = pBuffer;
00652: if (!parseRequestString((char*)pBuffer, totalBytes,
00653: cmdName, sizeof cmdName,
00654: urlPreSuffix, sizeof urlPreSuffix,
00655: urlSuffix, sizeof urlSuffix,
00656: cseq, sizeof cseq)) {
00657:

```

Point to RTSP source

```

parseRangeHeader
parseScaleHeader
RTSPServer
RTSPClientSession
handleCmd_PLAY
handleCmd_PAUSE
parseAuthorizationHeader
RTSPServer
RTSPClientSession
authenticationOK
parseRequestString
UserAuthenticationDatabase
UserAuthenticationDatabase
UserAuthenticationDatabase
01588: Boolean
01589: RTSPServer::RTSPClientSession
01590: ::parseRequestString(char const* reqStr,
01591: unsigned reqStrSize,
01592: char* resultCmdName,
01593: unsigned resultCmdNameMaxSize,
01594: char* resultURLPreSuffix,
01595: unsigned resultURLPreSuffixMaxSize,
01596: char* resultURLSuffix,
01597: unsigned resultURLSuffixMaxSize,
01598: char* resultCSeq,
01599: unsigned resultCSeqMaxSize) {

```

- Data parsing:
- 1.CmdName
 - 2.URLPreSuffix
 - 3.URLSuffix
 - 4.Cseq

接下來使用 for-loop 來解讀 CmdName 的命令，並且先把解讀出來的 RTSP 信令使用一個指標陣列 resultCmdName 來存放之，最後 parseRequestString() 函數把值返回給 CmdName 變數存放之：



```

RTSPServer.cpp
handleCmd_DESCRIBE
parseTransportHeader
RTSPServer
RTSPClientSession
handleCmd_SETUP
handleCmd_withinSession
handleCmd_TEARDOWN
parseRangeHeader
parseScaleHeader
RTSPServer
RTSPClientSession
handleCmd_PLAY
handleCmd_PAUSE
parseAuthorizationHeader
RTSPServer
RTSPClientSession
authenticationOK
parseRequestString
UserAuthenticationDatabase
UserAuthenticationDatabase
01614:
01615: 2.reqStrSize=socket所讀到Buffer的大小
01616: */
01617:
01618: //OPTIONS rtsp://140.113.13.87:8554/mpeg1or2AudioVideoTest
01619: //for- loop,用來產生resultCmdName的命令 ,[space]ASCII hex=20
01620: for (i = 0; i < resultCmdNameMaxSize-1 && i < reqStrSize; ++i) {
01621: char c = reqStr[i];
01622:
01623: //讀到空白或是 TAB便跳離resultCmdName迴圈
01624:
01625: if (c == ' ' || c == '\t') {
01626: parseSucceeded = True;
01627: break;
01628: } //end if
01629:
01630: //把c讀到字元丟到 resultCmdName所指向的指標陣列
01631: resultCmdName[i] = c;
01632: } //end for
01633: resultCmdName[i] = '\0';
01634: //讀完最後給一個結束字元
01635: if (!parseSucceeded) return False;

```

這邊要強調的關鍵點是 for-loop 中的 if 判斷式，在正常讀取 reqStr[i] buffer 內容的狀況下，字元 c 會一直接收到 RTSP 的封包字元內容，以下面的 RTSP 訊息為例，字元 c 也就是會把『O』、『P』、『T』、『I』、『O』、『N』、『S』等字元依序存放至 resultCmdName 的指標陣列中。

```

▽ Real Time Streaming Protocol
  ▸ OPTIONS rtsp://140.113.13.97:8554/mpeg1or2AudiovideoTest RTSP/1.0\r\n
    CSeq: 1\r\n
    User-Agent: VLC Media Player (LIVE.COM Streaming Media v2004.11.11)\r\n
    \r\n

```

而接下來我們看到用戶端所傳過來的『OPTIONS』訊令後面是一個空白字元(ASCII十六進制值 20) 如下圖 4.4 所示，而 if 判斷式中只要字元 c 接收到的值為空白字元，便會結束這個『產生 RTSP 命令名稱』的迴圈，這也表示 resultCmdName 指標陣列中已經正確地取得並存放了 cmdName=『OPTIONS』這個信令。

```

▽ Real Time Streaming Protocol
  ▸ OPTIONS rtsp://140.113.13.97:8554/mpeg1or2AudiovideoTest RTSP/1.0\r\n
    Method: OPTIONS
    URL: rtsp://140.113.13.97:8554/mpeg1or2AudiovideoTest
    CSeq: 1\r\n
    User-Agent: VLC Media Player (LIVE.COM Streaming Media v2004.11.11)\r\n
    \r\n

```

0000	00	04	23	b7	e9	52	00	02	44	63	fe	4e	08	00	45	00	..#..R.. DC..N
0010	00	bb	1f	c4	40	00	80	06	a6	de	8c	71	0d	57	8c	71	...@
0020	0d	61	09	fa	21	6a	70	5a	69	1b	15	e2	ca	ff	50	18P.
0030	ff	ff	d8	97	00	00	4f	50	54	49	4f	4e	53	20	72	74OPTIONS rt
0040	73	70	3a	2f	2f	31	34	30	2e	31	31	33	2e	31	33	2e	sp://140 .113.13.
0050	39	37	3a	38	35	35	34	2f	6d	70	65	67	31	6f	72	32	97:8554/ mpeg1or2
0060	41	75	64	69	6f	56	69	64	65	6f	54	65	73	74	20	52	Audiovid eoTest R

在『OPTIONS』後的
SPACE 空白字元
(ASCII 十六進制
值 20)

圖 4.4 Results RTSP command name

最後解讀出來的命令 cmdName 再利用 strcmp()來辨識要進入哪一個信令對應的執行函數，以 VLC player 為例的話，其發送信令順序會是『OPTIONS』->『DESCRIBE』->『SETUP』->『TEARDOWN』->『PLAY』->『PAUSE』。

```

if (!parseRequestString((char*)fBuffer, totalBytes,
    cmdName, sizeof cmdName,
    urlPreSuffix, sizeof urlPreSuffix,
    urlSuffix, sizeof urlSuffix,
    cseq, sizeof cseq)) {
    if (strcmp(cmdName, "OPTIONS") == 0) {
        handleCmd_OPTIONS(cseq);
    } else if (strcmp(cmdName, "DESCRIBE") == 0) {
        handleCmd_DESCRIBE(cseq, urlSuffix, (char const*)fBuffer);
    } else if (strcmp(cmdName, "SETUP") == 0) {
        handleCmd_SETUP(cseq, urlPreSuffix, urlSuffix, (char const*)fBuffer);
    } else if (strcmp(cmdName, "TEARDOWN") == 0
        || strcmp(cmdName, "PLAY") == 0
        || strcmp(cmdName, "PAUSE") == 0) {
        handleCmd_withinSession(cmdName, urlPreSuffix, urlSuffix, cseq,
            (char const*)fBuffer);
    } else {
        handleCmd_notSupported(cseq);
    }
} ? end else ?

```


4.3 RTSP 進入 RTP 傳送之機制

當伺服器端可以接收、解析以及執行用戶端所傳送的 RTSP 信令要求後，接下來尚須經過『DESCRIBE』、『SETUP』的手續方能到達『PLAY』的階段，『DESCRIBE』我們已於前面 2.4.2.2 (RTSP DESCRIBE) 章節中論述過，故在此不再贅述。接下來『SETUP』的說明我們亦在前面 2.4.2.3 章節 (RTSP SETUP) 中論述過，故在此我們僅就 Live555 對於 RFC2326 的實現作一個簡明的歸納整理，RFC2326 文件中的 RTSP State Machine 狀態圖如下圖 4.5 所示：

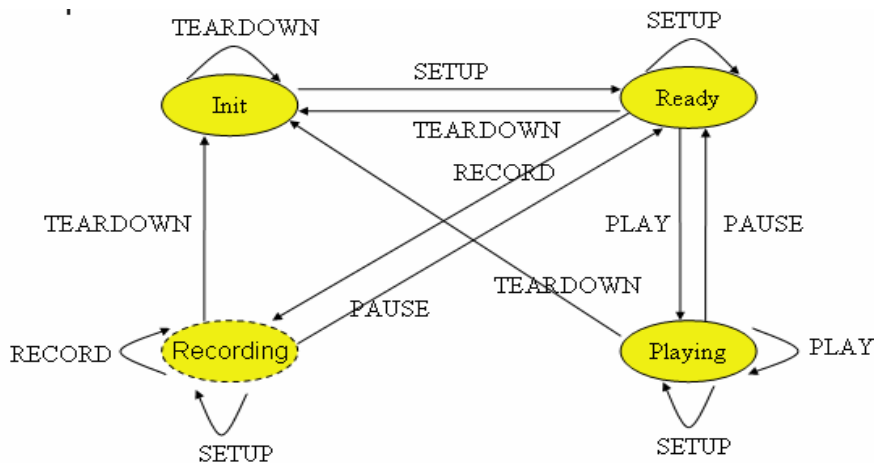


圖 4.5 RTSP State Machine

而截至目前為止(2006/12)，Live555 尚未支援圖 4.5 中的『RECORD』信令，故在此先以虛線表示，綜合比較起來 Live555 目前對於 RFC2326 RTSP 的實現如下表 4.3 所示。(註.RECORD 功能是用戶端可把串流接收到的資料寫入主機的儲存媒體中)

在得到伺服器端所回應的『SETUP』RTSP信令OK訊息之後，用戶端便可以發送『PLAY』信令要求伺服器端開始傳送RTP封包，此部分的程式流程說明我們已於前面3.1

章節(RTSP PLAY命令-handleCmd_PLAY())提過，參考下圖4.6所示，伺服器端在回應編號

92號的封包-也就是回應RTSP『PLAY信令OK』之後便開始一連串RTP影音封包的傳

送。

Schulzrinne, et. al. Standards Track [Page 29]
RFC 2326 Real Time Streaming Protocol April 1998

Live555 support method

method	direction	object	requirement
DESCRIBE	C->S	P,S	recommended
ANNOUNCE	C->S, S->C	P,S	optional
GET_PARAMETER	C->S, S->C	P,S	optional
OPTIONS	C->S, S->C	P,S	required
			(S->C: optional)
PAUSE	C->S	P,S	recommended
PLAY	C->S	P,S	required
RECORD	C->S	P,S	optional
REDIRECT	S->C	P,S	optional
SETUP	C->S	S	required
SET_PARAMETER	C->S, S->C	P,S	optional
TEARDOWN	C->S	P,S	required

Table 2: Overview of RTSP methods, their direction, and what objects (P: presentation, S: stream) they operate on

表 4.3 Implemented RTSP method of Live555



No. .	Time	Source	Destination	Protocol	Info
87	14.689709	140.113.13.97	140.113.13.97	RTSP	SETUP rtsp://140.113.13.97:8554/mpeg1or2Audiovide
88	14.672865	140.113.13.97	140.113.13.87	RTSP	Reply: RTSP/1.0 200 OK
89	14.681223	140.113.13.87	140.113.13.97	RTSP	SETUP rtsp://140.113.13.97:8554/mpeg1or2Audiovide
90	14.684104	140.113.13.97	140.113.13.87	RTSP	Reply: RTSP/1.0 200 OK
91	14.685751	140.113.13.87	140.113.13.97	RTSP	PLAY rtsp://140.113.13.97:8554/mpeg1or2Audiovide
92	14.686668	140.113.13.97	140.113.13.87	RTSP	Reply: RTSP/1.0 200 OK
93	14.686874	140.113.13.97	140.113.13.87	MPEG-1	MPEG-1 message
94	14.687027	140.113.13.97	140.113.13.87	MPEG-1	MPEG-1 message
95	14.687063	140.113.13.97	140.113.13.87	MPEG-1	MPEG-1 message
96	14.687160	140.113.13.97	140.113.13.87	MPEG-1	MPEG-1 message
97	14.687196	140.113.13.97	140.113.13.87	MPEG-1	MPEG-1 message
98	14.687270	140.113.13.97	140.113.13.87	MPEG-1	MPEG-1 message

▶ Frame 92 (338 bytes on wire, 338 bytes captured)
 ▶ Ethernet II, Src: 00:04:23:b7:e9:52, Dst: 00:02:44:63:fe:4e
 ▶ Internet Protocol, Src Addr: 140.113.13.97 (140.113.13.97), Dst Addr: 140.113.13.87 (140.113.13.87)
 ▶ Transmission Control Protocol, Src Port: 8554 (8554), Dst Port: 2554 (2554), Seq: 1101, Ack: 912, Len: 284
 ▼ Real Time Streaming Protocol
 ▶ RTSP/1.0 200 OK\r\n
 CSeq: 5\r\n
 Date: Tue, Aug 29 2006 15:26:09 GMT\r\n
 Range: npt=0.000-\r\n
 ▶ Session: 1\r\n
 RTP-Info: url=rtsp://140.113.13.97:8554/mpeg1or2AudiovideoTest/track1;seq=50878;rtptime=384912574,url=rt:\r\n\r\n

圖 4.6 『PLAY』 - Send RTP packet

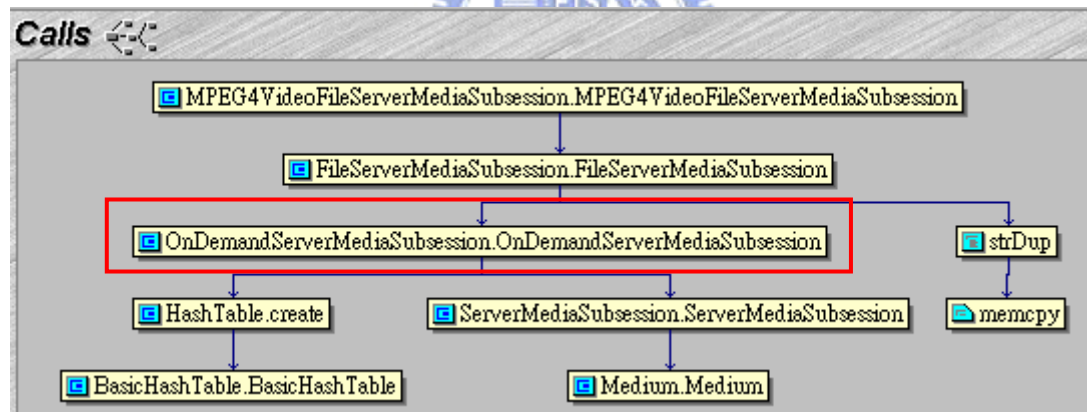
4.4 檔案資料內容之解讀及訊框之取出

當用戶端以及伺服器端的 RTSP PLAY 信令溝通協調部分已經成功之後，接下來伺服器端便是發送 RTP 封包;在第三章中我們已經討論了大部分的『封包包裝以及傳送』的過程，而這邊要更仔細討論的地方有下面三個~

1. 伺服器端如何取得來源檔案？
2. 伺服器端如何解析各種影音格式的檔案？

首先我們再次由第三章『封包包裝以及傳送』的主要程式

MPEG4VideoFileServerMediaSubsession.cpp 開始作快速的引導說明。



```
FramedSource* MPEG4VideoFileServerMediaSubsession
::createNewStreamSource(unsigned /*clientSessionId*/, unsigned& est
estBitrate = 500; // kbps, estimate

A-1 // Create the video source:
ByteStreamFileSource* fileSource
= ByteStreamFileSource::createNew(envir(), fileName);
if (fileSource == NULL) return NULL;
fileSize = fileSource->fileSize();

A-2 // Create a framer for the Video Elementary Stream:
return MPEG4VideoStreamFramer::createNew(envir(), fileSource);

*B RTPSink* MPEG4VideoFileServerMediaSubsession
::createNewRTPSink(Groupsock* rtpGroupsock,
unsigned char rtpPayloadTypeIfDynamic,
FramedSource* /*inputSource*/) {
return MPEG4ESVideoRTPSink::createNew(envir(), rtpGroupsock,
rtpPayloadTypeIfDynamic);
}

//取得檔案指標
FILE* fid = OpenInputFile(envir(), fileName);
if (fid == NULL) return NULL;

ByteStreamFileSource* newSource
= new ByteStreamFileSource(envir(), fid, preferredFr
newSource->fileSize = GetFileSize(fileName, fid);
//ByteStreamFileSource::createNew()函數回傳
return newSource;
} ? end createNew ?
```

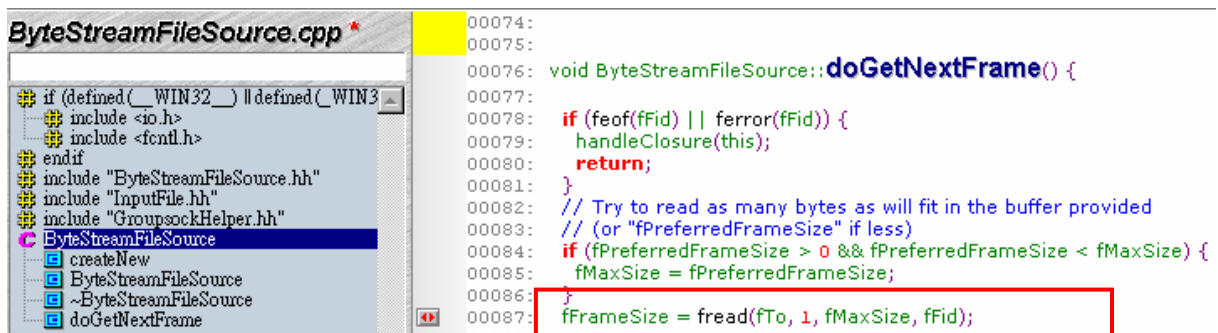
註：

- 1.*A、*B 函數已於前面第三章論述過，在此僅作讀取/解析檔案的重點論述
- 2.*A～請參考 3.2.2 章節，建立影像來源（createNewStreamSource()）
- 3.*B～請參考 3.2.3 章節，影像訊框的組成（createNewRTPSink()）
- 4.Live555 目前共支援九種串流格式，在此僅以 MPEG4ES 為例。

4.4.1：開檔與讀檔

在前面 3.2.2(開啟檔案並讀取檔案內容)章節中，我們知道 A-1

ByteStreamFileSource.createNew()會去呼叫 fopen()函數，如下圖 4.7(a)所示，目的是打開欲串流的檔案，並且 ByteStreamFileSource.cpp 的 doGetNextFrame()會去讀取 fFid 檔案指標指向的檔案位址，每次讀取 1 個 record(1 Byte)，共需讀取『fMaxSize』次。



```
ByteStreamFileSource.cpp *
00074:
00075:
00076: void ByteStreamFileSource::doGetNextFrame() {
00077:
00078:     if (feof(fFid) || ferror(fFid)) {
00079:         handleClose(this);
00080:         return;
00081:     }
00082:     // Try to read as many bytes as will fit in the buffer provided
00083:     // (or "fPreferredFrameSize" if less)
00084:     if (fPreferredFrameSize > 0 && fPreferredFrameSize < fMaxSize) {
00085:         fMaxSize = fPreferredFrameSize;
00086:     }
00087:     fFrameSize = fread(fTo, 1, fMaxSize, fFid);
00088: }
```

因為 Live555 用了很多建構子來完成串流物件初值的設定，故有一些指標名稱會有所不同，比如說當我們執行 ByteStreamFileSource 建構子時，因為 ByteStreamFileSource 子類別繼承了 FramedFileSource 父類別，故程式將會設定 fFid=fid。如欲自行利用 Live555

的函示庫來開發串流程式時，指標的使用需特別留意，而 fTo 指標指向的記憶體位址便是資料流的起始位址(進階的說明可以參考 Live DeviceSource.cpp)。

```
// Try to read as many bytes as will fit in the buffer provided
// (or "fPreferredSize" if less)
if (fPreferredSize > 0 && fPreferredSize < fMaxSize) {
    fMaxSize = fPreferredSize;
}
fFrameSize = fread(fTo, 1, fMaxSize, fFid);
```

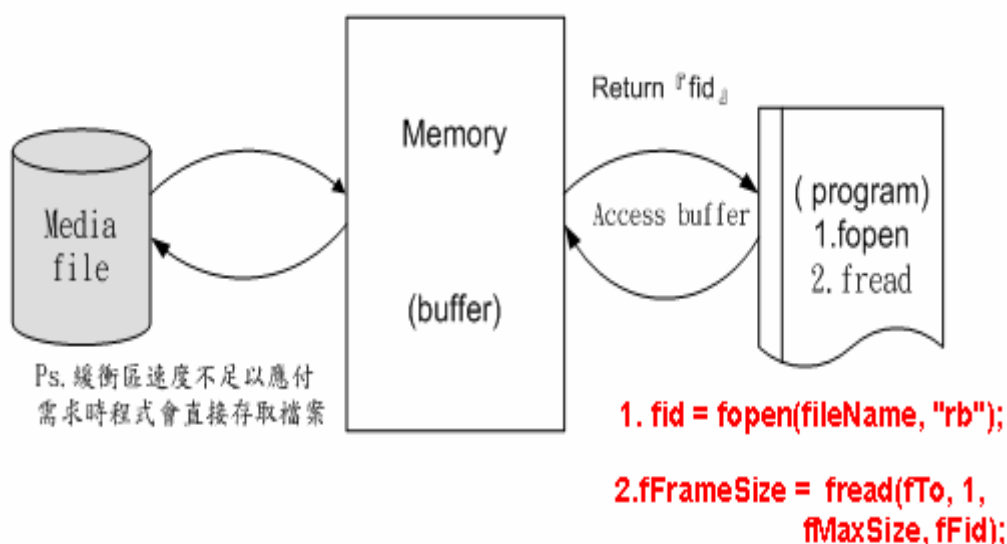


圖 4.7(a) fopen/fread

接下來要討論的是當打開一個多媒體檔案之後，要如何針對特定的影音格式資料流來解讀的部分，在此以 MPEG-4 ES 格式為例，請參考下圖 4.7(b)，其餘九種的影音格式的解析請自行參考 Live 的程式碼。

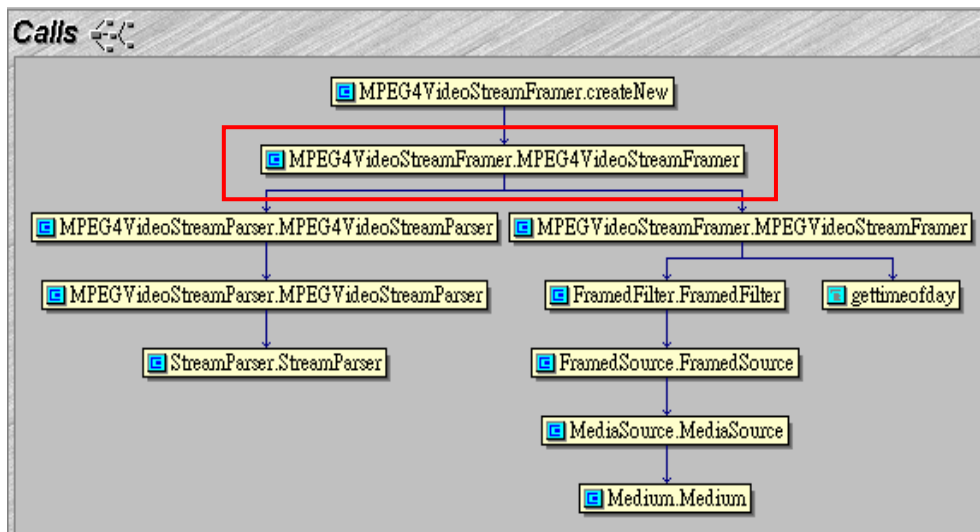


圖 4.7(b) MPEG4VideoStreamFramer function-call

4.4.2 建立以 MPEG4 為主的影像流訊框產生器

(MPEG4VideoStreamFramer)

如上圖 4.7(b)所示，MPEG4VideoStreamFramer 所完成的工作便是把 MPEG4 影像流中的 header 以及 frame 資訊給分離出來，特別是 VOP(說明請見 4.4.2.1 章節)，因為 VOP 是 MPEG-4 中互動的基本單位。

MPEG4VideoStreamParser 的功能是辨識 MPEG4 video ES 的 Visual Object Sequence (VS) Header + Visual Object (VO) Header + Video Object Layer (VOL) Header、Group of VOP (GOV) Header 以及 VOP frame 的能力，MPEG4 視覺位元串流各物件的說明及定義如下 4.4.2.1 章節所述。

4.4.2.1 MPEG4 概要

傳統的視訊壓縮技術是以一張畫面為壓縮的單位，而MPEG-4則將畫面再切割成更小的單位作為壓縮單位，我們稱為物件（Object），因此原本是一張張畫面的視訊序列 (video sequence) 會被分割成數個以物件為主（Object-based）的視訊物件序列 (Visual Object Sequence)。

MPEG4中一個標準的場景(scene)包括一個背景和一個或多個的前景物件，而在對此物件作處理之前，物件本身必須要先經取樣（sampled）處理。大部分物件都是在一段固定的時間間隔下被取樣的，而每一段時間的樣本就被稱為一個視訊物件圖（video object plane，VOP），比如說一個人、一台汽車便是一個VOP，MPEG4場景內每一個物件都是被一系列的VOP來表示。



4.4.2.2 MPEG-4 start codes

MPEG4 Structure and Syntax 如下圖 4.8 所示：

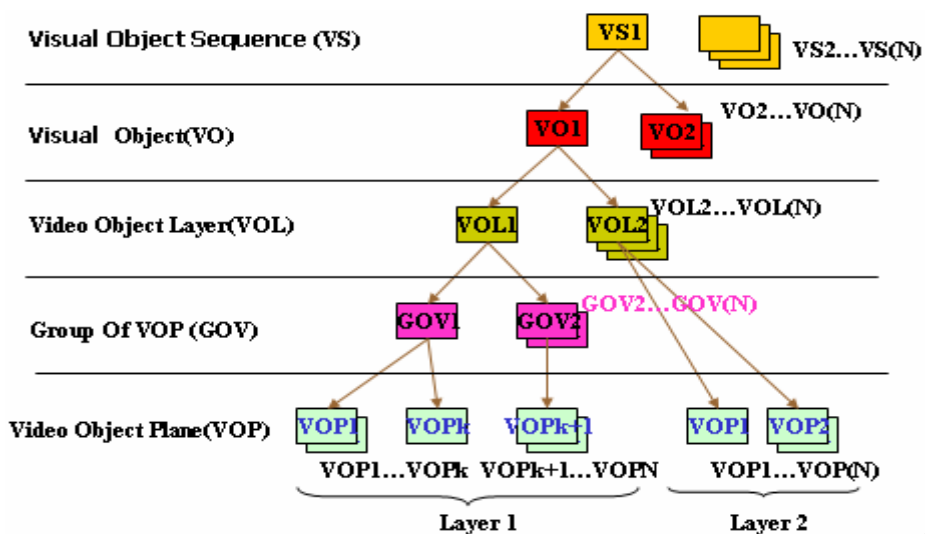


圖 4.8 MPEG4 video bitstream logical structure

- 1.視訊物件序列 (Visual Object Sequence, VS)：它是一個完整的 MPEG-4 場景，包含所有的物件。
- 2.視訊物件 (Visual Object, VO)：視訊物件會連結至場景中的某個 2D 物件，對應於場景中的某個物件或是背景，也就是資料流裡一個特定視訊物件的所有資訊。
- 3.視訊物件層 (Video Object Layer, VOL)：對一串列的視訊物件圖(VOP)或視訊物件圖群組(GOV)提供可調式壓縮(**scalable coding**)的能力，目的是因應網路傳輸率的變動。
- 4.視訊物件圖群組(Group of Video Object Planes, 簡稱GOV)：是一群視訊物件圖(VOP)的集合。視訊物件平面群組就和先前MPEG標準裡的圖片群 (group of pictures, GOP) 很類似，它提供在位元資料流 (bitstream) 裡視訊物件圖將被單獨編碼的位置。
- 5.視訊物件圖 (Video Object Planes, VOP)：如4.4.2.1的說明，VOP header後通常是由一連串MB (Macro Block)所組成的frame。

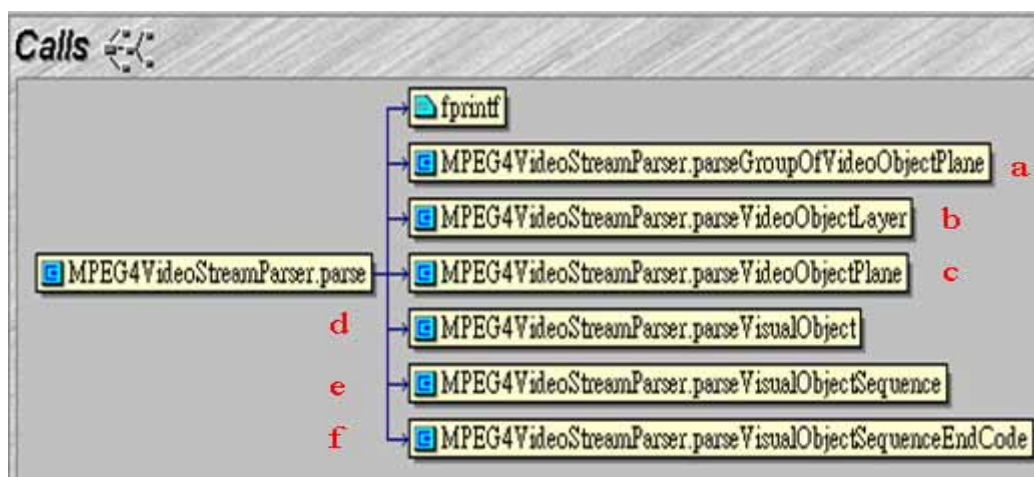
MPEG-4 Video stream 會提供階層式的視覺場景描述，起始碼 (start codes) 則是特殊的編碼值(請參考下表 4.4)，它們可以存取位元流的每一層階層架構，因此我們分離訊框的技巧便是要能辨識出這些視訊物件的起始碼，並從起始碼後找出重要的訊框資訊。

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
reserved	B7-B9
fba_object_start_code	BA
fba_object_plane_start_code	BB
mesh_object_start_code	BC
mesh_object_plane_start_code	BD
still_texture_object_start_code	BE
texture_spatial_layer_start_code	BF
texture_snr_layer_start_code	C0
texture_tile_start_code	C1
texture_shape_layer_start_code	C2
stuffing_start_code	C3
reserved	C4-C5
System start codes (see note)	C6 through FF
NOTE System start codes are defined in ISO/IEC 14496-1:1999	

表 4.4 MPEG-4 start codes

4.4.2.3 MPEG4 video stream parsing flow

要把 MPEG4 影像流中的 I/P/B frame 給過濾出來也意味著我們需要實作一個函數用來辨識出 MPEG4 起始碼(如表 4.3 所示)，並且可以把 MPEG4 資料流的 Visual Object Sequence (VS) Header、Visual Object (VO) Header、Video Object Layer (VOL) Header、Group of VOP (GOV) Header 以及 VOP(frame)等等資訊分離出來，這個函數便是 MPEG4VideoStreamParser.parse()，parse()的函數呼叫情形如下所示：



以一段 m4v 的影片來說，其 bitstream(如下圖 4.9(a))中包含了兩類資訊，第一類是 MPEG4 configuration information(組態資訊)，說明如下：

e.parseVisualObjectSequence():

負責解析 VisualObjectSequence，也就是解析全部的MPEG-4場景，包含所有物件的組態資訊。

d.parseVisualObject():

負責解析 VisualObject，也就是解析單一個 MPEG4 物件的組態資訊。

b.parseVideoObjectLayer ():

負責解析 VideoObjectLayer，也就是解析單一個視訊物件層的組態資訊。



第二類是Elementary stream data(基本流資料)，而上圖parse()所呼叫的a、c函數便是 Elementary stream的起始處入口(entry points)，說明如下：

a. parseGroupOfVideoObjectPlane():

負責解析GroupOfVideoObjectPlane。

c. parseVideoObjectPlane():

負責解析 VideoObjectPlane，在 VOP header 後的便會是 frame 的資料。

以下為程式解析流程說明，內容請參考Live555的MPEG4VideoStreamFramer.cpp，而此解析程式主要參考ISO/IEC 14496-2:2001(E)文件(Page 32)撰寫而成[8]。

(MPEG4VideoStreamFramer/MPEG4VideoStreamParser 解析流程說明)

1. 開始讀入 MPEG4 影像位元流，首先執行 parseVisualObjectSequence()，也就是由下圖 4.9(a) 的 m4v bitstream 中找出圖 4.9(b) 所定義的 visual bitstream 的 VS start-code，VS 即是 visual bitstream 的起始碼『00 00 01 B0』。

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
00000000h:	00	00	01	B0	F5	00	00	01	B5	09	00	00	01	00	00	00
00000010h:	01	20	08	C4	9D	C0	00	43	A9	C0	09	50	00	B0	D4	97
00000020h:	53	0C	1F	4C	2C	10	78	71	0F	00	00	01	B2	65	6D	34
00000030h:	76	20	34	2E	33	2E	32	2E	38	00	C9	FF	00	00	00	01
00000040h:	B6	11	8A	78	0D	EO	8E	AC	7A	B0	92	AA	FD	27	9B	6D
00000050h:	48	80	A9	AC	FF	74	78	C9	7B	45	C9	FE	20	24	1F	FC
00000060h:	18	6C	94	14	DF	67	04	04	AD	B4	DB	51	BD	1C	36	C8
00000070h:	8C	94	5A	78	21	8F	13	89	4C	B3	E4	A2	07	DA	FB	7F
00000080h:	6B	CC	73	44	A1	2D	58	06	32	06	04	3B	A2	50	2A	95
00000090h:	A6	D8	DE	36	D8	7D	FF	52	A2	E2	FA	3E	D5	99	D0	60
000000a0h:	14	39	98	3D	00	F0	3C	3C	4A	A8	15	6D	DC	52	AD	94
000000b0h:	F0	03	C1	10	71	D6	3C	C7	93	26	4C	3F	D0	38	0C	39
000000c0h:	C1	DF	CB	0B	9A	DO	FA	30	06	DB	49	78	AO	B0	2C	3C
000000d0h:	0D	C6	59	4A	0C	0A	D6	D5	DF	24	B1	AC	E5	OF	87	53
000000e0h:	FB	73	C5	FF	2E	12	87	4D	7D	2A	7F	89	65	EA	D3	C5
000000f0h:	41	F3	41	F2	62	E1	F6	37	59	B8	C8	80	03	39	EA	60

圖 4.9(a) m4v bitstream example

```
#define VISUAL_OBJECT_SEQUENCE_START_CODE 0x000001B0
#define VISUAL_OBJECT_SEQUENCE_END_CODE 0x000001B1
#define GROUP_YOP_START_CODE 0x000001B3
#define VISUAL_OBJECT_START_CODE 0x000001B5
#define YOP_START_CODE 0x000001B6
```

圖 4.9(b) Live555 visual bitstream start-code

2. 看見 VS start code 之後的是 1 Byte 的 "profile_and_level_indication" 『F5』。

3. 接下來是看到 VO start code 『00 00 01 B5』後進入 parseVisualObject()。

4. parseVisualObject() 首先把下一位元組 『09』 存起來，確認 is_visual_object_identifier 位元的值，非 0 的話，我們便找到 visual_object_verid(4 bit) 以及 visual_object_priority(3 bit)，如下表 4.5。

visual_object_verid	Meaning
0000	reserved
0001	object type listed in Table 9-1
0010	object type listed in Table V2 - 39
0011 - 1111	reserved

visual_object_priority: This is a 3-bit code which specifies the priority of the visual object. It takes values between 1 and 7, with 1 representing the highest priority and 7, the lowest priority. The value of zero is reserved.

表4.5 Meaning of visual_object_verid

5. 接下來是4位元的visual object type(視覺物件種類)

code	visual object type
0000	reserved
0001	video ID
0010	still texture ID
0011	mesh ID
0100	FBA ID
0101	3D mesh ID
01101	reserved
:	:
:	:
1111	reserved

表 4.6 visual object type

而目前 Live555 只支援"Video ID"，也就是 code=0001，如上表 4.6 所示。

6.接下來是 4 位元組的 video_object_start_code，10~13 Bytes：『00 00 01 20』，確認之

後即進入 parseVideoObjectLayer()。

7.進入 parseVideoObjectLayer()一開始必須看"video_object_layer_start_code"，

，也就是第 14~17 Bytes: 『00 00 01 20』，VS+VO+VOL 合稱為配置資訊(Configuration

Information)，是重要的 Decoder 訊息，因此 RFC 3016 規定串流的一開始要把配置資訊

放在最前面。

8.看到 video_object_layer_start_code 之後，把接下來的位元組都存起來，直到我們看到 GOV 或是 VOP 的 start code。

9.如下所示便是 VOP 的 start code 『00 00 01 B6』

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
00000000h:	00	00	01	B0	F5	00	00	01	B5	09	00	00	01	00	00	00
00000010h:	01	20	08	C4	9D	C0	00	43	A9	C0	09	50	00	B0	D4	97
00000020h:	53	0C	1F	4C	2C	10	78	71	0F	00	00	01	B2	65	6D	34
00000030h:	76	20	34	2E	33	2E	32	2E	38	00	C9	FF	00	00	00	01
00000040h:	B6	11	8A	78	0D	E0	8E	AC	7A	B0	92	AA	FD	27	9B	6D
00000050h:	48	80	A9	AC	FF	74	78	C9	7B	45	C9	FE	20	24	1F	FC
00000060h:	18	6C	94	14	DF	67	04	04	AD	B4	DB	51	BD	1C	36	C8

而其後便是 2 個位元的 "vop_coding_type"，如下表 4.7 所示，這 2 個位元便決定該

VOP 的編碼種類是 I/P/B frame 的哪一種，這便是 MPEG4VideoStreamParser 最主要的功

能所在-『從 MPEG4 video bitstream 中分離出 VOP I/P/B frame 的資訊』。

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

表4.7 Meaning of vop_coding_type

跟在 VOP header 後面便是一個 I/P/B frame 的所有 MB (Macro Block) 資料，在追蹤串流封包的位元組時亦可以仔細觀察所擷取下來封包的二進制碼以及時間戳記欄位，因為 RFC3016 中有規定包裝同一個 VOP (frame) 的數個封包其時間戳記皆要相同，並且 VOP start_code 必須是 『00 00 01 B6』，同時會設定 Marker bit 表示這是一個 VOP 的結束，實際情形請參考下圖 4.10(a)，而雖然目前的封包擷取軟體無法正確顯示 m4v 資料封包中有關

於payload type中的frame資訊，但經驗告訴我們『1』有可能是P/B frame，『2』可能是P frame，而『3』佔用了三個MTU封包，資料至少超過4.5KB(1500*3)故必定是最重要的I-frame訊息。

No.	Time	src	Destination	Protocol	Info	
50	0.000236	14*	140.113.13.89	RTP	Payload type=Unknown (96), SSRC=1400413477, Seq=21488, Time=95821823, Mark	1
51	0.031941	14*	140.113.13.89	RTP	Payload type=Unknown (96), SSRC=1400413477, Seq=21489, Time=95821826, Mark	
52	0.029150	14*	140.113.13.89	RTP	Payload type=Unknown (96), SSRC=1400413477, Seq=21490, Time=958219826	2
53	0.000227	14*	140.113.13.89	RTP	Payload type=Unknown (96), SSRC=1400413477, Seq=21491, Time=958219826, Mark	
54	0.029814	14*	140.113.13.87	TCP	11419 > 8554 [ACK] Seq=679 Ack=1230 win=16291 Len=0	3
55	0.004159	14*	140.113.13.89	RTP	Payload type=Unknown (96), SSRC=1400413477, Seq=21492, Time=95822829	
56	0.000212	14*	140.113.13.89	RTP	Payload type=Unknown (96), SSRC=1400413477, Seq=21493, Time=95822829, Mark	3
57	0.035318	14*	140.113.13.255	NBNS	Name query NB PC<00>	
58	0.002567	14*	140.113.13.89	RTP	Payload type=Unknown (96), SSRC=1400413477, Seq=21494, Time=958225832	3
59	0.000211	14*	140.113.13.89	RTP	Payload type=Unknown (96), SSRC=1400413477, Seq=21495, Time=958225832, Mark	
60	0.028102	14*	140.113.13.89	RTP	Payload type=Unknown (96), SSRC=1400413477, Seq=21496, Time=958228835, Mark	3
61	0.031291	14*	140.113.13.89	RTP	Payload type=Unknown (96), SSRC=1400413477, Seq=21497, Time=958231838	
62	0.000218	14*	140.113.13.89	RTP	Payload type=Unknown (96), SSRC=1400413477, Seq=21498, Time=958231838	3
63	0.000185	14*	140.113.13.89	RTP	Payload type=Unknown (96), SSRC=1400413477, Seq=21499, Time=958231838	
64	0.000181	14*	140.113.13.89	RTP	Payload type=Unknown (96), SSRC=1400413477, Seq=21500, Time=958231838, Mark	3
65	0.038413	14*	140.113.13.89	RTP	Payload type=Unknown (96), SSRC=1400413477, Seq=21501, Time=958234841, Mark	
66	0.031277	14*	140.113.13.89	RTP	Payload type=Unknown (96), SSRC=1400413477, Seq=21502, Time=958237844	3
67	0.000227	14*	140.113.13.89	RTP	Payload type=Unknown (96), SSRC=1400413477, Seq=21503, Time=958237844	
68	0.000209	14*	140.113.13.89	RTP	Payload type=Unknown (96), SSRC=1400413477, Seq=21504, Time=958237844, Mark	3
69	0.032823	14*	140.113.13.89	RTP	Payload type=Unknown (96), SSRC=1400413477, Seq=21505, Time=958240847	

圖4.10(a) VOP header and I/P/B frame

而MPEG4VideoStreamParser的程式動作原理便是把下圖4.10(b)中的所有ES流資料解出來並存起來，而在MPEG4VideoStreamParser程式的類別(class)中我們便可以看到上述解析過程中所有會用到的變數都有對應的資料成員(data member)，當建構子替物件設定初值完畢後也就代表我們已解析成功。

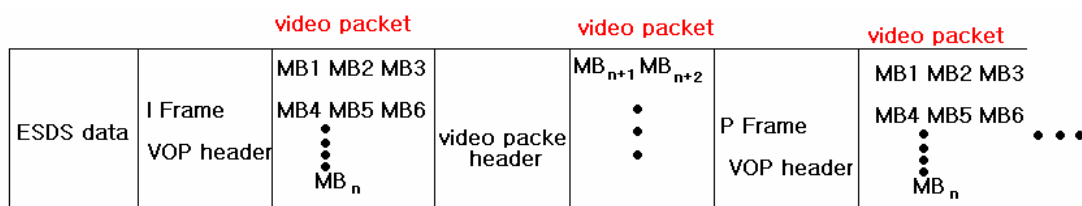


圖 4.10(b) mpeg-4 video stream example

10.最後當我們讀到 VISUAL_OBJECT_SEQUENCE_END_CODE(0x000001B1)時，我們必須把過程中所讀到的位元組都存下來，因為這就是一個場景的所有資訊，如下所示為 Live555 MPEG4VideoStreamFramer.cpp 的程式碼，我們會把 fPictureEndMarker 這個布林值設為 True，表示這是一個場景的結束，並利用 curFrameSize()函數傳回該 frame 的長度資訊，作為將來封包傳送的重要參數。

```
unsigned MPEG4VideoStreamParser::parseVisualObjectSequenceEndCode() {  
#ifdef DEBUG  
    fprintf(stderr, "parsing VISUAL_OBJECT_SEQUENCE_END_CODE\n");  
#endif  
    // Note that we've already read the VISUAL_OBJECT_SEQUENCE_END_CODE  
    save4Bytes(VISUAL_OBJECT_SEQUENCE_END_CODE);  
  
    setParseState(PARSING_VISUAL_OBJECT_SEQUENCE);  
  
    // Treat this as if we had ended a picture:  
    usingSource() -> fPictureEndMarker = True; // HACK #####  
  
    return curFrameSize();  
}
```



11.最後 MPEG4VideoStreamFramer 的功用則是把圖 4.10(b)中的 header 以及 frame 的資料分開，因為我們接下來要進行的便是依照 RFC 3016[7]的建議，去規劃一個有限空間的 RTP 封包中要如何配置表頭以及資料的全盤佈局，其主要目的是要因應網路有封包遺失的情形，因此我們需要去分配在一個有限大小的封包中(\leq MTU)，幫助解碼器進行錯誤恢復(error-resilience)用的表頭(VOP/VP header)與真正資料裝載量(payload)的比例，也就是要考慮封包以及通道使用率的問題，進而設計適當的封包演算法(packetization algorithm)。

4.5 建立以及傳送 RTP 封包

4.5.1 RFC3016 封裝演算法分析

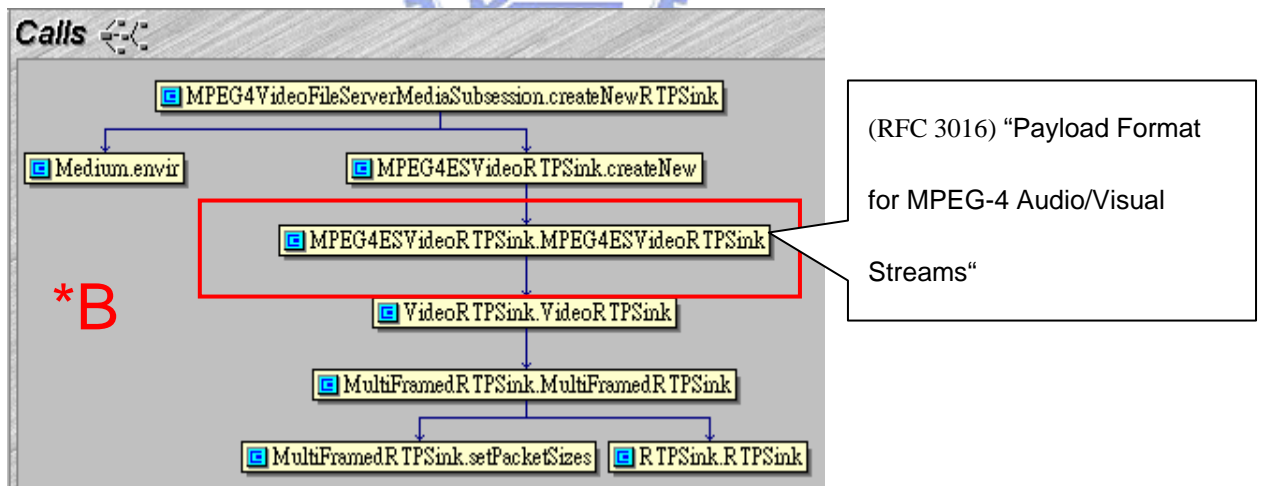
如下程式碼所示，我們已經由*A 的部分得到了 MPEG-4 的訊框，接下來要討論的是訊框如何被打包進一個封包的問題，也就是*B 的程式碼部分。

```
*A  FramedSource* MPEG4VideoFileServerMediaSubsession
    ::createNewStreamSource(unsigned /*clientId*/, unsigned& est
        estBitrate = 500; // kbps, estimate

    A-1 // Create the video source:
        ByteStreamFileSource* fileSource
        = ByteStreamFileSource::createNew(envir(), fileName);
        if (fileSource == NULL) return NULL;
        fileSize = fileSource->fileSize();

    A-2 // Create a framer for the Video Elementary Stream:
        return MPEG4VideoStreamFramer::createNew(envir(), fileSource);
    }

*B  RTPSink* MPEG4VideoFileServerMediaSubsession
    ::createNewRTPSink(Groupsock* rtpGroupsock,
        unsigned char rtpPayloadTypeIfDynamic,
        FramedSource* /*inputSource*/) {
    return MPEG4ESVideoRTPSink::createNew(envir(), rtpGroupsock,
        rtpPayloadTypeIfDynamic);
    }
```



MPEG4ESVideoRTPSink 建構子即是 Live555 RTSPServer 對於 RFC3016[7]的實現，RFC3016 中主要定義了 MPEG-4 影音流的 RTP 封包封裝方式(packetization)，在分析 Live555 RTSPServer 的程式碼後，我們發現 Live555 RTSPServer 的封裝法基本上是屬於

固定長度的封裝法[9]，因為每一個 RTP 封包的長度預設為 1448Bytes，其優缺點如下：

優點：提高封包/通道的使用率。

缺點：遺失的封包可能造成在其前後封包所屬的 video packet 無法解碼成功。

以下圖 4.11 為例，封包 2 遺失的話將會造成封包 1 的 video packet(2)以及封包 3 的 video packet(4)的影像資料無法解碼之。

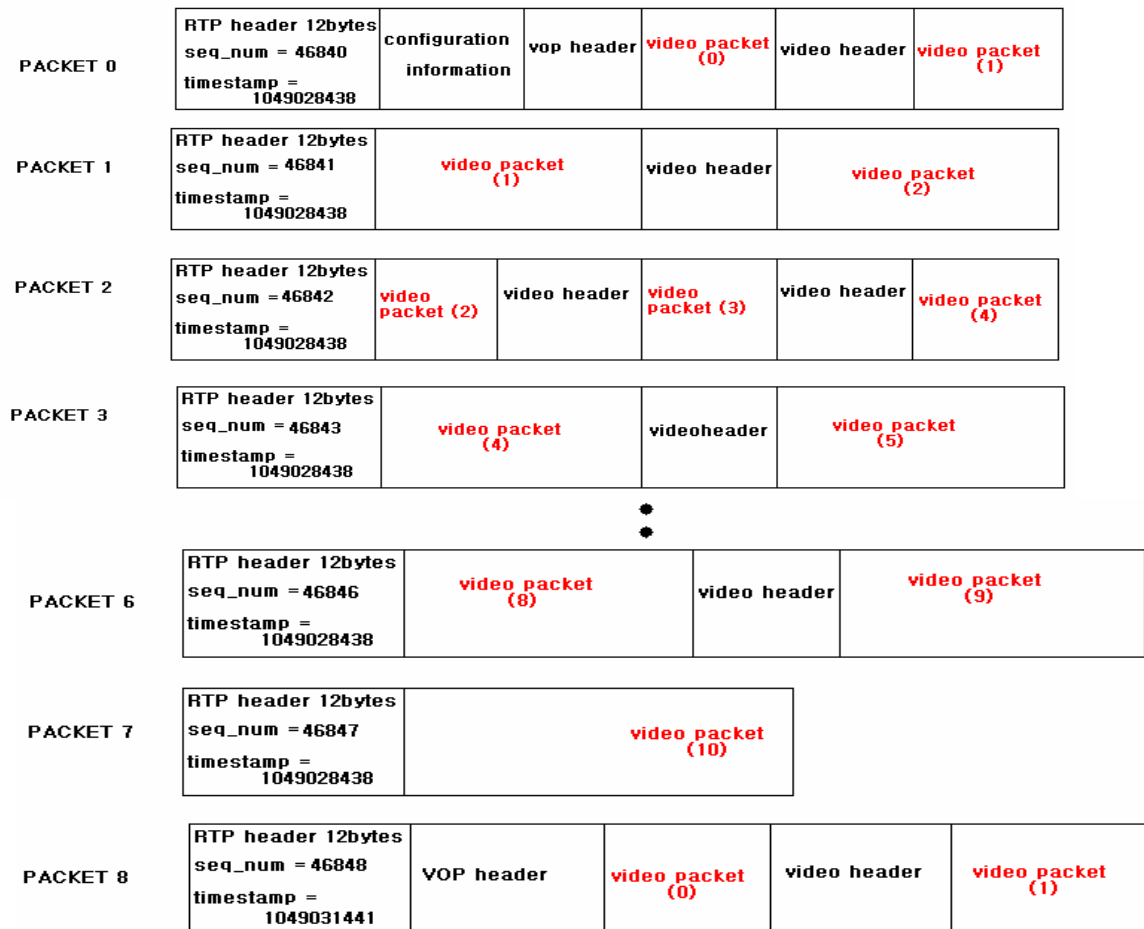


圖 4.11 mpeg-4 video stream in RTP payload

在 RFC 3016 的幾個封包封裝的方式中，Live555 RTSPServer 所選擇的封包裝載法
是比較偏向於下圖 4.12(f)的方法，其實際的演算法我們在 4.5.2 章節中(Live555

RTSPServer 封裝演算法)會作詳細的說明，因為在封裝演算法中還有一些增進串流系統效能的技巧是值得我們去討論的。

(註)RFC3016 中方法(d)、(e)、(f)的比較

(d)優點：適用於封包遺失率很高的網路環境下。因為即使封包n遺失，但是其後封包中的VP表頭所包含的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。

缺點：因為一個RTP封包中僅裝有一個VP封包，故UDP/IP/RTP表頭的overhead將會使得網路傳輸量變大。

(e)優點：提高封包/通道的使用率，適用於網路傳輸率很低的环境中。因為在一個RTP封包中含有多個VP，可以有效節省IP/UDP表頭的overhead。

缺點：減少封包遺失的錯誤恢復能力。因為封包遺失時，所有封包中的VP資訊亦無法知道。而一個RTP封包中最佳的VP數量可以由所處網路環境的封包遺失率以及傳輸速率來決定之。

(f)優點：沒有Video Packet表頭的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。

缺點：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。

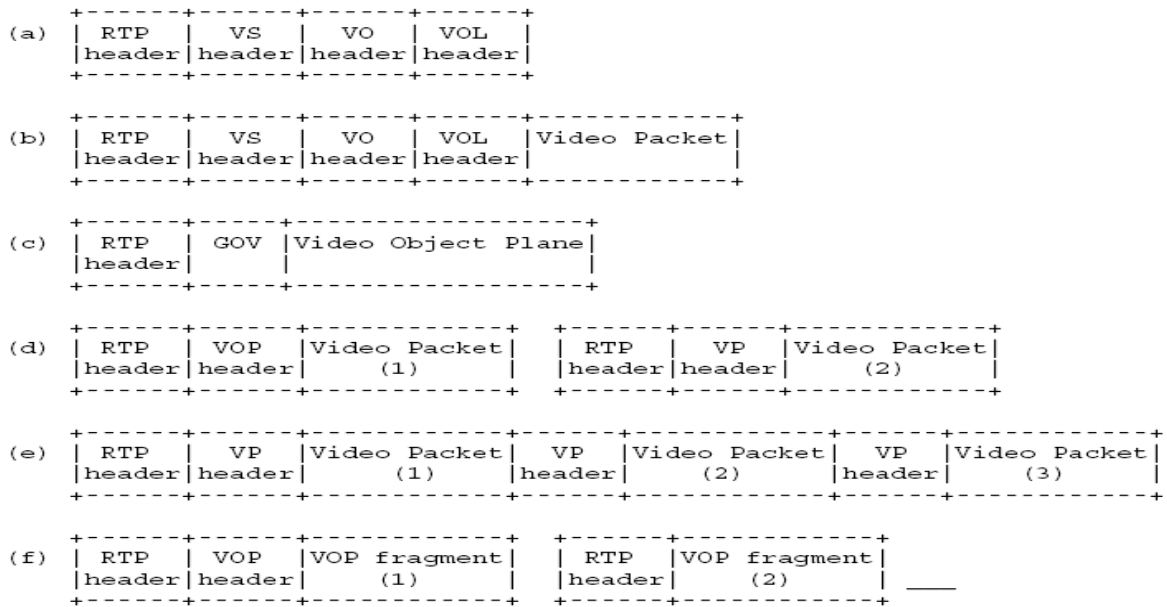


圖 4.12 Examples of RTP packetized MPEG-4 Visual bitstream

4.5.2 Live555 封裝演算法分析

因為封包大小不可大於MTU，不然將會被強制分割（IP fragment），而被強制切割出來的封包也會加上表頭成為新的overhead，在更糟的情況下會發生播延錯誤（error-propagation）的問題，也就是這些被強制分割的封包中如果損毀或遺失也將產生連鎖效應使得其他封包無法正常解碼播放，因此才會有RFC3016[7]的誕生。

因此，Live555 RTSPServer所實現的封裝演算法主要是以目前VOP(frame)與MTU中的較小值作為實際傳送封包的大小，若是VOP太大而無法放進一個封包時，則由程式主動對VOP進行切割(利用setPacketSizes(1000,1448)這個函數)，細分成數個封包來傳送，此目的在於使MPEG-4 ES所產生的封包數最少，同時避免IP fragmented帶給系統的負擔，程式碼如下所示，可以在MultiFramedRTSPSink.cpp程式中找到這個

isTooBigForAPacket()函數，在該函數中會以一個overflowBytes變數來記錄溢位的位元組大小。

```

/*****
檢查訊框大小是否有超過一個封包(MTU)的大小限制
*****/
Boolean MultiFramedRTSPSink::isTooBigForAPacket(unsigned numBytes) const {
    // Check whether a 'numBytes'-byte frame - together with a RTP header and
    // (possible) special headers - would be too big for an output packet:
    // (Later allow for RTP extension header!) #####
    numBytes += rtpHeaderSize + specialHeaderSize() + frameSpecificHeaderSize();
    return fOutBuf->isTooBigForAPacket(numBytes);
}

```

而超過 MTU 的 VOP 資料便會分裝到其他封包中，而 Live555 RTSPServer 原則上不會把 VOP 的表頭複製到各分裝的封包，其目的和優點是可以提高封包/通道的使用率，而其缺點便如圖 4.11 所看到的，單一封包遺失可能造成前後封包無法被解碼的情形發生。而最後一個封包即使有多餘的空間，我們也不會把下一個 VOP 的資料包裝進來。

不過以目前國內ISP(Internet Service Provider)的平均封包遺失率[10]來說，採用此封包演算法是相當符合實際需求的，即便是在國外，封包遺失率也大多在1~3%範圍之中，因此我們在封包演算法的實作上便是以達到較高封包/頻寬使用率為優先考量。

上述的演算法程式實作在 Live555 的 MultiFramedRTSPSink.cpp 程式中，主要實現函數如下所述：

isTooBigForAPacket()-檢查訊框大小是否有超過一個封包(MTU)的大小限制

sendPacketIfNecessary()-判斷是否有必要把封包傳送出去

上述兩個函數主要由 afterGettingFrame1()呼叫之，而在 packFrame 的過程中，Live

的封裝演算法難免會遇到下一個 frame 打包進來後會超過 MTU 的情形，通常我們一般的作法是直接將溢位的資料使用 memmove() 函數把資料搬到下一個封包中，但因為使用 memmove() 這個函數來進行溢位資料在記憶體中的不斷複製與搬移是很沒有效率的一個重複動作，也就是會增加這個演算法執行時帶給系統的負擔，下面所示的部分為 Live 封裝演算法的核心部分。

```
if (fOutBuf->isPreferredSize()
    || fOutBuf->wouldOverflow(numFrameBytesToUse)
    || (fPreviousFrameEndedFragmentation &&
        !allowOtherFramesAfterLastFragment())
    || !frameCanAppearAfterPacketStart(fOutBuf->curPtr(), frameSize) ) {
// The packet is ready to be sent now
sendPacketIfNecessary();
} else {
// There's room for more frames; try getting another:
packFrame();
}
} ? end else ?
} ? end afterGettingFrame1 ?
//end afterGettingFrame1
```



當封包有上述四種情形之一時便會進入 **sendPacketIfNecessary()** 函數，這四種傳送封包的條件解說如下：

1. 當封包大小 \geq 我們設定較喜歡傳送的封包大小，也就是 **setPacketSizes(1000,1448)** 中的 **1000**，則 **isPreferredSize()** 回傳為真，目的是確保在讀到下一個 VOP 前，封包大小至少會大於 1000 Bytes。
2. 有溢位情形發生時，溢位加速處理函數便是特別為了這個情形所設置的。
3. 上一個 frame 的切割標記布林值為 TRUE (以及) 允許切割之後把其它的 VOP frame 再加進來(預設為 false)，合起來的意思就是『當 frame 被切割之後，如果遇到下一個 VOP

的 frame，我們不會把它給加進來』。

4.封包開始打包時還允許 frame 一直加進來(但目前程式預設為 true，故第四個判斷條件不會進入 `sendPacketIfNecessary()`)。

而在 `sendPacketIfNecessary()` 中包含了一個加速溢位處理的機制，其程式碼如下所述，其目的在於算出每一個封包的『起始位置、封包大小以及封包偏移量』，舉例來說我們有一個大小為 4000 Bytes 的 frame，故我們將會分成 3 個封包來傳送之，也就是 $P1=1436$ ， $P2=1436$ ， $P3=1128$ ，而 $P2$ 以及 $P3$ 我們可以看到都屬於溢位的封包，一般沒有經驗的作法我們會使用 `memmove()` 函數進行溢位資料的複製與搬移，而 `memmove()` 這個動作相對於使用指標自動指向下一個溢位資料在 frame 中的起始點自然是複雜許多。



```
if (fOutBuf->haveOverflowData()
    && fOutBuf->totalBytesAvailable() > fOutBuf->totalBufferSize()/2) {
    unsigned newPacketStart = fOutBuf->curPacketSize()
        - (rtpHeaderSize + fSpecialHeaderSize + frameSpecificHeaderSize());
    fOutBuf->adjustPacketStart(newPacketStart);
} ? end if ? else {
// Normal case: Reset the packet start pointer back to the start:
// 一般溢位的處理情形(VOP不大於29999的情形之下)
fOutBuf->resetPacketStart();
if (fNoFramesLeft) {
// We're done:
    onSourceClosure(this);
}
```

```
void OutPacketBuffer::resetPacketStart() {
    if (fOverflowDataSize > 0) {
        fOverflowDataOffset += fPacketStart;
    }
    fPacketStart = 0;
}
```

上面的程式碼中，`newPacketStart` 便是每一個封包的起始位址，`curPacketSize()` 會傳回目前封包的大小並減掉相關表頭的大小 (`rtpHeaderSize + fSpecialHeaderSize + frameSpecificHeaderSize()`)，也就是我們如果以 `setPacketSizes` 設定最大封包長度為

1448，則 $\text{newPacketStart} = 1448 - 12 = 1436$ ，因此我們便可以呼叫 `fOutBuf->resetPacketStart()`

來設置下一個封包的起始位址。

由此我們可以瞭解要發展出一個高效能的串流應用程式，如同安德定理(Amdahl's Law)告訴我們的，我們要作的便是把程式『最常用』的部分加速之。事實上這些溢位處理函數的程式碼並不會很難，也不會是我們主要談論到的重點，我們在這邊觀察的重點應該是 Live 的此種作法帶給我們的程式實作上的『觀念啟發』。也就是在串流程式開發完畢後，應該活用一些 profile 之類的軟體去觀察串流程式的哪一個部分執行時間最多，進而想辦法去改良之，因為這個改良帶給系統的增益最大。



第五章 串流程式開發手札

由前面第二章至第四章的剖析中我們瞭解了串流伺服器的核心組成，因此在開發串流媒體程式時，基於物件導向程式設計(OOP, Object-Oriented Programming)的前提下，我們便可以直接運用 Live555 所提供的類別(class)來協助程式的開發。

同時 Live555 也已經把串流伺服器的核心元件給模組化，舉例來說便是第二章的 RTSPServer 模組(結合了 socket 與 RTSP)以及第三章提到的封包包裝與傳送模組，如果我們覺得模組(Module)中某個元件(Class/Object/Instance)的效能不夠好，我們亦可以動手自己修正或加入新的演算法來增進串流系統的效能，這也是我們為何需要對 Live555 所提供的原始碼(source code)作如此深入追蹤的原因。

Live555 給與我們的收穫在於串流程式設計上的經驗以及實作上的指引，在 4.1 章節串流伺服器的多工中，我們瞭解了 select() 以及 fork() 所帶給串流系統效能的影響；並且使用 select() 的話其第五個時間參數為何要設成一百萬秒，試想我們有一個即時監控伺服器平台，如果這個平台剛好有一個 11.5 天的週期都沒有用戶端連線，在 11.6 天後此伺服器便無法再接收連線，這當然是一個串流系統上的 nontrivial issue。

而在 4.2 章節中的是如何實作 RTSP parser 的指引；在 4.5 章節中，一個 RTP 封包經過封裝演算法的處理後便會變成一個結構化封包。同時在 Live555 所選擇的封裝演算法中因為常遇到 frame 大於 MTU 的情形，故必須作溢位資料處理的機制，而 Live 亦針對這些溢位資料的處理改用對系統負擔比較低的作法來完成之，目的是增進串流系統的

執行效能(安德定理(Amdahl's Law)的應用)。

綜合以上結論，我們可以說 Live555 可以快速地替我們要開發的串程式加入所需要的核心模組，避免重新發明輪子的窘境，如果我們想要替其中一個核心模組甚至是元件作演算法的再昇華亦是一個很好的研究方向。



第六章 結論

由本論文的第二章到第四章中我們可以看出串流伺服器的實作上包含了很多領域的知識，例如網路程式設計、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，在有限的資源下使串流系統發揮最大的效能比。

而目前的多媒體世界也出現了相當多的即時影音串流應用，例如目前當紅的 youtube[11]，該網站允許使用者自行上傳影片，並由該網站作為一個串流伺服器向網路上的任一個角落傳送影片；亦有相當多的嵌入式系統已經結合了影音串流的軟體技術而成為很好的 Surveillance system，如 NUUO[12]以及 ATEME[13]，透過本論文的剖析後，其背後所使用的根本技術相信大家能有一個基本的認識，相信這也是本論文的貢獻所在。

而上述的三個軟體程式在實際使用過後發現它們在效能以及使用者介面上都有不錯的表現，這也意謂著除了本論文所提及的一些串流基本的技術外，它們還花了很多心力在於系統的穩定性、效能性以及使用者介面的開發，而這也是本論文未來可以更進一步去研究的幾個方向，同時串流程式軟硬體間的協同設計也將會是一個很好的研究方向。

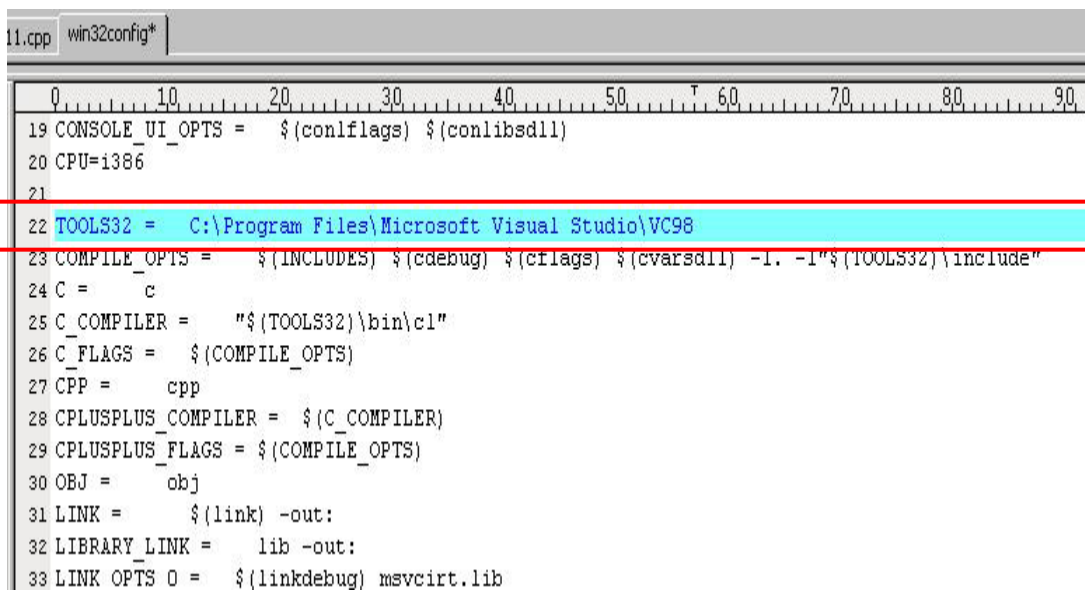
參考文獻

- [1] H. Schulzrinne, A. Rao, and R. Lanphier, "Real Time Streaming Protocol (RTSP)", RFC 2326 ,April 1998.
- [2] M. Handley, V. Jacobson, "SDP: Session Description Protocol",RFC2327, April 1998
- [3] <http://www.videolan.org/vlc/>
- [4] <http://www.live555.com/>
- [5] D. Hoffman,G. Fernando,"RTP Format for MPEG1/MPEG2 Video" , RFC 2250,January 1998 3.
- [6] H. Schulzrinne, Columbia University, "RTP: A Transport Protocol for Real-Time ", RFC3550. Titl, July 2003.
- [7] 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
- [8] ISO/IEC 14496-2:2001(E)(Page 32)
- [9] 張為棟，『整合式多媒體串流平台的發展與實作』，國立交通大學電信所碩士論文，95 學年度。
- [10]電信總局<http://www.dgt.gov.tw/Chinese/public-cares/12.5/12.5.2/lost-better.shtml>
- [11] <http://www.youtube.com>
- [12] <http://www.nuuu.com/>
- [13] <http://www.ateme.com>
- [14] H. Schulzrinne , "RTP Profile for Audio and Video Conferences with Minimal Control",RFC1890 ,January 1996
- [15] UNIX 網路程式設計網路應用程式設計介面：Socket 與 XTI
作者：W.Richard Stevens 譯者：林慶德 出版:培生

附錄一

1. How to configure and build the code on Windows

1. Unpack and extract the '.tar.gz' file (using an application such as "WinZip").
2. If the 'tools' directory on your Windows machine is something *other than* "c:\Program Files\DevStudio\Vc", change the "TOOLS32 =" line in the file "win32config".



```
11.cpp win32config*
0 10 20 30 40 50 60 70 80 90
19 CONSOLE_UI_OPTS = $(coniflags) $(conlibsdl)
20 CPU=i386
21
22 TOOLS32 = C:\Program Files\Microsoft Visual Studio\VC98
23 COMPILE_OPTS = $(INCLUDES) $(cdebug) $(cflags) $(cvarsdll) -I. -I"${TOOLS32}\include"
24 C = c
25 C_COMPILER = "${TOOLS32}\bin\cl"
26 C_FLAGS = $(COMPILE_OPTS)
27 CPP = cpp
28 CPLUSPLUS_COMPILER = $(C_COMPILER)
29 CPLUSPLUS_FLAGS = $(COMPILE_OPTS)
30 OBJ = obj
31 LINK = $(link) -out:
32 LIBRARY_LINK = lib -out:
33 LINK_OPTS 0 = $(linkdebug) msvcirt.lib
```

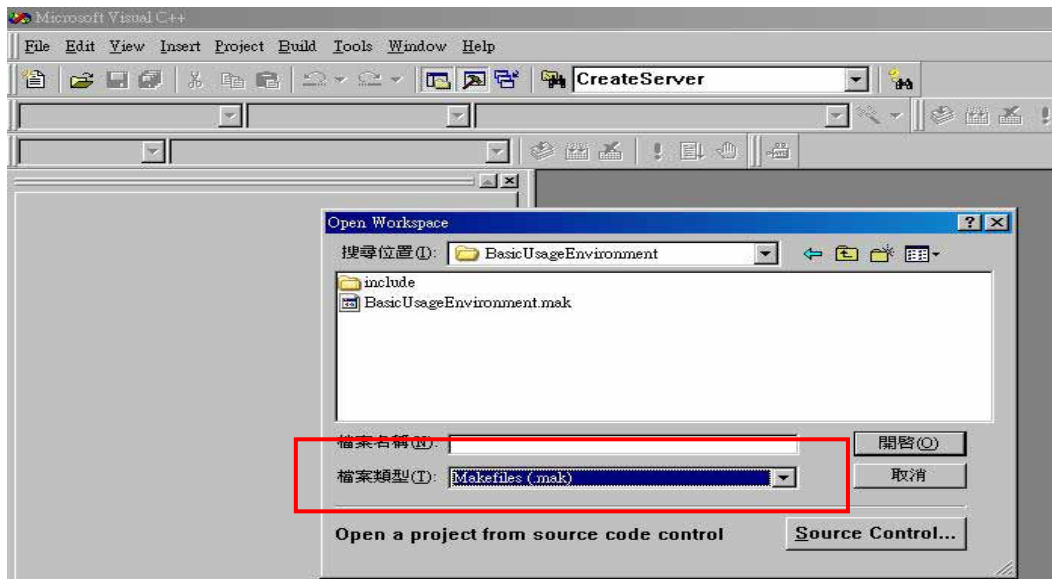
3. In a command shell, 'cd' to the "live" directory, and run genWindowsMakefiles

This will generate - in each subdirectory - a "*.mak" makefile suitable for use by (e.g.) Microsoft Visual Studio.

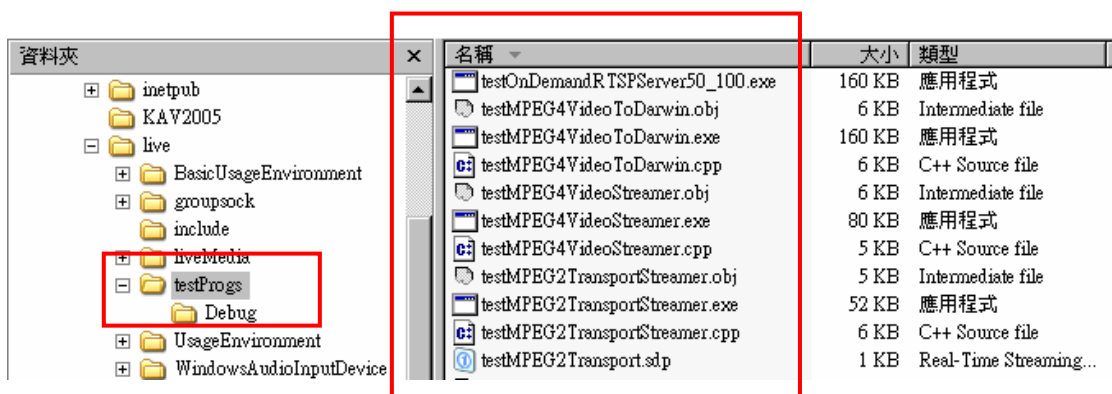
- Alternatively, if you're starting from a Unix machine, you can generate the Windows Makefiles by running ./genWindowsMakefiles

(after - if necessary - changing the "TOOLS32 =" line in the file "win32config", as noted above). Then, copy the "live" directories and its subdirectories (in ASCII mode) to a Windows machine.

- To use these Makefiles from within Visual Studio, use the "Open Workspace" menu command, then (in the file selection dialog) for "Files of type", choose "Makefiles (.mak)". Visual Studio should then prompt you, asking if you want to use this Makefile to set up a new project. Say "OK".
-



- Note that you will need to build each of the "UsageEnvironment", "groupsock", "liveMedia", and "BasicUsageEnvironment" projects *first*, before building "testProgs".
- If you wish, you can build the "WindowsAudioInputDevice" project also.
- Doug Kosovic notes: Visual C++ 2003 no longer comes with the old I/O Streams headers iostreams.h et al, or the corresponding library msvcirt.lib. So anybody trying to build the LIVE555 code with VC++ 2003 might find the following useful: A non-sourcecode modification workaround for VC++ 2003 is to copy the missing headers and msvcirt.lib from VC++ 2002. In file 'win32config' add an extra -I switch to COMPILE_OPTS to find the old headers and a -LIBPATH: switch to LINK_OPTS_0 to find msvcirt.lib.



After testProgs.mak compiled ,there will be executable programs generated in testProgs folder.

2.How to configure and build the code on Unix (including Linux, Mac OS X, QNX, and other Posix-compliant systems)

The source code package can be found (as a ".tar.gz" file) [here](#). Use "tar -x" and "gunzip" (or "tar -xz", if available) to extract the package; then cd to the "live" directory. Then run

```
./genMakefiles <os-platform>
```

(**os-platform** 指的是鍵入你所使用的作業系統)

where *<os-platform>* is your target platform - e.g., "linux" or "solaris" - defined by a "config.*<os-platform>*" file. This will generate a Makefile in the "live" directory and each subdirectory. Then run "make".

(完成"make"之後便會出現已經編譯可執行的範例程式)

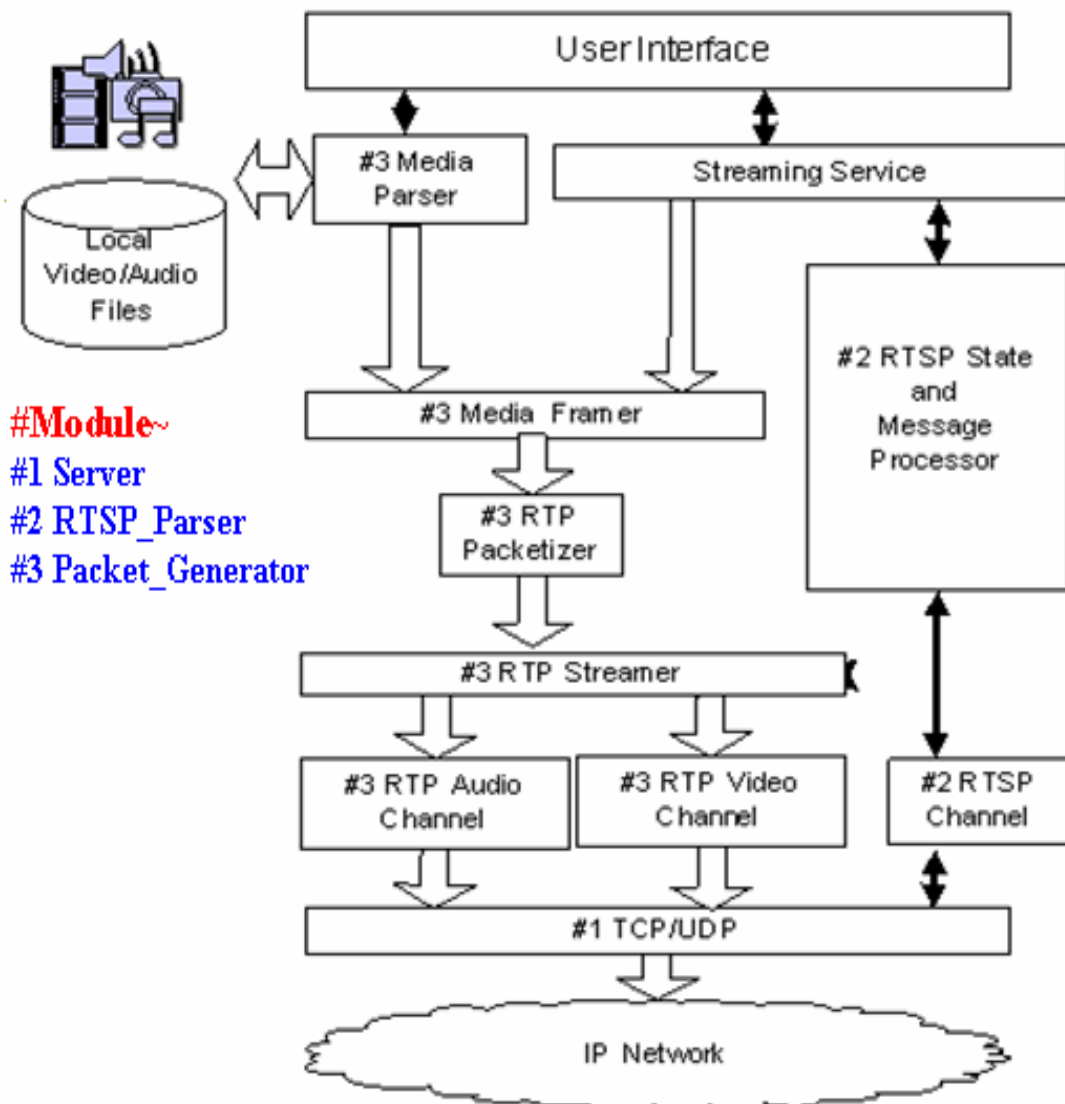


附錄二 Overview of Live555 Server's Module、Architecture and Performance

1. Live555 Server's module and Function

Live555 Module	Including Important Function		
#1 Server	Socket	Bind	Fcntl
	Listen	FD_ZERO	Select
	FD_ISSET	Accept	Recvfrom
	Sendto	Send	Close
#2 RTSP_Parser	parseRequestString	handleCmd_OPTIONS	handleCmd_DESCRIBE
	handleCmd_SETUP	handleCmd_PLAY	handleCmd_PAUSE
	handleCmd_TEARDOWN		
#3Packet_Generator	createNewStreamSource	MPEG4VideoStreamFramer	setPacketSizes
	buildAndSendPacket	packFrame	isTooBigForAPacket
	sendPacketIfNecessary		

2. Live555 Server's Streaming Architecture



自傳

吳宗修，民國六十七年生於台南市。民國九十四年一月畢業於國立台北科技大學電機工程系，同年二月進入國立交通大學IC設計產業碩士專班就讀，從事影音多媒體串流相關研究。民國九十六年一月取得碩士學位，論文題目是「串流伺服器特性剖析」。

研究興趣是C/C++程式開發，希望有一天可以搞懂其中物件導向以及資料結構的奧妙；生活興趣是電影、太空戰士(史克威爾系列)電玩、籃球以及羽球。

工作方面，曾在威盛電子以及金寶電子服務，分別任職於系統驗證部門(SV)以及硬體設計部門(CI-2)。

