國 立 交 通 大 學

電信工程學系

碩 士 論 文

整合式多媒體串流平台的發展與實作

Development and Implementation of Integrated Multimedia Streaming Platform

研究生：張為棟

指導教授：張文鐘 博士

中 華 民 國 九十五 年 八 月

整合式多媒體串流平台的發展與實做

Development and Implementation of Integrated Multimedia Streaming Platform

研 究 生：張為棟　　Student：Wei-Tung Chang

指導教授：張文鐘　　Advisor：Wen-Thong Chang

國 立 交 通 大 學

電信工程學系

碩 士 論 文

A Thesis

Submitted to Department of Communication Engineering

College of Electrical and Computer Engineering

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of Master

in

Communication Engineering

August 2006

Hsinchu, Taiwan, Republic of China

中華民國九十五年八月

整合式多媒體串流平台的發展與實做

研究生：張為棟　　指導教授：張文鐘　博士

國立交通大學電信工程學系碩士班

摘要

多媒體串流技術，使消費者在下載多媒體檔案的同時可以觀賞到已收到的部份，此項技術使得使用者不需要花費下載時間與硬體空間來儲存多媒體檔案。

多媒體串流技術，包括三個部份：1、檔案包裝格式，舉例來說*.avi 、*.mp4、*.wmv、*.mpg，這些都是包裝格式的一種，包裝格式的檔頭會說明如何從這些檔案格式中拆解出解碼器所需要的聲音流與影像流。2 影音壓縮解壓縮技術，舉例來說 mpeg-4、mp3、AAC 等都是影音壓縮解壓縮的一種，藉由這些影音壓縮技術可以把原始影音檔案大小壓縮成原來的數十分之一，３、網路串流技術，利用網路讓遠端接收到多媒體檔案並播放，所要求的為檔案播放的及時性與平順性。因此，一個完美的多媒體串流平台，基本上必須看得懂所有的檔案包裝格式與網路串流格式，同時必須準備各種影音壓縮技術來對影像流與聲音流進行編碼及解碼，這種看似複雜的串流平台在自由軟體基金會及全球軟體工程師的努力下正逐漸的建構當中，本論文的第三章將探討一個開放原碼的自由軟體 FFmpeg。

多媒體串流的一項重要應用為安全監控，本論文將建構一套雙攝影機模組的即時遠端安全監控系統，利用 VFW SDK 擷取雙攝影機影像並將其壓縮成 mpeg-4 simple profile，再利用串流平台串流播放之，由於將即時影像來源壓縮成 mpeg-4 simple profile 將會消耗大量 cpu 資源，本論文嘗試使用硬體代替軟體進行 mpeg-4 simple profile 的壓縮，至於解壓縮部份由於銷耗 cpu 資源有限因此仍以軟體進行解壓縮，而研究中也發現雖然軟、硬體皆是使用 mpeg-4 simple profile 但會因為 codec tool 的差異而造成解碼錯誤，本論文將解決差異所造成的解碼錯誤，並成功的串流由硬體所壓縮出來的影音檔案。

Development and Implementation of Multimedia Streaming Platform

Student : Wei-tung Chang Advisor : Dr. Wen-Thong Chang

Department of Communication Engineering
National Chiao Tung University

Abstract

Streaming technology allows people to enjoy the multimedia contents while downloading them. With the advantages of this technology, users don't need to spend extra hours on downloading multimedia contents or spare memory space to save them.

Streaming technology consists of three components. 1. File Format, for example, *.avi, *.mp4, *.wmv, and *.mpg. Decoder can parse video stream and audio stream via analyzing the header of a file format. 2. Codec, for example, mpeg-4, mp3, and AAC. By using Codec, users can compress original file several folds. 3. Network Streaming Technology. User can present smooth and real-time multimedia contents via network. In order to develop a perfect multimedia streaming platform, one have to realize all kinds of file formats and network streaming technology. In addition, the perfect platform has to possess all kinds of Codec to decode or encode multimedia contents. This complex perfect multimedia streaming platform is being built gradually by global software engineers. We will introduce an open source software which is produced by free software foundation in chapter 3.

Surveillance system is an important application in streaming technology, we will build a real-time two camera model multimedia streaming surveillance system in this thesis, we use VFW SDK to capture this two camera image and encode image to mpeg-4 simple profile, and stream the mpeg-4 video stream via internet.

In order to decrease the cpu loading, we try to use hardware mpeg-4 simple profile encoder to replace the software one. Although both hardware encoder and software decoder are mpeg-4 simple profile codec, the difference of codec tool will cause decoder error, we will resolve this problem in this thesis and successfully streaming multimedia contents which is made by hardware encoder,

致謝

首先，我要感謝我的指導教授：張文鐘老師，感謝老師兩年來辛苦的指導，不管在課業或是日常生活當中，老師的指導使我們獲益良多。

接者我要感謝我的實驗室夥伴，兩年多來認識了很多一起鑽研多媒體通訊領域的好朋友，不管是建華、其瑩、心賢、智維、義皓或是同屆的同學、陳博、小林、邱董、飄髮哥，以及新進來的學弟準 leader、阿修等，感謝這些夥伴的照顧與幫忙。

當然，還要感謝我的大學朋友們，在我最難過的時候安慰我鼓勵我，讓我能繼續完成該完成的目標，尤其是書安，幫了我許多忙，還有還有 Selena，讓我度過很多快樂的日子。

最後我要感謝我的家人，讓我這兩年可以沒有後顧之憂的唸書和做研究。

祝福我所有的朋友，老師，家人，身體健康、萬事順利。

目錄

圖列

表列

第一章　簡介

1.1 背景介紹

隨著網路傳輸頻寬的不斷成長，網際網路的應用已經從原本的文字或圖片傳輸，轉變成結合聲音、影像的互動式多媒體網路，這些多媒體資訊藉由稱為串流技術的新傳輸科技在網際網路上迅速擴張。

串流是一種具有即時〈Real-time〉特性的多媒體傳輸技術，其特色為客戶端不需要完整下載多媒體檔案，只需要先下載極小一部份資料即可開始播放，其優點除了減少使用者下載所需的時間外，更可節省磁碟儲存空間，達到智慧財產權的保護等。

目前串流媒體在網際網路上的傳輸方式主要有兩種方式，一種是以 HTTP/TCP 為基礎，利用 HTTP 協定可以讓串流媒體通過防火牆的阻礙，但缺點是 TCP 通訊協定將會導致傳輸速度的減緩，進而增加串流媒體所需要的頻寬，另一種方式則是使用 RTP/UDP，Real Time Protocol (RTP)，是由 Internet Engineering Task Force (IETF)所制定，並定義於 RFC-3550 ，根據[1]，RTP 提供點對點的傳輸服務，包括 payload type identification、sequence numbering、timestamping，利用 sequence number 來進行封包重建，而 timestamp 則記錄著各影像、聲音封包所應播放的時間，如此即可增進即時傳輸的品質。

另外，在目前的商業市場中，由於串流媒體伺服端的規格並未有

一個權威性標準產生，以至於各家廠商紛紛推出以自己的標準或是特有通訊協定所開發的串流媒體伺服器，導致一套串流伺服器只能對應一套編碼/解碼格式，綜觀市場目前串流媒體的三大主流分別是 RealNetworks.com 的 RealSystemR[2] 、 Microsoft 的 Media ServicesR[3]以及頻果電腦主推的 Quick Time ServerR[4]。也由於沒有一套標準的編碼解碼器(Codec)及串流媒體，Open Source 將在整合出重量級串流媒體伺服器上面扮演重要的角色。

1.2 研究動機

一個多媒體串流平台,包括 server 與 client，client 向 server 要求某種多媒體影音檔案，server 則對 client 所要求的多媒體影音檔案進行 Parse 並包裝成特殊網路封包格式--RTP Packet 進行網路傳送，所謂的 Parse 即是將多媒體影音檔案中的影像流與聲音流抽取出來，同時 client 必須對這影像流與聲音流進行解碼與播放，由上述的過程我們可以知道 client 與 server 存在某種默契,即 server 必須有能力抽取出 client 所要求的多媒體影音檔案中影像流與聲音流並包裝成 RTP Packet，而 client 必須有從 RTP Packet 中找出影音資料同時解碼這個影音資料的能力，現在的問題為當 client 所要求的多媒體影音檔案，server 沒有能力 Parse，或是當 Server 抽取出某種多媒體影音檔案的聲音流與影像流後，client 沒有相對應的解碼器對這些聲音流與影像

流進行解碼，此外，目前我們是使用 RTSP 當作 client 與 server 的溝通橋樑，不管是 client 要求暫停、快轉、倒轉甚至是要求某種影音檔案，都是藉由 RTSP Command 來進行溝通，server 也藉由 RTSP Command 來回應 client 的要求，當 server 看不懂 client 的 RTSP 指令或是 server 回應的 RTSP Command client 看不懂，這些都將造成串流的失敗，這些相容性問題，常常發生於不同串流平台之間的溝通，例如蘋果的 QuickTime Server 沒有辦法與 RealPlayer 互通。

要解決此相容性的問題，我們必須讓 server 有能力看得懂各種影音資料，任何多媒體影音資料都是由 0101…..這些 bitstreams 所組成，藉由讓 sever 看得懂這些 bitstreams 的物理意義，我們可以對多媒體影音資料進行 Parse，server 抽取出影像流與聲音流後必需包裝成適當的 RTP Packet，適當的 RTP Packet 可以幫助我們在錯誤發生時進行錯誤回復，client 則從這些 RTP Packet 中取得影音資料，再根據影音資料的種類給予適當的解碼器進行解碼與播放，因此我們必須在 Client 與 Server 端建立影音格式與解碼編碼的資料庫，在研究中也發現，光是建立一種影音資料庫是是不夠的，藉由建立各種多媒體影音資料庫的 Link List，可以處理各式多媒體影音資料。同時我們也必須試著建立一個符合標準的 RTSP 指令，使 client 與 server 可通用於目前流通於市面上的各種多媒體串流平台。

第二章 MPEG-2 System and M4v header syntax

MPEG-2 system 定義 DVD 與數位廣播電視的包裝格式，但為了增加壓縮比，在許多多媒體的應用上，藉由 mpeg-4 codec 取代 mpeg-2 codec，本章將介紹由 ISO 13818-1[5]定義的 mpeg-2system，包括現在 DVD 所使用的 mpeg-2 program stream 與數位廣播電視所使用的 mpeg-2 transport stream，在了解 mpeg-2 system 後我們便可以從中抽取出影像流與聲音流，抽取出來的影像流是 mpeg-4 simple profile，接下來為了找出這個影像流中各個 frame 的起始及結束位置，我們將研究 mpeg-4 simple profile 中各個語法的意義，在本章的最後將拆解實際的例子。

2.1 MPEG-2 System Introduction

MPEG-2 Standards 制訂於 Motion Picture Experts Group(MPEG)並由 International Standards Organisation(ISO)所出版，目前共分為九個部份列於表 1，其中的 ISO 13818-1[5]定義了兩種 Multiplexing 的格式，Program stream 以及 Transport stream，參考圖 1[5]，所謂的 Multiplexing 是指將壓縮好的影像流、聲音流、控制資料、使用者資料等結合成單一數據流，使此單一數據流能適用於儲存或者是傳輸，以下將就此兩種格式分別討論。

ISO/IEC DIS 13818_1	System—描述 video audio 同步與多路技術
ISO/IEC DIS 13818-2	Video—video 壓縮
ISO/IEC 13818-3	Audio—audio 壓縮
ISO/IEC DIS 13818-4	Compliance testing
ISO/IEC 13818-5	模擬軟體
ISO/IEC 13818-6	Extensions for DSM-CC is a full software implementation
ISO/IEC 13818-7	Advanced Audio Coding
ISO/IEC 13818-9	Extension for real time interface for systems decoders
ISO/IEC 13818-10	DSM-CC 規範

表 1 MPEG-2 Standards

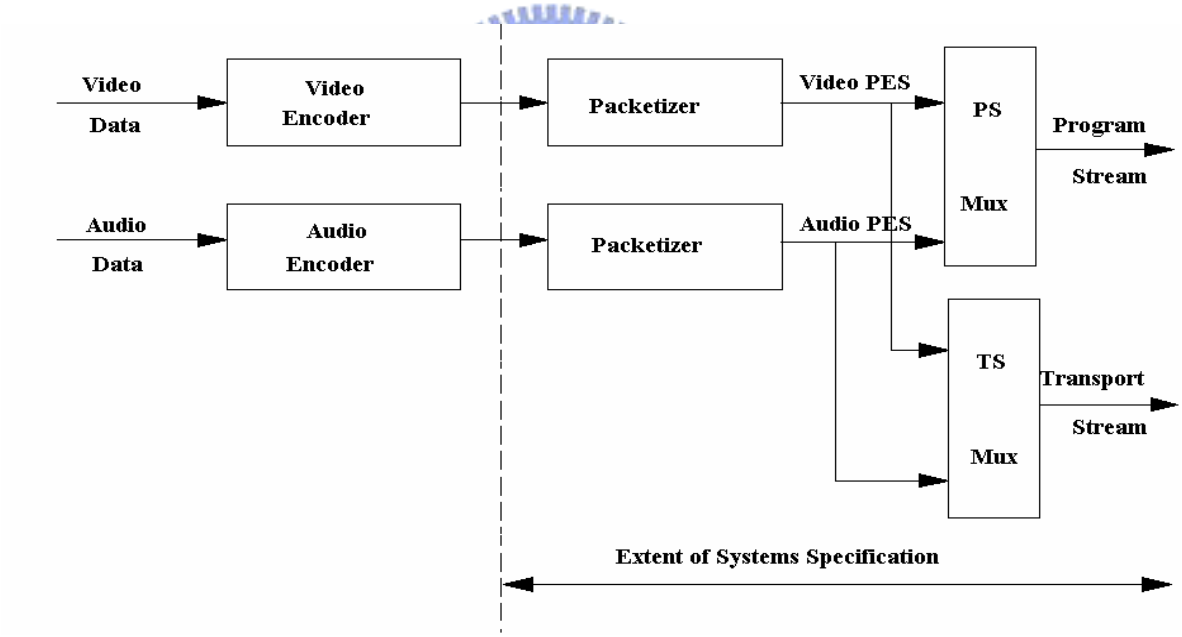


圖 1 Model of MPEG-2 System[5]

從圖 1，我們可以看到 MPEG-2 System Multiplexing 的過程，首先影像來源與聲音來源藉由 Viedo and Audio Encoder 壓縮成 viedo and audio elementary stream 而 Packetizer 則將這些 elementary stream 加上

適當的 PES Header 形成 PES Packet，同時 Packetizer 也會建立說明這些 elementary stream 位於檔案何處與種類的說明文件，把這些說明文件加上適當的 PES Header 也形成 PES Packet，所謂適當的 PES Header 如圖 2 所示，PES Header 開始於 3bytes 的 pes_start_code(00 00 01)，1byte 的 Stream_ID 及兩 bytes 的 PES Packet length，適當的 PES Header 即 Packtizer 會根據 payload 的種類給予適當的 stream_id，根據表 2[5] 可知，當 Stream ID 介於 0xC0~0xDF 之間則此 Payload 為 audio stream，當 Stream ID 介於 0xE0~0xEF 之間則此 Payload 為 video stream。

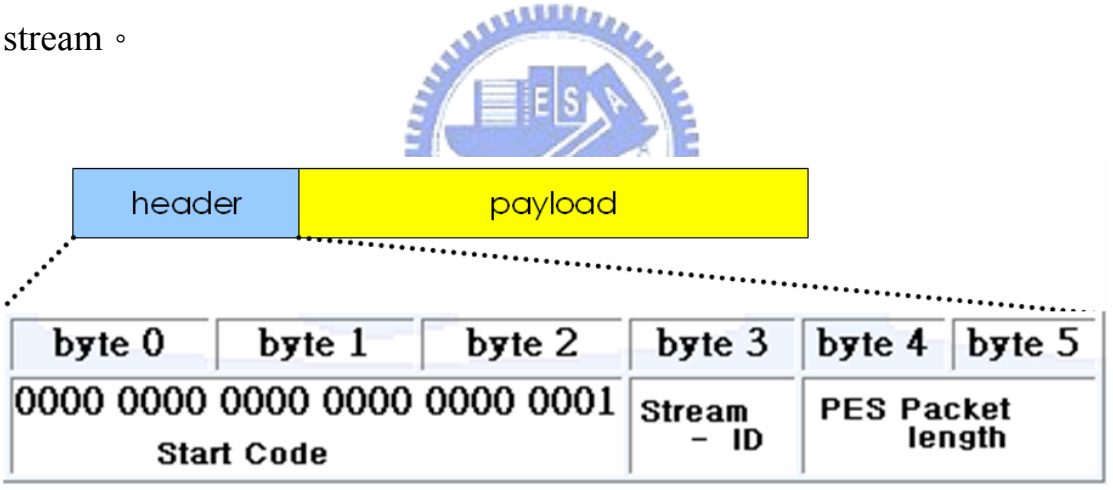


圖 2　PES Packet

Stream_id	Stream codind
1011 1100	Program_stream_map
1011 1101	Private_stream_1
1011 1110	Padding_stream
1011 1111	Private_stream_2
110x xxxx	ISO/IEC 13818-3 or ISO/IEC 11172-3 audio stream
1110 xxxx	ITU-T Rec H.262 | ISO/IEC 13818-2 or ISO/IEC 11172-2 video stream
1111 0000	ECM_stream

1111 0001	EMM stream
1111 0010	ITU-T Rec.H.222.0 | ISO/IEC 13818-1 Annex A or ISO/IEC 13818-6 DSMCC stream
1111 0011	ISO/IEC_13522_stream
1111 0100	ITU-T Rec H.222 | type A
1111 0101	ITU-T Rec H.222 | type B
1111 0110	ITU-T Rec H.222 | type C
1111 0111	ITU-T Rec H.222 | type D
1111 1000	ITU-T Rec H.222 | type E
1111 1001	ancillary_stream
1111(1010-1110)	reserved data stream
1111 1111	program_stream_directory

表 2 [5]Stream_id assignments

PES header 除了告訴 decorder 其 Payload data 為影像流或者是聲音流外，他還必須告知 decorder 此影像流或聲音流的格式。當 stream_id = 1011 1100 從表 2 可知此 PES 為 Program_stream_map，從表 3、表 4 中可以得知 Program stream map 藉由 stream_type、elementary_stream_id 來告知某一 stream_id 其 stream_type 為何。舉例來說，一個 mpeg-4 simple profile 的 mpeg-2 program stream，我們可以找到很多 PES header 中的 stream_id =0xE0 從表 2[5]可得知這代表這些 PES 的 Payload data 為影像流，也可以找到 stream_id = 0xBC 的 PES 根據表 2[5] 這代表這個 PES 為 Program Stream map，從這個 Program Stream map 的 Payload data 中我們將找到 stream_type = 16、elementary_stream_id = 0xE0，根據表 4[5]，stream_type = 16 為保留的型別，其定義為 mpeg-4 video codec。因此 stream_type =16、

elementary_stream_id =0xE0 代表只要影像流其　PES header 中的 stream id 為 0xE0 則此影像流為 mpeg-4 video codec。

Syntax	No. of bits	Mnemonic
program_stream_map() {		
packet_start_code_prefix	**24**	**bslbf**
map_stream_id	**8**	**uimsbf**
program_stream_map_length	**16**	**uimsbf**
current_next_indicator	**1**	**bslbf**
reserved	**2**	**bslbf**
program_stream_map_version	**5**	**uimsbf**
reserved	**7**	**bslbf**
marker_bit	**1**	**bslbf**
program_stream_info_length	**16**	**uimsbf**
for (i=0;i<N;i++) {		
descriptor()		
}		
elementary_stream_map_length	**16**	**uimsbf**
for (i=0;i<N1;i++) {		
stream_type	**8**	**uimsbf**
elementary_stream_id	**8**	**uimsbf**
elementary_stream_info_length	**16**	**uimsbf**
for (i=0;i<N2;i++) {		
descriptor()		
}		
}		
CRC_32	**32**	**rpchof**
}		

表 3[5] Program Stream map

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

表 4 [5]　Stream type assignments

2.2 MPEG-2 Program Stream

MPEG-2 Program Stream 就是 DVD 儲存的格式，根據 ISO13818-1[5] Program Stream 具有以下幾點特性

1 適用於儲存檔案格式，不適合用於傳輸檔案格式。

2 所有的 elementary streams(video、audio、data 等)皆具有相同的時間基準(time base)。

3 每一個 Program Stream 只能包含一個節目。

4 由於 Packet Size 比較大而沒有固定大小，因此適用於較少發生錯誤的環境，因為 decorder 很難去預測封包的開始及結束點。

由圖 1、圖 3 可以知道，要形成一個完整的 Program Stream 必須先將各種不同的 elementary stream 加上其應該有的檔頭，形成 PES (Packetized Elementary Stream)，再將許多 PES 集合起來並藉由加上 Pack header 形成一個 Pack，最後再將數個 Pack 集合成 Programe Streams，我們的目的是要拆解出藏在 PES payload 中的 mpeg-4 elementary stream 。

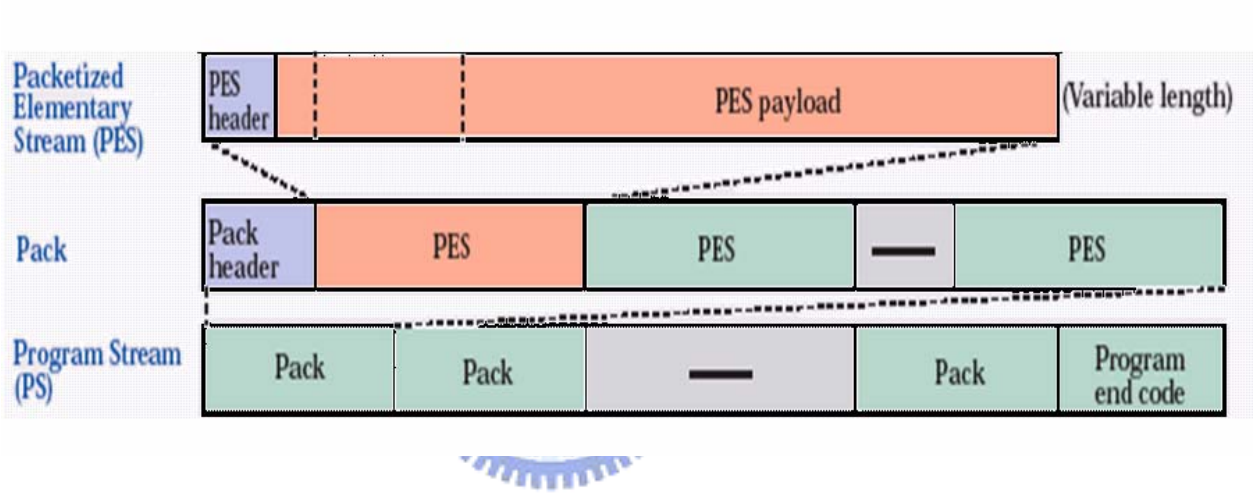


圖 3 Relationship between PES packets and Program Stream

2.2.1 MPEG-2 Program Stream—PACK and System header

根據 ISO 13818-1[6]以及圖 3，PES 在結合成 Program Stream 之前必須再加上 PACK header，PACK header 由 pack_start_code 0x000001BA 開始，表 5 為 PACK Header 的 syntax，此外當一個 program stream 的 PACK 中包涵 frame 的起始位置，則此 PACK 將會在 PACK Header 之後再加上 System Header 此 System Header 將描述這個 frame 所有影像流其 PES Header 中 Stream_ID 為何如表 6 所示

Syntax	No. of bits	Mnemonic
pack_header() {		
pack_start_code (00 00 01 BA)	32	bslbf
'01'	2	bslbf
system_clock_reference_base [32..30]	3	bslbf
marker_bit	1	bslbf
system_clock_reference_base [29..15]	15	bslbf
marker_bit	1	bslbf
system_clock_reference_base [14..0]	15	bslbf
marker_bit	1	bslbf
system_clock_reference_extension	9	uimsbf
marker_bit	1	bslbf
program_mux_rate	22	uimsbf
marker_bit	1	bslbf
marker_bit	1	bslbf
reserved	5	bslbf
pack_stuffing_length	3	uimsbf
for (i=0;i<pack_stuffing_length;i++) {		
stuffing_byte	8	bslbf
}		
if (nextbits() == system_header_start_code) {		
system_header ()		
}		
}		

表 5 PACK Header Syntax

Syntax	No. of Bits	Mnemonic
system_header () {		
system_header_start_code (00 00 01 BB)	32	bslbf
header_length	16	uimsbf
marker_bit	1	bslbf
rate_bound	22	uimsbf
marker_bit	1	bslbf
audio_bound	6	uimsbf
fixed_flag	1	bslbf
CSPS_flag	1	bslbf
system_audio_lock_flag	1	bslbf
system_video_lock_flag	1	bslbf
marker_bit	1	bslbf
video_bound	5	uimsbf
packet_rate_restriction_flag	1	bslbf
reserved_byte	7	bslbf
while (nextbits () == '1') {		
stream_id	8	uimsbf
'11'	2	bslbf
P-STD_buffer_bound_scale	1	bslbf
P-STD_buffer_size_bound	13	uimsbf
}		
}		

表 6 System Header Syntax

2.3 MPEG-2 Transport Stream

MPEG-2 Transport Stream 就是 HDTV 的傳輸格式，根據 ISO13818-1 Transport Stream 具有以下幾點特性：

1 一個 Transport Stream 可以擁有多個節目(program)

2 不同的節目可以擁有不同的時間基準(time base)]

3 擁有固定長度的封包大小，使解碼器能夠很容易的找出每一張照片的開始和結束，同時也比較容易在掉封包後進行補救

參考圖 4，與 Program Stream 不同，Transport Stream 藉由將 PES 以每 184bytes 為一個單位，再加上 4bytes 的 TS header 所組成的，參考表 7[5]，TS header 由 8bits 的 sync_byte 0x47 開始，另外還有一個 13bits 的 PID，PID 的目的是告訴解碼器這個 Transport Stream packet 的 payload 是什麼種類，由表 8[5]可以知道，我們有興趣的影像流與聲音流其 PID 將會介於 0x0010 與 0x1FFE 之間。

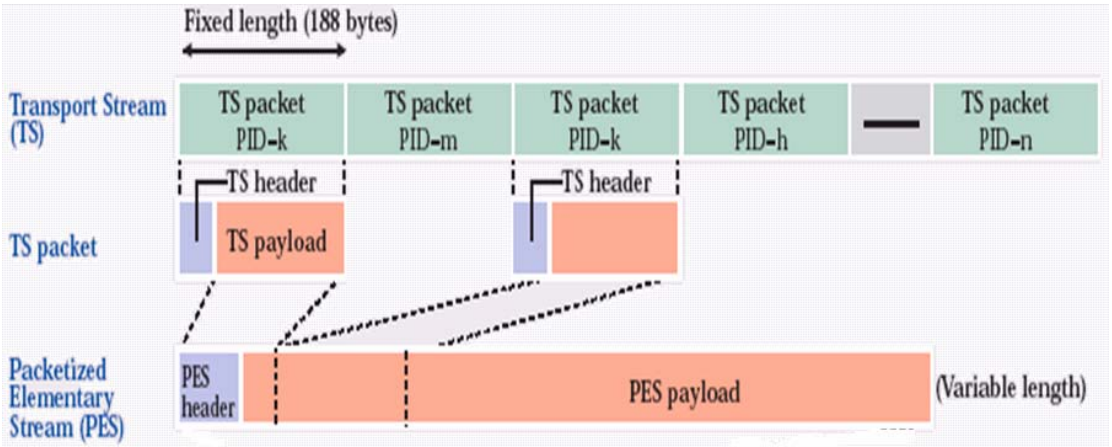


圖 4Relationship between PES and Transport Stream

Syntax	No. of bits	Mnemonic
transport_packet(){		
sync_byte	**8**	**bslbf**
transport_error_indicator	**1**	**bslbf**
payload_unit_start_indicator	**1**	**bslbf**
transport_priority	**1**	**bslbf**
PID	**13**	**uimsbf**
transport_scrambling_control	**2**	**bslbf**
adaptation_field_control	**2**	**bslbf**
continuity_counter	**4**	**uimsbf**
if(adaptation_field_control=='10' || adaptation_field_control=='11'){		
adaptation_field()		
}		
if(adaptation_field_control=='01' || adaptation_field_control=='11') {		
for (i=0;i<N;i++){		
data_byte	**8**	**bslbf**
}		
}		
}		

表 7[5] Transport Stream header

value	description
0x0000	Program Association Table
0x0001	Conditional Access Table
0x0002-0x000F	reserved
0x0010 … 0x1FFE	may be assigned as network_PID, Program_map_PID, elementary_PID, or for other purposes.
0x1FFF	Null packet

表 8 [5]PID Table

Transport Stream 與 Program Stream 一樣，必須要告訴解碼器此 Transport Stream 所攜帶的聲音流與影像流的格式，當 Transport Stream header 中的 PID = 0x0000 時，根據表 8[5]，這個 Transport Stream 的性質為 Program Association Table，表 9[5]指出 Program Association Table 有哪些成員，其中解碼器會參照表 9[5]中的 program_map_PID，

也就是當之後解碼器遇到 Transport Stream header 中的 PID 等於 program_map_PID 他就知道這個 packet 有 program_map 的資訊，program_map 會告知解碼器某一個 PID 其影像流與聲音流是用何種壓縮格式，表 10[5]指出 program_map_table 有哪些成員，從表 10[5]可以知道 Transport Stream 與 Program Stream 的 map table 相似，都是利用 stream_type 以及 elementary_PID 來告訴解碼器某一個 PID 其影像流與聲音流的壓縮格式，至於 stream_type 所代表的壓縮格式則與表 4[5]相同。

Syntax	No. of bits	Mnemonic
program_association_section() {		
table_id	8	uimsbf
section_syntax_indicator	1	bslbf
'0'	1	bslbf
reserved	2	bslbf
section_length	12	uimsbf
transport_stream_id	16	uimsbf
reserved	2	bslbf
version_number	5	uimsbf
current_next_indicator	1	bslbf
section_number	8	uimsbf
last_section_number	8	uimsbf
for (i=0; i<N;i++) {		
program_number	16	uimsbf
reserved	3	bslbf
if(program_number == '0') {		
network_PID	13	uimsbf
}		
else {		
program_map_PID	13	uimsbf
}		
}		
CRC_32	32	rpchof
}		

表 9 [5]Program association section

Syntax	No. of bits	Mnemonic
TS_program_map_section() {		
table_id	8	uimsbf
section_syntax_indicator	1	bslbf
'0'	1	bslbf
reserved	2	bslbf
section_length	12	uimsbf
program_number	16	uimsbf
reserved	2	bslbf
version_number	5	uimsbf
current_next_indicator	1	bslbf
section_number	8	uimsbf
last_section_number	8	uimsbf
reserved	3	bslbf
PCR_PID	13	uimsbf
reserved	4	bslbf
program_info_length	12	uimsbf
for (i=0; i<N; i++) {		
descriptor()		
}		
for (i=0;i<N1;i++) {		
stream_type	8	uimsbf
reserved	3	bslbf
elementary_PID	13	uimsnf
reserved	4	bslbf
ES_info_length	12	uimsbf
for (i=0; i<N2; i++) {		
descriptor()		
}		
}		
CRC_32	32	rpchof
}		

表 10 [5]Transport Stream program map table

2.4 M4v header syntax

從 2.2、2.3 中，目前我們已經知道 Program Stream 以及 Transport Stream 分別將影像流資料與聲音流資料藏在哪裡，但對串流來說，我們還必須知道每張 frame 的開始以及結束，那是因為只要是同一張 frame 的資料我們都要分配給他相同的 RTP timestamp，(關於 RTP 我們將再第四章進行討論)，因此本章將描述壓縮好的 mpeg-4 elementary stream 也就是 M4v，其檔頭包含哪些重要的消息。

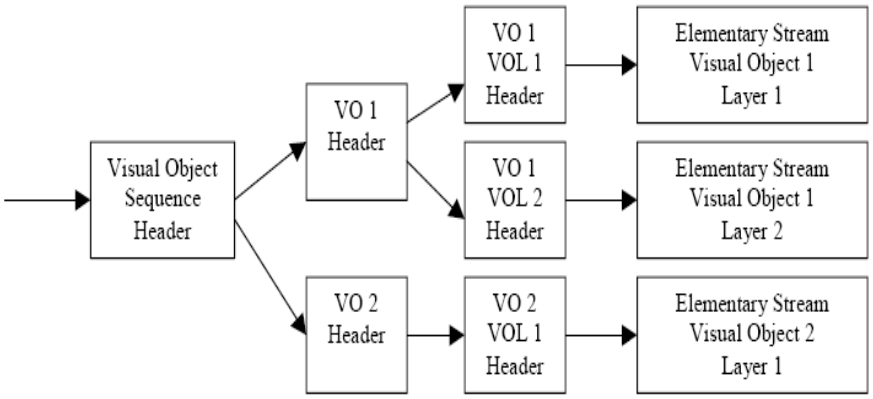


圖 5 Visual Informaton-Logical Structure[6]

圖 5[6]指出了一個完整的 M4v，是由 Visual Object Sequence Header、Visual Object Header、Video Object Header、Video Object Layer Header 所組成，這些 header 將會夾帶解碼器所需要的一些訊息，例如照片的尺寸、量化矩陣的參數等資料，我們稱這些資料為 Elementary Stream Descriptors(ESDS data)，根據 ISO 14496-2[6]，ESDS data 結束後將進入 Video Object Plane，也就是每張 frame 的起始位置，我們就是藉由 Video Object Plane 的 header 來判斷這張照片是 I 或 P frame 以及每張 frame 的起始位置。圖 6 顯示一個完整 m4v 檔是由 ESDS data 以及 I frame 與 P frame 組合而成。

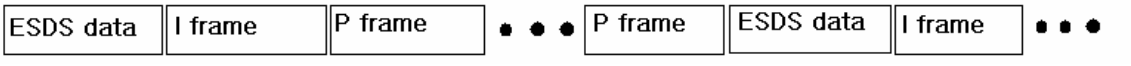
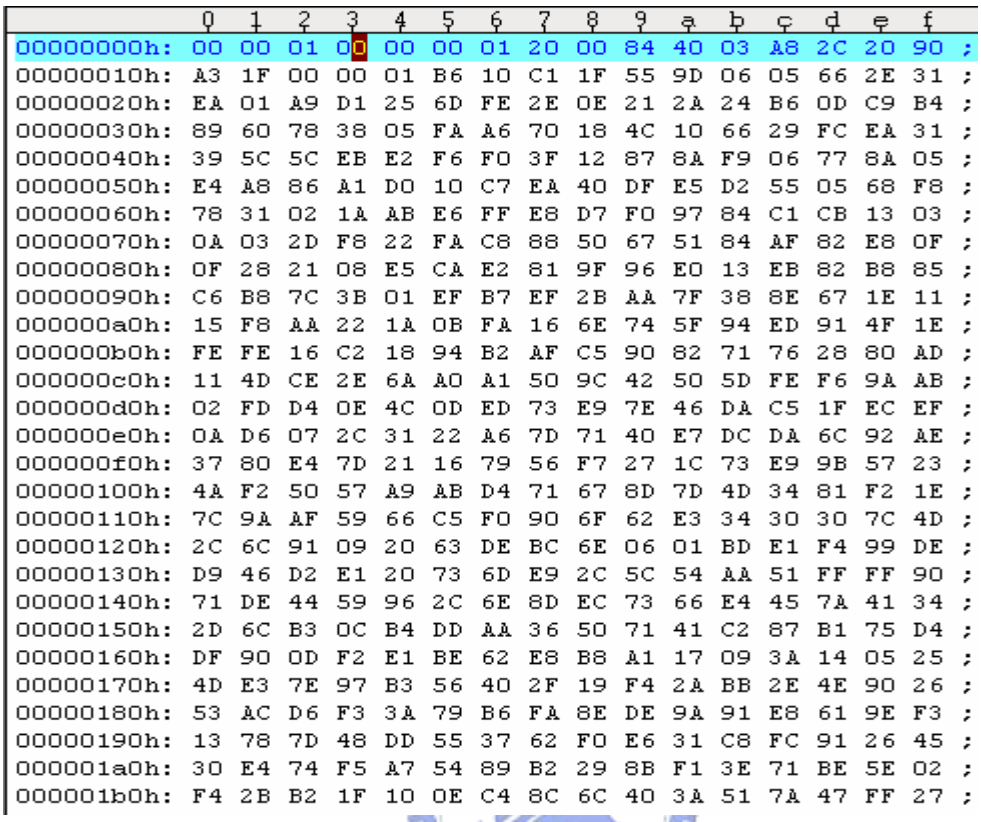


圖 6 Structure Of M4V

圖 7 則是利用 UltraEdit 以 16 進位的方式顯示某一 m4v 檔，雖然根據 ISO-14496-2[6]以及圖 5，m4v 開始於 Visual Object Sequence

Header

.



	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	
00000000h:	00	00	01	00	00	00	01	20	00	84	40	03	A8	2C	20	90	;
00000010h:	A3	1F	00	00	01	B6	10	C1	1F	55	9D	06	05	66	2E	31	;
00000020h:	EA	01	A9	D1	25	6D	FE	2E	0E	21	2A	24	B6	0D	C9	B4	;
00000030h:	89	60	78	38	05	FA	A6	70	18	4C	10	66	29	FC	EA	31	;
00000040h:	39	5C	5C	EB	E2	F6	F0	3F	12	87	8A	F9	06	77	8A	05	;
00000050h:	E4	A8	86	A1	D0	10	C7	EA	40	DF	E5	D2	55	05	68	F8	;
00000060h:	78	31	02	1A	AB	E6	FF	E8	D7	F0	97	84	C1	CB	13	03	;
00000070h:	0A	03	2D	F8	22	FA	C8	88	50	67	51	84	AF	82	E8	0F	;
00000080h:	0F	28	21	08	E5	CA	E2	81	9F	96	E0	13	EB	82	B8	85	;
00000090h:	C6	B8	7C	3B	01	EF	B7	EF	2B	AA	7F	38	8E	67	1E	11	;
000000a0h:	15	F8	AA	22	1A	0B	FA	16	6E	74	5F	94	ED	91	4F	1E	;
000000b0h:	FE	FE	16	C2	18	94	B2	AF	C5	90	82	71	76	28	80	AD	;
000000c0h:	11	4D	CE	2E	6A	A0	A1	50	9C	42	50	5D	FE	F6	9A	AB	;
000000d0h:	02	FD	D4	0E	4C	0D	ED	73	E9	7E	46	DA	C5	1F	EC	EF	;
000000e0h:	0A	D6	07	2C	31	22	A6	7D	71	40	E7	DC	DA	6C	92	AE	;
000000f0h:	37	80	E4	7D	21	16	79	56	F7	27	1C	73	E9	9B	57	23	;
00000100h:	4A	F2	50	57	A9	AB	D4	71	67	8D	7D	4D	34	81	F2	1E	;
00000110h:	7C	9A	AF	59	66	C5	F0	90	6F	62	E3	34	30	30	7C	4D	;
00000120h:	2C	6C	91	09	20	63	DE	BC	6E	06	01	BD	E1	F4	99	DE	;
00000130h:	D9	46	D2	E1	20	73	6D	E9	2C	5C	54	AA	51	FF	FF	90	;
00000140h:	71	DE	44	59	96	2C	6E	8D	EC	73	66	E4	45	7A	41	34	;
00000150h:	2D	6C	B3	0C	B4	DD	AA	36	50	71	41	C2	87	B1	75	D4	;
00000160h:	DF	90	0D	F2	E1	BE	62	E8	B8	A1	17	09	3A	14	05	25	;
00000170h:	4D	E3	7E	97	B3	56	40	2F	19	F4	2A	BB	2E	4E	90	26	;
00000180h:	53	AC	D6	F3	3A	79	B6	FA	8E	DE	9A	91	E8	61	9E	F3	;
00000190h:	13	78	7D	48	DD	55	37	62	F0	E6	31	C8	FC	91	26	45	;
000001a0h:	30	E4	74	F5	A7	54	89	B2	29	8B	F1	3E	71	BE	5E	02	;
000001b0h:	F4	2B	B2	1F	10	0E	C4	8C	6C	40	3A	51	7A	47	FF	27	;

圖 7 Display*.m4v by Hex

及 Visual Object Header 但事實上為了因應個別需求所開發出來的 mpeg-4 codec 其格式會有些許的差別，從圖 7 可以發現，這個 M4v 開始於 32bits 的 video_object_start_code 00 00 01 00，而非 Visual_object_sequence_start_code。圖 8 列出了這個檔案的 ESDS data，其大小為 18bytes。

byte 0	byte 1	byte 2	byte 3
0000 0000	0000 0000	0000 0001	0000 0000
	video_object_start_code		

圖 8-a byte 0 ~ byte 3

byte 4	byte 5	byte 6	byte 7
0000 0000	0000 0000	0000 0001	0010 0000
video_object_layer_start_code			

圖 8-b byte 4 ~byte 7

byte 8	byte 9		byte 10			
0000 0000 10 random_accessible_vol video_object_type_indication is object layer identifier	0001 aspect_ratio_info	0 vol_control_parameters	0 0 vol_shape	1 M	000 0000	

圖 8-c byte 8 ~ byte 10

byte 10	byte 11	byte 12				
01	000000 0000 0011 vop_time_increment_resolution	10	1 M	0 fix_vop_rate	1 M	000

圖 8-d byte 10 ~ byte 12

byte 12	byte 13	byte 14	byte 15
1010 1 M	000 0010 1100 video_object_layer_width	00 1 M	00000 1001 0000 video_object_layer_height

圖 8-e byte 12~byte15

byte 16		byte 17	
1 M	010 0011 interlaced + obmc_disable + sprite_enable + not_8_bit + quant_type + complexity_estimation_disable + resync_marker_disable	000 data_partitioned + scability + resync_marker_disable	11111

圖 8-f byte 16~byte17

從圖 8-c 中，可以發現 ESDS data 定義了 2bits 的 vol_shape，在這裡 vol_shape = 00 代表這個 vol 的形狀是長方形，而圖 8-e 可以發

現有兩個 13bytes 的值分別代表 vol_width 以及 vol_height，在這裡 width = 176，height = 144，因此解碼器可以從 ESDS data 中得到如何解碼 I frame 與 P frame 的資訊。

在 ESDS data 中，也會因為 profile 選擇上的差異，造成 codec tool 使用上的不同，以圖 8-c 來看，其 vol_control_parameters 設定為 0，若此 bit 設定為 1，則會增加 2bits 的 chroma_format，與 1bits 的 low_delay、vbv_parameters，當 vbv_parameters 設定為 1，則之後將設定一些 VBV 參數，包括 15bits 的 first_half_bit_rate、latter_half_bit_rate、first_half_vbv_buffer_size、latter_half_vbv_occupancy 以及 3bits 的 latter_half_vbv_occupancy，此外，也可以在 ESDS Data 中設定量化矩陣參數，圖 8-f 中 quant_type 設定為 0，當此值設定 1 為則我們可以設定 64bytes 的 intra_quant_mat，與 64bytes 的 non_intra_quant_mat。

根據圖 6，ESDS data 結束後會進入 I frame，frame 由 vop_start_code 開始，圖 9 則表示出這個 m4v 檔的 I frame 開始於 32bits 的 vop_start_code 00 00 01 B6，以及 2bits 的 vop_coding_type，從表 11 可以知道當 vop_coding_type = 00 時代表此 frame 為 I frame 。

byte 18	byte 19	byte 20	byte 21	byte 22	
0000 0000	0000 0000	0000 0001	1011 0110	00	01 0000
vop_start_code				vop_coding_type	

圖 9 byte 18 ~ byte 22

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

表 11 Meaning of vop_coding_type

2.5 Parsing video stream from Mpeg-2 System

從前面三小節中，我們可以知道如何從 Program Stream 以及 Transport Stream 中拆解出 elementary stream，同時也知道每一張 frame 的開始與結束點為何。

以 Program Stream 來說，首先，要先拿掉 PACK header 以及 SYSTEM header，接下來找出藏有 program_map 的 PES，這個 PES 的 stream_id 根據表 2 會等於 0xBC，而且這個 PES 根據表 3 會有 stream_type 以及 elementary_stream_id 等資訊，從 elementary_stream_id 我們可以知道接下來要找的 PES 其 header 中的 stream_id 若是等於 elementary_stream_id 則這個 PES 的 payload data 中就有我們要的訊息，而從 payload data 中可以藉由 vop_start_code 00 00 01 B6 找到每張 frame 的開頭與結尾，在這邊要注意的地方是在 I frame 之前還會有 ESDS data 必須將其保留下來，作為解碼器解碼的依據。接下來將實際拆解一個 mpeg-4 simple profile 所組成的 mpeg-2 program stream

file 並將結果顯示於圖 10。

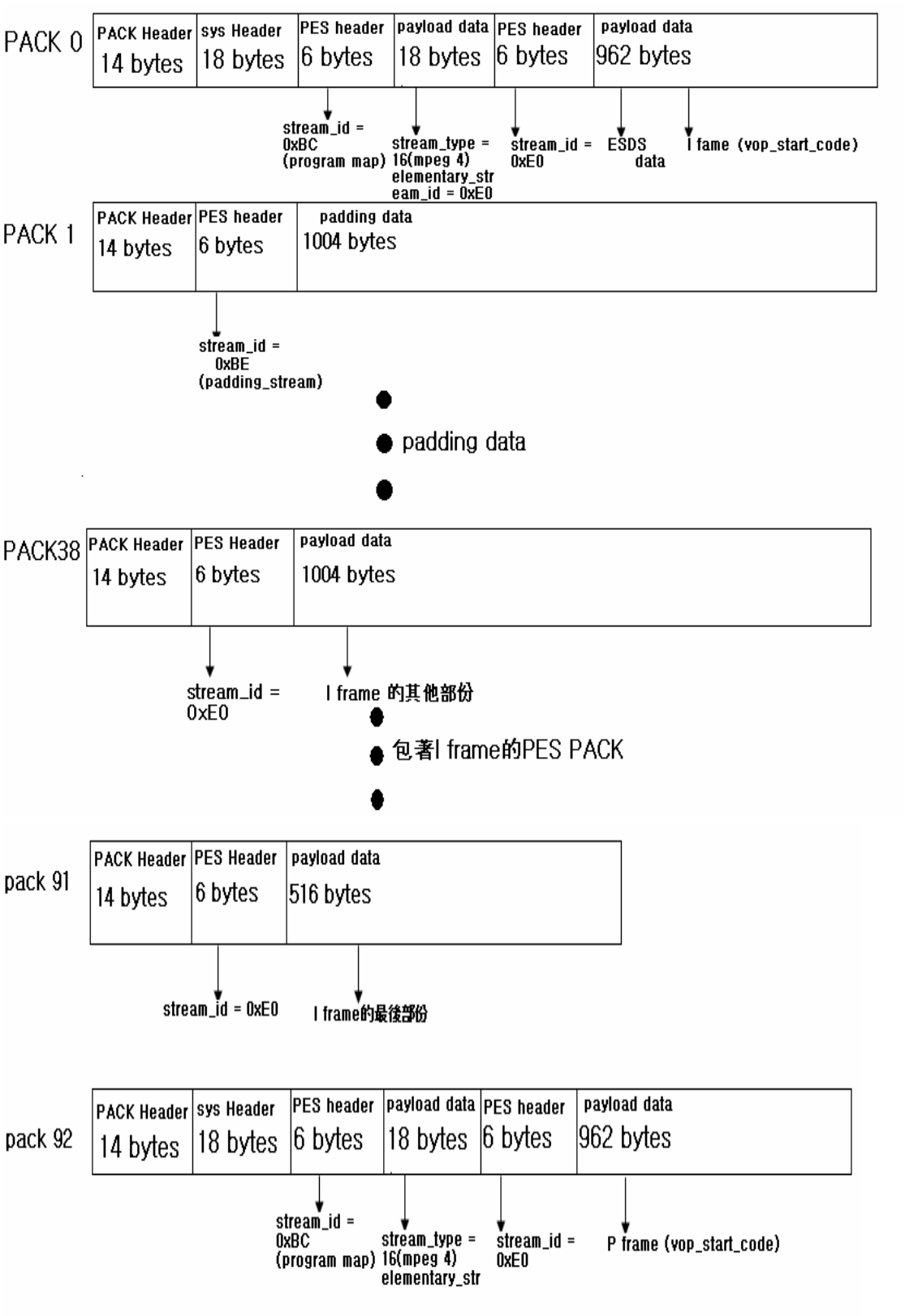


圖 10 實際拆解 program stream 的結果

從圖 10 中我們可以發現，pack 0 與 pack 92 分別是一張 frame 的開始，所以這兩個 pack 將會包含比其他 pack 更多的資訊，例如這兩個 pack 都有 program stream map，從這個 program stream map 可以知道 video stream 為 mpeg 4，因為 stream_type = 16，而且其 PES 的 payload data 中更包含有 PTS(presentation time stamp)及 DTS(decoding time stamp)等幫助播放及解碼的資訊。

Transport stream 的拆解過程與 Program stream 相似，首先我們要找出 transport stream header 中的 PID = 0x0000 的 packet，因為根據表 8 這代表這個 packet 的 payload data 為 program association table，而根據表 9 我們可以從 program association table 中找到一個 program_map_PID，接下來我們要找出某一個 Transport stream header 中的 PID = program_map_PID，因為這個 Transport stream 的 payload data 中根據表 8 會告訴我們 elementary_PID 及其 stream_type，之後只要 PID = elementary_PID 我們就保留其 payload data，在這裡要注意的是根據圖 3 假如這個 tansport stream packet 剛好包含 PES header，則我們必須在保留其 payload data 前先拿掉 PES header。圖 11 為分析一個 Transport stream 檔案的結果其壓縮格式為 mpeg-2。

從圖 11 的 pack 2 中我們可以找到 elementary_PID 接下來只要將 Transport stream header 中 PID = elementary_PID 的 payload data 保留

起來即可。

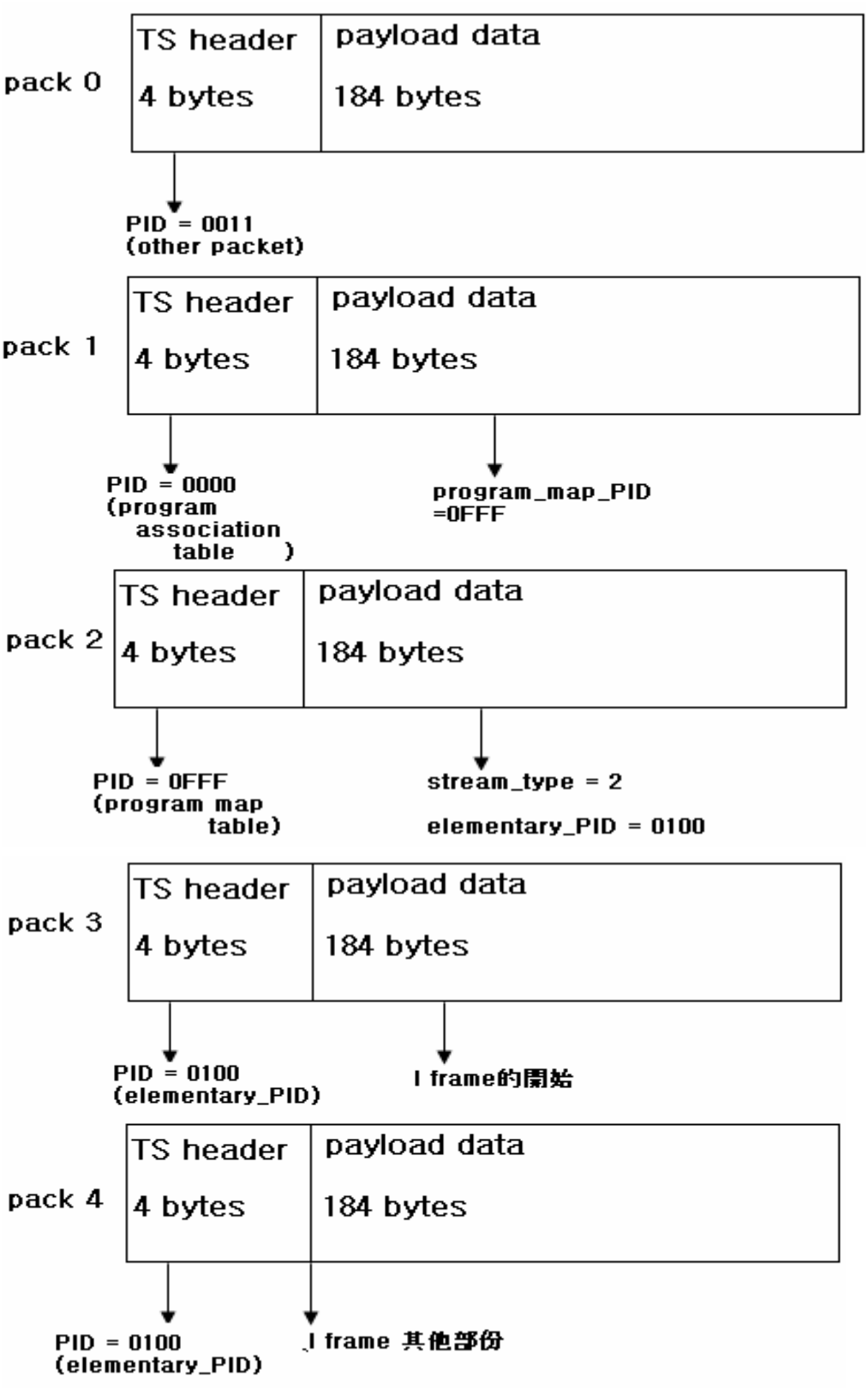


圖 11 實際拆解 transport stream 的結果

第三章 多媒體串流之自由軟體介紹及使用

在第二章中，我們建立了一種多媒體檔案格式的資料庫 MPEG-2 System 以及 mpeg-4 simple profile，但世界上的多媒體檔案格式太多，要建立各種多媒體檔案格式的資料庫所需耗費的工程可以預期將會相當巨大，所幸這些問題藉由自由軟體的發布可以獲得解決，藉由自由軟體我們可以建立各種影音格式的資料庫，以這個資料庫來處理各種影音檔案。

自由軟體基金會(Free Software Foundation)[7]提供的免費且能自由流通的軟體，開發者除了能擁有軟體的完整程式碼，同時也被允許能對此程式碼進行修改及開發，那是因為由自由軟體基金會所發布的軟體都擁有 GNU 通用公共許可證(GNU Lesser General Public License)，這個許可證讓程式接收人可以擁有運行、修改、發行及公開的自由。本章將介紹及使用由自由軟體基金會所發布與多媒體串流有關的軟體，FFmpeg[8]，另外也將介紹一套由 Intel 所開發的 OpenCV(Open Source Computer Vision Library)[9]這是一套與影像處理相關的自由函式庫。

3.1 FFmpeg

FFmpeg 是一套功能強大的多媒體開發軟體，可以用於影像流與

聲音流的分離、轉換、編碼、解碼，主要是由三個應用程式 ffmpeg、ffserver、ffplay 以及兩個函式庫 libavcodec、libavformat 組成：

1 ffmpeg，是個轉碼工具，也就是將某一個影音格式轉碼成另一個影音格式，例如 ffmpeg –i inputfile.mpg outputfile.mp4 就是將一個.mpg 檔轉換成.mp4 檔。

2 ffserver，是個串流伺服器，藉由 HTTP 進行即時串流。

3 ffplay，是一個用 FFmpeg 函式庫所開發出來的多媒體播放器。

4 libavcodec 是 FFmpeg 所提供的函式庫，幾乎涵蓋了所有影音格式的編碼及解碼功能，程式開發者只要將此函式庫引入 project 即可使用其所提供的編碼/解碼功能。

5 libavformat 是 FFmpeg 所提供的函式庫，利用這個函式庫將聲音流與影像流從檔案中抽取出來或是將壓縮過的聲音流與影像流結合成某種格式的檔案。

3.1.1 應用程式的的編譯及使用

本節將介紹如何在 Windows 底下連結應用程式與 FFmpeg 所提供的兩個 library，藉由連結這兩個函式庫 libavcodec 與 libavformat 應用程式將可以使用這兩個函式庫所提供的各種影音檔案的編碼、解碼等功能。

由於 VC++ 6.0 並非是一個標準的 C 編譯器，因此 FFmpeg 所使

用的語法無法在 VC++底下編譯通過，因此若希望在 WinXP 底下編譯 FFmpeg 就必須使用 MINGW 加上 MSYS，MSYS 是 Windows 底下的一個虛擬 linux shell 環境，此環境提供了標準的 C 編譯器，我們藉由他來執行 configure、make 等指令，在這邊需要注意的是 MSYS 必須安裝在 MINGW 底下的 bin 資料夾下。

接下來就是下載 FFmpeg 的 source code，下載的方式是使用 CVS(Concurrent Versions System)協作/併發版本系統，CVS 是一種版本控制系統，它可以很方便的讓開發者將最新版本上傳，也可以讓使用者能隨時下載最新版本，下載到最新的 FFmpeg 後我們必須執行裡面的 configure，執行 configure 的目的是將必要的參數傳給 Makefile，我們可以使用最簡單的參數配置方式，也就是打開 MSYS，將所在目錄切換到剛剛下載的 FFmepg 底下，執行以下的指令"./configure –enable-memalign-hack"，至於其他的配置方式例如希望得到.lib 檔或是.dll 檔則可藉由執行"./configure --help"得到說明，圖 12 可以看到輸入./configure –help 所得到的結果。

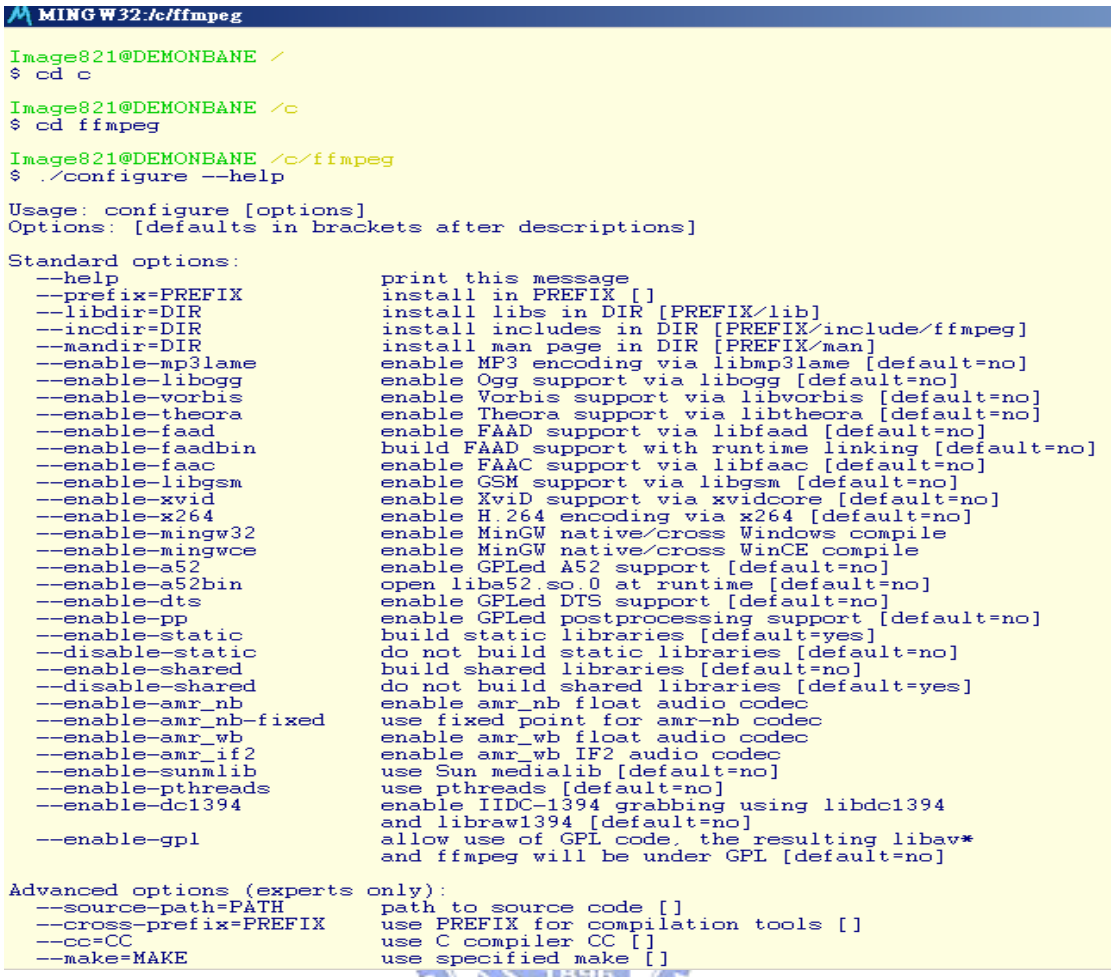


圖 12 ./configure –help 列出支援的參數配置

執行完./configure -- enable-memalign-hack 後，接下來就是執行 make 以及 make install，在 linux 環境下，我們是藉由 Makefile 將各種 library 的連接與輸出輸入的關係設定好，根據上面的步驟我們可以成功的編譯出 ffmpeg.exe 這個執行檔。但若是想要編譯出 ffplay.exe 檔則必須連結 SDL(Simple DirectMedia Layer)，基本上 SDL 就如同 Windows 所提供的 DirectDraw 一般，不同之處在於 SDL 需在 Linux 的作業系統上使用，其主要目的是為了讓使用者能夠藉由直接存取記憶體的方式，達到快速顯示、播放多媒體的

功能。安裝的方法是將下載下來的 SDL 的 bin、include、lib、share 等四個目錄拷貝到 mingw 的目錄底下，接著修改 i386-mingw32msvc-sdl-config 的第一行指令為 prefix=/mingw，之後重新編譯 ffmpeg 即可得到 ffplay.exe 這個執行檔，圖 13 是使用 ffplay 播放影片的畫面。

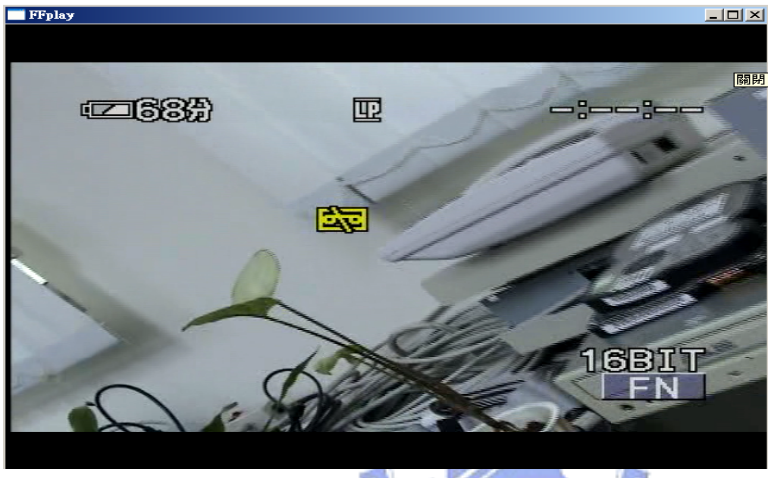


圖 13 ffplay 播放畫面

3.1.2 av_register_all( )—各種影音格式模板的建立

為了使用 FFmpeg 所提供的函式庫，必須要試著了解 FFmpeg 提供哪些 API(Application Program Interface)以及這些應用程式開發介面具有哪些功能。

首先，FFmpeg 幾乎可以看的懂目前大部分的影音包裝格式，那是因為函式庫 libavformat 支援了大部分的影音包裝格式例如 mpg(mpeg1/2)、mp4、mp3、avi、wav、rm、quick time 等，FFmpeg 的做法為在應用程式的一開始先建立好一個鏈結串列，這個鏈結串列

的單位為一種影音格式模板而每個模版都是一個 Structure AVInputFormat，圖 14 則列出 AVInputFormat 包含哪些成員。不同的影音格式模板其成員的內容將不盡相同，例如 mpeg-2 program stream 這個影音模版其成員 long name 會定義成 MPEG PS format 而 mpeg-2 transport stream 這個影音模版其成員 long name 則會定義成 MPEG TS format。

```
typedef struct AVInputFormat {
const char *name;
const char *long_name;
int priv_data_size;
int (*read_probe)(AVProbeData *);
int (*read_header)(struct AVFormatContext *, AVFormatParameters *ap);
int (*read_packet)(struct AVFormatContext *, AVPacket *pkt);
int (*read_close)(struct AVFormatContext *);
int (*read_seek)(struct AVFormatContext *,
                 int stream_index, int64_t timestamp, int flags);

int64_t (*read_timestamp)(struct AVFormatContext *s, int stream_index,
                          int64_t *pos, int64_t pos_limit);
int flags;
const char *extensions;
int value;
int (*read_play)(struct AVFormatContext *);
int (*read_pause)(struct AVFormatContext *);
struct AVInputFormat *next;
```

圖 14 member of AVInputFormat

鏈結串列會將所有的影音格式模板串接其來，在這裡的影音格式模板是指一個屬於 AVInputFormat 的結構，例如圖 15 就是 FFmpeg 所定義的 mpeg-2 program stream 的影音格式模板當然他也屬於

AVInputFormat 這個結構，而輸入的檔案只要對這個鏈結串列進行循序搜尋，當搜尋到某一個影音格式模版時，我們必須將輸入檔案的檔頭餵給這個影音格式的成員函式 read_probe( )，舉例來說，若是搜尋到的影音格式是 mpeg-2 program stream 則根據圖 15 我們必須把輸入檔案的檔頭餵給 mpegps_probe( )，read_probe( )這個成員函式會去判斷輸入檔案的檔頭符不符合這個影音格式，若是符合則 read_probe( )會回傳非零整數，代表輸入檔案即為此一影音格式，反之若是回傳值為零則代表還沒有找到符合的影音格式，程式必須繼續搜尋下去。

```
AVInputFormat mpegps_demux = {
    "mpeg",
    "MPEG PS format",
    sizeof(MpegDemuxContext),
    mpegps_probe,
    mpegps_read_header,
    mpegps_read_packet,
    mpegps_read_close,
    NULL,       mpegps_read_dts,
    .flags = AVFMT_SHOW_IDS,
};
```

圖 15 mpeg-2 program stream 的結構成員

上面建立鏈結串列與搜尋檔案格式的過程看似複雜，但 FFmpeg 的 API 能幫助我們完成大部分的工作，由 libavformat 這個函式庫所提供的 API av_register_all( )其目的就是建立起有所有檔案格式的鏈結串列，圖 15 則是 av_register_all( )的水平關係圖，從圖 16 中我們可

以發現 av_register_all( )呼叫了很多影音格式的初始化函式，例如 mp3_init( )、mpegps_init( )、mpegts_init( )等，這些初始化函式分別都會呼叫 av_register_input_format( )，程式就用 av_register_input_format( )建立起所有影音格式的鏈節串列，圖 17 則 av_register_input_format( )的原始程式碼，從圖 16、17 我們可以知道 av_register_all( )呼叫了所

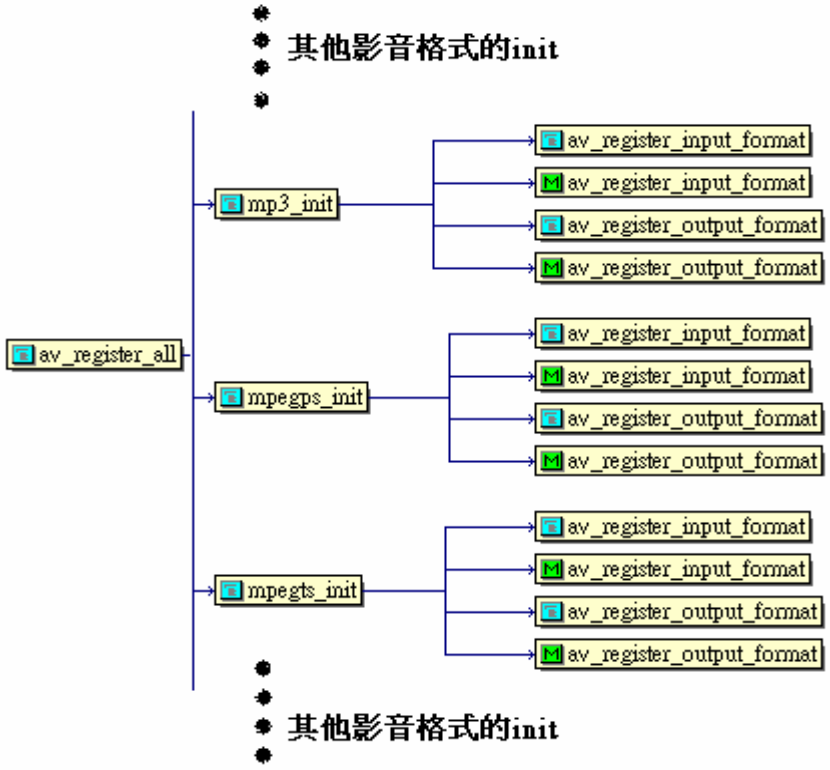


圖 16 av_register_all( )之水平關係圖

```
AVInputFormat *first_iformat = NULL;

void av_register_input_format(AVInputFormat *format)
{
    AVInputFormat **p;
```

```
    p = &first_iformat;
    while (*p != NULL) p = &(*p)->next;
    *p = format;
    format->next = NULL;
}
```

圖 17 av_register_input_format( )程式碼

有影音格式的初始化函式例如 mp3_init( )、mpegps_init( )，而這些初始化函式會將它們所代表的影音包裝格式以 Structure AVInputFormat 傳給 av_register_input_format( )，舉例來說，mpegps_init( )會將圖 15 的 mpegps_demux 傳給 av_register_input_format( )，而 av_register_input_format( )則會將 mpegps_demux 與前面的影音格式串接起來，因此我們只要很呼叫 av_register_all( )這個 api，就可以建立如圖 18 所示的鏈結串列，將各種模版串接起來。而 first_iformat 則會指向這個鏈結串列的起始位置。

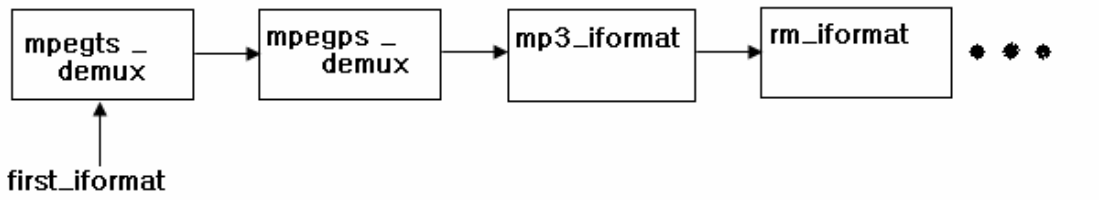


圖 18 各種影音格式的鏈結串列

3.1.3 av_open_input_file( )—確認輸入檔案包裝格式

在建立好各種影音格式的鏈結串列後，接下來就要判斷輸入檔案為何種影音格式，判斷的方式如上一節所述，輸入檔案會跟每種影音

格式模板進行比對，每個影音格式模板的成員函式 read_probe( )會試著判斷輸入檔案的檔頭符不符合這種影音格式的檔頭標準，一旦符合則會回傳一個非零整數，這個判斷的過程 libavformat 這個函式庫也提供一個 api 供程式開發者使用，那就是 av_open_input_file( )，圖 19 則列出 av_open_input_file( )的水平關係圖。av_open_input_file( )藉由呼叫 av_probe_input_format( )來進行搜尋鏈結串列的過程。

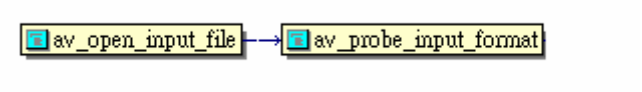


圖 19 av_open_input_file( )水平關係圖

圖 20 則是 av_probe_input_format( )的原始碼，根據圖 18，first_iformat 會指向這個鏈結串列的開頭，因此 av_probe_input_format( )利用一個 for 迴圈從鏈結串列的開頭 first_iformat 一直搜尋到鏈結串列的結尾，而 for 迴圈每執行一次就代表搜尋到某一個影音格式，同時每執行一次都會呼叫一次 fmt1->read_probe(pd)，這裡的 fmt1 是代表某一個影音格式，pd 則是輸入檔案的檔頭，read_probe( )在輸入檔案的檔頭與這個影像格式所定義的檔頭相符時，會回傳一個大於零的整數值，並將 fmt 設定為 fmt1，而 fmt 則為最後確定好的影音格式。

```
AVInputFormat *av_probe_input_format(AVProbeData *pd, int is_opened)
{
    AVInputFormat *fmt1, *fmt;
    int score, score_max;
    fmt = NULL;
    score_max = 0;
    for(fmt1 = first_iformat; fmt1 != NULL; fmt1 = fmt1->next) {
            if (!is_opened && !(fmt1->flags & AVFMT_NOFILE))
                continue;
        score = 0;
        if (fmt1->read_probe) {
            score = fmt1->read_probe(pd);
        } else if (fmt1->extensions) {
            if (match_ext(pd->filename, fmt1->extensions)) {
                score = 50;
            }
        }
        if (score > score_max) {
            score_max = score;
            fmt = fmt1;
        }
    }
    return fmt;
}
```

圖 20 av_probe_input_format( )的原始碼

至於 read_probe( )的函式內容則根據影音格式的不同而有不同的定義，在這邊我們將以第二章出現過的 mpeg-2 program stream 與 transport stream 為例子分別說明在 FFmpeg 中 read_probe( )如何成功的判斷出檔案格式為 program stream 與 transport stream，從圖 20 我們可以知道當搜尋到某一個影音格式時程式會呼叫 fmt1->read_probe( )來判斷輸入的檔案其檔頭是否符合此影音格式的定義，當此影音格式

為 mpeg-2 program stream 也就是當圖 20 的 fmt1 等於圖 14 的 mpegps_demux 時，呼叫 fmt1->read_probe( )事實上就等同於呼叫圖 20 的 mpegps_probe( )，圖 21 則為 mpegps_probe( )的程式碼，首先 mpegps_probe( )會先讀進這個 program stream 的前面 2048bytes 的資料，事實上以第二章的圖 10 來看，讀進 2048bytes 就是讀進 PACK 0 與 PACK 1，而判斷的規則則是這 2048bytes 至少要有一個 system_header_start_code 0x000001BB，而且 system header 的個數必需小於或等於 PACK header。只要符合這兩項，mpegps_probe( )就會回傳一個大於零的整數，也就確定此輸入檔案為 program stream。

```
static int mpegps_probe(AVProbeData *p)
{
    uint32_t code= -1;
    int sys=0, pspack=0, priv1=0, vid=0;
    int i;

    for(i=0; i<p->buf_size; i++){
        code = (code<<8) + p->buf[i];
        if ((code & 0xffffff00) == 0x100) {
            switch(code){
            case SYSTEM_HEADER_START_CODE:      sys++; break;
            case          PRIVATE_STREAM_1:    priv1++; break;
            case           PACK_START_CODE: pspack++; break;
            case     (VIDEO_ID + 0x100):       vid++; break;
            }
        }
    }

    if(sys && sys*9 <= pspack*10)
        return AVPROBE_SCORE_MAX/2+2;
    if((priv1 || vid) && (priv1+vid)*9 <= pspack*10)
        return AVPROBE_SCORE_MAX/2+2;
    return 0;
}
```

圖 21 mpegps_probe( )程式碼

Transport stream 也跟 Program stream 相似，當圖 20 的 fmt1 等於

transport stream 時，呼叫 ftm1->read_probe( )等同於呼叫 mpegts_probe( )，圖 22 則是 mpegts_probe( )的水平關係圖以及 mpegts_probe( )呼叫 analyze( )的程式碼，不同於 mpegps_probe( )直接

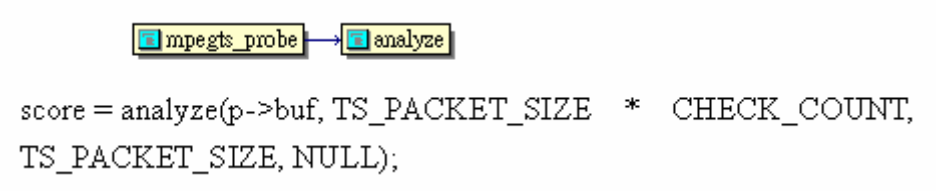


圖 22 mpegts_probe( )與 analyze( )關係

判斷輸入檔案的檔頭有幾個 system_header 與 pack_header，mpegts_probe( )則是在副函式 analyze( )中進行判斷，其判斷的方式是讀入輸入檔案(就是圖 22 的 p->buf)的前 1880bytes，接下來判斷這 1880bytes 中有幾個 bytes 是 Transport stream header 中的 sync_byte 0x47，每讀進一個 sync_byte 則圖 22 中的 score 會加一，從第二章的圖 11 我們可以知道 Transport stream 中每個 packet 其大小為 188bytes 且其開頭會有 4 bytes 的 header，sync_byte 0x47 就包含在這 4bytes 中，因此我們可以預期讀進這 1880bytes 並判斷 sync_byte 後 score 的值將為 10，程式就是藉由這個方法判斷出輸入檔案為 mpeg-2 transport stream。圖 23 則是 analyze( )的原始碼。

```
static int analyze(const uint8_t *buf, int size, int packet_size, int *index)
    int stat[packet_size];
    int i;
    int x=0;
    int best_score=0;
    memset(stat, 0, packet_size*sizeof(int));

    for(x=i=0; i<size; i++){

        if(buf[i] == 0x47){

            stat[x]++;
            if(stat[x] > best_score){
                best_score= stat[x];
                if(index) *index= x;
            }
        }
        x++;
        if(x == packet_size) x= 0;
    }

    return best_score;
}
```

圖 23 analyze( )原始碼

3.1.4 av_find_stream_info( )—獲得 codec 資訊

從 3.1.2 與 3.1.3 中我們可以知道，FFmpeg 如何建立起鏈節串列，以及如何獲知輸入檔案其影音格式為何，在這邊要注意的是，影音格式只是個包裝，舉例來說，一個*.mp4 的檔案，mp4 只是他的包裝格式，但其裡面包裝的聲音流與影像流其 codec 可能有很多種可能，聲音流可能使用 MP3(**MPEG-1 Audio Layer 3**)，也有可能使用 AAC(Advanced Audio Coding)，影像流則有可能是 mpeg-4，av_find_stream_info( )這個 api 的其中一項重要任務就是找出這些包

裝格式其聲音流與影像流到底是用何種 codec，圖 24 是 av_find_stream_info( )的水平關係圖，從圖 24 我們可以知道 av_find_stream_info( )最後將會呼叫 iformat->read_packet( )，在這邊 iformat 是屬於如圖 14 的 Structure AVInputformat，因此假若輸入檔案是 mpeg-2 program stream 則 iformat 會因為 3.1.3 介紹的 API av_open_input_file( )而被設定成如圖 15 的 mpegps_demux，因此在這邊呼叫 iformat->read_packet( )事實上就等於呼叫圖 15 的 mpegps_read_packet( )。

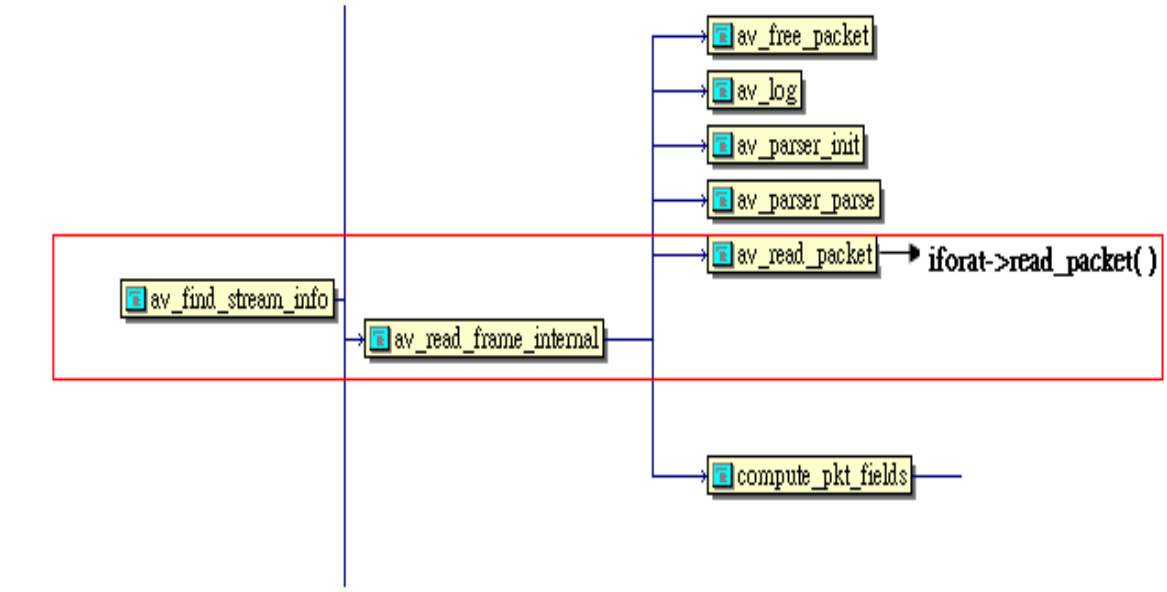


圖 24 av_find_stream_info( )之水平關係圖

接下來我們將介紹 FFmepg 如何利用 mpegps_read_packet( )找出 program stream 其影像流與聲音流的 codec。圖 25 是

mpegps_read_packet( )的水平關係圖，其中 mpegps_read_pes_header( )

圖 25 mpegps_read_packet( )水平關係圖

會藉由呼叫 find_next_start_code( )找到下一個 PES header 中的 start code，第二章表 2 則列出所有 PES 的 start code，一旦 find_next_start_code( )找到的 start code 等於 0xBC 根據表 2 這個 PES 所代表的意義為 Program Stream map，則 mpegps_read_pes_header( )就會呼叫函式 mpegps_psm_parse( )來進行 Program Stream map 的分解動作，根據表 3 我們可以得知對 Program Stream map 的分解動作其最終的目的為找到各為 1byte 的 stream_type 與 elementary_stream_id，我們就是藉由 stream_type 與 elementary_stream_id 來判斷出影像流與聲音流的 codec，圖 26 則為 mpegps_psm_parse( )的原始碼，對照表 3 與圖 26，type 與 es_id 分別等同於 stream_type 與 elementary_stream_id，至於 stream_type 與 codec 的關係則列於表 4。

```
static long mpegps_psm_parse(MpegDemuxContext *m, ByteIOContext *pl
{
    int psm_length, ps_info_length, es_map_length;

    psm_length = get_be16(pb);
    get_byte(pb);
    get_byte(pb);
    ps_info_length = get_be16(pb);

    /* skip program_stream_info */
    url_fskip(pb, ps_info_length);
    es_map_length = get_be16(pb);

 while (es_map_length >= 4) {
        unsigned char type = get_byte(pb);
        unsigned char es_id = get_byte(pb);
        uint16_t es_info_length = get_be16(pb);
        /* remember mapping from stream id to stream type */
        m->psm_es_type[es_id] = type;
        /* skip program_stream_info */
        url_fskip(pb, es_info_length);
        es_map_length -= 4 + es_info_length;
    }
    get_be32(pb); /* crc32 */
    return 2 + psm_length;
}
```

圖 26 mpegps_psm_parse( )的原始碼

3.2 OpenCV 與串流平台結合

目前應用於多媒體串流的例子很多，安全監控是其中之一，安全監控的應用很多，例如人臉追蹤與辨識，移動物偵測等，結合各種影像處理與壓縮串流是目前多媒體串流的一個重要應用，本節將實際結合影像處理與壓縮串流，驗證影像處理與壓縮串流結合。

OpenCV(Open Source Computer Vision Library)是 Intel 開放原碼電腦視覺函式庫，由一系列 c 函式和少量 c++類別所構成，實現了影像處理方面許多的應用，其特色為擁有 300 多個 c 函式 API，而且對商業與非商業的應用都是免費的，要使用 OpenCV 的函式庫，首先要先編譯出我們所需要的函式庫(Library)與靜態聯結函式庫(dll)，從下載下來的 OpenCV 原始程式中的 make 資料夾我們可以找到 opencv.dsw 這個 workspace，只要編譯這個 workspace 就可以得到我們想要的函式庫，當然我們也可以在這個 workspace 進行程式修改以得到更符合我們需要的影像處理函式庫。

3.2.1 串流平台簡介[11]

串流平台可以分為兩個部份，Server 以及 Client，Server 端將影像來源進行 mpeg-4 壓縮，也就是將不含 bmp header 的 bmp raw data 壓縮成如第二章所示 mpeg-4 video elementary stream，壓縮好的

elementary stream 則包裝成 rtp packet 進行網路傳送，而 Client 端則接收 rtp packet 並且進行 mpeg-4 的解壓縮，也就是將 mpeg-4 video elementary stream 解壓縮成 bmp raw data，由於 client/server 系統必須在同一個時間處理不同的工作，例如 server 必須一邊進行 mpeg-4 壓縮一邊進行網路傳輸的工作，因此整個系統是建構成多執行緒系統 (multi-thread system)，作業系統分配資源給各個執行緒，讓系統能同時間的進行多項工作。圖 27 是同一個時間 server 端會執行的執行緒

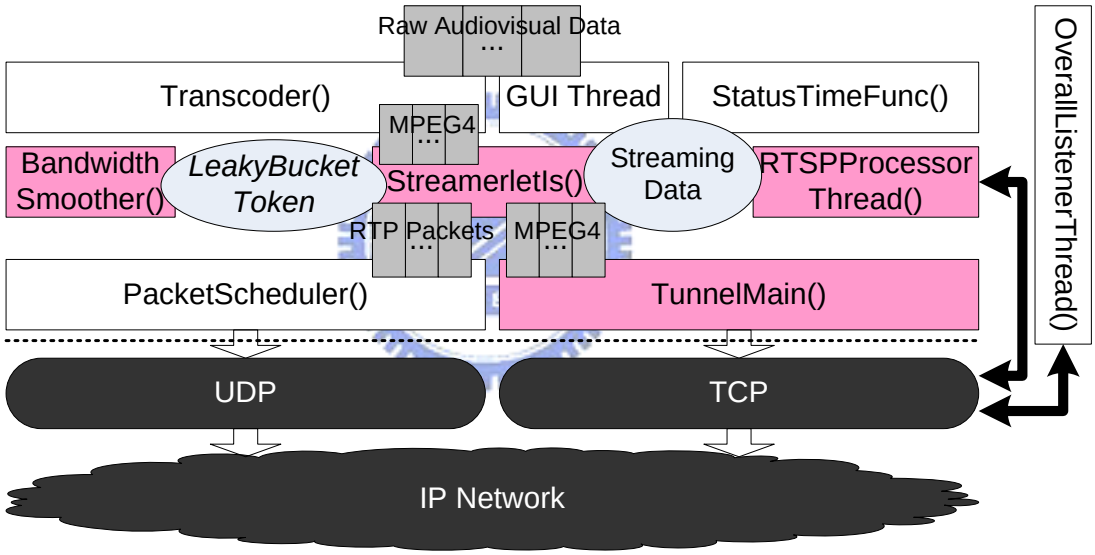


圖 27 Threads of server system

.● GUI thread

作為使用者與 server 端溝通的介面

● Transcoder( )

將各種攝影機擷取而來的 raw data 壓縮成 mpeg-4 elementary stream，然後將這些 bitstreams 放到 buffer 裡讓 StreamerletIs( )存取

● StatusTineFunc( )

收集並且顯示 client 的資訊於 GUI 上

● OverallListenerThread( )

負責接收新的 client 連線，並且每次當成功的建立起新的連線會創造另外一個執行緒 RTSPProcessorThread( )來管理這個新的連線

● PacketScheduler( )

將已經包裝好的 RTP pacets 藉由 UDP sockets 送入網際網路中

下面這些執行緒，在每次有新的 client 連線進來時就會被建立起來

● RTSPProcessorThread( )

負責接收、分析、抽取以及回應 RTSP messages，並且改變目前 server 的狀態，圖 28 是 RTSP State Machine，舉例來說，當 Server 端目前在 Ready state，而且收到來自 client 端的 "PLAY" message，RTSPProcessorThread( )會將 state 改變成 Play state 並且做出對應的動作

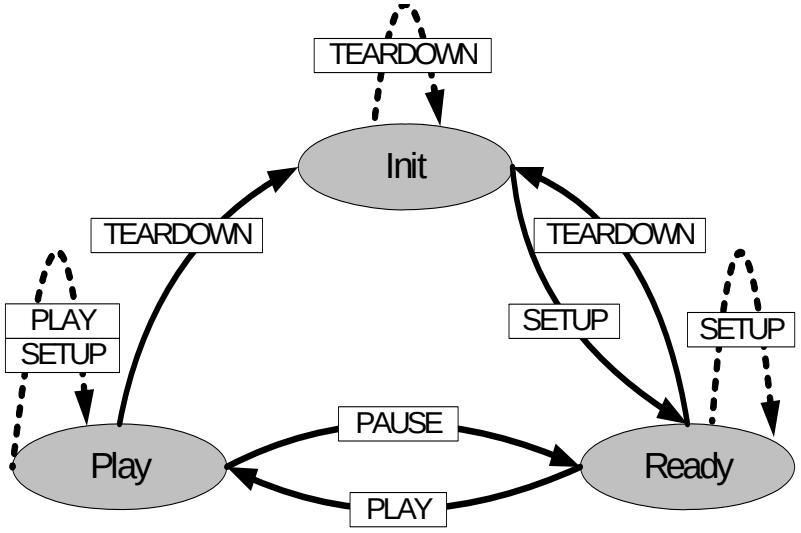


圖 28 RTSP State Machine

● StreamerletIs( )

如果是採用 UDP，則此執行緒負責將剛剛 Transcoder( )壓縮好的 mpeg-4 video elementary stream 包裝成 RTP packets，並將這些 packets 放在 PacketScheduler( )能夠存取的 buffer 上，另一方面如果是採用 TCP 當做傳送協定，則此執行緒便不做包裝的動作，直接將壓縮好的 elementary stream 放在 TunnelMain( )能夠存取的 buffer 上

● TunnelMain( )

藉由 TCP Sockets 將 elementary stream 送入網際網路中

圖 29 則是當 client 啟動時，同時間會執行的執行緒

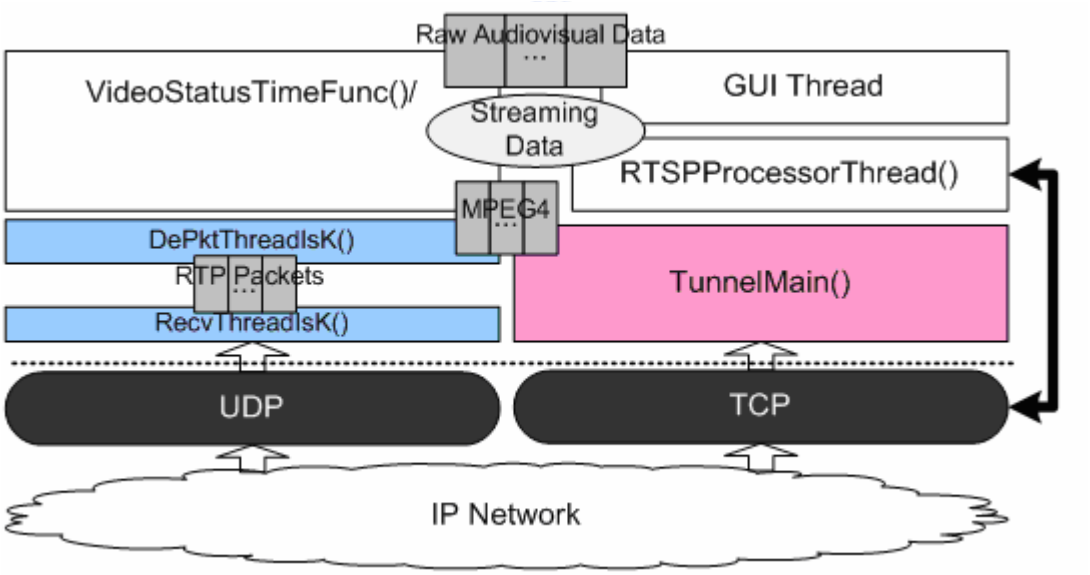


圖 29 Threads of client system

• GUI thread

作為使用者與 server 端溝通的介面

•VideoStatusTimeFunc( )

負責在 GUI 上面顯示目前接收封包的情況，同時把接收的 mpeg-4 elementary stream 解壓縮成 bmp 的 raw data，並把解壓縮的結果播放

出來

- RTSPProcessorThread( )

與 server 端相同，負責 RTSP 指令分析以及改變 client 的 RTSP 狀態

- TunnelMain( )

藉由 TCP 接收串流封包，並且把這些串流封包放在 VideoStatusTime Func( )能夠存取的地方

- RecvThreadIsK( )

藉由 UDP 接收 RTP packets，並且把這些封包放在 DepktThreadIsK( ) 能夠存取的地方

- DePktThreadIsK( )

拆解 RTP packets 取得裡面的 elementary streams，並將這些資料放在 VideoStatusTimeFunc( )能存取的地方

從以上這些介紹，我們可以在 server 端或是 client 端加上影像處理，在 server 端我們可以加在 Transcoder( )這個執行緒裡，因為這個執行緒將從各種攝影機抓取資料，抓到的資料必須轉成 bmp 檔接著我們可以在這邊進行影像處理以及串流處理，在 client 端我們可以加在 VideoStatusTimeFunc( )這個執行緒之上，因為這個執行緒能將 elementary stream 解碼成 bmp 的 raw data，我們可以再建立另外一個執行緒來存取這個 raw data 並做影像處理。

在 server 端我們將透過以下幾個 API 以及一些參數配置將影像處理過後的 bmp raw data 進行 mpeg-4 壓縮以及網路串流。首先，必須

配置好 mpeg-4 壓縮輸入資料與輸出資料的位置，在這邊輸入資料是指每一張進行過影像處理的 bmp raw data 而輸出資料是指壓縮過後的 mpeg-4 video elementary stream，有可能是 ESDS data +I frame 也有可能是 P frame 其配置方式如圖 30 所示，其中 imageData 是經過影像處理後的 bmp raw data 的起始位置，而 TargetBuffer 則是壓縮過後的 mpeg-4 frame 的起始位置。接下來我們就要對此 bmp raw data 進行

```
typedef struct _ENC_FRAME_
{
    void *image;        // the image frame to be encoded
    void *bitstream;    // the buffer for encoded bitstream
    long length;        // the length of the encoded bitstream
    int        IsKeyFrame;
    int write_coded;    // flag to built reconstruct image
    unsigned char *recon_image; // buffer for reconstruct image

}    ENC_FRAME;
ENC_FRAME  SP4EncFrame;
SP4EncFrame.image = imageData;
SP4EncFrame.bitstream = TargetBuffer;
```

圖 30 配置 codec input data 與 output data

mpeg-4 壓縮，使用 API encore( )，此 API 會將壓縮好的 mpeg-4 elementary stream 放到 TargetBuffer 裡。

- encore((HANDLE*)(&handle)，ENC_OPT_ENCODE，
  &ENC_FRAME，NULL)

接下來，我們要把 TargetBuffer 裡面的 bitstream 從圖 26 中 Trancoder( )

這個執行緒送到與 StreamerletIs( )共享的 share memory 中，這個動作則是藉由 CapturToServer( )這個 API 來完成。

- CaptureToServer(void** StreamServer，int stream_type，int input_channel，int real_time_capture，MediaItem *mediabuffer) ;

至於 client 端則只要再建立一個執行緒，跟圖 29 的 VideoStatusTimeFunc( )這個執行緒共同分享 raw audiovisual data 即可，圖 29 的 raw audiovisual data 是放在 global_a->Display_bmp 中。

3.2.2 一個 OpenCV 與串流平台結合的例子

在這裡我們利用 OpenCV 提供的影像處理函式，與串流平台進行整合，首先必須先將 OpenCV 所提供的 header 檔加入我們的 Project，如圖 31 所示，如此我們就可以使用這些 header 檔所提供給我們的 Structure，OpenCV 提供了一個 Structure IplImage 專門用來處理 bmp 的 raw data，因此不管是再 server 端從攝影機得到的 raw data 或是

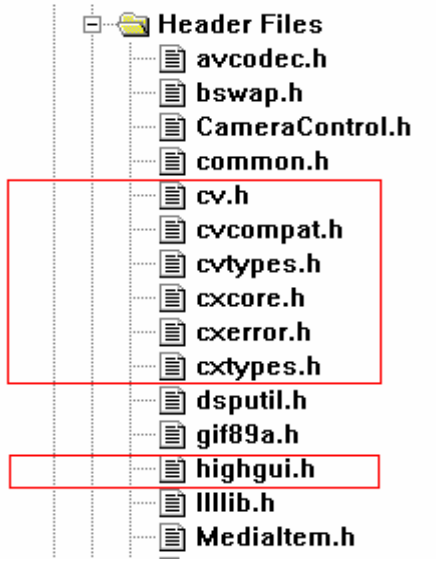


圖 31 將 header 檔加入 Project 中

Client 端經過解碼而得的 bmp raw data 我們都必須將這些 raw data 的資料填進這個 Structure 中以便後續影像處理的程序，圖 32 則是 IplImage 的成員，我們必須將這個結構裡面的 imageData 指向想要進行影像處理的 bmp raw data，同時還必須設定好 width，height 等資料

```
typedef struct _IplImage
{
    int  nSize;             /* sizeof(IplImage) */
    int  ID;                /* version (=0)*/
    int  nChannels;         /* support 1,2,3 or 4 channels */
    int  alphaChannel;      /* ignored by OpenCV */
    int  depth;             /* pixel depth in bit */
    char colorModel[4]; /* ignored by OpenCV */
    char channelSeq[4]; /* ditto */
    int  dataOrder;         /* 0 - interleaved color channels, 1 - separate color
channels cvCreateImage can only create interleaved images */
    int  origin;            /* 0 - top-left origin,
                               1 - bottom-left origin (Windows bitmaps style) */
    int  align;             /* Alignment of image rows (4 or 8).
                               OpenCV ignores it and uses widthStep instead */
    int  width;             /* image width in pixels */
    int  height;            /* image height in pixels */
    struct _IplROI *roi;/* image ROI. if NULL, the whole image is selected */
    struct _IplImage *maskROI; /* must be NULL */
    void  *imageId;         /* ditto */
    struct _IplTileInfo *tileInfo; /* ditto */
    int  imageSize;         /* image data size in bytes*/
    unsigned char *imageData;   /* pointer to aligned image data */
    int  widthStep;     /* size of aligned image row in bytes */
    int  BorderMode[4]; /* ignored by OpenCV */
    int  BorderConst[4]; /* ditto */
    unsigned char *imageDataOrigin; /* pointer to very origin of image data*/
}
IplImage;
```

圖 32 IplImage 的成員

接下來，為了使用 OpenCV 所提供的 API，我們必須將之前編譯出來的 library 加入到我們的 Project 中，加入的方法如圖 33 所示，在 IDE 介面中選 Project->Settings->Link，接著將 highgui.lib，cxcore.lib，cv.lib

加進來，這樣就可以使用 OpenCV 的 API。在這邊我們將列出幾個

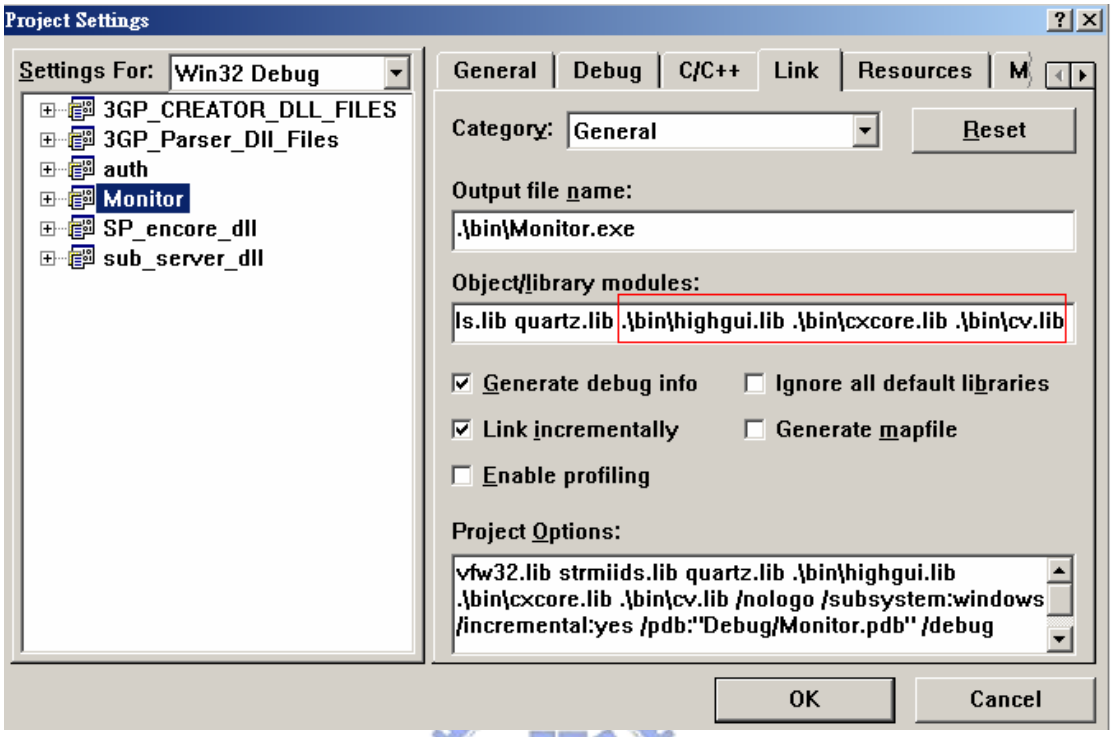


圖 33 於 IDE 介面中加入 Library

OpenCV 與人臉追蹤有關的函式。

• CvHaarClassifierCascade* cvLoadHaarClassifierCascade(const char* directory,
CvSize orig_window_size);

此函式會載入利用海爾特徵的級聯分類器，OpenCV 已經幫我們訓練了幾個級聯分類器都是以 xml 的格式儲存，例如

haarcascade_frontalface_alt.xml 就是一個以正臉為目標樣本的分類器

• CvSeq* cvHaarDetectObjects(const CvArr* image, CvHaarClassifierCascade* cascade, CvMemStorage* storage, double scale_factor=1.1,int min_neighbors=3, int flags=0,CvSize min_size=cvSize(0,0));

在這邊 image 是待檢驗的影像來源，cascade 則是

cvLoadHaarClassifierCascade( )的回傳值，cvHaarDetectObjects( )會回傳待檢驗的影像來源有幾個我們有興趣的目標例如”人臉”。

• void cvRectangle(CvArr* img, CvPoint pt1, CvPoint pt2, CvScalar color,
int thickness=1, int line_type=8, int shift=0);

繪制簡單、指定粗細或帶填充的矩形，藉由 cvHarrDetectObjects( )我們可以得到目標在影像來源的位置，使用 cvRectangle( )將目標以矩形標記。

圖 34 是結合臉部追蹤於 client 端的結果。

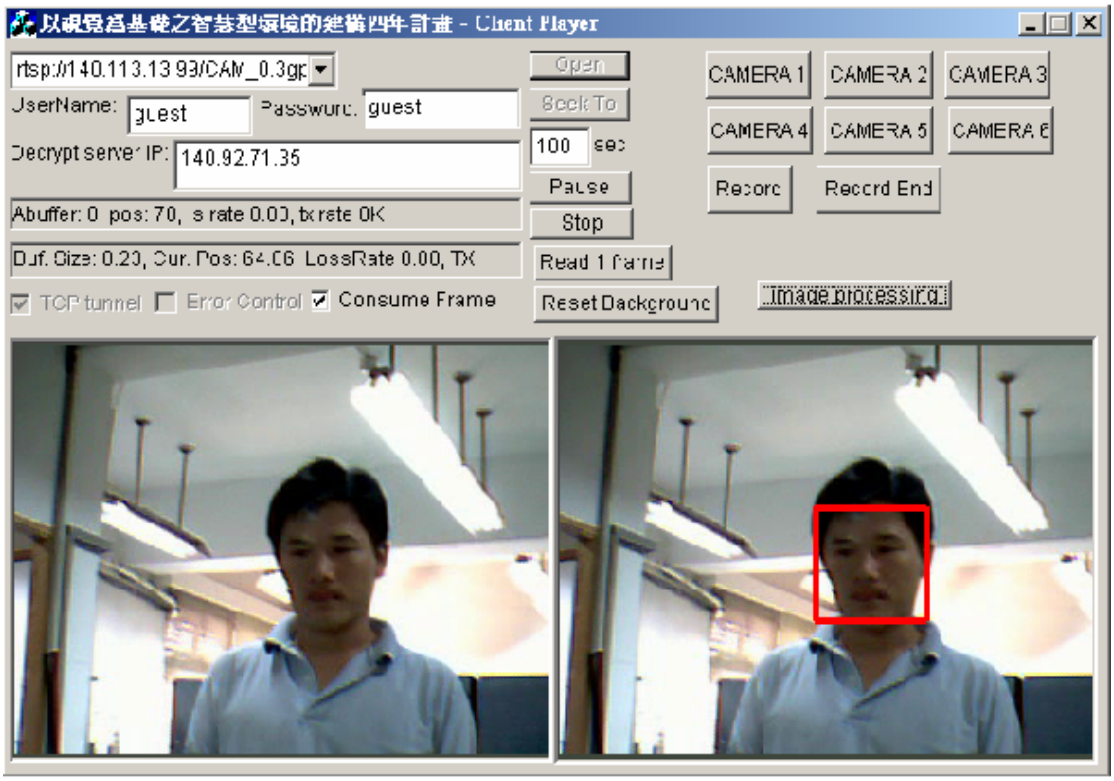


圖 34 結合影像處理與串流平台結果

第四章 RTP Payload Format for MPEG-4 Video Stream and MPEG-2 System

在本章我們將介紹一套標準的網路串流 Open Source Live555，Live555 實做了 RTP/RTSP 等標準網路串流協定，此套標準協定將幫助我們建立一套標準的 RTSP 串流平台。

在多媒體串流的應用中，為了達到即時傳輸的特性，我們捨棄了 TCP(Transmission control protocol)傳輸控制協議，因為 TCP 给與每個封包序號，這個序號除了保證接收端能夠按照順序處理封包外，同時也保證當封包遺失後，傳送端能夠重傳遺失的封包，但這個重傳的動作將會影響到我們所要求的即時傳輸特性，因此，在要求即時傳輸的多媒體串流應用中，我們捨棄了 TCP 而改採用 UDP(User datagram protocol)用戶數據報協議，UDP 只提供資料的不可靠交付，也就是當封包遺失後，UDP 將不採取任何動作，但同時我們也失去了當封包順序錯誤時的回復能力，為了解決這個問題，我們再 UDP 之上再加上了 RTP(Real-time transport protocol)，RTP 在其檔頭裡面加了 16bits 的 sequence number，利用此 sequence number 我們可以訂正封包順序錯誤的情形。本章將敘述 RTP 如何包裹 mpeg-4 video stream 及 mpeg-2 system，並藉由 live555[11]這套支援 RTP 串流格式平台驗證實際串流情形。

4.1 RTP(Real-Time Transport Protocol)

RTP 提供點對點的網路傳輸功能，適用於各種即時傳輸的應用程式當中，例如各種多媒體資料的傳輸，這些多媒體資料被包裝成特定格式的 RTP 封包，然後藉由 UDP 等網路協定傳送到網際網路上。圖 35 則為 RTP 封包的格式，從圖 35 我們可以知道 RTP 封包包含一個檔頭以及在檔頭後面的負載資料，RTP 封包的檔頭格式在 RFC 3550 中有明確定義，而負載資料格式的切割方式，則根據負載資料不同而有不同的定義，例如，RFC 3016[12]定義了 MPEG-4 影像流與聲音流在 RTP 封包的負載資料格式。

圖 35　RTP 封包格式

4.1.1 RTP 檔頭資訊

參考圖 36，RTP 檔頭有以下訊息

- Version (V) : 2bits

指出 RTP 的版本為何，RFC3550 把此值設定為 2。

- Padding (P) : 1bits

此 bit 指出封包的結尾有無 padding data，padding data 可以用於各種加密演算法，padding 的最後一個 byte 則指出有幾個 padding bytes 是可以忽略的。

● Extension(X) : 1bits

當此 bit 設定為 1 時，代表這個 RTP 封包除了固定大小的檔頭外，還放了額外的檔頭資訊。

● CSRC count(CC) : 4bits

一個 RTP Packet 可能是由兩個以上的來源藉由 Mixer 所組合而成的，CSRC(Contributing source)則是來源串流的 ID，因此 CSRC count 會說明這個 RTP packet 是由幾個來源組成的。

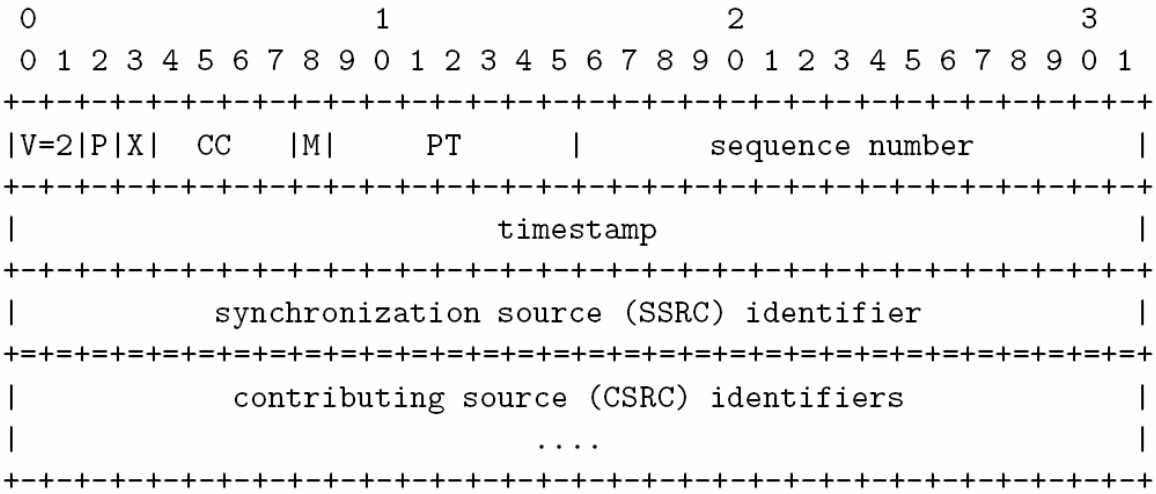


圖 36 RTP Header Format[12]

● Marker (M) : 1 bit

用來說明這個封包是否有重要的事件，例如這個 RTP 封包的 Payload data 中有無 frame 的分界。

● Payload type(PT) : 7 bits

指出 payload data 的類型。

● Sequence number : 16 bits

每增加一個 RTP packet 這個數字就會線性增加，起始值是隨機的，藉由這個數字，接收端可以偵測出封包遺失，以及將順序錯誤的封包進

行重新排列。

- Timestamp : 32 bits

在即時擷取的影音資料時，這個值代表 payload data 中第一個 byte 的取樣時間。當我們要傳送的資料是已經儲存好的多媒體資料時，這個值則是代表 payload data 的播放時間點，因此只要 payload data 的影像資料是屬於同一個 VOP 則這些 RTP packets 的 Timestamp 的值都會是一樣的，初始值則是隨機選擇。

- SSRC : 32 bits

Synchronization source(SSRC)代表這個 RTP packet 的 ID，也就是當這個 RTP packet 藉由 Mixer 組合成其他 RTP 封包時，其他 RTP 封包的 CCRC 值就是這個 RTP 封包的 SSRC 值。

- CSRC list : 0 到 15 個項目，每個項目 32bits

列出這個 RTP 封包是由哪些來源所組成的，列出那些來源的 SSRC 值。

4.1.2 RTP Payload Format for MPEG-4 Audio/Visual Streams

RFC3016 中定義了如何切割 MPEG-4 video stream 於 RTP payload 當中，但首先我們要先介紹 mpeg4 video stream 中 video packet 的觀念，在 mpeg4 video 的壓縮中，使用 video packet 的目的是為了幫助解碼器進行錯誤回復，一個 vop 當中可能會包含很多個 video packet，video packet 如圖 36 所示，在網路傳輸的過程當中，常常會有封包遺

失的情形發生，不幸的是，當遺失的封包包含 VOP header 資料時，將會造成屬於這個 VOP 的所有 Macorblock 都無法解碼，藉由 video packet 將可以解決這個問題，根據 ISO 14496-2，一個擁有 video packet 的 mpeg-4 simpel profile 將會如圖 37 所示，video packet 由許多 Macroblock 所組成，當解碼端讀取到 video packet 時，會開始讀取 Macroblock，每讀取完一個 Macroblock 解碼端會判斷接下來是否出現 video packet header 或 VOP header 若是沒出現這兩個 header 解碼端就判斷接下來仍然是 Macroblock，圖 38 則是 video packet 的細部結構

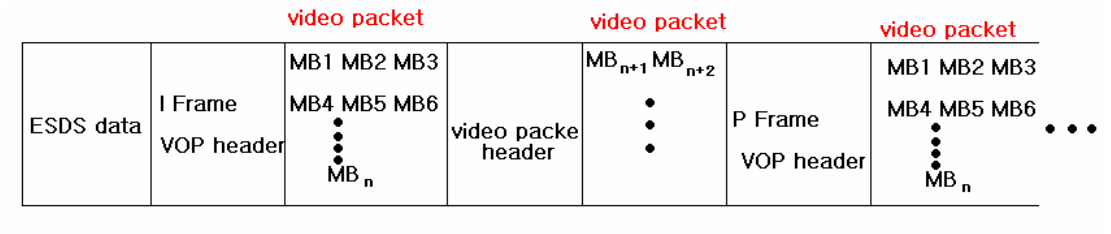


圖 37 一個有 video packet 的 mpeg-4 video stream

參考圖 38，video packet 首先由連續 16 個 0 形成 resync_marker，在 resync_marker 之後會有 Macroblock number，

Sync	Header	HEC	(Header)	Macroblock data	Sync

圖 38 video packet structure

Macroblock number 的 bit 數可以從 1bit 到 14bits，這個 bit 數是跟 VOP 的大小有關，舉例來說，一個 vop_width = 320 而 vop_height = 240 參

考表 12[6]，則這個 Macroblock number 需要 9bits 來表示所有 VOP 的 Macroblock。每一個 VOP 其 Macroblock 都有特定的 Macroblock number 最左上角的 Macroblock 其 Macroblock number 為 0，最右下角的 Macroblock 其 Macroblock number 最大，每增加一個 Macroblock

length of macroblock_number code	((vop_width+15)/16) * ((vop_height+15)/16)
1	1-2
2	3-4
3	5-8
4	9-16
5	17-32
6	33-64
7	65-128
8	129-256
9	257-512
10	513-1024
11	1025-2048
12	2049-4096
13	4097-8192
14	8193-16384

表 12　Length of macroblock_number code[6]

則 Macroblock number 會增加 1，以剛剛那個例子來說，若一個 video packet 其 Macroblock number = 000011001，則代表這個 video packet 所攜帶的 Macroblock 位於如圖 39 黑色部份所示的位置，在

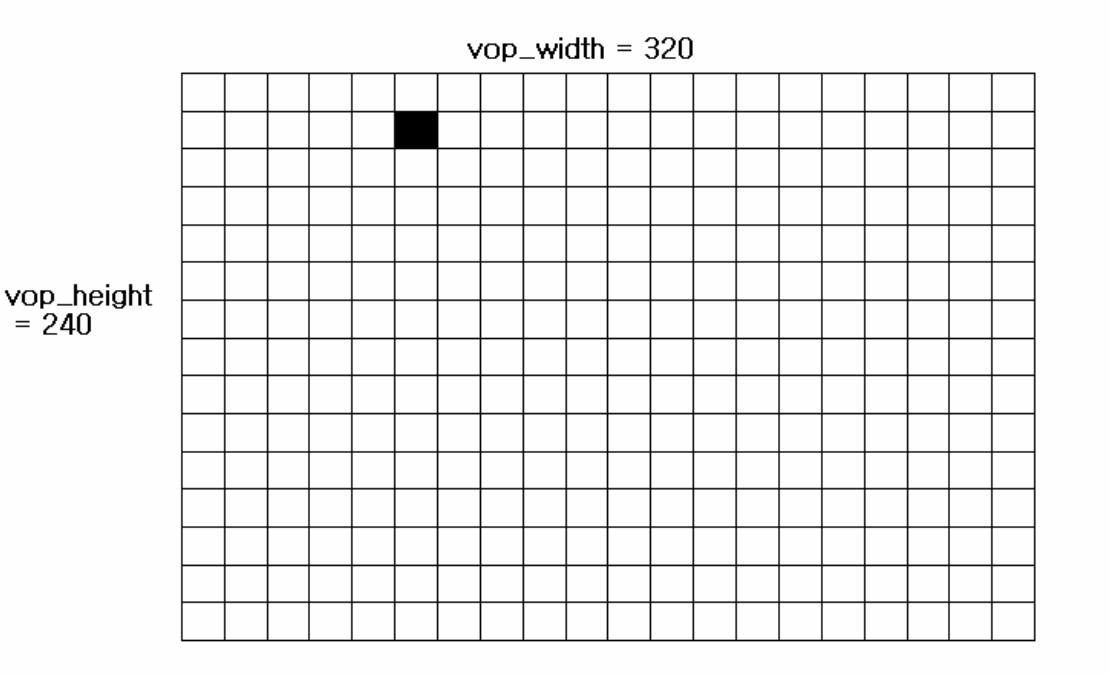


圖 39 Macroblock number = 25

Macroblock number 之後就是 1 個 bit 的 HEC(header_extension_code) flag，若是這個 flag 被設定成"1"，則在 HEC 之後會列出這個 video packet 所屬的 VOP 其檔頭資訊，例如 vop_width、vop_height、vop_time_increment 等資訊，因此藉由 video packet 所攜帶的資訊，就算之前的 rtp 封包遺失，解碼器還是能找到目前這個 rtp 封包其 Macroblock 位於 VOP 何處，同時也可以藉由重複出現的 VOP 檔頭資訊得到如何解碼的方式。

RFC3016 基本上建議一個 RTP Packet 裡面只放一個 VOP，這是因為每個 RTP Packet 檔頭的 Timestamp 會隨著不同 VOP 而不同，但是在實際的應用上，這種作法會造成頻寬的浪費，因為有時候 VOP 會只有 VOP header 而沒有 video packet(vop_coder = 0)，有時候某種形

狀的 VOP 其 coding block 很小也就是 video packet 很小，這種情形下若是仍舊讓一個 RTP packet 只包含一個 VOP 將會造成 OVERHEAD 的效應，因此 RFC3016 允許串聯多個 VOP 於一個 RTP packet 中，有鑑於此 RFC3016 對於 Timestamp 的設定有額外定義如下

- 若有多個 VOP 包在同一個 RTP 裡，則 RTP 的 Timestamp 定義為這些 VOP 中最早出現的一個，至於其他 VOP 的 Timestamp 則可由 VOP header 的 modulo_time_base 與 vop_time_increment 進行推算。
- 假如這個 RTP packet 只包含 Configuration information(包括 Visual Object Sequence Header、Visual Object Header 和 Video Object Layer Header)則 RTP Timestamp 定義為下一個出現的 VOP。
- 假如 RTP packet 只包含 visual object sequence end code 則 Timestamp 會設定為上一個 VOP 的 Timestamp。

RTP Timestamp 在 mpeg4 設定中其解析度為 90kHz，Timestamp 的算法則如圖 40 所示，因此若是每秒出現 25 個 VOP 則每增加一個 VOP 其 Timestamp 會增加 3600。

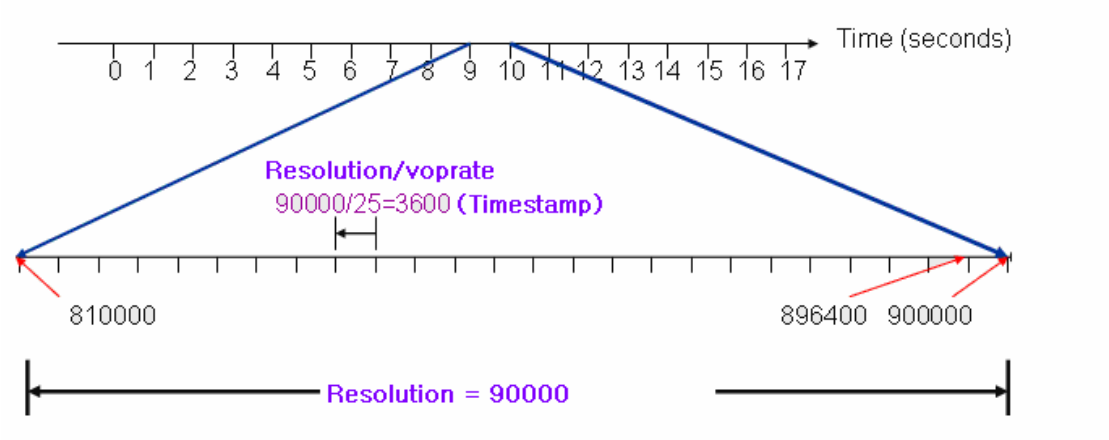


圖 40 RTP Timestamp

接下來我們要介紹 RFC3016 中如何切割 MPEG-4 video stream 於 RTP payload 中，在那之前我們先定義檔頭的意義

- Configuration information 包括 Visual Object Sequence Header，Visual Object Header 和 Video Object Layer Header。
- visual_object_sequence_end_code
- Group_of_VideoObjectPlane( )或者是 VideoObjectPlane( )、video_plane_with_short_header( )，MeshObject( )及 FaceObject( )的檔頭。
- video packet header
- gob_layer( )的檔頭。

知道 mpeg-4 video codec 的檔頭定義後，接下來將說明 RFC 3016 對於 MPEG-4 video stream 的切割規則。

(1) Configuration information 和 Group_of_VideoObjectPlane( )應該要放在 RTP payload 的最前面。

(2) 若同時有多個 header 存放在 RTP payload 中，則在語法排序上最高的要放在最前面，visual_object_sequence_end_code 是最低的語法排序。

(3) 同一個 header 不能拆解到兩個不同的 RTP packet。

(4) 除非 VOP 的 size 太小，否則盡量一個 VOP 放在一個 RTP packet 裡面。

(5) 盡量讓同一個 video packet 放在同一個 RTP packet 中。

根據以上五項規則，RFC3016 列出了幾個 RTP Packet 的例子，如圖 41 所示，圖 41-d 是一個 RTP packet 包裝一個 video packet，這種包裝方式適用於當網路常有掉封包的情形，因為即使包裝著 VOP header 的 RTP packet 遺失，其他的 RTP packet 也能藉由存放於 video packet header 中的 HEC information 進行解碼，這種作法就不需要額外的 RTP header 來幫助錯誤回復。圖 41-e 則是同時讓多個 video packet 放在一個 RTP packet 的例子，這種包裝方式適用於網路頻寬較差的情形，可以減少因為 RTP/IP header 所造成的 overhead 的問題，然而這也同時減少當 RTP packet 遺失後解碼端進行錯誤回復的能力，因為遺失一個 RTP packet 將會損失多個 video packet，至於一個 RTP packet 要放多少個 video packet 則是依據網路頻寬與封包遺失率來決定。在 VOL header 中有個 resync_marker_disable bit，當這個 bit 被設定為”1”

代表這個 mpeg-4 codec 不支援 video packet 的作法，當我們希望每個 RTP packet 都是固定大小時，往往就會形成如圖 41-f 的情形，這種情形下的 RTP packet 不適用於容易掉封包的網路，因為解碼端無法進行錯誤回復。

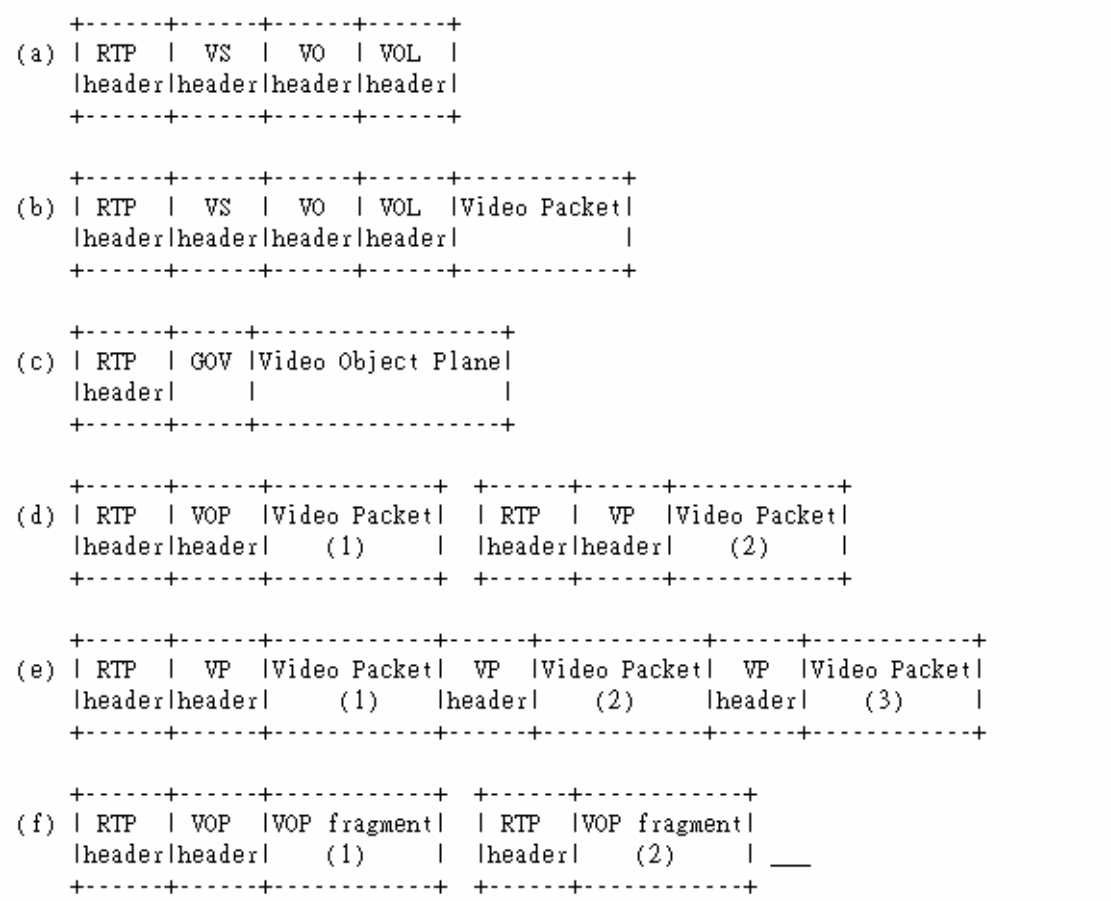


圖 41　RTP packet 例子

4.2 Live555[11]

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

Live555 包含了 server 以及 client 部份，這套函式庫可以用來傳送和接收多媒體串流資料，配合各種編碼器可以整合成 video on demand 串流模組，也可以配合相關之解碼器以及應用程式整合成為串流播放模組，目前已經有 Player 整合 Live555 成為其支援串流視訊的主要模組，例如 VideoLan(VLC Player)以及 MPlayer。Live555 目前可以支援多種 codec 的標準 RTP 封裝及解封裝，如 AC3、AMR、H.261、JPEG、MP3、MPEG1-2、MPEG-2 TS 、MPEG-4 ES、MPEG-4 Audio、CELP、以及 WAV 等。本節將描述 Live555 如何包裝 MPEG-4 ES 以及抽取出 RTP 封包並對其進行分析。

4.2.1 Rtsp Command in Live555

Live555 提供 RTSP 用於當做遠端連線的控制器，以下將介紹 Live 555 所提供的 Command function。在這邊 Client 使用 VideoLanClient(VLC) 而這些 Command function 的意義為當 Server 端收到某一個 Command 時會利用這個 Command 對應的 Command funtion 做出對應的動作，

例如 Server 收到”PLAY”Command 則會利用 handleCmd_PLAY( )做出對應的動作。

• handleCmd_DESCRIBE( )

Client 以”DESCRIBE”要求 Server 以何種描述方式來描述播放資訊，描述方式有三種 application/sdl、application/rtsl、application/mheg，例如 client 要求 server 以 application/sdp 的方式來描述 rtsp:\\140.113.13.98:554/test.m4v 則 client 的要求將會如下所示

```
DESCRIBE rtsp://140.113.13.98:554/test.m4v RTSP/1.0
CSeq: 29
Accept: application/sdp
User-Agent: VLC Media Player (LIVE.COM Streaming Media v2004.11.11)
```

而 server 的回應會如下所示，在這邊由於我們傳送的是 mpeg-4 ES 所以 server 的回應訊息會包括這個 test.m4v 檔的 configuration information(包括 visual object sequence header 等資訊)。

```
RTSP/1.0 200 OK
CSeq: 29
Date: Mon, Aug 14 2006 08:47:10 GMT
Content-Base: rtsp://0.0.0.0/test.m4v/
Content-Type: application/sdp
Content-Length: 546

v=0
o=- 1155545228609000 1 IN IP4 0.0.0.0
s=Session streamed by "IIIVideoOnDemandRTSPServer"
i=test.m4v
t=0 0
a=tool:LIVE.COM Streaming Media v2005.02.09
a=type:broadcast
a=control:*
a=range:npt=0-
a=x-qt-text-nam:Session streamed by "IIIVideoOnDemandRTSPServer"
a=x-qt-text-inf:test.m4v
m=video 0 RTP/AVP 96
c=IN IP4 0.0.0.0
a=rtpmap:96 MP4V-ES/90000
a=fmtp:96 profile-level-id=245;
config=000001B0F5000001B509000001000000012008C49DC00043A9C0095000B0D497530C1F4C2C1078710F
000001B2656D34762034 2E332E322E3800C9FF00
a=control:track1
```

• handleCmd_Setup( )

利用 SETUP 描述串流資料的傳輸機制，底下的例子 client 想要建立

一個 rtsp://140.113.13.98:554/test.m4v 的 RTSP 連線，而 RTP 封包則是再 port 1470 和 1471 進行接收，Server 則回應會從 port 1472 和 1473 丟出 RTP 封包。

Client 對 Server 的要求

```
SETUP rtsp://140.113.13.98:554/test.m4v/track1 RTSP/1.0
CSeq: 38
Transport: RTP/AVP;unicast;client_port=1470-1471
User-Agent: VLC Media Player (LIVE.COM Streaming Media v2004.11.11)
```

Server 對 Client 的回應

```
RTSP/1.0 200 OK
CSeq: 38
Date: Mon, Aug 14 2006 11:01:34 GMT
Transport: RTP/AVP;unicast;destination=140.113.13.98;client_port=1470-1471;
server_port=1472-1473
Session: 1
```

- handleCmd_OPTIONS( )

Client 利用 OPTIONS 來詢問 Server 支援哪些功能，Server 則將其支援的功能送出，如下面所示

```
RTSP/1.0 200 OK
CSeq: 41
Date: Mon, Aug 14 2006 11:24:48 GMT
Public: OPTIONS, DESCRIBE, SETUP, TEARDOWN, PLAY, PAUSE
```

- handleCmd_PLAY( )

Client 藉由 PLAY 這個指令通知 Server 開始傳送所需的多媒體資料，如下面所示為 Client 對 Server 的 Play 要求，npt＝0 代表要求檔案從一開始的地方播放。

```
PLAY rtsp://140.113.13.98:554/test.m4v RTSP/1.0
CSeq: 9
Session: 1
Range: npt=0.000-
User-Agent: VLC Media Player (LIVE.COM Streaming Media v2004.11.11)
```

下面則為 Server 收到 PLAY request 所做的回應

```
RTSP/1.0 200 OK
CSeq: 9
Date: Mon, Aug 14 2006 12:42:26 GMT
Range: npt=0.000-
Session: 1
RTP-Info: url=rtsp://0.0.0.0/test.m4v/track1;seq=43580
```

4.2.2 M4V RTP payload in live555

從上一小節我們可以知道當 Client 送出"PLAY"Command 後，Server 將會開始傳送 RTP packet，在 Live555 中藉由 buildAndSendPacket( ) 進行 RTP packet 的包裝與傳送。如圖 42 所示，packFrame( )會將

```
void MultiFramedRTPSink::buildAndSendPacket(Boolean isFirstPacket) {
  fIsFirstPacket = isFirstPacket;

  // Set up the RTP header:
  unsigned rtpHdr = 0x80000000; // RTP version 2
  rtpHdr |= (fRTPPayloadType<<16);
  rtpHdr |= fSeqNo; // sequence number
  fOutBuf->enqueueWord(rtpHdr);
  fTimestampPosition = fOutBuf->curPacketSize();
  fOutBuf->skipBytes(4); // leave a hole for the timestamp

  fOutBuf->enqueueWord(SSRC());

  // Allow for a special, payload-format-specific header following the
  // RTP header:
  fSpecialHeaderPosition = fOutBuf->curPacketSize();
  fSpecialHeaderSize = specialHeaderSize();
  fOutBuf->skipBytes(fSpecialHeaderSize);

  // Begin packing as many (complete) frames into the packet as we can:
  fTotalFrameSpecificHeaderSizes = 0;
  fNoFramesLeft = False;
  fNumFramesUsedSoFar = 0;
  packFrame();
}
```

圖 42　buildAndSendPacket( )

Payload 所需的資料放進去每個 RTP packet 中最後再利用 UDP 將這些封包傳送出去。圖 43 則為實際擷取這些 RTP 封包並對其封包內容進行分析的結果，從圖 43 我們可以看到 PACKET 0 到 PACKET7，因為其 payload data 都包裝同一個 VOP 所以其 RTP header 的 timestamp

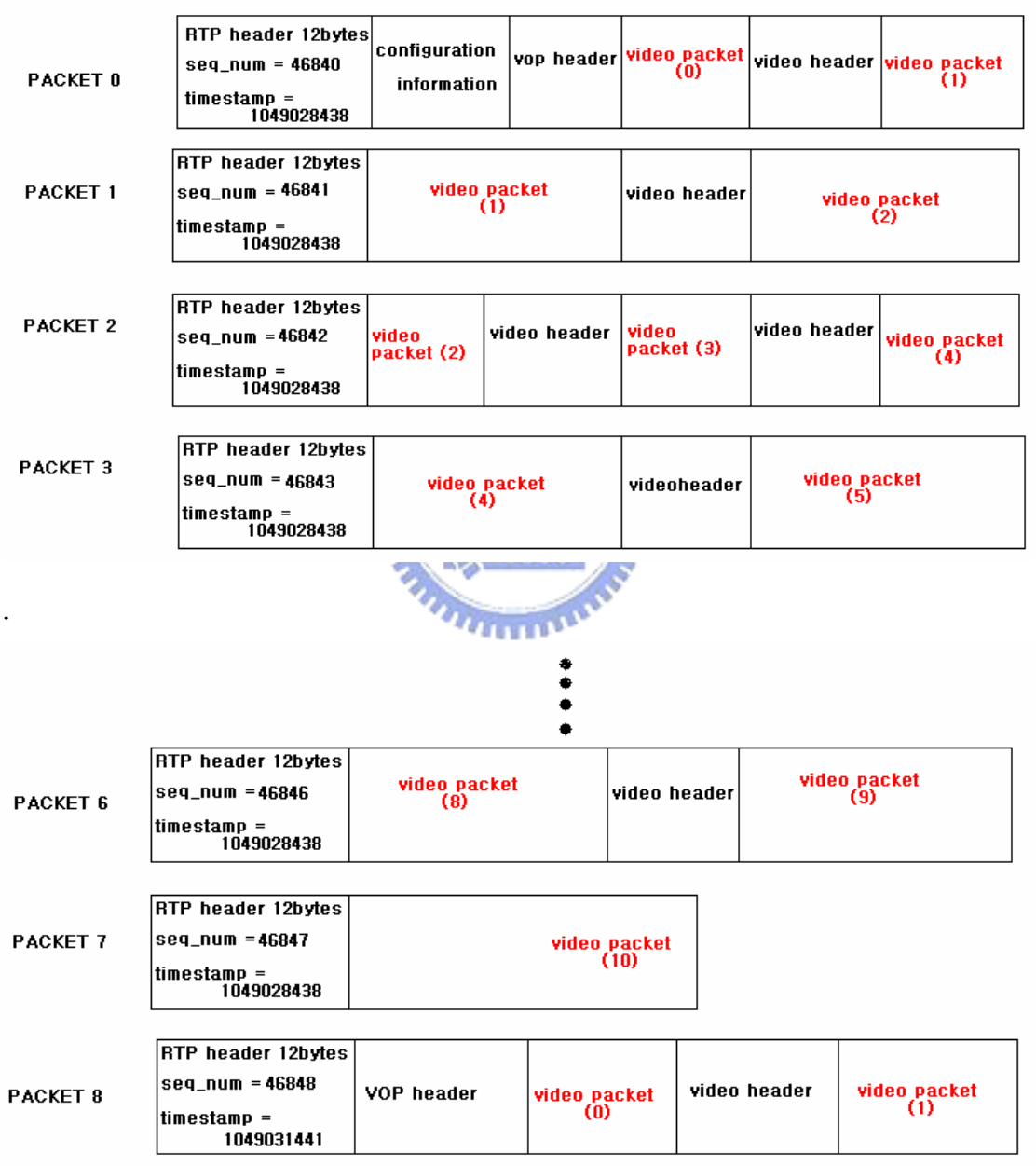


圖 43　mpeg-4 video stream in RTP payload

會是同一個值 1049028438，而 PACKET 8 的 Timestamp，因為這理

的 voprate=29.97/sec 則根據圖 39 其 Timestamp 會增加 90k/29.97 = 3003。至於 seq_num 則一開始會給個隨機的值之後每增加一個封包其 seq_num 會增加 1，Live555 包裝 RTP packet 的方式是藉由讓每個 RTP payload 固定大小為 1436bytes 來減少因為檔頭所造成的 overhead，但這樣卻失去了 mpeg-4 video packet 所提供的錯誤回復能力，以圖 43 為例，當 PACKET 2 這個封包遺失，將會影響到 PACKET 1 的 video packet (2)以及 PACKET 3 的 video packet (4)，但若是使用圖 40-d 的包裝方式，則每遺失一個封包將不會影響其他的 video packet。

4.3 RTP payload format for mpeg-2 system

以隨選視訊系統來說，多媒體串流必須有能力串流儲存在硬碟中的各種影音檔案，這種影音檔案除了影像流與聲音流外，還包括描述這些影像流與聲音流的檔頭，例如第二章曾經介紹過的 mpeg-2 program stream 就是一種包裹影像流與聲音流的檔案格式，以串流伺服器的角度來說，其重點是抽取出這些影音檔案的聲音流與影像流，再將這些影像流與聲音流包裹於 RTP payload 中，本節將以 mpeg-2 program stream 為例子說明 live555 拆解 mpeg-2 program stream 並將 mpeg-2 影像流包裹於 RTP payload 中。

4.3.1 mpeg-2 video stream

在第二章中我們曾經介紹過 mpeg-2 program stream 如何包裝影像流與聲音流，本小節將介紹由 ISO 13818-2[13]所定義的 mpeg-2 video stream，mpeg-2 video stream 的結構如圖 44 所示，mpeg-2 video stream 是由 I Frame 、B Frame 以及 P Frame 所組成，每個 Frame 由許多 Slice 組成，Slice 中包括許多 Macroblock 這些 Macroblock 是組成 frame 的最基本單位，同一個 Slice 的所有 Macroblock 會位於一張 frame 的同一個水平位置上如圖 45 所示，而在每個 I Frame 之前會有 ESDS data 包括 sequence_header、group of picture header 等，這些檔頭將描述此影像流的相關資訊包括 framerate，frame size 等資訊。

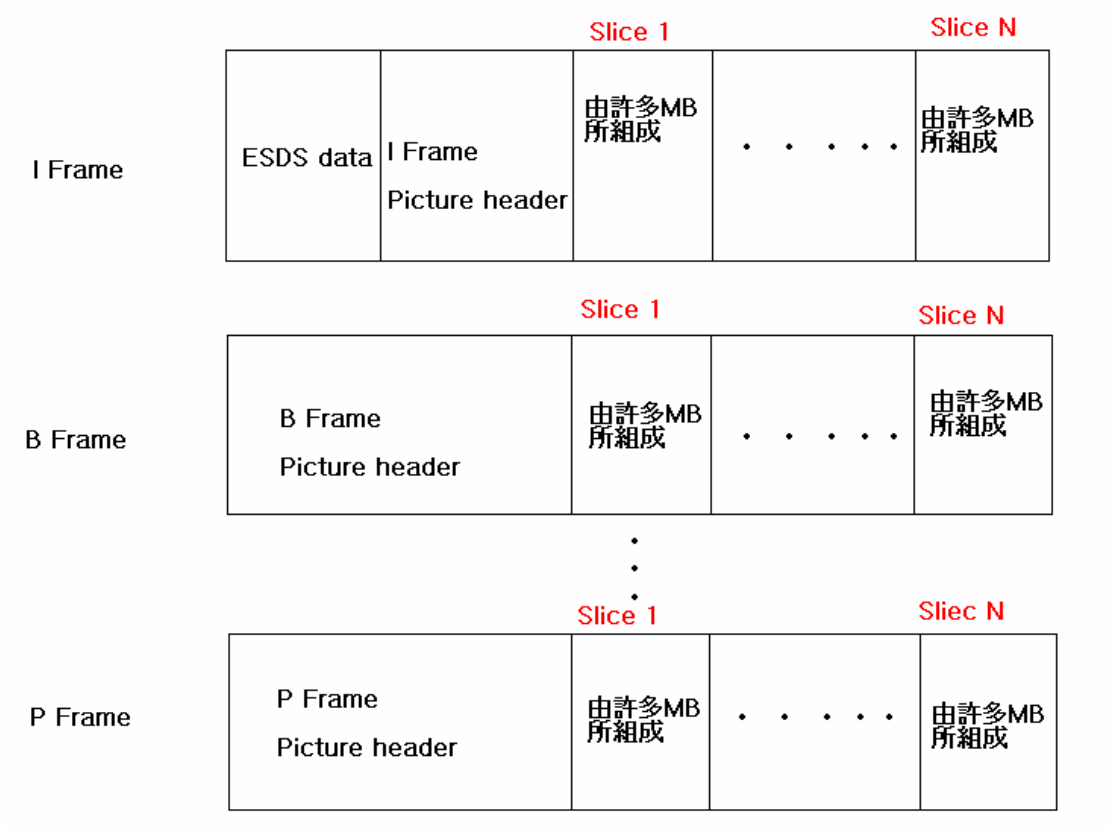


圖 44 a mpeg-2 video stream example

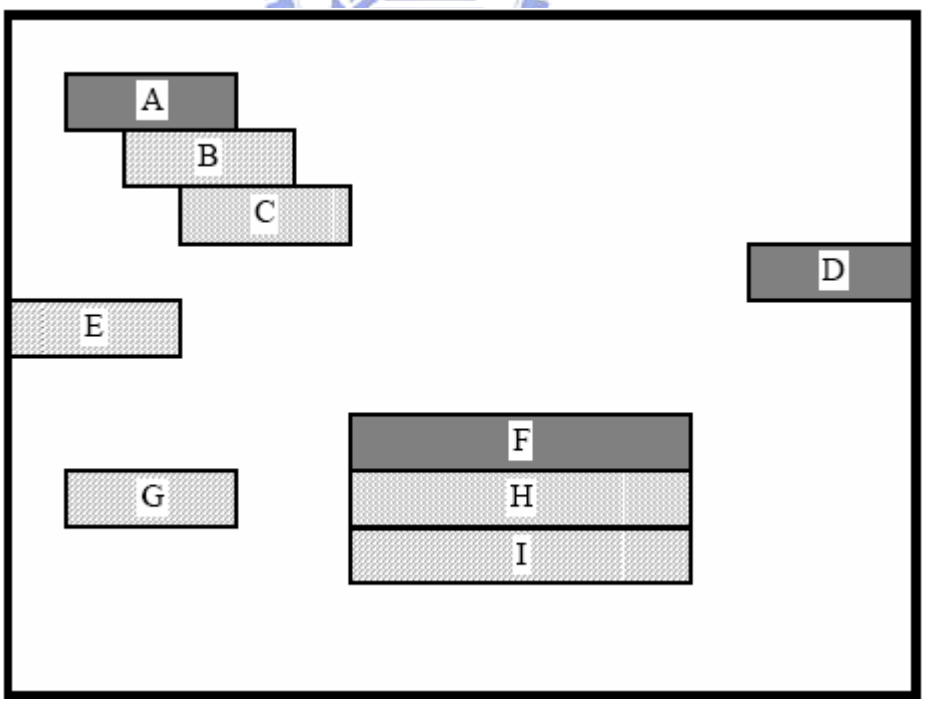


圖 45　frame 中 slice 分布示意圖[13]

4.3.2 Live555 包裝 Mpeg-2 System 的方式

根據第二章，mpeg-2 program stream 包括 PACK header、System header、PES header 以及 PES payload，圖 44 所示的 mpeg-2 video stream 是放在 mpeg-2 program stream 的 PES payload 中，live555 抽取出 mpeg-2 video stream 並將其包裹於 RTP packet 中，其包裹方式如圖 46 所示，不同於圖 43，live555 對 mpeg-2 video stream 是以 Slice 為單位，因此並不要求每個 RTP packet 都有固定大小，這樣的作法讓 RTP packet 遺失後不會影響前後封包，以圖 45 為例，當包裹 Slice F 的 RTP packet 遺失後，剩下的 Slice 仍可以順利解出，因為包裹 Slice F 的 RTP packet 不會包裹其他的 Slice，但這樣的作法需要消耗較大的頻寬。

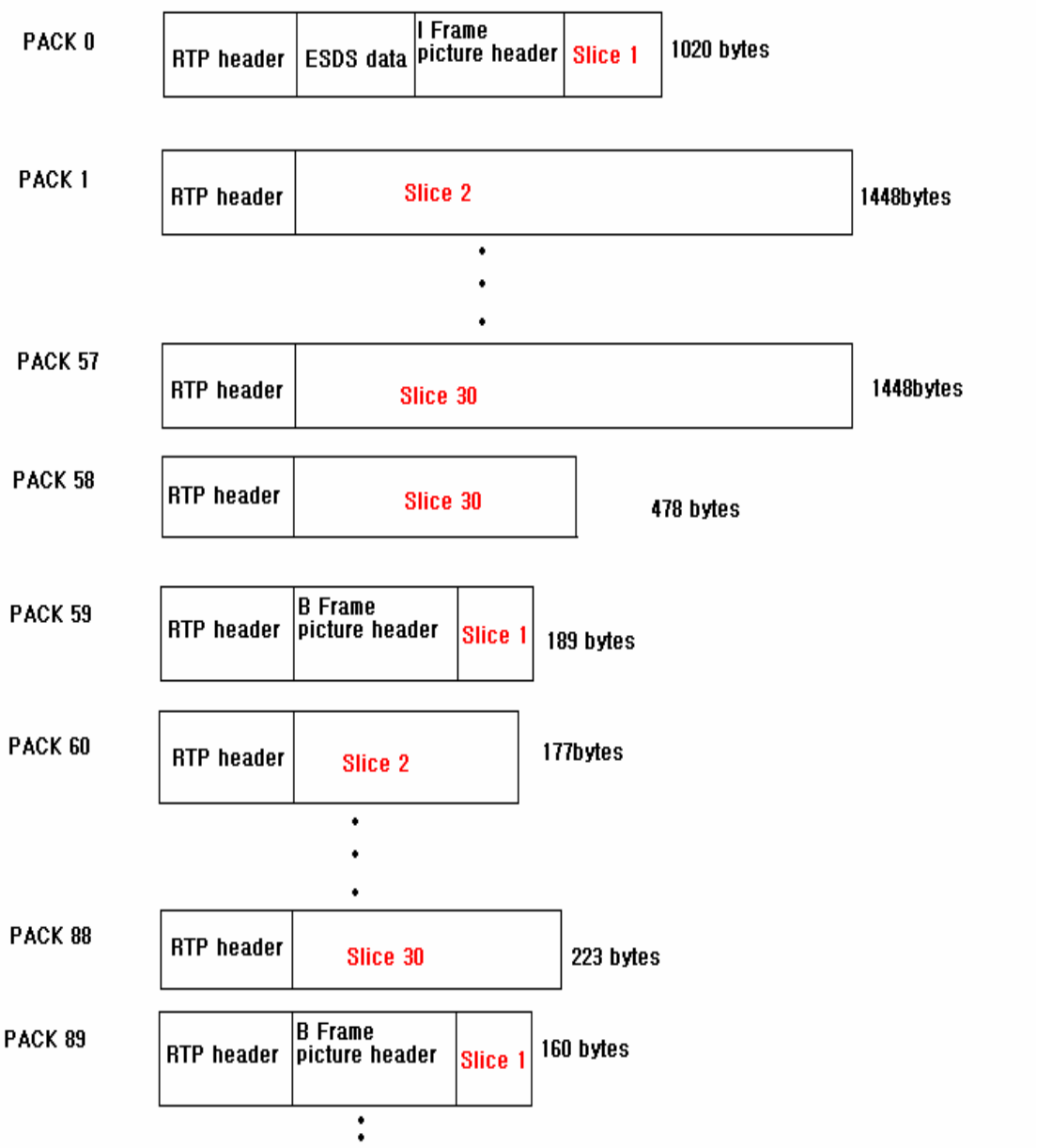


圖 46 mpeg-2 video stream in RTP payload

第五章　多媒體串流環境建構

Mpeg-4 simple profile 提供了極好的壓縮比，但卻會消耗很多 cpu 資源，為了解決這個問題，本章以硬體進行 mpeg-4 simple profile 壓縮，至於解壓縮則因為消耗的資源有限因此仍以軟體進行 mpeg-4 simple profile 的解壓縮，藉由硬體壓縮晶片將影像來源進行 mpeg-4 simple profile 的壓縮，同時將此 mpeg-4 video stream 包裝成 mpeg-2 program stream 或是 mpeg-2 transport stream，本章的第一小節將抽取出 mpeg-2 program stream 的影像流與聲音流，並利用 3.2.1 介紹的串流平台，對這個影像流進行串流及解碼播放。

多媒體串流的一項重要應用是作為遠端安全監控，藉由即時擷取影像並對影像做智慧型處理達到安全監控的目的，本章第二節將建構一個雙攝影機來源的串流平台，利用 Directshow 從影像擷取卡中擷取影像並利用 3.2.1 的串流平台進行串流及遠端播放。

5.1 軟硬體多媒體串流平台整合

在第二章中我們曾經介紹過如何拆解出 mpeg-2 program stream 的影像流與聲音流，利用硬體壓縮晶片壓縮出來的 mpeg-2 program stream 其影像流將會是 mpeg-4 simple profile，圖 47-a 及 b 則藉由建立 MPEG-2 System 的資料庫抽取出此硬體壓縮晶片壓縮出來檔案的影像流並進行串流的結果。

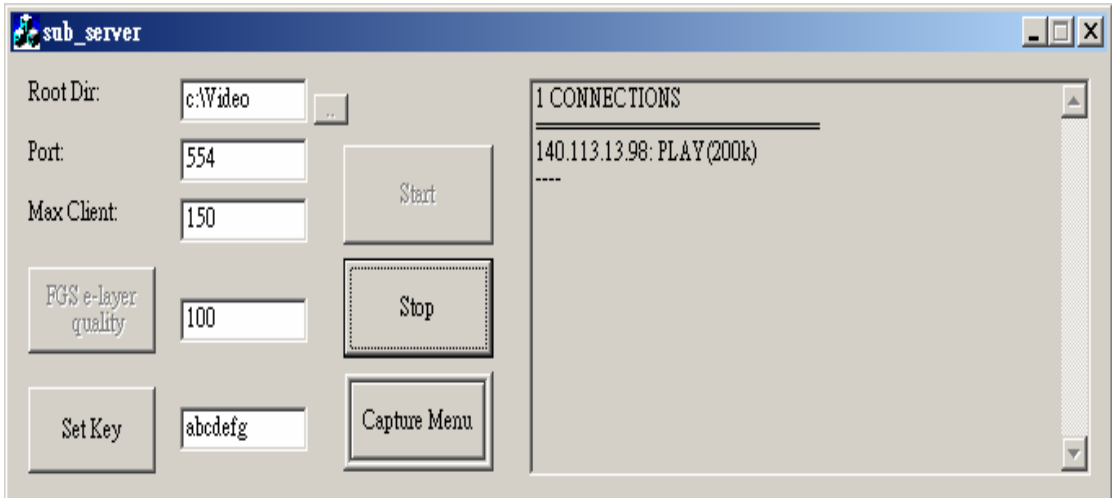


圖 47-a 串流平台伺服端介面

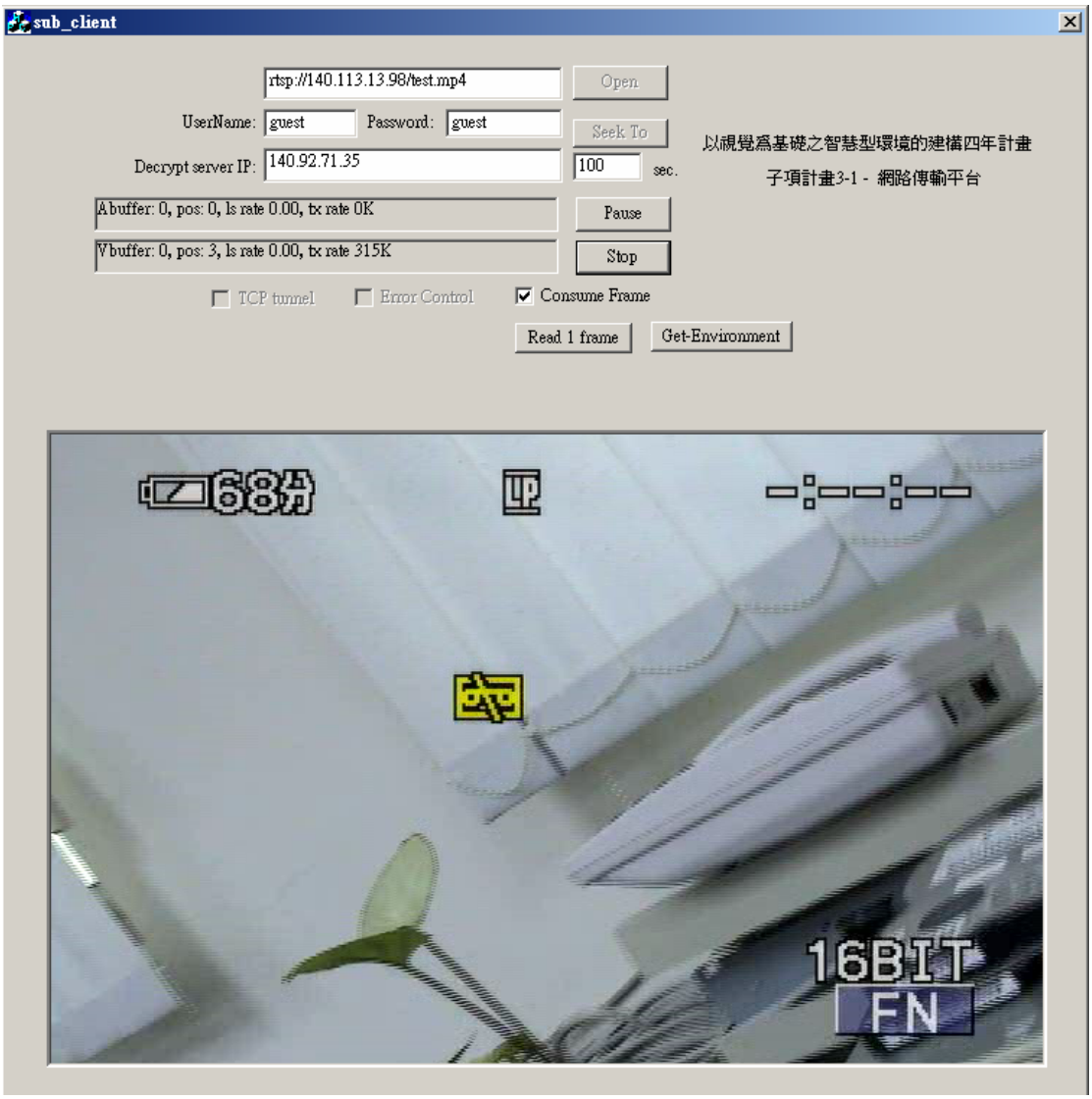


圖 47-b 串流平台接收端介面

5.2 雙攝影機串流平台整合

在智慧型安全監控的環境下，我們可以使用雙攝影機模組於大型空間中進行偵測與追蹤，藉由場景攝影機監控整個大型場景，而目標物攝影機則是利用其 PTZ 功能取得目標物之清晰、可辨識的影像，由於場景攝影機可以監控整個大型空間，因此可以同時偵測並追蹤多個目標物，但現行的作法只能在本地端進行監控，此平台的建立將可整合雙攝影機監控模組應用於遠端監控上。

5.2.1 雙攝影機影像擷取

在雙攝影機的影像擷取部份，我們是利用微軟提供的 VFW(Video for windows)SDK，VFW SDK 提供了在 Windows 系統中進行影像擷取所需要的 API，利用此 SDK 可以大大的降低應用程式的開發難度。開發視訊擷取程式需要用到以下幾個 VFW API[14]

- capCreateCaptureWindow( )

創建擷取窗，可以在擷取窗上面進行影像的預覽，此函式會回傳一個 handle 利用這個回傳的 handle 我們可以控制影像擷取的各種設定，包括影像來源、擷取速率等。

- capDriverConnect( )

單獨定義一個擷取窗是不能工作的，擷取窗必須與一個影像輸入裝置連接，這樣才能取得影像訊號，此函式可使一個擷取窗與一個影像擷取驅動程式相連。

- capPreviewScale( )、capPreviewRate( )

當擷取窗連接了影像擷取裝置的驅動程式後，利用此兩函式進行影像預覽時候的設定，capPreviewScale( )用來設定預覽影像的預覽比例，而 capPreviewRate( )則用來設定擷取影像的 frame rate。

- capPreview( )

利用此函式，將影像在擷取裝中進行預覽。

透過以上幾個函式，可以建立一個基本的影像擷取程式，接下來則是利用 VFW SDK 提供的函式，得到影像的 bmp raw data。

- capSetCallbackOnFrame( )

此函式為應用程式建立一個 callback function，每次當收到新的 frame 時，由此函式所建立的 callback function 將會執行一次，因此我們藉由此 callback function 得到每個 frame 的 raw data。

5.2.3 雙攝影機串流平台建構

圖 48 為雙攝影機伺服端介面，此介面將場景攝影機放在畫面的上方，將目標物放在界面下方，此介面的左邊則是使用 VFW SDK 的預覽模式所畫出的影像，右邊則是利用 callback function 擷取每張 frame 的 bmp raw data，藉由設定 bmp header 將結果繪於 MFC 中，擷取下來的 bmp raw data 壓縮成 mpeg-4 simple profile 再包裝成如第四章所示的 RTP packet 進行串流。

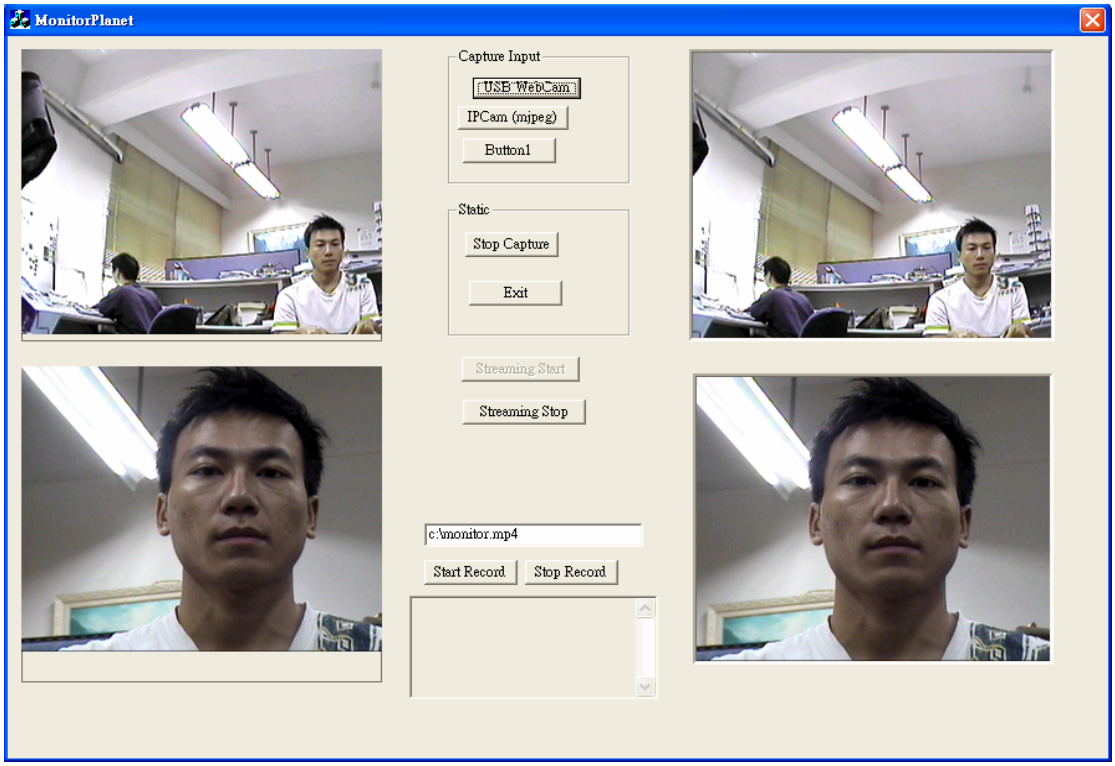


圖 48 雙攝影機伺服端介面

圖 49 則為雙攝影機接收端介面，接收到 RTP 封包後將 RTP 封包的 payload data 以 frame 為單位組合，組合好的 frame 即為一個 mpeg-4 simple profile 的 VOP，接著將這些 VOP 進行解碼，解碼後即為一張照片的 bmp raw data，圖 49 則將這些 bmp raw data 加上 bmp header 後畫在 MFC 的介面中。

圖 49 雙攝影機接收端介面

第六章　結論

Codec、檔案包裝格式及網路傳輸是多媒體串流的三大要素，一個完美的平台必須能夠拆解出各種包裝格式的影音流與聲音流，同時這個平台更需要準備各種 Codec 來對拆解出來的影音流與聲音流進行編碼及解碼，這樣的平台在自由軟體基金會開放原碼，以及全世界的軟體工作者的努力下已經逐漸形成，在本篇論文的第三章曾介紹 FFmpeg 這個開放原碼如何看的懂所有的包裝格式。

在 real-time 影像擷取，並對其進行壓縮串流的應用上，本篇論文也利用 VFW SDK 建構出雙攝影機的串流平台，同時在第三章中也利用 OpenCV 進行人臉追蹤。在 real-time 的應用中擷取出來的影像為了減少頻寬的使用勢必要對其進行影像壓縮，在將來的應用當中尤其是安全監控的應用上，攝影機的數量將會越來越多，這樣將造成 cpu 極大的負擔，為了解決這個問題在影像壓縮上勢必需要硬體晶片的幫忙，本篇論文嘗試串流一個硬體晶片所壓縮出來 mpeg-4 simple profile 其包裝格式為 mpeg-2 system 包括 program stream 與 transport stream，由於在包裝成 RTP packet 前必須先抽取出影像流與聲音流，第二章提供了關於如何在 mpeg-2system 中抽取影像流與聲音流的相關知識，但抽取出的影像流卻因為 codec tool 使用上的不同，使得同樣是 mpeg-4 simple profile 的解碼器無法解碼，這個差異在第二章與第五章

曾做過介紹。

在未來，我們除了希望建構出一套完美的串流平台外，在硬體壓縮上，希望能做到即時性的 mpeg-4 simple profile 壓縮串流平台。

第七章　參考文獻

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

[2] RealNETWORKS Corp. ,http://www.realNetworks.com

[3] MicroSoft Corp,.http://www.microsoft.com

[4]Apple Computer, Inc., http://www.apple.com

[5]ISO/IEC 13818-1 Generic coding of moving pictures and associated audio:systems

[6]ISO/IEC14496-2 Information technology-coding of audio-visual objects-Part2:Visual

[7]Free Software Foundation http://www.fsf.org/

[8]FFmpeg http://ffmpeg.mplayerhq.hu/

[9]OpenCV http://www.intel.com/research/mrl/research/opencv/

[10]Chien-hua Chen Design and Implementation of a Real-time Interactive RTP/RTSP Multimedia Streaming Monitoring System with Bandwidth Smoothing Technique

[11]live555 http://www.live555.com/liveMedia/

[12]RFC 3016 - RTP Payload Format for MPEG-4 Audio/Visual Streams

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

[14] VFW API available at http://msdn.microsoft.com/library/default.asp?url=/library/en-us/multimed/htm/_win32_video_for_windows.asp

簡歷

張為棟，民國七十一年出生於台北縣，民國九十三年畢業於國立中央大學電機工程學系，同年進入國立交通大學電信研究所從事多媒體通訊相關研究，民國九十五年取得碩士學位，論文題目為：整合式多媒體串流平台的發展於實做，研究興趣為多媒體通訊相關。