

行政院國家科學委員會專題研究計畫 成果報告

總計畫(III)

計畫類別：整合型計畫

計畫編號：NSC94-2219-E-009-007-

執行期間：94 年 08 月 01 日至 95 年 07 月 31 日

執行單位：國立交通大學電子工程學系及電子研究所

計畫主持人：杭學鳴

共同主持人：蔡淳仁，蔣迪豪，林大衛，李素瑛，王聖智

計畫參與人員：唐之璇，蔡家揚，施瑛姿，鍾國洋，朱育成，劉建志，
鄭凱庭，陳威年，葉尚諭

報告類型：完整報告

處理方式：本計畫可公開查詢

中 華 民 國 95 年 10 月 17 日

行政院國家科學委員會補助專題研究計畫 ☒ 成果報告
☐ 期中進度報告

基於 MPEG 標準之多媒體通訊與串流整合平台及其應用(總計畫)(III)
MPEG Integrated Multimedia Platform and Applications (III)

計畫類別：☐ 個別型計畫 ☒ 整合型計畫

計畫編號：NSC 94-2219-E-009 -007

執行期間：94 年 8 月 1 日至 95 年 7 月 31 日

計畫主持人：杭學鳴 國立交通大學電子工程學系 教授
共同主持人：王聖智 國立交通大學電子工程學系 教授
李素瑛 國立交通大學資訊工程學系 教授
蔡淳仁 國立交通大學資訊工程學系 助理教授
蔣迪豪 國立交通大學電子工程學系 副教授

計畫參與人員：唐之璇，蔡家揚，施瑛姿，鍾國洋，朱育成，劉建志，鄭凱庭，陳威年，葉尚諭

成果報告類型(依經費核定清單規定繳交)：☐ 精簡報告 ☒ 完整報告

本成果報告包括以下應繳交之附件：

- ☐ 赴國外出差或研習心得報告一份
- ☐ 赴大陸地區出差或研習心得報告一份
- ☒ 出席國際學術會議心得報告及發表之論文各一份
- ☐ 國際合作研究計畫國外研究報告書一份

處理方式：除產學合作研究計畫、提升產業技術及人才培育研究計畫、列管計畫及下列情形者外，得立即公開查詢

☐ 涉及專利或其他智慧財產權，☐ 一年 ☐ 二年後可公開查詢

執行單位：國立交通大學電子工程學系

中 華 民 國 95 年 10 月 15 日

行政院國家科學委員會專題研究計畫成果報告

基於 MPEG 標準之多媒體通訊與串流整合平台及其應用(總計畫)(III)

MPEG Integrated Multimedia Platform and Applications (III)

計畫編號：NSC 94-2219-E-009 -007

執行期限：94 年 8 月 1 日至 95 年 7 月 31 日

主持人：杭學鳴 國立交通大學電子工程學系教授

共同主持人：王聖智，李素瑛，蔡淳仁，蔣迪豪 國立交通大學

計畫參與人員：唐之璇，蔡家揚，施瑛姿，鍾國洋，朱育成，劉建志，鄭凱庭，陳威年，葉尚諭 國立交通大學學生

中文摘要

本群體計畫建構符合 MPEG 標準之網際網路即時(real-time)多媒體傳輸測試平台並在其上開發相關之技術與應用。此測試平台含媒體伺服器、使用端與模擬網路。本計畫以此平台為核心開發各項多媒體傳輸相關技術，如：可調層次式視頻編碼技術，串流多媒體伺服器與資料庫，智財管理與保護等等。本團隊將此測試平台提案到 MPEG 標準委員會，已被接受成為 MPEG-21 Part 12: Multimedia Test Bed for Resource Delivery 的技術報告類標準文件。

以多媒體傳輸測試平台為核心台，上述多媒體傳輸相關技術分別在五個子計畫中進行：(1) 可調層次式視訊編碼技術之前處理與後處理技術研究，(2) 視訊串流伺服器及視訊資料庫整合，(3) MPEG 多媒體傳輸機制及通訊協定在嵌入式行動平台上的分析設計，(4) 高等精細可調層次式視頻編碼技術之研究，以及(5) MPEG 智財管理與保護系統及視訊小波編碼器之設計與模擬。整體計畫時程擬為三年，本報告將摘要敘述第三年之各子計畫成果，並略敘多媒體傳輸測試平台的整合成果。

關鍵詞：MPEG-4、MPEG-7、MPEG-21、串流視訊、多媒體測試平台、精細可調層次式編碼

英文摘要

The goal of this project is to construct a real-time multimedia streaming test bed and develop related technologies and applications. This test bed can be used to verify multimedia coding algorithms and MPEG-21 Digital Item processing schemes. In this project, we provide such a test bed prototype system for the MPEG committee to experiment with various user scenarios. Because of the value of this test bed, it has been accepted as the Technical Report of MPEG-21 Part 12: Multimedia Test Bed Resource Delivery.

Based on this test bed, we develop various types of application technologies and systems, for example, fine granularity scalability codec, streaming video servers and database, intellectual property rights protection and management, and preprocessing and postprocessing algorithms for scalable video streaming. These technologies are studied in 5 sub-projects: (1) Preprocessing and Postprocessing Techniques of Scalable Video Streaming, (2) Video Streaming Server and Video Database Integration, (3) Design and Analysis of MPEG Multimedia Transport Mechanisms and Protocols for Embedded Mobile Platforms, (4) Advanced Fine Granularity Scalability, and (5) MPEG IPMP System and Interframe Wavelet Codec Design and Simulation. The entire duration of this project is 3 years. This report contains the research results of the third year. Also, we summarize the integration results of the multimedia streaming test bed.

Keywords: MPEG-4, MPEG-7, MPEG-21, Video Streaming, Multimedia Test Bed, Fine Granularity Scalable Coding

目錄 Table of Contents

1. 背景與目的	4
2. 可調層次式視訊編碼技術之前處理與後處理技術研究 -- 王聖智教授	5
3. 視訊串流伺服器及視訊資料庫整合 --李素瑛教授	8
4. MPEG 多媒體傳輸機制及通訊協定在嵌入式行動平台上的分析設計——蔡淳仁教授 10	
5. 高等精細可調層次式視頻編碼技術之研究 —— 蔣迪豪教授.....	12
6. MPEG 智財管理與保護系統及視訊小波編碼器之設計與模擬-- 杭學鳴教授	14
7. 整合成果:多媒體通訊與串流平台	17
8. 計畫整體成果與自評	21

1. 背景與目的

本整合型計畫希望達成下述共同目標：

1. 本計畫建構符合 MPEG 標準之網際網路即時(real-time)多媒體傳輸測試平台。此測試平台含使用端與模擬網路。在 MPEG-21 應用中，媒體伺服器與編解碼器均為使用端(之部分)。本計畫以此平台為核心開發各項多媒體傳輸相關技術，如：可調層次式視頻編碼技術，串流多媒體伺服器與資料庫，智財管理與保護等等。2002 年 12 月將此測試平台提案到 MPEG 標準委員會，成為 MPEG-21 Part 12: Multimedia Test Bed for Resource Delivery 的工作草案(Working Draft)。經過持續努力，這個工作草案於 2003 年 12 月晉升至大會草案(Committee Draft/Proposed Draft of Technical Report)。本案已正式成為 MPEG 技術報告類標準文件: ISO/IEC JTC 1/SC 29, *Information Technology – Multimedia Framework (MPEG-21) – Part 12: Test Bed for MPEG-21 Resource Delivery*, ISO/IEC TR 21000-12: 2005(E), 2005.
2. 本計畫在上述多媒體傳輸測試平台上整合各子計畫成果，建構應用實例展示，規劃中有即時串流視訊與多點視訊會議兩系統。利用本平台之即時網路特性，驗證所開發之技術在整合環境下之成效。並可依測試數據，改良演算法與系統設計，此功能是一般非即時模擬或單獨個別技術測試所無法達成的。
3. 由於此一多媒體之測試平台可望由 MPEG 標準委員會接納，成為標準參照軟體。因此可提供媒體製作者、媒體伺服器業者、通訊與多媒體服務系統業者、多媒體相關研究單位與學者等，一個實用的、可重複的、易嵌入的實驗與測試平台，作為網際網路應用技術最佳的研發工具。

隨著多媒體標準 MPEG 系列之制定，2.5G 及 3G 無線通訊標準之發表，與日新月異的數位電視，掌上型電腦與行動網路之廣受歡迎，更前瞻的多媒體應用日漸普及。多媒體技術的發達促使資訊的存取與資料的傳遞之相關應用廣泛地發展。多媒體應用在百家爭鳴情況下，卻造成彼此之間在資料傳遞與存取發生互動困難的問題。因此，MPEG-21 制定了多媒體通用存取的架構與法則，不同的消費者群(customers)可以透過談判來連接與互動。因此，在 MPEG-21 應用中基於可調層次式視頻編碼技術之隨選視訊與多點視訊會議之多媒體系統，特別受到產學官各界的青睞；因其能夠契合 2.5G 及 3G 視訊服務的需求與網路變異時之可調層次性。

可調層次式視頻編碼技術被 ISO/IEC 組織規劃為 MPEG-21 Part 13: Scalable Video Coding (SVC)。後來轉為 MPEG-4 part 10 (ITU H.264) Advanced Video Coding 的 extension。2004 年 3 月 MPEG 第 68 次會議上評估世界多家知名研究機構的提案，以角逐成為 SVC 視訊壓縮標準之起始點。在 14 個付費參與視覺品質評估的候選方案中，除了 Microsoft Research Asia 之外，交通大學是唯一提出兩案的單位。之後在 2005 年及 2006 年我們持續提案參加 MPEG SVC 的 Core Experiments。在未來幾年之內，本計畫所提的整合平台與基於這個整合平台的設計概念所衍生的媒體製作與視訊服務之相關產業、產品、研發等將成為工業界與學界的趨勢，其所研發原型系統也可帶動產品的開發並增加其產值。

此「基於 MPEG 標準之多媒體通訊與串流整合平台及其應用」整合計畫的分工情形大概如下：

- (一) 可調層次式視訊編碼技術之前處理與後處理技術研究 —— 王聖智教授
- (二) 視訊串流伺服器及視訊資料庫整合 —— 李素瑛教授
- (三) MPEG 多媒體傳輸機制及通訊協定在嵌入式行動平台上的分析設計 —— 蔡淳仁

教授

(四) 高等精細可調層次式視頻編碼技術之研究 —— 蔣迪豪教授

(五) MPEG 智財管理與保護系統及視訊小波編碼器之設計與模擬 —— 杭學鳴教授

2. 可調層次式視訊編碼技術之前處理與後處理技術研究 -- 王聖智教授

在這次的進度報告中，我們提出了兩項成果。一是接續去年的以影像內容為基礎的視訊對比加強技術，所發展出來的「以影像內容為基礎的視訊暗部加強技術」另一項是一種針對 H.264/AVC 視訊縮放的技術，我們稱之為「視訊空間解析度加強技術」。在「以影像內容為基礎的視訊暗部加強技術」方面，我們根據影像內容分析的結果，產生相對應的亮度調整機制。運用此項技術，我們可以用來改善視訊品質，並達到特定應用之目的。在「視訊空間解析度加強技術」方面，我們以 MAP 估測器為基礎，實作一視訊解析度改善的技術，另外，我們並針對現有方法做一些改進，使其在場景變換及快速運動時依然能有效的來加強視訊的解析度。

A. 視訊暗部加強技術

(1) 背景

現實世界中的光源強度分配的範圍是非常大的，但人類的視覺系統會對其做一定的調整以得到較清楚的視覺感知。但在數位影像中，由於位元數的限制，所以無法儲存完整的亮度變化。使用較少位數儲存影像資訊的結果會使得所儲存之數位影像，其亮度分佈可能會很集中，也就是整張影像會很黑或是很亮，因此喪失了影像的辨識性。因此我們的目的就是希望能解決上述的問題。在暗部加強的現有技術中，Retinex 是一普遍被運用的技術。雖然 Retinex 可提供一定的處理效果，但是其運算複雜度較高(log 運算)。因此也增加其硬體實現的困難度。

(2) 演算法說明

我們所提出的方法是接續去年的「以影像內容為基礎的視訊對比加強技術」所發展出來的。首先我們定義一 $n \times n$ 大小的視窗作為分析的基準。假設當視窗移動到影像中座標軸(x,y)時，我們會以函數 f_1, f_2 分析視窗內亮度值的特性。其中 f_1 主要是分析視窗中像素亮度值之平均趨勢，而 f_2 主要是分析視窗中像素亮度值的變異度資訊。我們一方面以視窗掃描影像產生 f_1, f_2 的資訊時，另一方面，我們會累計(f_1, f_2)的出現頻率。接著我們將從影像特徵對分布函數出發，推演出相對應的對比加強演算法。當我們考慮完所有特徵對的貢獻後，我們可以求出一最終的柱狀圖資訊 $M(g)$ ，並以此產生相對應的亮度轉換函數 $T_c(r)$ 。

接著，影像的暗部加強，可藉由壓縮影像中亮度對比較強的邊角資訊同時保持影像中亮度對比較弱的邊角資訊來達成。若換一個角度來思考這個問題，等同於影像的暗部加強可藉由加強影像中較弱的邊角資訊來達成。對我們的去年所發展的對比加強演算法架構而言，此概念相當容易被實現，我們只需要將去年研究成果中的柱狀圖的公式改為

下式即可：
$$M(g) = \int_{f_2=T1}^{f_2=T2} \int_{f_{1min}}^{f_{1max}} H(g; f_1, f_2) df_1 df_2$$

(3) 實驗結果

圖 1 為所提方法和 Retinex 的比較，我們可以清楚的觀察到，所提方法能提供和 Retinex 類似的處理效果，但是以運算複雜度的觀點而言，所提方法則是較 Retinex 簡單需多，也更易於硬體的實現。

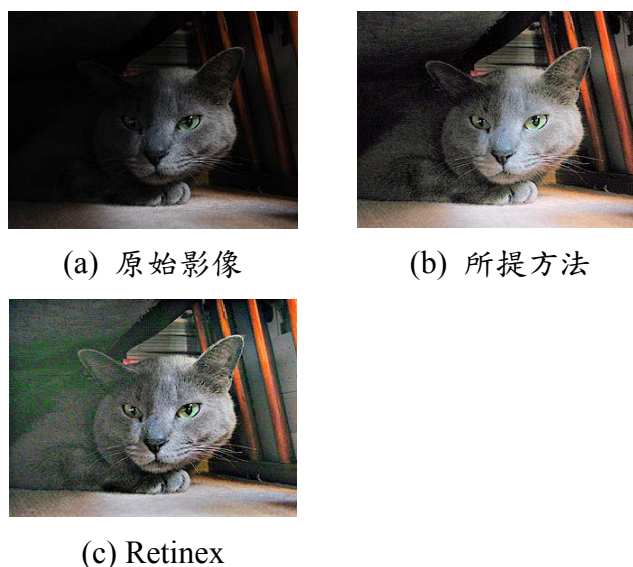


圖 1 暗部加強實例

(4) 結論

在本研究中，我們根據去年所提出的對比加強處理演算法為基礎，成功的發展出一套適用於影像暗部加強的演算法，且較 Retinex 而言，更易於硬體實現。另外，最後的實驗結果也證實了所提方法的強健性及實用性。

B. 視訊空間解析度加強技術

(1) 背景

畫面解析度改善(Resolution Enhancement)或是超解析度(Super-Resolution)的技巧，主要處理的問題是希望能夠藉由觀察低解析度(Low-Resolution)的影像或視訊去重建回高解析度的影像或視訊，可以利用圖 2 來做說明。超解析度的演算法利用低解析度影像間 sub-pixel 移動的關係去重建回高解析度的影像。而這些移動關係是由於視訊序列裡發生的運動所造成。由於有這些移動，所以才能在多張低解度影像中，找出對於重建高解析度影像有用的資訊。幾種常用來解決超解析度問題的方法為：(1) Projection onto Convex Set 方法；(2) Stochastic 方法；(3) Hybrid 方法。

(2) 架構

為了要進一步探討針對壓縮視訊畫面解析度改善的方法，首先要先把會使用到的系統模型建立起來。這個系統模型，將會把原始的高解析度影像跟解壓縮過後的低階析度影像的關係連結起來。我們的方法，是先建立視訊取像模型、視訊壓縮、以及雜訊模型。接著，我們假設高解析度影像的資訊可能會出現在多張低解析度的影像中，那麼每張低解析度的影像都能夠提供關於高解析度影像的額外資訊。因此，藉由觀察多張壓縮過的低解析度影像就可以重建回高解析度影像。



(a) 157th frame; 原始超解析度方法； (b) 157th frame; 改善過的超解析度方法

圖 2 原始超解析度方法跟改善過的超解析度方法的模擬比較：Foreman 視訊序列

因為 MAP 的方法在 model 雜訊和先前知識上很有彈性及韌性，所以我們以此為基礎，推導 MAP estimator。針對超解析度的 spatial-domain MAP estimator 的推導，我們必須定義 Fidelity constraint 與 Prior model，接著推導 MAP estimator 的 cost function，最後採用 steepest descent 方法去找到超解析度問題的最佳解。然而在整個流程中，motion estimation 是整個方法理最關鍵的一個步驟，所以如果有不正確的 motion estimation 出現，就會造成 warp-back 過程發生錯誤。為了處理這樣的問題，我們分別針對瑕疵減少與場景轉換與快速運動偵測兩個議題提出解決方法。

(3) 實驗結果

在實驗中，我們將畫面的解析度由 QCIF 放大為 CIF。我們使用四張 QCIF 的低解析度影像去重建單張 CIF 的高解析度影像。在這個情況下，我們必須要作三次的 motion estimation。然後我們便會得到三張 warped-back images 及三張的 warped-back error images。由圖 2，我們可以看到視覺品質獲得改善。而我們的方法跟 bilinear 的比較可以在圖 3 中看到。可以看出我們方法重建回的高解析度影像，比 bilinear 方法的結果來的少瑕疵且在邊緣的部分比較銳利。



(a) 5th frame; bilinear 方法； (b) 5th frame; 改善過的超解析度方法

圖 3 Bilinear 方法跟改善過的超解析度方法的模擬比較：Mobile 視訊序列

(4) 結論

在這次的進度報告中，我們以 MAP estimator 實現了針對 H.264/AVC 壓縮視訊的畫面解析度改善方法。針對重建結果中產生的瑕疵部分，也提出兩種改善的方式去壓抑瑕疵。第一種，是一個 spatial domain 的中位數濾波器，用來壓抑 salt-and-pepper 型的雜訊。第二種，則是 pixel-wise temporal median filter 用來移除重建過程中的例外者。針對 SC/FM

的問題，我們也提出了可能的改善方式去解決這樣的問題。跟原始的超解析度方法比較，我們提出的方法可以克服 SC/FM 的問題，並壓抑大部分的瑕疵。

3. 視訊串流伺服器及視訊資料庫整合 --李素瑛教授

本計畫主要目的在於研究與實踐視訊資料庫以及視訊伺服器。在視訊資料庫方面，除了視訊資料內容之分析與高階與語意特徵擷取之外，並結合音訊資料之特徵加上影片內容字幕之偵測及擷取，以內容分析之結果為依據的影片調整方法，讓資源能夠有效地利用，並提高影片的視覺品質。並且利用分析所得之語意資訊加以解譯及推測，以偵測影片資料內容之事件，藉此達成高階特徵之擷取並以此為基礎產生既簡短又具語意意義之視訊資料摘要提供視訊資料預覽之用，除此之外在產生影片內容摘要之過程中，同時產生符合 MPEG-7 之低階至高階特徵之描述（descriptor）以及描述結構（description scheme），以利資料庫索引之建立並藉此建立互動式影片瀏覽系統。

(1) 緣由與目的

在視訊伺服器方面，我們希望設計支援單點傳送與多點傳送的視訊串流伺服器。在單點傳送伺服器設計方面，主要有兩項任務，一項是傳輸速率控制(rate control)，另一項則是錯誤控制(error control)。在多點傳送視訊串流伺服器方面，由於多點傳送有效率地使用頻寬，因此我們的重心在於錯誤控制。在多點傳送的錯誤控制中，當封包遺失發生時，有兩個主要的問題需要解決，一個是區域性地錯誤恢復(Local Error Recovery)，一個是 NACK 爆炸(NACK Implosion)的議題，前者是希望能盡量縮短錯誤恢復所需要的時間，後者則避免區域性網路壅塞的問題。

最後，我們分別實作出支援單點傳送與多點傳送的視訊串流伺服器以及視訊資料庫，最後再將它們整合以達到快速地從資料庫中找到最佳的位元配置與影片調整方法，來做視訊串流的調整。從實驗結果可以看出所提出的方法相當實用並可達到較好的品質。

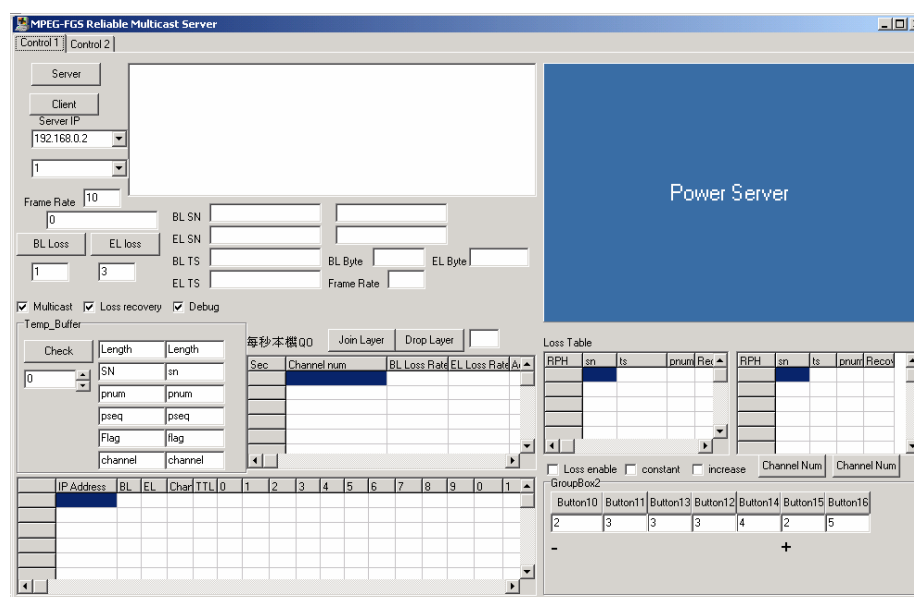


圖 4 視訊串流伺服器介面

(2) 系統設計與實驗結果

◆ 單點傳送與多點傳送視訊串流伺服器之整合

由於單點傳送視訊串流伺服器與多點傳送視訊串流伺服器屬於不同的傳輸模式，因此在實際運作時，其負責的模組也不相同，在單點傳送視訊串流伺服器中包含傳輸速率控制模組、傳輸錯誤更正模組以及單點傳送網路模組；在多點傳送視訊串流伺服器中包含傳輸頻道分割模組、應用層協定模組以及多點傳送網路模組。另外，由於視訊串流伺服器傳輸模式的不同，使得接收端不管在接收資料緩衝區的設計以及錯誤恢復的機制皆不盡相同，所以要將這兩種視訊串流伺服器整合起來時，必須先判斷是否是單點傳送要求再進入單點或多點傳送模組，方可確保不論在單點傳送或是多點傳送的運作下，皆可正確無誤地接收資料並且執行錯誤恢復的能力。整合之視訊串流伺服器如圖 4 所示。

◆ 視訊影片資料庫

隨著無線網路的發展與可攜式隨身裝置能力的提升，在這樣的環境中傳遞影片的應用越來越廣泛。在有限的資源和與生俱來的限制下，如何適當地調整影片長度以及品質成為在無線多媒體應用相關領域中的一個最重要且富挑戰性的議題。

由於不同種類之視訊資料則有不同之特性，如運動類影片與新聞類及一般電影類影片其特性則相當不同，視訊片段之分類及其影片長度及品質調整之設定則有不同之標準，在本研究中，我們提出了一個以內容為依據的影片調整方法，讓資源能夠有效地利用，並提高影片的視覺品質。藉由分析影片中物件的亮度、位置、移動向量與能量等屬性，配合客戶端裝置能力與關聯性統計模型取得場景中較吸引注意的部分如圖 5 所示。利用區域加權速度失真模型、位元配置與影片調整方法，動態地從物件、畫面與 GOP 三個層面來做視訊串流的調整，如圖 6 所示。

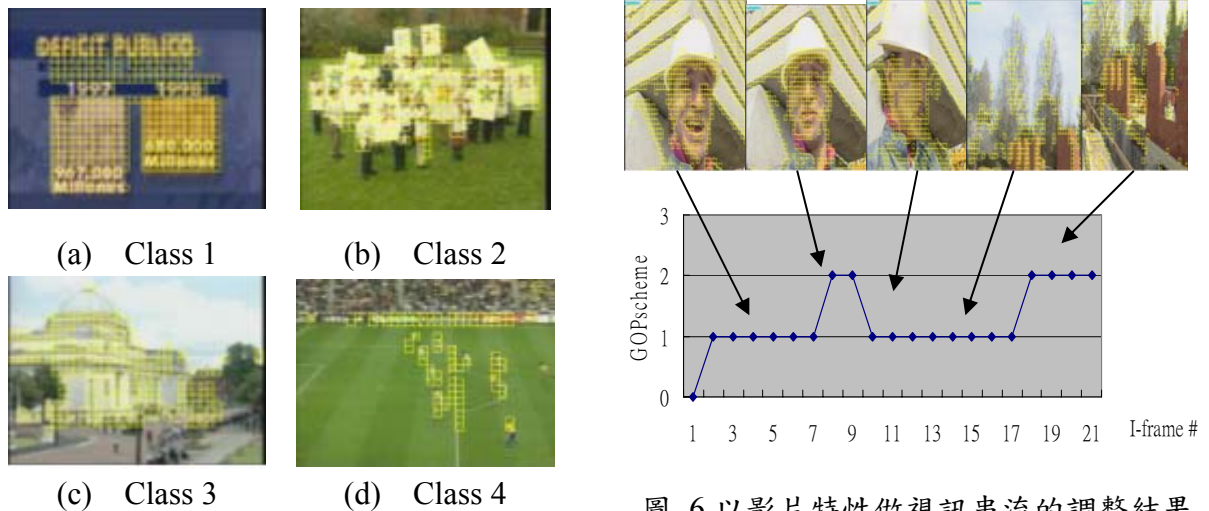


圖 6 以影片特性做視訊串流的調整結果

圖 5 不同影片類型中，感興趣區域的實驗結果

另外，我們利用物件軌跡作為基礎，再加上 domain knowledge 作為推測之依據，並利用研發之事件推測演算法判斷視訊片段是否重要，是否隱含高階語意，之後即可將重要之片段擷取並合併成為更具意義之視訊摘要，相信這將是在結構化視訊計算範疇當中相當重要之進程。我們亦結合音訊內容以增進視訊摘要之正確性。另外 MPEG-7 亦定義

了 Summary DS，我們也將產生符合其架構及格式之 descriptors 及 description schemes 來建立索引並藉此建立互動式影片瀏覽系統(如圖 7 所示)，使得使用者做視訊資料擷取時能快速有效地比對出正確之資料。

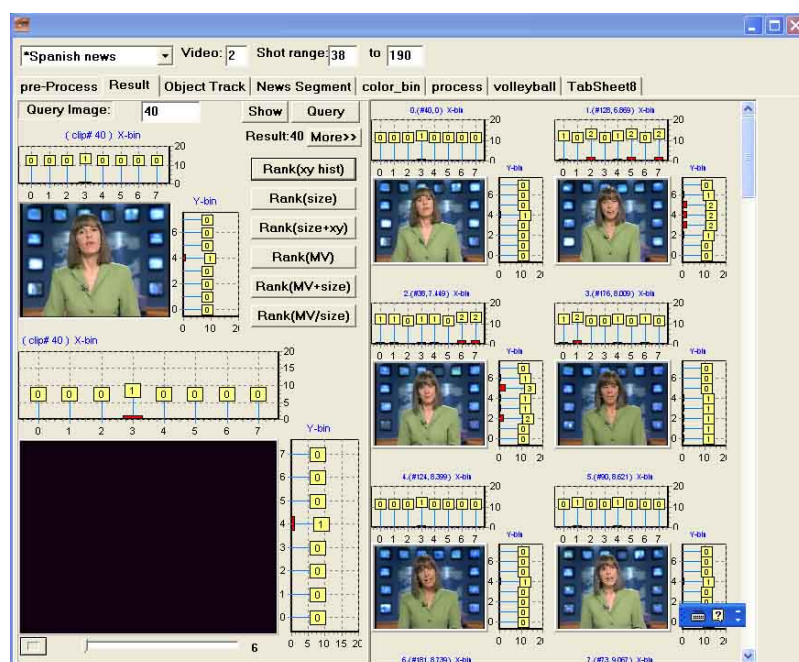


圖 7 互動式影片瀏覽系統

4. MPEG 多媒體傳輸機制及通訊協定在嵌入式行動平台上的分析設計——蔡淳仁教授

在三年的整合計畫中，本子計畫除了配合總計畫團隊，協助 MPEG 標準 ISO/IEC TR 21000-12: Test Bed for MPEG-21 Resource Delivery 的制定之外，在過去三年共發展了以下系統。在第一年的時候，除了開發 Test Bed，主要是研究現有的流量控制和容錯機制的設計。第二年則改進了現有的碼率失真最佳化 (rate-distortion optimized) 可調式串流傳輸機制並整合進計畫團隊開發的 MPEG Test Bed 標準測試環境中。另外，在第二年的計畫中，也針對未來多媒體嵌入式系統的相當有潛力的多次內容調適(multiple adaptations)的應用，設計出以 wavelet 為基礎之 scalable video codec 的流量控制機制。最後，在第三年的研究中，我們更進一步提出了更好的碼率失真最佳化 (rate-distortion optimized) 可調式串流傳輸機制。在此系統中，我們會根據封包內容的重要度進行不同的 Reed-Solomon 容錯保護。在研究成果的產出方面，根據第二年和第三年的研究，我們分別寫了兩篇期刊論文，目前正在審核中。

(1) 緣由與目的

過去的串流傳輸系統的設計重點是在於同一時間內要能服務最多的客戶端為主，而且多半假設所有連到伺服器的終端機都是透過類似的網路以及具有相似的計算能力。本計畫的主要目的在於配合 MPEG-21 的理念，設計出一個具有實用性系統，可以橫跨不同的網路架構和在不同的客戶端設備上 (PC、手機、PDA 等等) 提供高品質多媒體傳輸播放服務。因此，這個傳輸系統必須能動態地根據不同的應用環境調整資料包裝和傳

輸的方式。簡言之，一個數位多媒體傳輸系統的架構必需包含流量控制和容錯機制。另外，依據客戶端的能力來調整媒體資料流品質的能力也是十分重要的。

為了在頻寬和封包掉失率隨時在改變的環境下給使用者最好的串流傳輸品質，許多人嘗試對傳輸的資料進行碼率失真最佳化(Rate-Distortion)分析，然後根據分析結果來傳輸視訊串流。過去所提出的碼率失真最佳化的網路傳輸的流量控制機制裏，最大的問題在於封包漏失所造成的失真(distortion)沒有辦法量化。許多現有系統的做法在封包漏失所造成的失真的估測都十分不切實際（或完全忽略了）。我們提出了一個較實際的方法，可以合理的把封包失真量化。

要達到碼率失真最佳化的網路串流傳輸，除了要根據網路頻寬和封包漏失率來估測出最好的串流資料量，還必須能在算出的串流資料量限制下，從 scalable video 的位元流中，抽出最好的子位元流。所以我們針對這方面的需求，設計出了一個快速收斂的小波轉換視訊壓縮法(wavelet video coding)的流量控制機制，另外，更進一步設計了可以有效進行多次碼率失真最佳化的子位元流抽取(multiple adaptation)的機制。

另外，多媒體串流資料傳輸之前，一般的系統會進行 Forward Error Correction (FEC) 或 Automatic Retransmission reQuest (ARQ) 的保護。然而，在目前公開發表的系統中，最多只是針對漸層式編碼可調式編碼的不同傳輸層做不同程度的 FEC 保護。而我們則改進了這項缺點。在本計畫第三年的成果中，我們提出了一套機制，在碼率失真最佳化的原則下，能較精確地對視訊內容依其重要性進行不同程度的 Reed-Solomon 編碼保護，並配合 data interleaving 的封包包裝法來強化資料對封包掉失的容錯度。

(2) 結果與討論

前面大致提到了本計畫在過去三年除了參加 MPEG 標準制訂之外的三項主要研究成果，首先是一個較實用的碼率失真最佳化串流傳輸系統、其次是一個較適合嵌入式系統使用的低複雜度可多次內容調適的流量控制機制、最後則是一個根據 IP 封包網路串流傳輸特性所設計的自動隨內容重要性調整的 (content-adaptive) FEC 保護機制。

本節首先描述本計畫在碼率失真最佳化串流傳輸系統方面的創新設計。目前在這方面較知名的是由 P. A. Chou 等人所發展的系統。這個系統有兩大缺點。首先是他們使用封包掉失率來估測碼率失真最佳化分析中的失真。這是很不實際的做法。其次，該系統為了避開傳統 ARQ 對封包回傳時間很敏感的問題，因此採用封包預先重覆傳送的方法來降低失真，這也是很沒有效率的容錯保護機制。因此這套方法目前發表的成果以理論分析為主，在實作上有很多細節並沒有提出解決方案，而且在頻寬變化大的網路環境下，串流傳輸最難達到的平滑播放要求也沒有考量。

在本計畫的碼率失真最佳化串流傳輸系統中，我們把封包漏失所造成的失真，轉化為不同程度的 FEC 保護所造成的失真。舉例而言， 10^{-3} 的封包漏失率造成的失真，就相當於 10^{-3} 的 FEC 的 error protection 導致 data rate 降低所造成的失真。整個系統可以分成兩大部份：媒體封包相依性控制：媒體封包相依控制和碼率失真最佳化傳輸控制。在提出的系統中，碼率失真最佳化傳輸模組會根據資料內容的移動量資訊，來調整流量控制時的最佳化決定。簡單地說，一般的碼率失真最佳化控制架構必需在資料單元群組之間利用碼率及失真的 Lagrangian cost function 來算出最小值的解來有效率的分配時間和頻寬的網路資源。而本計畫提出的系統則會同時利用 FEC 的編碼率和每一段視訊資料所含的移動量大小來校正傳統的 cost function 以求得更佳的效果。

在可調式位元串流傳輸中，影像資料可以分成好幾次傳送，每次的傳送都可以幫助

解碼端得到更接近於原影像資料的重建訊號，因此可調式位視訊編碼法必須能支援多樣化的調適運作(adaptation operations)以針對不同應用產生有效可解碼的位元流。前面所提到的系統設計，主要是以 MPEG-4 FGS 的編碼法（從 2005 年起，FGS 已經不屬於 MPEG-4 標準的一部份）為可調式編碼的核心。但在第一年的研究過程中，我們發現 FGS 的諸多限制，因此在第二年開始的研究中，就開始轉向採用小波轉換視訊壓縮法。

小波轉換視訊壓縮法的流量控制機制，一般系統(e.g. JPEG 2000 及 3-D ESCOT) 是利用在編碼階段所產生的碼率失真資料表，配合二分搜尋法，來決定在某個頻寬條件下最佳的視訊位元流子集合。在本計畫中，我們設計了一個比碼率失真表更有效的雙參數碼率失真模型來進行最佳的位元流切割點的快速搜尋。也因為我們所提出的視訊視訊資料模型比較精簡也更有效率，我們可以把它隨著抽取出的位元流一起傳輸到接收端以進行多次的碼率失真最佳化子位元流切割。

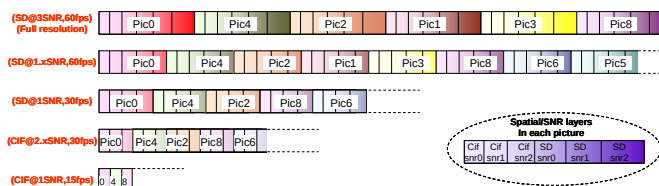
在第三年的計畫中，我們開發了一個能自動隨視訊內容重要性和即時封包掉失率來調整 FEC 保護強度的機制。這部份的設計主要是針對 IP 網路封包串流傳輸的應用而開發的。目前所有的 IP 網路的串流傳輸系統都會由 layer-2 以下的通訊協定處理位元錯誤，而在 layer-3 以上往往只能看到封包掉失的錯誤。現有系統多半以 ARQ 封包重傳的方法或是用 Reed-Solomon 編碼配合資料交錯安排 (data interleaving) 來達到修復掉失的封包的功能。前者往往不適用封包回傳時間 (round-trip time) 較長的網路，而後者則是很難達到碼率失真最佳化的原則。在本計畫提出的方法，則是利用我們在第二年時所發展的雙參數碼率失真模型，並根據小波轉換的視覺效應，定出一個能夠隨視訊內容和即時網路封包掉失率來調整 Reed-Solomon 保護強度的 FEC 封包編碼保護機制。

5. 高等精細可調層次式視頻編碼技術之研究 —— 蔣迪豪教授

為了支援接收端依據不同的能力解碼所接收的資料，ISO/IEC MPEG 和 ITU-T 兩大標準委員會共同組織一個名為 Joint Video Team (JVT) 的團隊共同發展可調視訊編碼技術 (SVC)，以期達成單一位元流能夠支援空間、時間、品質 (SNR) 三種不同解析度的目標，並且滿足低複雜度與低延遲的限制。可調視訊編碼技術是基於 H.264/AVC 的一種延伸：它能產生與 H.264/AVC 相容的基底層還有一個完全可調的加強層，即加強層可根據不同的情況被截取在任何一個位元以達到不同的空間、時間、品質解析度。

(1) 緣由與目的

MPEG-4 標準委員會制訂了細緻可調視訊編碼以進行視訊串流及廣播。細緻可調視訊編碼的加強層使用了幀內預測以及位元層編碼，使其位元流能被截斷於任意位置並提供了細緻的影像畫質調節。因為缺乏幀間預測，MPEG-4 細緻可調視訊編碼之加強層有良好的容錯能力，卻有較差的編碼效率。本計畫提出了新的技術以加強幀間預測及編碼效率，同時仍保持良好的錯誤容忍度。這些技術也能夠使用於正制定中的 H.264/AVC 可調視訊編碼標準，並且顯著的改善了壓縮效果。



(a) SVC bitstream extraction

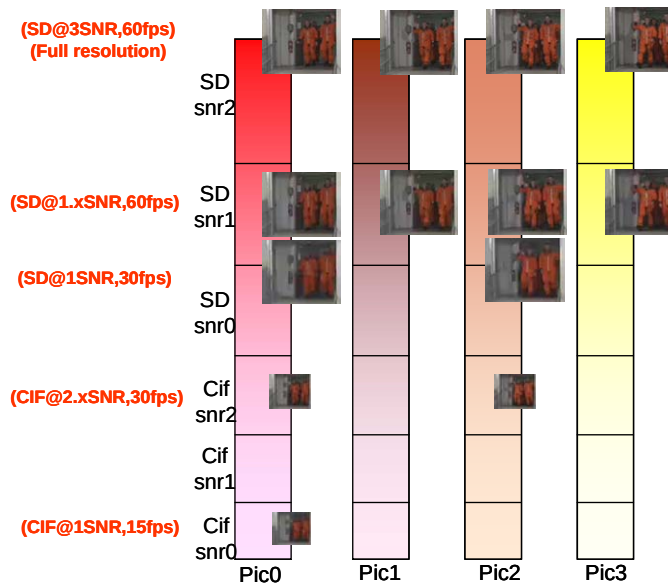
圖 8 列舉出可調視訊編碼的應用：假設系統包含了三個擁有不同連接速度的設備(伺服器、路由器、無線存取點)。任一時間，多個客戶端的使用者可以連上網路，可調視訊編碼位元串有兩個不同的空間解析度(SD, 704x576 和 CIF, 352x288)、三個不同的時間解析度(60, 30, 15 fps)，針對每一個空間解析度又有三個不同的品質解析度。圖 8(a)顯示每一個連結的位元串架構，此位元串包括多個畫面而每一個畫面包含不同的空間跟品質解析度。最後，整個可調視訊編碼所能提供的解析度金字塔如圖 8(b)所示。

(2) 方法與實驗

圖 9 顯示一個可調視訊編碼器的主要架構：這個編碼器有三個空間層(或是品質層)。每一層均被獨立的壓縮器所壓縮(如圖二的框線所示)。至於時間解析度的壓縮方法則有兩種方式可以採用：(a)運動補償時間濾波器，(b)階層雙向預測架構。

(3) 結果與討論

如圖 10(a)所示，當有一半的畫面被跳過的時候，H.264/AVC 比 SVC 差了 1.5 到 2.0 dB 左右。當更多的畫面被跳過的時候，H.264/AVC 的效果會更糟。所以證明當網路的頻寬不穩定的時候，SVC 擁有比 H.264/AVC 更好的適應能力。在圖 10(b)中，在最起始的 QCIF 層，SVC 有跟 H.264/AVC 幾乎一致的效果。在 CIF 層，SVC 有 0.5 dB 的品質下降。在 4CIF 層，SVC 則有將近 1.0 dB 的品質下降。所以，SVC 可以在可允許的品質下降範圍內，達到空間解析度的可調性。圖 10(c)顯示 SVC 在時間解析度的效能，雖然 SVC 比 H.264/AVC 有 1.8 dB 的 PSNR 下降，但是這樣的品質下降情況可以經由較負責精細調整來改善。最後，當三種解析度可調同時被支援的時候，圖 10(d)顯示其效能評比，一般而言，SVC 相較於 H.264/AVC 有 0.5 到 1.2 dB 的 PSNR 損失。



(b) SVC decoded video

圖 8 An example of SVC

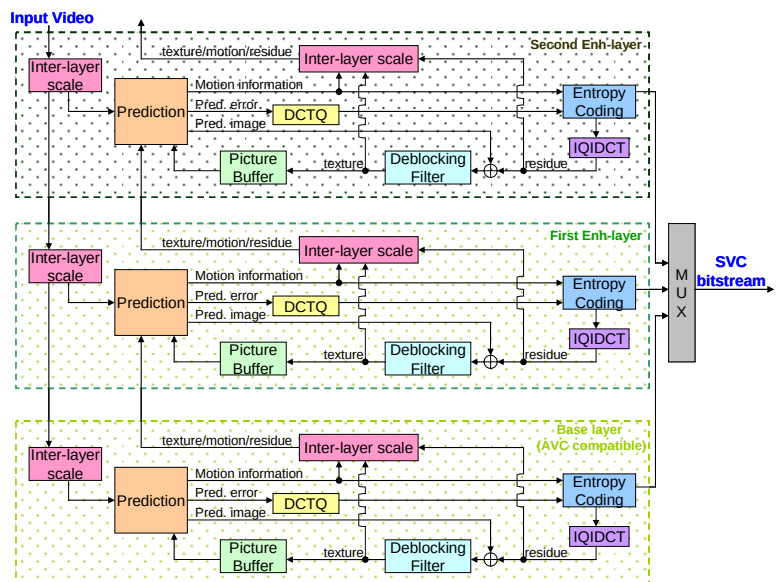
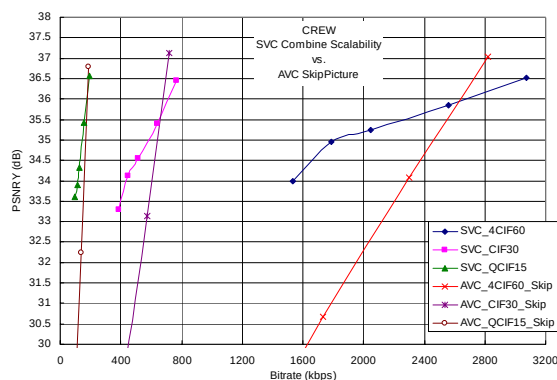
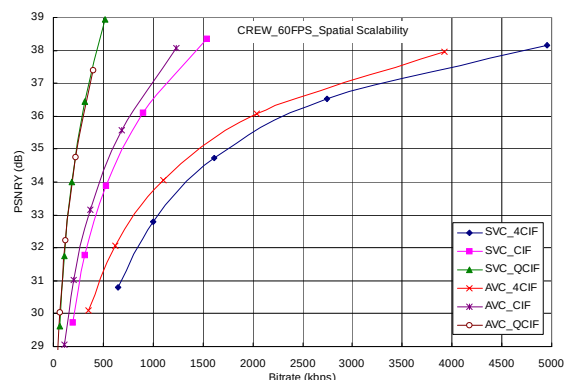


圖 9 SVC encoder structure with three spatial/SNR layers

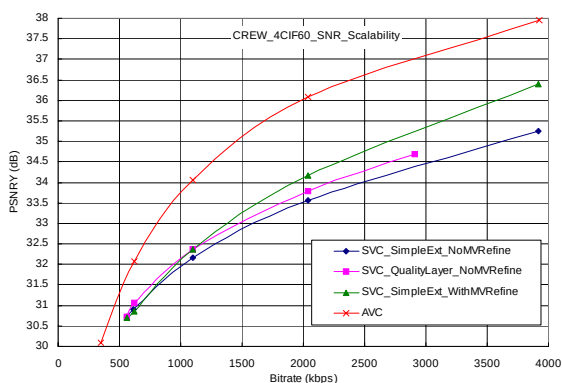
總結，作為 H.264/AVC 的延伸，SVC 提供了一個 H.264/AVC 相容的基底層跟一個完全可調的加強層來支援空間、時間、SNR 可調性。針對空間可調性，金字塔的架構被用來改善兩層之間的預測。針對時間可調性，階層雙向架構被用來改善壓縮效率。針對 SNR 可調性，強韌細緻可調視訊編碼可以顯著改善 MPEG-4 細緻可調視訊編碼的壓縮效果，並仍保持良好的細緻性及容錯能力。實驗結果正如預期，SVC 犧牲了些許的壓縮效率，但是卻能提供三種不同解析度的可調性。



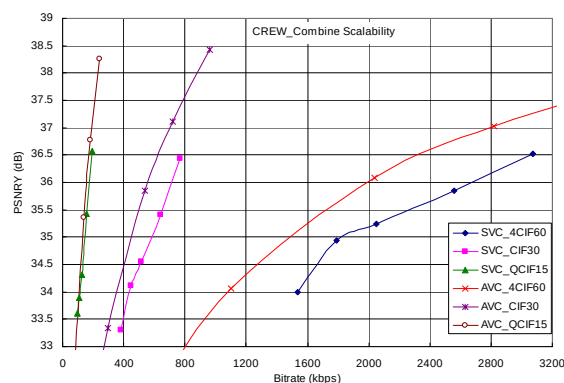
(a) SVC with combined scalability vs. AVC with picture skipping



(b) SVC with spatial scalability only



(c) SVC with SNR scalability only



(d) SVC with combined scalability

圖 10 Performance comparison between H.264/AVC and SVC

6. MPEG 智財管理與保護系統及視訊小波編碼器之設計與模擬-- 杭學鳴教授

在網際網路快速成長之下，導致大量的數位資訊在網際網路與無線通道上傳輸。因此，也衍生出著作權保護的問題。如何去保護交易進行時的資料交換、如何使得交易後商品得到適當保護、如何保障付費人享有自己的權益，其概念在 MPEG 中稱為智慧財產管理與保護 (Intellectual Property Management and Protection, IPMP)。如何整合各個不同的系統，進而使消費者能輕易的使用 IPMP，則為目前亟需解決的重點，此亦為本計畫的第一個研究課題。此外，隨者網際網路以及廣播事業的發展與普及，多媒體的應用的標準就日趨重要。因為應用的變化幅度大，要求的規格也有許多變化，因此，新訂定的標準也就沒有特別為某種應用做設計，而反而以可調性 (scalability) 為主。傳統上可調性以 wavelet 相關研究為主，但 wavelet 在視訊方面有其先天限制，因此產生其他概念的

可調分析方法，此為本計畫的第二個研究課題。

A. MPEG-4 IPMP 與 MPEG-21 REL

(1) 緣由與目的

常見的多媒體保護機制大多使用各種的加密演算法對內容進行保護。但是對於日趨複雜的使用環境，被動的內容加密機制已經不敷所需，必須考慮一個進階的機制來描述特定使用者的權限，因此我們透過權利描述語言(Rights Expression Language, REL)來滿足此需求。

(2) 系統設計

目前有許多權利描述語言，其中由 MPEG-21 標準所訂定的權利描述語言不但能描述使用者的權限，還能在 MPEG-21 標準所訂定的平台上流通。本研究使用 MPEG-21 IPMP、MPEG-21 REL 和 MPEG-21 Test Bed with MPEG-4 IPMPX 之觀念與工具來創建我們的系統。MPEG-21 IPMP 定義高層次的 Digital Item 保護機制。MPEG-21 REL 能夠描述廣泛的使用權利，亦提供清晰的權利認證與控管機制。然而 MPEG-21 IPMP 與 REL 僅提供概念上的定義，因此我們選用 MPEG-21 Test Bed 作為實現的平台。藉由修改 MPEG-4 IPMPX 定義的介面，我們在 Test Bed 上加入 IPMPX 功能，並據以實現數個 IPMP 與 REL 工具(圖 11)。

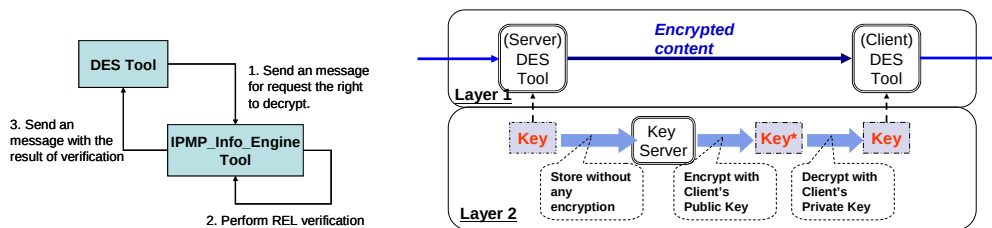


圖 11 串流媒體保護與相關 IPMP Tool

(3) 結果與討論

我們利用三個範例展現此 DRM 系統。範例(一)展示在串流應用的環境下使用即時的權利認證和內容保護的能力。範例(二)表現如何設計和使用 license 讓使用者預視影像短片。範例(三)則展現一個比較複雜的 Super Distribution 範例，利用我們設計的 REL license 提供線上與離線的驗證(圖 12)。

B. 輪廓轉換

(1) 緣由與目的

自然界的影像在經過空間轉換到頻域(frequency domain)之後，大部分的能量都集中在低頻上，搭配有效率的熵編碼(entropy coding)後，可以提升壓縮效率。近年來由於可調整性(scalability)的需求越來越熱門，因此小波轉換(wavelet transform)的研究也越來越熱門。但是小波轉換對二維訊號的表示法卻不是很理想，因為它忽略了二維方向的連續性，故後來有人提出了方向性濾波器頻帶(directional filter bank)的轉換(圖 13)，如曲線轉換(curvelet transform)和輪廓轉換(contourlet transform)。

(2) 方法與實驗

本研究主要基於 Minh N. Do 的輪廓轉換，觀察各次頻帶(subband)的輸出結果。影像經過兩次拉式金字塔的拆解後，把最高頻的部分作 8 個方向的 2 維方向性濾波頻帶拆解，中間的頻率部分則是作 4 個方向的頻帶拆解，最低頻的部分不作方向性的拆解。

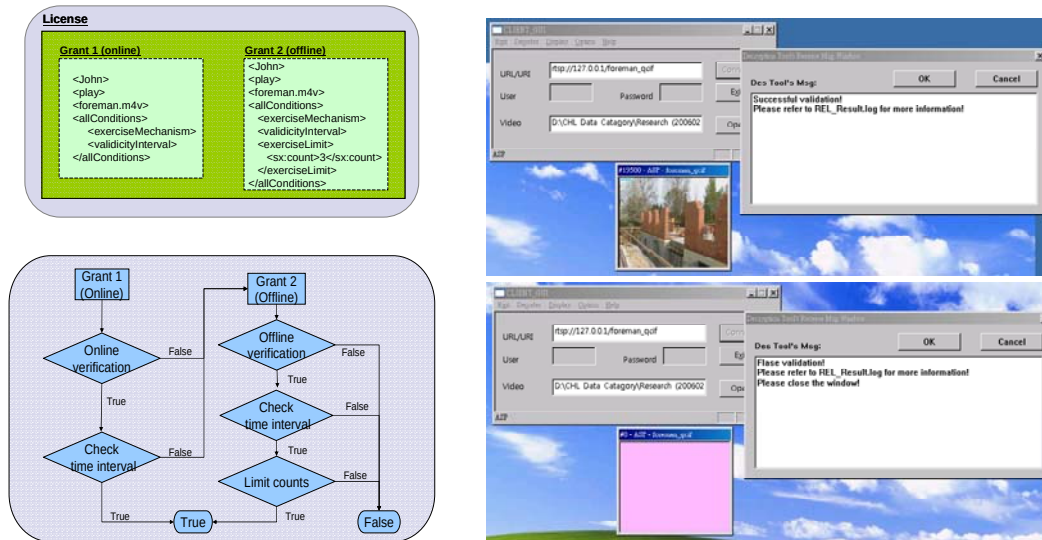


圖 12 Super Distribution 範例程式之驗證流程與執行畫面

(3) 結果與討論

觀察實驗結果可以發現，只要影像的邊緣(edge)和某個頻帶一致性越大，在那個方向上就會有越強的訊號出現(圖 14)。

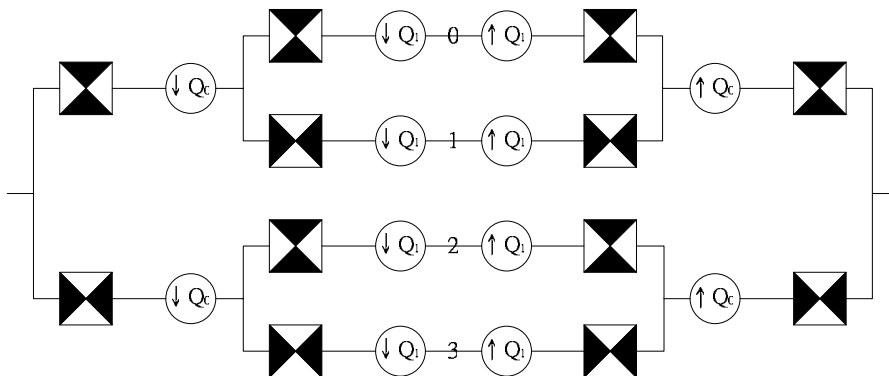


圖 13 4 個方向的 2 維方向性濾波頻帶

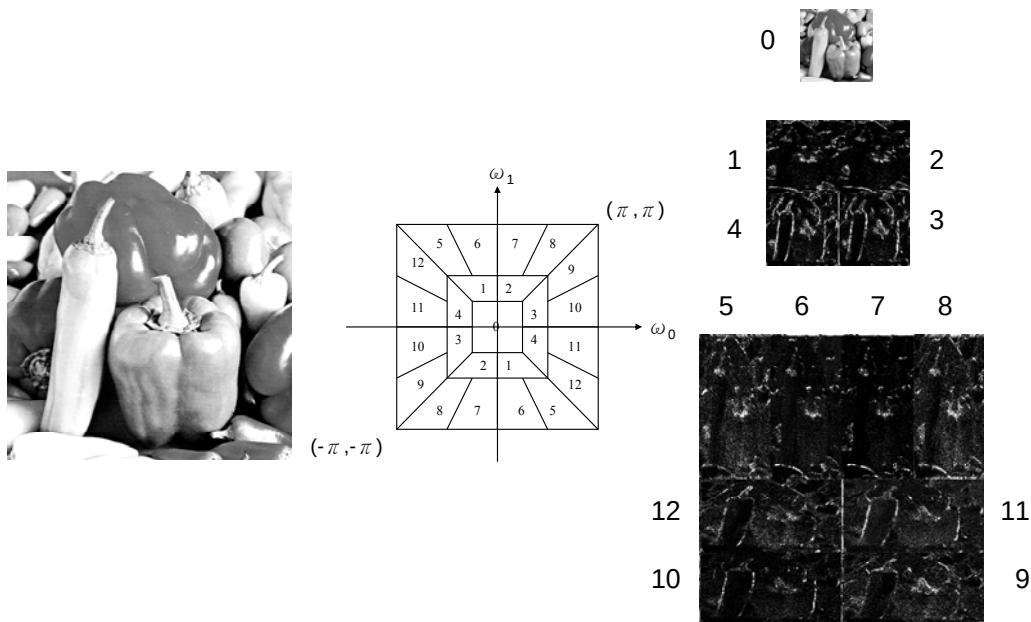


圖 14 經過拉式金字塔和 2 維方向性濾波頻帶後拆解的圖片

7. 整合成果:多媒體通訊與串流平台

本計畫之多媒體通訊與串流整合平台為 MPEG-21 標準 Part 12 (以下簡稱 Test Bed)。其主要目的是將多種先進的多媒體技術實作並整合於此架構中，例如可調層次式視頻編碼技術，串流多媒體伺服器與資料庫，智慧財產管理與保護等等。平台的發展歷程自 2002 年 12 月開始，先將此測試平台提案到 MPEG 標準委員會，成為 MPEG-21 Part 12: Multimedia Test Bed for Resource Delivery 的工作草案(Working Draft)。經過持續努力，這個工作草案於 2003 年 12 月晉升至大會草案(Committee Draft/Proposed Draft of Technical Report)。目前本案已正式成為 MPEG 技術報告類標準文件: ISO/IEC JTC 1/SC 29, Information Technology– Multimedia Framework (MPEG-21) – Part 12。其後，本研究群在此定案的 Test Bed 參考軟體上繼續進行研發，例如針對當時 MPEG-21 尚不明確的智慧財產權保護機制，持續追蹤標準的進展，進而加強 Test Bed 原有功能，結合另一參考軟體，目前已可展現部份 MPEG-21 Rights Expression Language (REL) 功能，並從中窺見未來多媒體內容保護技術的輪廓。

以下，將大略描述本計畫 Test Bed 的設計與實作。整個 Test Bed 的執行環境可區分為三大部份，如圖 15 所示。Server 端負責多媒體串流傳輸，將 Client 端要求的內容自資料庫中取出，加以適當切割後傳至網路上；Client 端接收到串流資訊後加以組合、解碼、以及播放；為了在區域網路中模擬可能發生的傳輸錯誤，Test Bed 包含了一個 Network Emulator，用以模擬諸如 Packet Loss、Packet Delay、Bandwidth 變化等等傳輸狀況。此種執行環境雖然較為複雜，卻提供相當大的彈性，在研究時可以單獨使用任何一個部份進行除錯或測試。

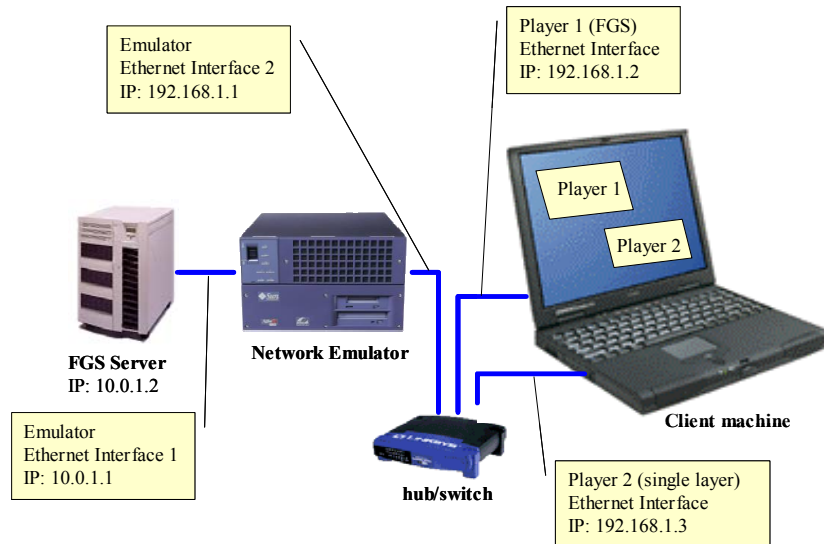


圖 15 MPEG-21 Test Bed 執行環境

圖 16 所示為 Test Bed 的架構設計。以下依序說明架構中的重要元件：

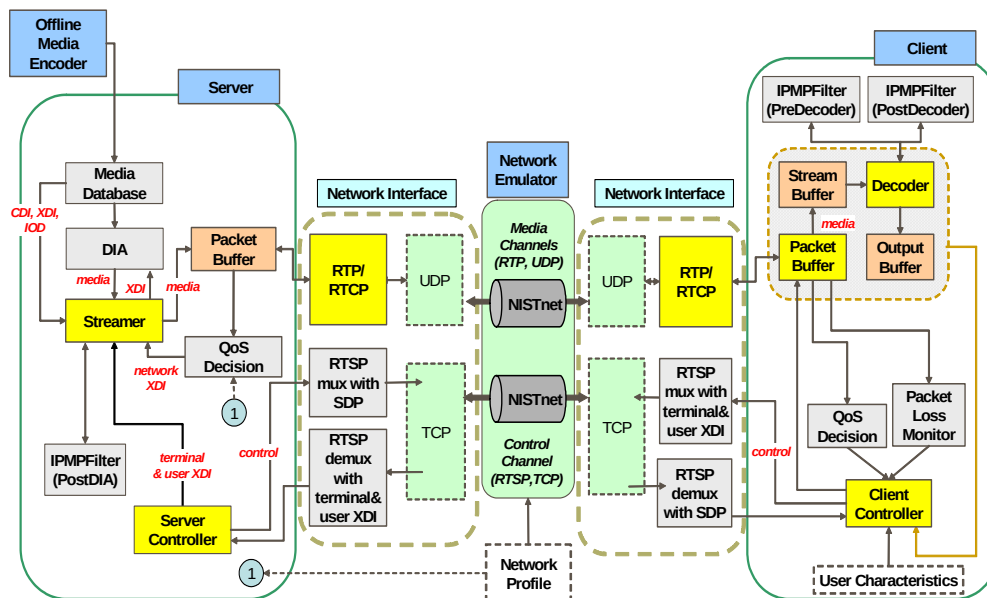


圖 16 MPEG-21 Test Bed 架構圖

● Server 端

- Media Database: 此介面用以存取壓縮好的影音資料。在 Test Bed 中，所有的多媒體資訊，皆為 Off-line 壓縮後，放置於多媒體資料庫結構之下，到執行時期再透過此介面讀取所需的資料片段。
- Digital Item Adaptation (DIA): 這是 MPEG-21 中相當重要的一個概念。同樣的數位內容，因為網路傳輸條件或接收端的能力的限制，必須視情況加以調整，使接收者能得到在此限制下最好的品質。Test Bed 中的 DIA 元件必須與 Codec

搭配，在 session 建立時即可根據多媒體類型決定 DIA 元件，接著在依據不同的條件 (Test Bed 中目前實作 network profile) 調整輸出串流的資訊量。

- Streamer: 主要功能為切割多媒體 bitstream，使其符合網路傳輸，處理後的資料形成 Packet。此元件也會根據輸入的 QoS 資訊決定該如何向 DIA 元件取得資料。
- Packet Buffer: 為一 ring buffer，用以暫存尚未送出的 Packet。
- IPMPFilter (PostDIAFilter): 在 Test Bed 的設計中，Server 端只有一個 IPMP Control Point 可供處理資料，其位置在 DIA 與 Streamer 之間。其意義為處理即將送出的 bitstream，例如對串流加密等等。
- Server Controller: 這是 Server 端最複雜的元件，主要功能是控制 Server 端其他元件的啟動、運作、關閉，而 session 的建立與結束也是由此元件負責。
- QoS Decision: 收集不同的傳輸限制條件，並將控制訊號送至相關的元件，以便控制傳輸品質。目前的條件來源為 Network Profile 與 Packet Buffer 的狀態。

● Client 端

- Packet Buffer: 與 Server 端的 Packet Buffer 為相同的 ring buffer 結構，但則是作為已接收尚未處理的 Packet 暫存區。不同的 bitstream 會對應一個專屬的 Packet Buffer。
- Stream Buffer: 簡單的說，Stream Buffer 是將 Packet 組合成 bitstream 的暫存區。但因為 Client 端可能碰到 Packet Loss 的情形，所以 Stream Buffer 在組合 bitstream 時仍以 ring buffer 的管理方式，且依據時間順序排列，未收到的時序則以 place holder 暫時取代。一個 Stream Buffer 對應一個 Packet Buffer，以方便處理 packet re-transmission。
- Decoder: 將組合後的 bitstream 解壓縮為可播放的格式。如果要對抗傳輸錯誤，error concealment 功能可在此元件內實作。
- Output Buffer: 解壓縮後的資料置放於此暫存區中，供播放元件讀取。
- IPMPFilter (PreDecoderFilter): 在 Client 端，IPMP Control Point 有兩個位置。這是第一個 Control Point，位於 Stream Buffer 與 Decoder 之間，目的是處理接收到的 bitstream。其功能對應於 Server 端的 PostDIAFilter。因此，如果 PostDIAFilter 中放入加密 Tool，則此處放入的就是解密 Tool。
- IPMPFilter (PostDecoderFilter): 在 Client 端，IPMP Control Point 有兩個位置。這是第二個 Control Point，位於 Decoder 與 Output Buffer 之間，目的是處理解 decode 後的資料，例如加入 digital watermark，防止隨意散佈數位內容。
- Client Controller: 這是 Client 端相當複雜的元件，負責管理 Client 端其他元件的運作。當 session 建立或結束時，Client Controller 與 Server Controller 互相通訊並交換相關資訊。當傳輸有錯誤時，Client Controller 也負責發出重傳命令，要求 Server 端重送特定的 Packet。
- QoS Decision: Client 端的 QoS 設計目的是根據網路狀況分析，再將調整參數送至 Client Controller 進行處理。不過目前因為 Network Profile 為已知，所以此部份的實作為無動作。

- Packet Loss Monitor: 負責監控 Packet Loss 狀態，如果還在時間容許範圍內，則透過 Client Controller 發出 re-transmission 的要求。
- Network Emulator
 - NISTnet: 由 NIST 開發的 Linux kernel module 以及相關的 user mode utilities，用來在兩張網路卡之間模擬各種網路傳輸狀況，可設定多種參數，例如 packet loss rate 與 transmission delay 等等。
 - Network Profile: 為一 Test Bed 定義的文字檔案，其格式可描述某一時間區間之 packet loss rate、delay、bandwidth 等等。

利用上述的各項元件，整個 Test Bed 的運作流程簡述如下：Client 確定欲播放的媒體 URL 後，Client Controller 與 Server Controller 透過 RTSP 建立 session；Server 端確認媒體存在後，根據要求將相關的元件設定好，並開始把媒體的設定資訊透過 Server Controller 送至 Client 端；Client 根據接到的設定資訊將各元件啟動，並傳送播放指令到 Server；Server 從 Media Database 中取出 bitstream，根據 QoS 資訊進行 DIA，切割 bitstream 成適合傳輸的大小，並轉換為 Packet；Packet 經過 RTP channel 傳送到 Client 端；Client 將收到的 Packet 組合成 bitstream，送入 Decoder 解碼，解碼結果放入 Output Buffer 供 player 播放。在運作過程中，Client 端可根據 Packet Loss 情況，透過 Client Controller 發出 re-transmission 需求，而 Server 端則據此透過 RTP 重送 Packet。如果有 IPMP 的需求，則必須在 session 建立時，在 Server 端啟動必須的 Tool，並透過 RTSP 將 IPMP 相關資訊傳至 Client 端，待 Client 將相關的 Tool 啟動並連接至 IPMPFilter 後，才能發出播放命令。

本計畫完成的 MPEG-21 Test Bed 標準 reference software 功能如下：

- 支援的媒體格式：MPEG-4 ASP (Advanced Simple Profile)、FGS (Fine Granularity Scalability)、BSAC (Bit Sliced Arithmetic Coding)
- 支援的 IPMP Tool：DES Tool (Data Encryption Standard)
- 可展示的多媒體傳輸：
 - 單一 Video (ASP or FGS)、單一 Audio (BSAC)、影音 (ASP+BSAC or FGS+BSAC)
 - 在無 IPMP 狀況下，可搭配各式 Network Profile 測試
 - 可使用 GUI 或 Command-line 版本的 Client 端。GUI 版本直接播放媒體於視窗中；Command-line 版本則將 Output Buffer 的資料依序寫入指定的檔案中。

本計畫於 MPEG-21 Test Bed 標準制定程序定案後，仍持續注意 MPEG-21 標準的進展，並適時修正或增加功能。例如：

- 修正 Initial Object Descriptor (IOD) 處理程序 buffer 過小的問題。
- 結合 MPEG-21 Rights Expression Language (REL) 的 ContentGuard reference software，實作 IPMP REL Tool，為 Test Bed 加上 REL 處理功能。
- 目前可展現的 IPMP 功能為：
 - 以 DES 加密傳送的 bitstream。
 - 每隔一段時間改變 DES key 以提高安全度。
 - 修改 Client 端使其直接讀取 local file 播放。

- 利用 REL 達到 preview 類型的應用。此種應用允許一小段時間內無限制播放 bitstream，當時間超過時，則必須有 license 才能取得播放權限。
- 利用 REL 達到類似 OMA Super Distribution 類型的應用。在此種應用環境中，加密的 bitstream 可以任意散佈，播放端必須擁有相對應的 license 才能取得播放權限。

8. 計畫整體成果與自評

本計畫旨在建構符合 MPEG 標準之網際網路即時(real-time)多媒體傳輸測試平台，並以此平台為核心開發各項多媒體傳輸相關技術，如：可調層次式視頻編碼技術，串流多媒體伺服器與資料庫，智財管理與保護等。這個年度中，本校團隊的三位老師杭學鳴、蔡淳仁、蕭旭峯及博士後研究員彭文孝，在國科會計畫補助旅費情況下參與 MPEG 標準會議。過去三年中，提出 70 件 MPEG 貢獻文件(Contributions)。將上述技術提案至 MPEG 標準組織，有助我國技術之進入國際舞台。參與 MPEG 國際會議之具體成果簡述如下。

蔡淳仁教授擔任 MPEG-21 Part-12 Multimedia Test Bed for Resource Delivery 之共同編輯者(editor)。交大 MPEG-21 團隊之成員人數約在 10 人左右。該團隊的目標是建構並改進 MPEG-21 Part-12 Multimedia Test Bed。這個團隊合作成果已經在 2004 年被 MPEG 標準委員會接受為技術報告類標準文件: ISO/IEC JTC 1/SC 29, *Information Technology – Multimedia Framework (MPEG-21) – Part 12: Test Bed for MPEG-21 Resource Delivery*, ISO/IEC TR 21000-12: 2005(E), 2005.

蔣迪豪教授擔任 MPEG-4 14496-7 Optimized Reference Software 之共同編輯者(editor)。負責編撰的最佳化 MPEG-4 14496-2 之 Simple Profile Video Codec 亦已完成，成為正式標準之修正案。蔣迪豪教授亦擔任 MPEG-4 Part-2 Video Subgroup 之 Co-Chair，參與 MPEG-4 Part-2 標準書之修訂。並與 MPEG-4 Part-9 之專家合作以完成 Software-Hardware Codesign 的平台。

2004 年 3 月 MPEG 標準委員會舉行 scalable video coding Call-for-Proposal 競賽。交通大學團隊提出兩項提案，除了 Microsoft Research Asia 之外，交通大學是唯一提出兩案的單位。我方案品質評估為中上，在世界強隊競爭下，結果尚佳。2004 年 7 月 MPEG 的第 69 次會議，交通大學團隊提出一項改善 Interframe Wavelet 提案，Response to CE1b in SVC-- SB-Reach Method for Entropy Coding。2005 年 4 月 MPEG 的第 72 次會議，交通大學提出另一項改善 Part 10 Scalable Video Coding (SVC)提案，Adding Selective Enhancement Functionality to Scalable Video Coding，可加強 SVC 的 Region-of-Interests (RoI) 能力，依此成立一項 Core Experiment。

此外，本計畫有多項學術與應用技術方面之創新：在視訊串流伺服器、可調層次式視訊編碼技術、多媒體傳輸網路機制、多點視訊會議系統、及視訊分割合成與編解碼等等技術與實作方面的新發現。發表國內外期刊與學術研討會論文多篇，詳見各子計畫報告。

更直接並且對工商業更有價值的貢獻將是人才訓練。同學們在學校階段已熟悉較前瞻的世界標準，畢業後進入產業，直接有助於產業界開發新產品，提昇我國工業技術能力。

綜合評估：本計畫產出相當多具有學術與應用價值的成果，並積極參與國際 MPEG 標準會議，將我人研發成果推廣到國際舞台。此外亦達到人才培育之效，整體成效良好。

2003.7 – 2006.7 三年中本研究群發表之 MPEG 標準文件

1. Y.-S. Tung, C.-N. Wang, T. Chiang, and H. Sun “ISO/IEC JTC1/SC29/WG11 M13393: AHG on Maintenance of MPEG-4 Visual related Documents, Reference Software and Conformance”, July 2006 (77th, Klagenfurt, Austria)
2. Y.-S. Tung, C.-N. Wang, T. Chiang, and H. Sun “ISO/IEC JTC1/SC29/WG11 M13032: AHG on Maintenance of MPEG-4 Visual related Documents, Reference Software and Conformance”, April 2006 (76th, Montreux, Switzerland)
3. Y.-S. Tung, C.-N. Wang, T. Chiang, and H. Sun “ISO/IEC JTC1/SC29/WG11 M12724: AHG on Maintenance of MPEG-4 Visual related Documents, Reference Software and Conformance”, January 2006 (75th, Bangkok, Thailand)
4. Y.-S. Tung, C.-N. Wang, T. Chiang, and H. Sun “ISO/IEC JTC1/SC29/WG11 M12428: AHG on Maintenance of MPEG-4 Visual related Documents, Reference Software and Conformance”, October 2005 (74th, Nice, France)
5. F.-C. Chang, C.-P. Ho, C.-Y. Tsai, C.-H. Li, J.-D. Cheng, W.-C. Chang, Y.-T. Shih, C.-L. Lin, C.-H. Lu, C.-C. Cheng, C.-Y. Liu, J.-C. Ma, K.-C. Lee, and C.-J. Tsai, “ISO/IEC JTC1/SC29/WG11 M12373: Update to the FGS-Based Multimedia Resource Delivery Test Bed Software”, July 2005 (73rd, Poznan, Poland)
6. Y.-S. Tung, C.-N. Wang, T. Chiang, and H. Sun “ISO/IEC JTC1/SC29/WG11 M12116: AHG on Maintenance of MPEG-4 Visual related Documents, Reference Software and Conformance”, July 2005 (73rd, Poznan, Poland)
7. E. S. Jang, K. Asai, and C.-J. Tsai, “ISO/IEC JTC1/SC29/WG11 N7329: Study of Video Coding Tool Repository v5.0”, July 2005 (73rd, Poznan, Poland)
8. C.-J. Tsai, “ISO/IEC JTC1/SC29/WG11 M12074: Suggestions on the direction of VCTR”, April 2005 (72nd, Busan, Korea)
9. T. Chiang, Y.-S. Tung, C.-N. Wang, “ISO/IEC JTC1/SC29/WG11 M11785: AHG on Maintenance of MPEG-4 Visual related Documents, Reference Software and Conformance”, April 2005 (72nd, Busan, Korea)
10. W.-H. Peng, Tihao Chiang and H.-M. Hang, “ISO/IEC JTC1/SC29/WG11 M11914: Adding Selective Enhancement Functionality to Scalable Video Coding,” April 2005 (72nd, Busan, Korea)
11. T. Chiang, Y.-S. Tung, C.-N. Wang, “ISO/IEC JTC1/SC29/WG11 M11475: AHG on Maintenance of MPEG-4 Visual related Documents, Reference Software and Conformance”, Jan 2005 (71th, Hong Kong, China)
12. H.-M. Hang, “ISO/IEC JTC1/SC29/WG11 M11482: AHG Report on Further Exploration in Wavelet Video Coding,” January 2005 (71st, Hong Kong, China)
13. Y.-S. Wang, C.-N. Wang, T. Chiang and J.-R. Ohm, “ISO/IEC JTC1/SC29/WG11 N6710, MPEG-4 Visual: List of Problems Reported, v.14”, Oct 2004 (70th, Palma de Mallorca, Spain)
14. T. Chiang, Y.-S. Tung, C.-N. Wang, “ISO/IEC JTC1/SC29/WG11 M11156: AHG on Maintenance of MPEG-4 Visual related Documents, Reference Software and Conformance”, Oct 2004 (70th, Palma de Mallorca, Spain)
15. Y.-S. Tung, C.-N. Wang, T. Chiang, J.-R. Ohm, “ISO/IEC JTC1/SC29/WG11 M11327: MPEG-4 Visual: List of Problems Reported, v.13.1”, Oct 2004 (70th, Palma de Mallorca, Spain)
16. C.J. Tsai, “ISO/IEC JTC1/SC29/WG11 N6629, DoC on ISO/IEC 21000-12/PDTR Test Bed for MPEG-21 Resource Delivery,” July 2004 (69th, Redmond, Washington, USA)
17. Y.-S. Wang, C.-N. Wang, T. Chiang and J.-R. Ohm, “ISO/IEC JTC1/SC29/WG11 N6514, MPEG-4 Visual: List of Problems Reported, v.13” July 2004 (69th, Redmond, Washington, USA)

18. C.J. Tsai, "ISO/IEC JTC1/SC29/WG11 M11142: Universal Video Codec Architecture Suggestions", July 2004 (69th, Redmond, Washington, USA)
19. C.-N. Wang, C.-H. Li, C.-W. Fan, Y.-T. Shih, J.-P. Ho, C.-L. Lin, F.-C. Chang, G.-C. Li, C.-N. Chiu, C.-Y. Tsai, H.-C. Chuang, J.-H. Chen, J.-C. Ma, Chi-Yu Liu, C.-C. Chen, C.-J. Tsai, Tihao Chiang, S.-Y. Lee, and H.-M. Hang, "ISO/IEC JTC1/SC29/WG11 M11117: Scalable Multimedia Streaming Test Bed for Media Coding and Testing in Streaming Environments," July 2004 (69th, Redmond, Washington, USA)
20. H.-K. Hsu, C.-Y. Tsai, H. -C. Huang, H.-M. Hang and T. Chiang, "ISO/IEC JTC1/SC29/WG11 M10934, Response to CE1b in SVC-- SB-Reach Method for Entropy Coding," July 2004 (69th, Redmond, Washington, USA)
21. M. Van Der Schaar, A. Becker, J. Ridge, C.-J. Tsai, "ISO/IEC JTC1/SC29/WG11 M10758: AHG on Core Experiments in Scalable Video Coding," July 2004 (69th, Redmond, Washington, USA)
22. T. Chiang, Y.-S. Tung and C.-N. Wang "ISO/IEC JTC1/SC29/WG11 M10755, AHG on Maintenance of MPEG-4 Visual related Documents, Reference Software and Conformance", July 2004 (69th, Redmond, Washington, USA)
23. C.-J. Tsai and C.-N. Wang, "ISO/IEC JTC1/SC29/WG11 N6471, Study of ISO/IEC 21000-12/PDTR Test Bed for MPEG-21 Resource Delivery," March 2004 (68th, Munich, Germany)
24. Y.-S. Wang, C.-N. Wang, T. Chiang and J. R. Ohm, "ISO/IEC JTC1/SC29/WG11 N6363, MPEG-4 Visual: List of Problems Reported, v.12," March 2004 (68th, Munich, Germany)
25. H.-C. Huang, W.-H. Peng, Y.-C. Lin, C.-N. Wang and T. Chiang, "ISO/IEC JTC1/SC29/WG11 M10724, Update of the Response to Cfp on Scalable Video Coding Technology: Proposal S07 -- A Robust Scalable Video Coding Technique," March 2004 (68th, Munich, Germany)
26. C.-Y. Tsai, H.-K. Hsu, H.-M. Hang, and T. Chiang, "ISO/IEC JTC1/SC29/WG11 M10569/S08, Response to Cfp on Scalable Video Coding Technology: Proposal S08 -- A Scalable Video Coding Scheme Based on Interframe Wavelet Technique," March 2004 (68th, Munich, Germany)
27. H.-C. Huang, W.-H. Peng, Y.-C. Lin, C.-N. Wang, T. Chiang and H.-M. Hang, "ISO/IEC JTC1/SC29/WG11 M10569/S07, Response to Cfp on Scalable Video Coding Technology: Proposal S07 -- A Robust Scalable Video Coding Technique," March 2004 (68th, Munich, Germany)
28. C.-N. Wang, T. Chiang, and H. Sun, "ISO/IEC JTC1/SC29/WG11 M10723, Text of DTR 14496-7 Fifth Edition", March 2004 (68th, Munich, Germany)
29. Y.-S. Tung, C.-N. Wang, and T. Chiang, " ISO/IEC JTC1/SC29/WG11 M10517, AHG on Maintenance of MPEG-4 Visual: List of Problems Reported, v11.1," March 2004 (68th Munich, Germany)
30. C.J. Tsai, M. van der Schaar, Y.-K. Lim, and Doug Young Suh, "ISO/IEC JTC1/SC29/WG11 M10486, AHG on MPEG Multimedia Test Bed," March 2004 (68th, Munich, Germany)
31. M. van der Schaar, A. Becker, J. Ridge, C.-J. Tsai, and Vittorio Baroncini, " ISO/IEC JTC1/SC29/WG11 M10480, AHG on Scalable Video Coding," March 2004 (68th, Munich, Germany)
32. T. Chiang, Y.-S. Tung and C.-N. Wang, " ISO/IEC JTC1/SC29/WG11 M10476, AHG on Maintenance of MPEG-4 Visual related Documents, Reference Software and Conformance," March 2004 (68th, Munich, Germany)
33. C.-N. Wang, C.-Y. Tsai, H.-C. Chuang, J.-H. Chen, J.-C. Ma, C.-N. Chiu, C.-Y. Liu, C.-W. Fan, F.-C. Chang, C.-J. Tsai, T. Chiang, S.-Y. Lee, and H.-M. Hang, "ISO/IEC JTC1/SC29/WG11 M10298: Scalable Multimedia Streaming Test Bed for Media Coding and Testing in Streaming Environments," Dec. 2003 (67th, Kona, Hawaii, USA)

34. J.-R. Ohm, Y.-S. Tung, T. Chiang, H. Thoma, C.J. Tsai, "ISO/IEC JTC1/SC29/WG11 N6183: Study Text of ISO/IEC 14496-2:200X/DCOR1," Dec. 2003 (67th, Kona, Hawaii, USA)
35. C.J. Tsai, C.-N. Wang, "ISO/IEC JTC1/SC29/WG11 N6275: Requirements for ISO/IEC TR21000-12 Multimedia Test Bed for Resource Delivery, v. 4," Dec. 2003 (67th, Kona, Hawaii, USA)
36. C.J. Tsai, M. van der Schaar, Y.-K. Lim, "ISO/IEC JTC1/SC29/WG11 N6255: Text of ISO/IEC 21000-12/PDTR Test Bed for MPEG-21 Resource Delivery, " Dec. 2003 (67th, Kona, Hawaii, USA)
37. T. Chiang, Y.-S. Tung, C.-N. Wang, "ISO/IEC JTC1/SC29/WG11 M10215: AHG on Maintenance of MPEG-4 Visual Draft Corrigendum, Reference Software and Conformance," Dec. 2003 (67th, Kona, Hawaii, USA)
38. Y.-S. Tung, C.-N. Wang, T. Chiang, J.-R. Ohm, "ISO/IEC JTC1/SC29/WG11 M10296: MPEG-4 Visual: Updated List of Problems Reported, v.10.1," Dec. 2003 (67th, Kona, Hawaii, USA)
39. C.J. Tsai, C.-N. Wang, "ISO/IEC JTC1/SC29/WG11 M10297: Proposed Draft Requirements for ISO/IEC TR21000-12 Multimedia Test Bed for Resource Delivery, v. 4," Dec. 2003 (67th, Kona, Hawaii, USA)
40. C.J. Tsai, M. van der Schaar, Y.-K. Lim, "ISO/IEC JTC1/SC29/WG11 M10299: Working Draft 3.0 of ISO/IEC TR21000-12 Multimedia Test Bed for Resource Delivery," Dec. 2003 (67th, Kona, Hawaii, USA)
41. M. van der Schaar, A. Becker, J. Ridge, and C.J. Tsai, "ISO/IEC JTC1/SC29/WG11 M10219: AHG on Scalable Video Coding," Dec. 2003 (67th, Kona, Hawaii, USA)
42. C.J. Tsai, M. van der Schaar, Y.-K. Lim, and D. Y. Suh, "ISO/IEC JTC1/SC29/WG11 N6259: AHG on MPEG Multimedia Test Bed," Dec. 2003 (67th, Kona, Hawaii, USA)
43. C.J. Tsai, M. van der Schaar, Y.-K. Lim, and D. Y. Suh "ISO/IEC JTC1/SC29/WG11 M10422: AHG on MPEG Multimedia Test Bed," Dec. 2003 (67th, Kona, Hawaii, USA)
44. D. Y. Suh, C.J. Tsai, A. Cotarmanach, and K. Kim, "ISO/IEC JTC1/SC29/WG11 M10354: MOEG and Internet (1) : the avt Working Group," Dec. 2003 (67th, Kona, Hawaii, USA)
45. T. Rusert and C.J. Tsai, "ISO/IEC JTC1/SC29/WG11 M10402: Generation of Original Test Sequences and AVC Anchors for Call for Proposals on Scalable Video Coding Technology," Dec. 2003 (67th, Kona, Hawaii, USA)
46. Y.-S. Tung, C.-N. Wang, T. Chiang, J.-R. Ohm, "ISO/IEC JTC1/SC29/WG11 N6185: MPEG-4 Visual: List of Problems Reported version 11.0," Dec. 2003 (67th, Kona, Hawaii, USA)
47. J.-R. Ohm, Y.-S. Tung, T. Chiang, H. Thoma, C.J. Tsai, "ISO/IEC JTC1/SC29/WG11 N5951: Text of ISO/IEC 14496-2:200X/DCOR1", Oct. 2003 (66th, Brisbane, Australia)
48. Y.-S. Tung, C.-N. Wang, T. Chiang, J.-R. Ohm, "ISO/IEC JTC1/SC29/WG11 N5952: MPEG-4 Visual: List of Problems Reported, v.10," Oct. 2003 (66th, Brisbane, Australia)
49. Y.-S. Tung, C.-N. Wang, T. Chiang, J.-R. Ohm, "ISO/IEC JTC1/SC29/WG11 M10173: MPEG-4 Visual: Updated List of Problems Reported, v.9.1," Oct. 2003 (66th, Brisbane, Australia)
50. J.-R. Ohm, T. Chang, and C.-N. Wang, "ISO/IEC JTC1/SC29/WG11 N5954: Proposed Text of ISO/IEC 14496-7/Amd.1:200X/DCOR1," Oct. 2003 (66th, Brisbane, Australia)
51. C.J. Tsai and C.-N. Wang, "ISO/IEC JTC1/SC29/WG11 N6048: Draft Requirements for ISO/IEC TR21000-12 Multimedia Test Bed for Resource Delivery, v. 3", Oct. 2003 (66th, Brisbane, Australia)
52. T. Chiang, Y.-S. Tung, C.-N. Wang, "ISO/IEC JTC1/SC29/WG11 M9951: AHG on Editorial Convergence and Maintenance of MPEG-4 Reference Software," Oct. 2003 (66th, Brisbane, Australia)
53. C.J. Tsai and C.-N. Wang, "ISO/IEC JTC1/SC29/WG11 M10159: Draft Requirements for

- ISO/IEC TR21000-12 Multimedia Test Bed for Resource Delivery, v. 3”, Oct. 2003 (66th, Brisbane, Australia)
54. C.-Y. Tsai, H.-C. Chuang, J.-H. Chen, J.-C. Ma, C.-Y. Liu, C.-W. Fan, F.-C. Chang, C.-N. Wang, C.J. Tsai, T. Chiang, S.-Y. Lee, and H.-M. Hang, “ISO/IEC JTC1/SC29/WG11 M10160: Scalable Multimedia Streaming Test Bed for Media Coding and Testing in Streaming Environments”, Oct. 2003 (66th, Brisbane, Australia)
 55. C.J. Tsai, M. van der Schaar, “ISO/IEC JTC1/SC29/WG11 M9955: AHG on Scalable Video Coding,” Oct. 2003 (66th, Brisbane, Australia)
 56. C.J. Tsai, M. van der Schaar, Y.-K. Lim, “ISO/IEC JTC1/SC29/WG11 N6072: AHG on MPEG Multimedia Test Bed,” October, 2003 (66th, Brisbane, Australia)
 57. C.J. Tsai, M. van der Schaar, Y.-K. Lim, “ISO/IEC JTC1/SC29/WG11 M9972: AHG on MPEG Multimedia Test Bed,” Oct. 2003 (66th, Brisbane, Australia)
 58. D. Y. Suh, C.J. Tsai, K. Kim (ETRI), A. Perkis (NTNU, Norway), A. Cotarmanach (FT, France), “ISO/IEC JTC1/SC29/WG11 M10047: Comprehensive interoperability between MPEG and Internet,” Oct. 2003 (66th, Brisbane, Australia)
 59. C.J. Tsai, M. van der Schaar, Y.-K. Lim, “ISO/IEC JTC1/SC29/WG11 M10163: Working Draft 3.0 of ISO/IEC TR21000-12 Multimedia Test Bed for Resource Delivery,” Oct. 2003 (66th, Brisbane, Australia)
 60. Y.-S. Tung, C.-N. Wang, T. Chiang, J.-R. Ohm, “ISO/IEC JTC1/SC29/WG11 N5691: MPEG-4 Visual: List of Problems Reported, v.9,” July 2003 (65th, Trondheim, Norway)
 61. T. Chiang, Y.-S. Tung, C.-N. Wang, “ISO/IEC JTC1/SC29/WG11 M9632: AHG on Editorial Convergence and Maintenance of MPEG-4 Reference Software,” July 2003 (65th, Trondheim)
 62. C.J. Tsai, M. van der Schaar, “ISO/IEC JTC1/SC29/WG11 M9636: AHG on Scalable Video Coding,” July 2003 (65th, Trondheim)
 63. C.J. Tsai, M. van der Schaar, Y.-K. Lim, “ISO/IEC JTC1/SC29/WG11 M9662: AHG on MPEG Multimedia Test Bed,” July 2003 (65th, Trondheim)
 64. Sam S Tsai, H.-M. Hang, T. Chiang, “ISO/IEC JTC1/SC29/WG11 M9756: Motion Information Scalability for MC-EZBC: Response to Call for Evidence on Scalable Video Coding,” July 2003 (65th, Trondheim)
 65. Y.-S. Tung, C.-N. Wang, T. Chiang, J.-R. Ohm, “ISO/IEC JTC1/SC29/WG11 M9763: MPEG-4 Visual: Updated List of Problems Reported, v.8,” July 2003 (65th, Trondheim, Norway)
 66. C.-N. Wang, T. Chiang, “ISO/IEC JTC1/SC29/WG11 M9764: Improvement of Optimized MPEG-4 14496-2 Simple Codec with EPFL SIT Analyzer,” July 2003 (65th, Trondheim, Norway)
 67. C.J. Tsai, C.-N. Wang, “ISO/IEC JTC1/SC29/WG11 M9765: Proposed Draft Requirements for ISO/IEC TR21000-12 Multimedia Test Bed for Resource Delivery, v. 2,” July 2003 (65th, Trondheim, Norway)
 68. C.-N. Wang, C.-Y. Tsai, Y.-C. Lin, H.-C. Chuang, K.-L. Tong, J.-H. Chen, F.-C. Chang, H.-C. Lin, C.-J. Tsai, “ISO/IEC JTC1/SC29/WG11 M9766: FGS-Based Video Streaming Test Bed for Media Coding and Testing in Streaming Environments,” July 2003 (65th, Trondheim, Norway)
 69. H.-C. Huang, W.-H. Peng, C.-N. Wang, T. Chiang, H.-M. Hang, “ISO/IEC JTC1/SC29/WG11 M9767: Stack Robust Fine Granularity Scalability: Response to Call for Evidence on Scalable Video Coding,” July 2003 (65th, Trondheim, Norway)
 70. C.-N. Chiu, Frank Ho, Spencer Hsu, C.J. Tsai, “ISO/IEC JTC1/SC29/WG11 M9769: FGS Audio Integration for TR-21000 Multimedia Test Bed WD 2.0,” July 2003 (65th, Trondheim, Norway)

**Information technology — Multimedia
framework (MPEG-21) —**

**Part 12:
Test Bed for MPEG-21 Resource Delivery**

Technologies de l'information — Cadre multimédia (MPEG-21) —

Partie 12: Lit d'essai pour livraison de ressources MPEG-21

PROOF/ÉPREUVE

Reference number
ISO/IEC TR 21000-12:2005(E)



PDF disclaimer

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. The ISO Central Secretariat accepts no liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies. In the unlikely event that a problem relating to it is found, please inform the Central Secretariat at the address given below.

© ISO/IEC 2005

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Contents

Page

Foreword	iv
Introduction	vi
1 Scope	1
2 Overview of Functionality	1
2.1 Functionality of Test Bed	1
2.2 Target Use Cases	1
2.3 Relation with Reference Software	1
2.4 MPEG Technologies within Test Bed	1
2.5 API Overview and Language	2
3 Overall Architecture	2
4 Client Components	2
4.1 Introduction	2
4.2 Decoder Object	3
4.3 OutputBuffer Object	4
4.4 StreamBuffer Object	5
4.5 PacketLossMonitor Object	6
4.6 ClientController Object	6
5 Server Components	7
5.1 Introduction	7
5.2 MediaDatabase Object	8
5.3 ServerController Object	8
5.4 Streamer Object	9
5.5 File Format	10
5.6 DIA Object	10
6 Common Components	11
6.1 Introduction	11
6.2 PacketBuffer Object	11
6.3 QoSDecision Object	14
6.4 IPMP Objects	14
6.4.1 MessageRouter Object	15
6.4.2 ToolManager Object	16
6.4.3 IPMPTool Object	16
6.4.4 IPMPFilter Object	17
6.4.5 Terminal	18
7 Network Emulator and Network Profile Format	18
7.1 Introduction	18
7.2 Network profile file format	19
7.3 Synchronization between network profiles and streaming sessions	20
Bibliography	21

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

In exceptional circumstances, the joint technical committee may propose the publication of a Technical Report of one of the following types:

- type 1, when the required support cannot be obtained for the publication of an International Standard, despite repeated efforts;
- type 2, when the subject is still under technical development or where for any other reason there is the future but not immediate possibility of an agreement on an International Standard;
- type 3, when the joint technical committee has collected data of a different kind from that which is normally published as an International Standard (“state of the art”, for example).

Technical Reports of types 1 and 2 are subject to review within three years of publication, to decide whether they can be transformed into International Standards. Technical Reports of type 3 do not necessarily have to be reviewed until the data they provide are considered to be no longer valid or useful.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

ISO/IEC TR 21000-12, which is a Technical Report of type [3], was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

ISO/IEC TR 21000 consists of the following parts, under the general title *Information technology — Multimedia framework (MPEG-21)*:

- *Part 1: Vision, Technologies and Strategy*
- *Part 2: Digital Item Declaration*
- *Part 3: Digital Item Identification*
- *Part 5: Rights Expression Language*
- *Part 6: Rights Data Dictionary*
- *Part 7: Digital Item Adaptation*

- *Part 8: Reference Software*
- *Part 9: File Format*
- *Part 10: Digital Item Processing*
- *Part 11: Evaluation Tools for Persistent Association Technologies*
- *Part 12: Test Bed for MPEG-21 Resource Delivery*
- *Part 16: Binary Format*

The following parts are under preparation:

- *Part 4: Intellectual Property Management and Protection*
- *Part 15: Event Reporting*

Introduction

This document describes the component API of ISO/IEC TR 21000-12: Test Bed for MPEG-21 Resource Delivery. The test bed is mainly composed of a streaming player, a media server, and an IP network emulator. This document describes the API of each components of the test bed to facilitate a component oriented development process. This platform provides a flexible and fair test environment for evaluating scalable media streaming technologies for MPEG contents over IP networks.

Information technology — Multimedia framework (MPEG-21) —

Part 12:

Test Bed for MPEG-21 Resource Delivery

1 Scope

This Technical Report specifies a test bed that is designed to assist in performance assessment of MPEG-21, Scalable Video Codec (SVC) for streaming applications and for the evaluation of resource delivery technologies over unreliable packet-switched networks. A subset of MPEG-4 IPMP is also included in the test bed so that encrypted streaming and layered access functionality of a DRM system can be tested for different SVC designs.

2 Overview of Functionality

2.1 Functionality of Test Bed

This platform provides a flexible and fair test environment for evaluating scalable media streaming technologies for MPEG contents over IP networks. In particular, the test bed is designed for the evaluation of MPEG Scalable Video Codec. This test bed has capabilities of simulating different channel characteristics of various networks, therefore,

- Various scalable codec (audio, video, scene composition) technologies could be evaluated.
- Various packetization methods and file formats can be evaluated.
- Various multimedia streaming rate control and error control mechanisms can be plugged into the test bed and evaluated.

2.2 Target Use Cases

Currently, the test bed is targeted at scalable audio and video streaming applications with some DRM support.

2.3 Relation with Reference Software

It must be emphasized that the software provided with this TR is not part of the MPEG-21 reference software. In addition to providing some useful utility software for resource delivery system development, this TR tries to show a solid example of how MPEG technologies can be integrated together in a working system for scalable audio/video streaming applications.

2.4 MPEG Technologies within Test Bed

The following MPEG technologies are supported by the test bed:

- MPEG scalable audio and video codecs
- MPEG-4 on IP
- MPEG-4 IPMP (a small subset)
- MPEG-21 DIA Network Adaptation QoS

The architecture and the API are not tied to any particular media codecs. However, currently only the MPEG-4 FGS video and MPEG-4 BSAC audio are officially supported by the software.

2.5 API Overview and Language

The API is divided into three parts, namely, server component API, client component API, and common component API. The network emulator described in clause 7 is a standalone application that is not part of the component-based framework. The programming language used to describe the API is C++. This is because the project is mainly implemented using the C++ language, except for the network emulator GUI, which is done in Java. This will make it easier for the implementers to follow the API and to design different modules to merge into the test platform.

3 Overall Architecture

The overall architecture of the Test Bed is illustrated in Figure 1.

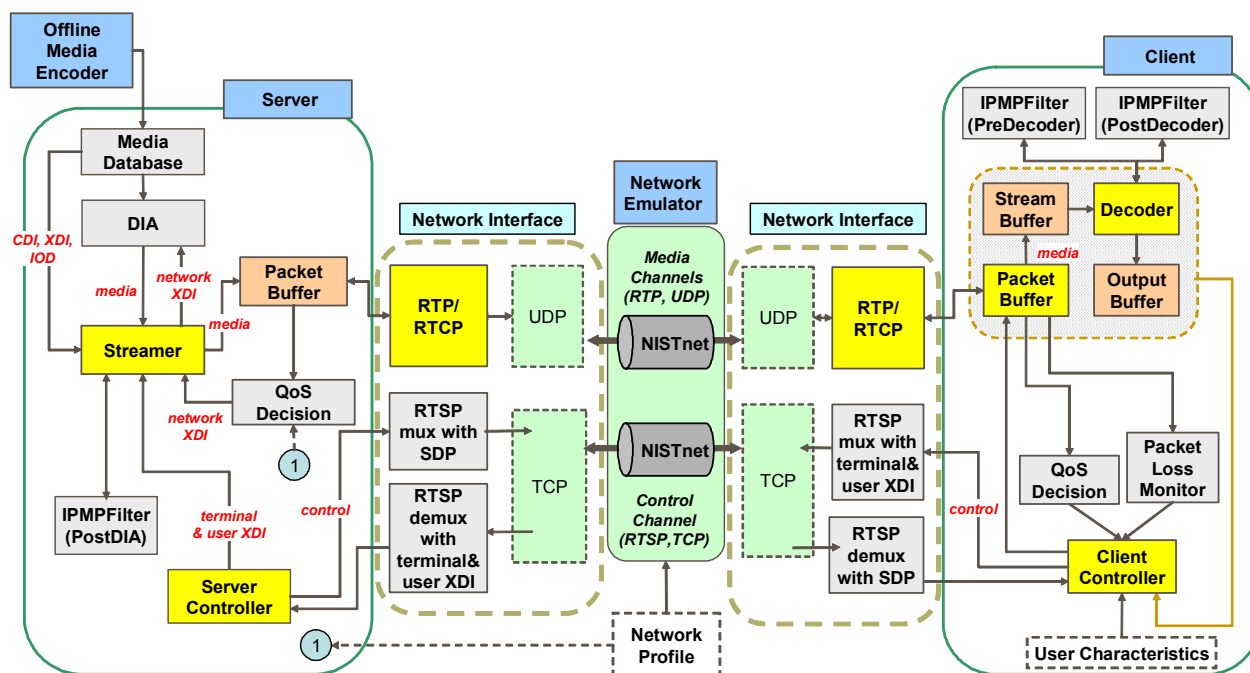


Figure 1 — Architecture of the resource delivery test bed

Subsequently, the various server and client components that can fit in this test-bed architecture are described in clause 4.

4 Client Components

4.1 Introduction

This section lists the detail component APIs so that third party component providers can design their own client modules that can be integrated into this test bed.

4.2 Decoder Object

This is the base object for all decoders, including real-time and offline decoders. The task of fetching coded units from the input streams and storing decoded units into the output stream is carried out by this base object. This is done as follows:

1. **Decoder** gets some coded bits from the input bitstream by calling **StreamBuffer::GetBitstreamData()**. A bitstream can contain multiple layers of sub-bitstreams. If no data in any layers of the sub-bitstream is available, the **Decoder**'s thread is suspended till data is available.
2. **Decoder** calls a private virtual function, *Decode()*. This function is overloaded by specific decoders and performs the actual decoding of a coding unit.
3. The output of *Decode()* is saved in a buffer via the method **OutputBuffer::WriteDecodedData()**. The GUI can later call **OutputBuffer::GetDecodedData()** to display (video) or play (audio) the decoded content. A presentation timestamp will be provided in this later API.

Methods

The following methods are implemented by the base class **Decoder**.

Decoder()

The object constructor (serious initialization is done in *Setup()*). No work is done till *Start()* is called.

int Setup(int track_id, uint8 *config, int32 config_length, int32 codec_type, void* streambuffer_link, void* outputbuffer_link)

This function initializes the decoder. The video header (or video configuration descriptor is passed in for parameters like video width & height or audio frame size). *config_length* is the length in bits of *config*, and *streambuffer_link* and *outputbuffer_link* are the pointers to **StreamBuffer** object and **OutputBuffer** object respectively. A non-zero error code returns if it fails.

void Start()

Start the **Decoder** thread, *Run()*, to perform the decoder task.

void Close()

Must be called before the **Decoder** object is deleted in order to terminate the **Decoder** thread.

static void Run(void *)

This method, which runs the **Decoder** thread, fetches compressed units from the input **StreamBuffer** object, calls *Decode()* to decode them, and saves the decoded presentation units to the output **OutputBuffer** object through the **OutputBuffer::WriteDecodedData()** method. The function terminates when the input stream returns *END_OF_STREAM* on a call to *GetBitstreamData()*.

bool Decode()

Decode one coding unit. The function returns TRUE if successful.

uint32 GetAttributes(ATTRIBUTE_CODE code, void *value)

Depending on the value of *code*, the requested attribute is returned through the pointer parameter *value*. In some codecs, this function returns meaningful values after the *Setup()* function (i.e. after the decoding of the audio-visual configuration header). A nonzero error code returns if it fails. For some codecs (e.g. ITU-T H.263), this function only returns meaningful values after decoding of a frame (after calling the function *Decode()*).

When the attribute “SUPPORTED_CODEC_NAMES” is requested, the function returns a pointer of an array of UTF-8 strings of codec names. Each string in the array begins with an one byte length field, followed by the characters. Strings are concatenated one after another, the whole string arrays ends with a zero byte. For example, a **Decoder** object may returns a pointer points to the following string array:

```
{ 0x03, 'A', 'S', 'P', 0x03, 'F', 'G', 'S', 0x00 }.
```

The default implementation of this function always returns zero.

ATTRIBUTE_CODE is defined as follows:

```
typedef enum
{
    // common attributes
    SUPPORTED_CODEC_NAMES

    // for visual
    FRAME_WIDTH,      // the width in pixels of a video frame
    FRAME_HEIGHT,     // the height in pixels of a video frame
    FRAME_BITS_PER_PIXEL, // bits per pixel in a video frame
    FRAME_PIXEL_FORMAT, // pixel format in a video frame

    // for audio
    PITCH,            // the pitch of an audio sample
    AUDIO_FREQUENCY,  // the frequency of an audio sample
    NUM_CHANNELS,     // number of audio channels
    BITS_PER_SAMPLE,  // number of bits per audio sample
    AUDIO_CU_DURATION, // the duration in milliseconds of a sample
} ATTRIBUTE_CODE;
```

As you can see from this definition, codes for different types of decoders are mixed. A decoder is expected to return value only for attribute codes that are relevant to it. Otherwise it returns zero. The returned attribute is always type-casted to uint32. The method returns zero if the requested attribute is not recognized or available by the decoder.

4.3 OutputBuffer Object

This object class is used as an abstraction of the output device (for either audio and video). The decoder, after decoding, should dump the decoded data into this object. The client GUI would then send the data to the actual device based on the timestamp.

Methods

OutputBuffer()

This is the object constructor.

int Setup(int32 buffer_size)

buffer_size is the size of one decoded data unit (e.g. an YCbCr video frame or a PCM audio frame) in bytes.

int WriteDecodedData(uint32 timestamp, uint8 *data, int32 data_length)

This method sends a decoded unit to the output buffer. The pointer **data* points to the decoded frame and the size of the decoded data is specified by *data_length*. The timestamp, *timestamp*, of the decoded frame is in milliseconds. The function returns zero upon success, otherwise, it returns a non-zero code when it fails.

int GetDecodedData(uint32 *timestamp, uint8 **data, int32 *data_length)

The client GUI will call this method to render the decoded content (audio or video). The indirect pointer ***data* returns a pointer to the decoded data and the size of the decoded data is returned via *data_length*. The presentation time is returned via the parameter **timestamp*. The client GUI should render the data at time *timestamp* (in millisecond). The function returns zero upon success, otherwise, it returns a non-zero code when it fails.

int ReleaseBuffer(void)

After client player gets decoded data, it will call this method to release the control to the buffer which is obtained by using **GetDecodedData()**.

4.4 StreamBuffer Object

StreamBuffer is a FIFO buffer that holds raw bitstream data. Even though the data structure is intended for raw data bits, the actual implementation should record raw bitstream boundary “markers” set by the transport layers, e.g. IP packet boundaries (see method **GetBitstreamData()** for more explanations), through some auxiliary data structure.

Methods**StreamBuffer(int number_of_tracks)**

This is the object constructor. *number_of_tracks* is the total number of media tracks (e.g. audio-visual tracks or multiple levels/layers for scalable codecs).

int Setup(int track_id, int32 buffer_size, void *packetbuffer_link)

buffer_size is the size of the raw bitstream buffer in bytes. *Packetbuffer_link* is the link to **PacketBuffer** object.

int SetBookmark(int track_id, int32 bookmark_number)

This method records the current position in stream buffer to a bookmark array for track *track_id*, and the corresponding index in this array is *bookmark_number*. This method is always called by **Decoder** object to do backward searching preparation. It returns a non-zero code when it fails.

int GotoBookmark(int track_id, int32 bookmark_number)

This method will move the current position to that recorded in the bookmark array by index *bookmark_number* for track *track_id*. It returns a non-zero code when it fails.

int32 GetBitstreamData(int track_id, int32 nbytes, uint8 *data)

This method retrieves *nbytes* bytes from the track *track_id* of a **StreamBuffer** object, and returns the value to the caller. The returned data is stored in the pointer *data*. The function returns zero upon success, otherwise, returns a non-zero code when it fails.

int32 GetOffsetToNextDataBoundary(int track_id)

In a streaming-over-IP system, bitstream data has a natural boundary set by the transport layer (e.g. for each frame, slice or video packet, etc.). This function returns the number of bits to the next marker (or packet boundary) position in track *track_id*. Some decoders can use this information to avoid resync marker searching. It returns zero if there is no further boundary point.

int PutBitstreamData(int track_id, int32 nbytes, uint8 *data)

This method adds *nbytes* bytes into track *track_id* of the **StreamBuffer** object. The output data is stored in the pointer *data*. The function returns zero upon success, otherwise, returns a non-zero code when it fails.

void SetDataBoundaryPoint(int track_id)

This method set the current bit position as a data boundary point of track *track_id*. This information will be used by succeeding calls to *GetOffsetToNextDataBoundary()*. This method is usually called by a **PacketBuffer** object.

4.5 PacketLossMonitor Object

This module handles lost packet monitoring and retransmission. The method *CheckPacketLoss()* will be triggered occasionally from client-side timer to issue retransmission requests.

Methods**PacketLossMonitor(int32 number_of_track)**

This is the object constructor. *number_of_tracks* is the total number of media tracks (e.g. audio-visual tracks or multiple levels/layers for scalable codecs).

int Setup(void *packetbuffer_link, void *clientcontroller_link)

The parameter *packetbuffer_link* is a pointer to the **PacketBuffer** object, and *clientcontroller_link* is a pointer to the **ClientController** object.

void EnableRetransmission(int track_id, bool on_off_flag)

This method turns on/off the retransmission mechanism for track *track_id*. The state is controlled by *on_off_flag*. If *on_off_flag* = 0, the mechanism is turned off, otherwise, it's turned on.

void CheckPacketLoss()

This function checks all the track buffers whose retransmission mechanism has been turned on to see if there are any packet losses. If there are packet losses and the condition is valid for retransmission, this method calls **ClientController::RetransmissionRequest()** to issue retransmission requests.

4.6 ClientController Object

This object class integrates and controls the components of client (include the component of IPMP). The design principle of the **ClientController** APIs tries to follow the “maximum freedom” rule. Even though the naming of the methods is adopted from the RTSP model (such as, DESCRIBE, PLAY, etc.), there is no coercing that RTSP has to be used as the control protocol.

Methods**ClientController()**

This is the object constructor.

int Setup(const void *client_info)

This method set up the **ClientController** object. The input parameter *client_info* is intentionally left as a free-form parameter. In the current implementation, this parameter is a pointer to a structure that contains information about session ID, media info (for SDP [5]), terminal capabilities, etc. The structure is highly

dependent on the control protocols used. The function returns zero upon success, otherwise, it returns a non-zero code when it fails.

void Start()

Start the **ClientController** thread, *Run()*, to perform the control event processing loop.

void Close()

Must be called before the **ClientController** object is deleted in order to terminate the **ClientController** thread.

static void Run(void *)

This method, which runs the **ClientController** thread, sits in a loop waiting for the GUI user inputs and processing these inputs. If failure or exception occurred, this thread should set an event to acknowledge the UI. The event handler can be fetched via *GetAttributes()* method.

int GetAttributes(const char *attribute_name, void *value)

This method returns a server or client attribute (specified by *attribute_name*). The requested attribute is returned through the pointer parameter *value*. If the returned value is NULL, then *attribute_name* is an invalid (or unsupported) attribute. If it is a server attribute, the client could use the GET_PARAMETER method of RTSP to get it. The function returns zero upon success, otherwise, it returns a non-zero code when it fails.

int SetAttributes(const char *attribute_name, void *value)

This method sets the value of client or server attributes. If it is a server attribute, the client could use the SET_PARAMETER method of RTSP to set it. The attribute is specified by *attribute_name*, and its value is pointed to by *value*. The function returns zero upon success, otherwise, it returns a non-zero code when it fails.

int RetransmissionRequest(uint32 *packet_numbers, int nPackets, int track_id)

A **PacketLossMonitor** object calls *RetransmissionRequest()* to issue a retransmission with specified packet numbers in packet number list (*packet_numbers*). The number of packets in the list is given by the parameter *nPackets*. The requested missing packets belong to the media track *track_id*. The function returns zero upon success, otherwise, it returns a non-zero code when it fails.

void OnStartPresentation(void *parameters)

This call-back method will be called upon the receiving of a “start media presentation” request from the GUI. The input parameter *input* is intentionally left as a free-form parameter. When the user issues a PLAY request through the client GUI, this methods will be called. Three types of (RTSP) control requests will be processed here, namely DESCRIBE, SETUP, and PLAY.

void OnStopPresentation(void *parameters)

This call-back method will be called upon the receiving of a “end media presentation” request from the GUI. The input parameter *input* is intentionally left as a free-form parameter. The control requests TEARDOWN will be processed in this method.

5 Server Components

5.1 Introduction

This section lists the detail component APIs so that third party component providers can design their own server modules that can be integrated into this test bed.

5.2 MediaDatabase Object

This object class currently uses a simple directory tree structure for content storage. The server configuration file must sit in the root directory of the content database.

Methods

MediaDatabase()

This is the object constructor.

int OpenMediaFile(int track_id, char *filename)

This method opens file *filename* for track *track_id*. This method is usually called by the **Streamer** object.

int CloseMediaFile(int track_id)

This method closes media file for track *track_id*. This method is usually called by the **Streamer** object.

int GetNextResourceUnit(int track_id, uint8 *buffer, int32 *buffer_length, uint32 *timestamp)

This method retrieves one resource unit (can be a video frame, video packet, or an audio frame) from track *track_id* along with its timestamp. This function is usually called by the **DIA** object and the data will be stored in the buffer pointed to by *buffer*. The parameter *buffer_length* is an input/output parameter that stores maximum buffer length (in bytes) as an input parameter and returns the actual data size. Timestamp of the data will be returned via *timestamp*. The function returns a user-defined status code that controls the transmission behavior of the streamer.

int GetMediaConfigHeader(int track_id, uint8 *buffer, int32 *buffer_length)

This method retrieves the media configuration header (for example, the MPEG-4 VOL header) from track *track_id* for SDP composition. This function is usually called by the **ServerController** object and the data will be stored in the buffer pointed to by *buffer*. The parameter *buffer_length* is an input/output parameter that stores maximum buffer length (in bytes) as an input parameter and returns the actual data size. The function returns zero upon success, otherwise, it returns a non-zero code when it fails (for example, if the buffer is too small to store the next decoding unit).

5.3 ServerController Object

The design principle of the **ServerController** APIs, just like the **ClientController** APIs, tries to follow the “maximum freedom” rule. Even though the naming of the methods is adopted from the RTSP model (such as, DESCRIBE, PLAY, etc.), there is no coercing that RTSP has to be used for the implementation. To create a multicast **ServerController**, for example, most of these APIs will be NULL APIs that does not do anything.

For a unicast system, an individual server controller will be created for each connecting client.

Methods

ServerController()

This is the object constructor.

int Setup(const void *server_info)

This method set up the **ServerController** object for a connecting client. The input parameter *server_info* is intentionally left as a free-form parameter. In existing implementation, this parameter is a pointer to a structure that contains information about session ID, media info (for SDP [5]), etc. The structure is highly

dependent on the control protocols used. The function returns zero upon success, otherwise, it returns a non-zero code when it fails.

void Start()

This method starts the **ServerController** thread, *Run()*, to perform the control event processing loop.

void Close()

This method must be called before the **ServerController** object is deleted in order to terminate the **ServerController** thread properly.

static void Run(void *)

This method, which runs the **ServerController** thread, sits in a loop waiting for the control message inputs from the control channel and processing these inputs by calling the callback function. To adapt the framework to different protocols, only the callback function needs to be changed.

void Callback(int32 message_id, void *parameters)

This callback function is called by control protocol module when a message is received. The pointer *parameters* specifies the input parameters from the message. The callback method shall configure the server state, process the requests, and send ACK back to the client.

int GetAttributes(const char *attribute_name, void *value)

This method returns a server or client attribute (specified by *attribute_name*). The requested attribute is returned through the pointer parameter *value*. If the returned value is NULL, then *attribute_name* is an invalid (or unsupported) attribute. If it is a client attribute, the server could use the GET_PARAMETER method of RTSP to get it. The function returns zero upon success, otherwise, it returns a non-zero code when it fails.

int SetAttributes(const char *attribute_name, void *value)

This method sets the value of client or server attributes. If it is a client attribute, the server could use the SET_PARAMETER method of RTSP to set it. The attribute is specified by *attribute_name*, and its value is pointed to by *value*. The function returns zero upon success, otherwise, it returns a non-zero code when it fails.

5.4 Streamer Object

The **Streamer** object performs several tasks: packing bitstreams into RTP payload and stores them into packet buffer. This object also performs rate adaptation and packet scheduling.

Methods

Streamer(char *ip_addr, void *DIA, void *QoS, int number_of_tracks)

This is the object constructor. The parameter *ip_addr* is the IP of the remote host, *DIA* is a pointer to the associated **DIA** object, *QoS* is a pointer to the **QoSDecision** object, and *number_of_tracks* is the total number of media tracks (e.g. audio-visual tracks or multiple levels/layers for scalable codecs).

void Setup(int track_id, int32 client_port, void *rtp_info)

This method commands the streamer to create a session for a client and retrieve the related RTP information such as SSRC, time base, sequence number offset, and local port. These are necessary information for the sending controller to communicate with receiving controller via SDP. The data type of *rtp_info* can be a user-defined data structure that contains variables for storing necessary information.

void Start()

Start the streamer thread, *Run()*, to perform the streaming task.

void Close()

Must be called before the **Streamer** object is deleted in order to terminate the streamer thread.

static void Run(void *)

This method runs the streamer thread. It call **DIA::GetNextResourceUnit()** to read some data for packetization. It performs rate adaptation by discarding some data from enhancement layers. The *Run()* thread is created as soon as a client connects. However, it will not start sending data until an RTSP "PLAY" request from the client is received. Upon receiving a PLAY request, the **ServerController** object calls **Streamer::StartSending()** to start sending the RTP packets. The *Run()* thread also calls the method **QoSDecision::CalculateCurrentBandwidth()** to retrieve available network bandwidth in order to perform rate adaptation.

void StartSending(int track_id)

This method sets the sending flag for the corresponding client so that the *Run()* thread starts pumping out RTP packets. The parameter *track_id* informs the streamer which media track to send.

void RetransmissionRequest(int track_id, uint32 seq_no)

This method sets retransmission flag for packet *seq_no* of track *track_id*. **ServerController** calls this method to notify the **Streamer** object to resend the missing packet.

void SetAdaptationParameters(void *parameters)

The **ServerController** object uses this function to pass the client terminal capability information (such as maximal width, height, and video frame rate or audio sampling rate it can support) into the **Streamer**. The pointer parameter is left as a free form pointer so that different implementation of **ServerController** and **Streamer** can design their own data type.

5.5 File Format

Currently, the file format for a content maintained by **MediaDatabase** object is just a set of raw bitstreams (ASP, FGS, and BSAC) and XML files (CDI, XDI) located in a directory under the name of the content. The design simplify the system and the authoring tools. However, for a large scale system, this type of content management is not very efficient. Eventually, ISO file format should be used in this reference platform.

5.6 DIA Object

The DIA object performs media resource adaptation by DIA engine processing. At first, the DIA engine will get the CDI and static XDI information from **ServerController** object for initial setup. Besides, the dynamic XDI will be set in the adapting process. The adapted resource will be generated with CDI and these two XDIs information. After adaptation, **Streamer** object will get adapted resource by **GetNextResourceUnit** in DIA object.

Methods

DIA()

This is the object constructor.

int Setup(int track_id, int engine_id, string *CDI, string *XDI, MediaDatabase *MDB)

The **ServerController** object uses this function to initialize the **DIA**. The *engine_id* indicates what kind of adaptation engine to be used. *CDI* provides the properties (ex: location, type, frame rate) of media resource. *XDI* is specified for the adaptation guide settings of media resource. *MDB* is a pointer of **MediaDatabase**. So that **DIA** can access original media resource.

int SetXDInfo(int track_id, string *XDI)

This method is used to set the dynamic XDI information. For example, the **Streamer** object can use this method to pass the dynamic network information into the **DIA** object for setting adaptation parameters.

int GetCDInfo(int track_id, string *CDI)

This method retrieves the adapted CDI information from the **DIA** object.

int GetNextResourceUnit(int track_id, uint8 *buffer, int32 *buffer_length, uint32 *timestamp)

This method retrieves one adapted resource unit (e.g., a video frame, video packet, or an audio frame) from track *track_id* along with its timestamp. This function is usually called by the **Streamer** object and the data will be stored in the buffer pointed to by *buffer*. The parameter *buffer_length* is an input/output parameter that stores maximum buffer length (in bytes) as an input parameter and returns the actual data size. Timestamp of the data will be returned via *timestamp*. The function returns a user-defined status code that controls the transmission behavior of the streamer.

6 Common Components

6.1 Introduction

This section lists the APIs of the components that are common to both the client module and the server module. Even though these components function differently when they are used in the client and in the server, they are listed under common components since both the client side and the server side requires matching pairs of these components.

6.2 PacketBuffer Object

Real-time Transport Protocol/Real-time Control Protocol (RTP/RTCP, RFC-1889) is used as the media transport mechanism. This object class implements an RTP packet buffer data structure. This class is RTP payload independent and calls jRTPLib ([1]) for standard RTP/RTCP composition and transporting. The main method, *Run()*, of this object class should run in its own thread.

A data structure, *rtp_header*, used in this object is defined as follows,

```
typedef struct _rtp_header
{
    uint32 seq_no;           // extended sequence number
    uint32 rtptime;         // rtp timestamp
    uint8 PT;               // payload number
    uint8 marker;           // marker bit(s)
} rtp_header;
```

Both client and server uses the same **PacketBuffer** object with different initializations. The reason is to force this module to be updated simultaneously and consistently for both the client and the server.

Methods

PacketBuffer(bool for_server, int32 number_of_tracks)

This is the object constructor (serious initialization is done in *Setup()*). The parameter *for_server* is set to true if this object is intended for server, false if it is intended for a client. The parameter *number_of_tracks* specifies the total number of media tracks in the streaming session.

This class has two different overloading *Setup()* methods, one for the client and the other for the server. These methods are defined as follows.

int Setup(int track_id, int32 number_of_packets, int32 max_packet_size)

On the client side, this function initializes the packet buffer object for media track *track_id*. The parameter *number_of_packets* is the total (maximum) number of packet allowed in the packet buffer. *max_packet_size* is the permissible maximal size of a single packet. The function returns zero upon success, otherwise, returns a non-zero code when it fails.

int Setup(int track_id, int32 number_of_packets, int32 max_packet_size, void* extra_parameter)

On the server side, this method initializes the packet buffer object for media track *track_id*. The parameter *number_of_packets* is the total (maximum) number of packet allowed in the packet buffer for retransmission. *max_packet_size* is the permissible maximal size of a single packet. The function returns zero upon success, otherwise, returns a non-zero code when it fails.

The parameter *extran_parameter* is the pointer of RTP related information (Ex: SSRC) for sever setup.

void Start()

Start the packet buffer thread, *Run()*, to perform the packet buffer task.

void Close()

Must be called before the **PacketBuffer** object is deleted in order to terminate the packet buffer thread.

static void Run(void *)

This method runs the packet buffer thread. On the client side, it fetches RTP packets from the input sockets, stores them in the packet ring buffer, and waits for **StreamBuffer** object to retrieve bitstream data. On the server side, this method is not used.

int ReadCodedUnit(int track_id, uint8 *buffer, int32 *buffer_length, uint32 *timestamp)

This method retrieves enough packets from the packet buffer to compose one decoding unit (can be a video frame, video packet, or an audio frame) from track *track_id* along with its timestamp. This function is usually called by the **StreamBuffer** object and the data will be stored in the buffer pointed to by *buffer*. The parameter *buffer_length* is an input/output parameter that stores maximum buffer length (in bytes) as an input parameter and returns the actual data size. Timestamp of the data will be returned via *timestamp*. The function returns zero upon success, otherwise, it returns a non-zero code when it fails (for example, if the buffer is too small to store the next decoding unit).

int32 WritePacketData(int track_id, rtp_header *header, uint8 *data, uint32 length, uint32 time_schedule)

This method composes some RTP payloads and the corresponding RTP information for track *track_id*. The parameter *data* points to the input bitstream data with size *length* bytes. *time_schedule* is the scheduled time

(in milliseconds) for this packet to go out. The function returns zero upon success, otherwise, it returns a non-zero code when it fails.

int GetLostPacketSequenceNumber(int track_id, int32 **seq_no_list)

This method returns channel condition to the caller(s), namely the **QoSDecision** object and/or the **PacketLossMonitor** object. The parameter *track_id* specifies the media track in interest. The function returns the total number of lost packet since last call. A pointer to an array of lost sequence number is returned via the output parameter *seq_no_list*.

int SendRetransmissionPacket(int track_id, uint32 seq_no, int32* payload_length)

This function is usually called by **Streamer** object for sending retransmission packet (extended sequence number *seq_no*) for track *track_id*. The parameter *payload_length* is the pointer of payload length of the retransmission packet. This method returns a non-zero code when it fails.

void GetChannelCondition(int track_id, int32 *FPL, int32 *CPL, int32 *xSeqNum, int32 *jitter, int32 *RTT, void *app_specific_parameters)

This method returns the current channel condition of track *track_id* to the caller, namely the **QoSDecision** object class. On the server side, these information mainly comes from the RTCP feedbacks. On the client side, these information comes directly from the receiving statistics of RTP packets. All the parameters are output parameters; and the meanings of parameters are listed as follows [3]:

1. *FPL*: Fraction of packet loss.
2. *CPL*: Culmulative number of packet loss.
3. *xSeqNum*: the highest extended sequence number.
4. *jitter*: packet inter-arrival jitter.
5. *RTT*: estimated round-trip time.
6. *app_specific_parameters*: user-defined channel parameters. All the above parameters are standard channel condition parameters defined in RTP specification [3]. The pointer *app_specific_parameters* is a free-form parameter for user to define extended rate adaptation mechanisms. The information may pass back from the clients to the server via the RTCP APP packets.

int GetCurrentBufferDepth(int track_id)

This method returns the current depth (in milliseconds) of media data in the track *track_id* buffer. The method returns zero upon success, and a non-zero error code upon failure.

int GetNetworkAttribute(int track_id, uint8 *attributes, void* value)

This method returns the network information (such as UDP ports) via the parameter *attribute*, and the requested attribute is returned through the pointer parameter *value*. This method is usually called by the **ClientController** object to retrieve network related information maintained by the **PacketBuffer** object. This parameter is intentionally left as a free form parameter. The method returns zero upon success, and a non-zero error code upon failure.

int SetNetworkAttribute(int track_id, uint8 *attributes, void* value)

This method sets the network information specified by the parameter *attribute*, and the requested attribute is set through the pointer parameter *value*. This parameter is intentionally left as a free form parameter. A typical use of this function is for **ClientController** to set network related information (such as IP address). The method returns zero upon success, and a non-zero error code upon failure.

6.3 QoSDecision Object

This object class estimates the channel condition and provides some QoS information for rate adaptation.

Methods

QoSDecision()

This is the object constructor.

int Setup(bool for_server, void *extra_parameter, void *PB)

This function initializes the **QoSDecision** object. The parameter *for_server* is set to true if this object is intended for server, false if it is intended for a client. *extra_parameter* is a free form pointer that can be used to pass some extra info to setup the object. Finally, *PB* is a pointer to the associated **PacketBuffer** object. In the existing implementation, this parameter passes in the file name of the simulating network profile. The function returns zero upon success, otherwise, returns a non-zero code when it fails.

int CalculateCurrentBandwidth(void *QoS_parameter)

This method returns the current bandwidth to the caller. For a system with true dynamic streaming rate control, the method calls the method **PacketBuffer::GetChannelCondition()** to obtain statistics about the channel and calculates the bandwidth accordingly. The parameter *QoS_parameter* is a free form pointer that can be used to provide implementation specific parameters. The returned value is the available channel rate (in bps) associated with the input *QoS_parameter* of a specific media.

6.4 IPMP Objects

The MPEG-4 IPMP system is mainly composed of three parts, namely Terminal, Message Router, and Tool Manager. The Terminal is the consumer of IPMP protected contents. The functions of a Terminal in IM1 are mapped to the functions of the client of the test bed, with some modifications.

All the messages come from an IPMP Tool to another IPMP Tool or the Terminal to an IPMP Tool are routed by Message Router. IPMP Message Router could just be a conceptual entity that implements the Terminal-side interface between the Terminal and an IPMP Tool.

The function of the Tool Manager is to create instances of IPMP Tools, and to connect the IPMP Tool to the corresponding position to perform IPMP functions.

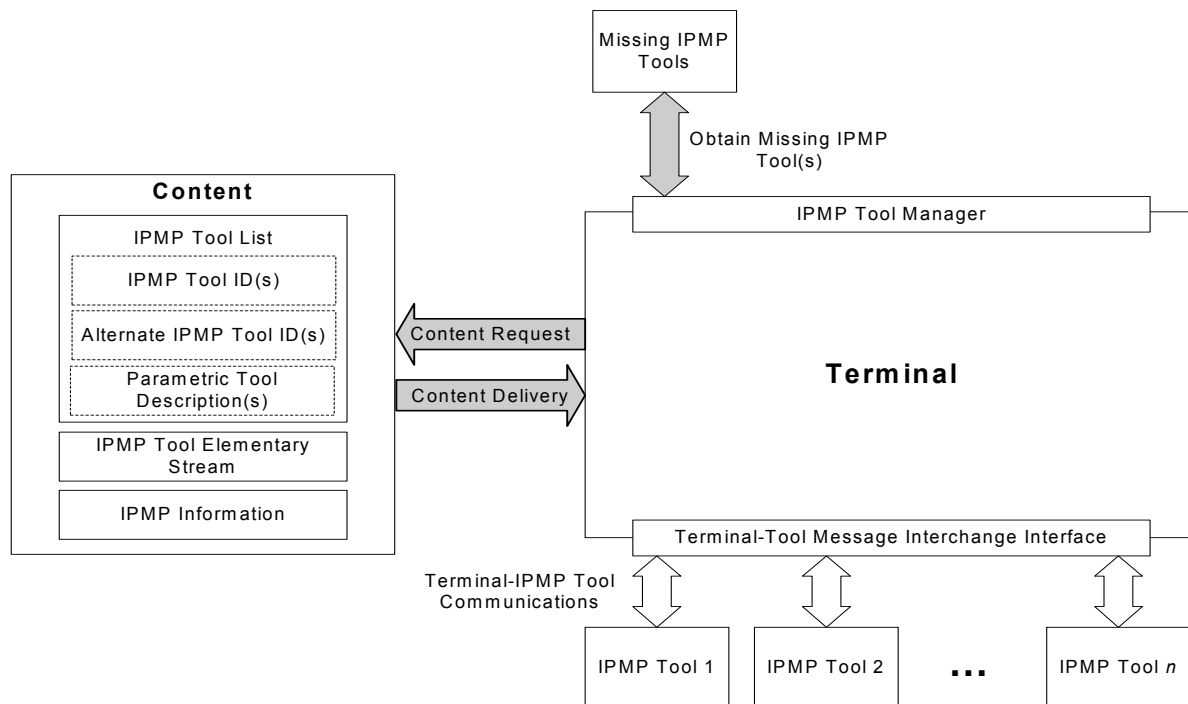


Figure 2 — IPMP architecture

The detailed component behaviors and data structure definitions could be found in [6]. Based on the original MPEG-4 IPMPX architecture, we define the IPMP APIs of the Testbed in the following sections.

6.4.1 MessageRouter Object

All IPMP messages are routed by **MessageRouter**. It also handles **IPMPToolDescriptors** and **ESDs**. On receiving an **IPMPDescriptorPointer**, it asks **ToolManager** to instantiate tool at given control point.

Methods

int ProcessIPMPToolDescriptor(IPMPToolDescriptor* IPMPtooldescriptor)

This API is called by the **Terminal** to process the IPMP Tool Descriptor. When the **Terminal** receives the IOD from the server, it calls this function to pass *IPMPtooldescriptor* contained in IOD to **MessageRouter**. The function returns zero when succeed.

int ProcessESD(ESD* esd)

This API is called by the **Terminal** and processes the ES Descriptor. When the **Terminal** receives the IOD from the server, it calls this function to pass all ESDs contained in IOD to **MessageRouter**. The function returns zero when succeed.

int ProcessIPMPDescriptorPointerD (uint32 contextID, IPMPDescriptorPointer* ptr)

This API is called when processing an ES Descriptor. **MessageRouter** instructs **ToolManager** to instantiate an **IPMPTool** at the given control point of the given ES. The *contextID* specifies the context where the pointer resides. The *ptr* contains the information of which IPMPToolDescriptor is to be used. The function returns zero when succeed.

int ReceiveMessage(IPMPToolMessageBase* msg)

This API is called by an **IPMPTool** or the **Terminal**. On receiving a message (the parameter *msg*),

MessageRouter determines and sends it to the corresponding recipient. The function returns zero when succeed.

IPMPContext* GetContext()

This API provides means to access the corresponding context. All IPMP objects can refer to each other by navigating through the context tree.

6.4.2 ToolManager Object

ToolManager is responsible to parse the **IPMPToolListDescriptor** and to resolve the **IPMPTools** on the list. It is also responsible to instantiate and destroy the **IPMPTools**.

Methods

int ReceiveToolListDescriptor(ToolListDescriptorD* tool_list)

This API is called by the **Terminal** and processes the IPMP tool list. When **Terminal** receives the Tool List Descriptor in the IOD, the function is called to pass *tool_list* to **ToolManager**. Then, the **ToolManager** resolves the tool IDs for later instantiation. Note that the server-side and client-side resolving process may be quite different. The function returns zero when succeed.

IPMPTool* ConnectTool(uint32 context_id, IPMPToolDescriptorD* descr)

Instantiate and connect a tool. When this method is invoked, the tool is instantiated by the parameters specified in *descr*. Then, the **ToolManager** connects it to the specified location (*context_id* specifies the ES to apply, and the control point is specified in *descr*).

int DisconnectTool(IPMPTool* tool_ptr)

Disconnect the given tool pointed by *tool_ptr* from the **IPMPFilter**. The function returns zero when succeed.

IPMPContext* GetContext()

This API provides means to access the corresponding context. All IPMP objects can refer to each other by navigating through the context tree.

6.4.3 IPMPTool Object

IPMPTool perform the function of IPMP, such as en/decryption, or watermarking.

Methods

int ReceiveMessage(IPMPToolMessageBase* msg)

This API is called by the **MessageRouter** to pass a message to the **IPMPTool** object. The parameter *msg* is the IPMP message to be handled by this tool. The function returns zero when succeed.

int ProcessData(uint32 timestamp, uint8* in_data, uint32 in_size, uint8 out_data, uint32* out_size)**

This API is called by the **IPMPFilter** to process the given content data. The parameter *in_data*, is the input buffer of length *in_size*. The indirect pointer ***out_data* returns a pointer to the processed data, and the size returned as *out_size*. The output buffer should be created during the invocation of the method, and should be released somewhere else. The **IPMPFilter** should take care of the release operations, except the last output buffer of the IPMP tool chain. The function returns zero when succeed.

void Initialize(IPMPToolDescriptorD* init)

Initialize the **IPMPTool** by the given *init* object. This method is called by the **ToolManager** after this tool object is instantiated.

IPMPContext* GetContext()

This API provides means to access the corresponding context. All IPMP objects can refer to each other by navigating through the context tree.

6.4.4 IPMPFilter Object

An **IPMPFilter** object serves as a ControlPoint in IPMP. We can add and/or remove an **IPMPTool** into/from the **IPMPFilter**. On the Testbed, there are three **IPMPFilters** for each ES. To distinguish them in the following notations, we call them **PreDecoderFilter**, **PostDecoderFilter**, and **PostDIAFilter**. The **PreDecoderFilter** and **PostDecoderFilter** are in the client side. However, **PostDIAFilter** is in server side. On Testbed, an **IPMPFilter** is implemented more like a co-processor than a filter. For example, the **Decoder** get data from the **StreamBuffer**, and call the `ProcessData()` of the **PreDecoderFilter** to produce the “real” encoded data. After decoding, the **Decoder** passes the decoded data to the **PostDecoderFilter**, then puts the processed data to the **OutputBuffer**.

Methods**int AddTool(IPMPTool* tool, uint8 sequence_code)**

Add the given **IPMPTool** into the filter. The *tool* object is an initialized **IPMPTool**, and the *sequence_code* specifies the priority of the tool. Higher value of *sequence_code* denotes higher priority when processing data. This method returns zero when succeed.

int RemoveTool(IPMPTool* tool)

Remove the specified **IPMPTool** from the filter. The parameter *tool* specifies the **IPMPTool** to be removed. This method returns zero when succeed.

int ProcessData(uint32 timestamp, uint8* in_data, uint32 in_size, uint8 out_data_ptr, uint32* out_size_ptr)**

This method iterates through each contained **IPMPTool** and chained all data process operations to produce the final result. The *in_data* is the input data buffer of length *in_size*. The created output buffer is returned in *out_data_ptr* with length *out_size_ptr*. The method returns zero when succeed.

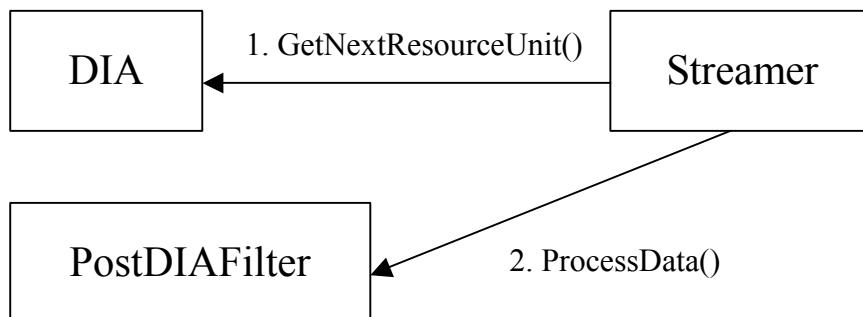


Figure 3 — Call-steps from a Streamer to IPMPFilter

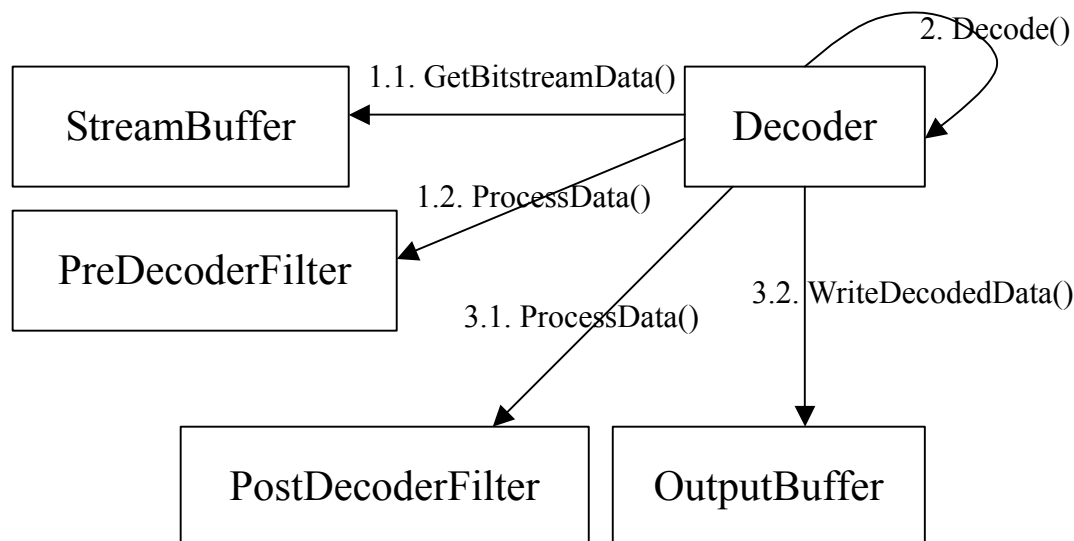


Figure 4 — Call-steps from a Decoder to IPMPFilters

6.4.5 Terminal

The Terminal object is an abstraction of the Testbed without IPMP. The object is provided as the interface to interact with non-IPMP part of the system.

Methods

int ReceiveMessage(IPMPToolMessageBase* msg)

This method is called by the **MessageRouter** to pass terminal specific IPMP messages. The *msg* object is the message to be processed. The method returns zero when succeed.

int SendMessageToBitstream(IPMPToolMessageBase* msg)

This method is called by the **MessageRouter** to pass remote-terminal specific IPMP messages. The *msg* object is the message to be sent to the other side. The method returns zero when succeed.

IPMPContext* GetContext()

This API provides means to access the corresponding context. All IPMP objects can refer to each other by navigating through the context tree.

7 Network Emulator and Network Profile Format

7.1 Introduction

In the test bed, an IP network emulator, NIST Net [2], is used to provide repeatable network environments that are close to practical wide-area heterogeneous network environments. NIST Net is a public domain Linux-based IP network emulator developed by the National Institute of Standard Technology. It is a raw IP packet filter with many controllable channel parameters such as packet loss ratio, jitter, bandwidth variation, delay, and network buffer size. To simulate different network environments, a GUI that controls the NIST Net kernel module is required. Current reference test bed includes a GUI module (nctu_simunet) that parses a network

profile text file and controls the time-varying network conditions via NIST Net. A typical test bed setup is shown in Figure 5.

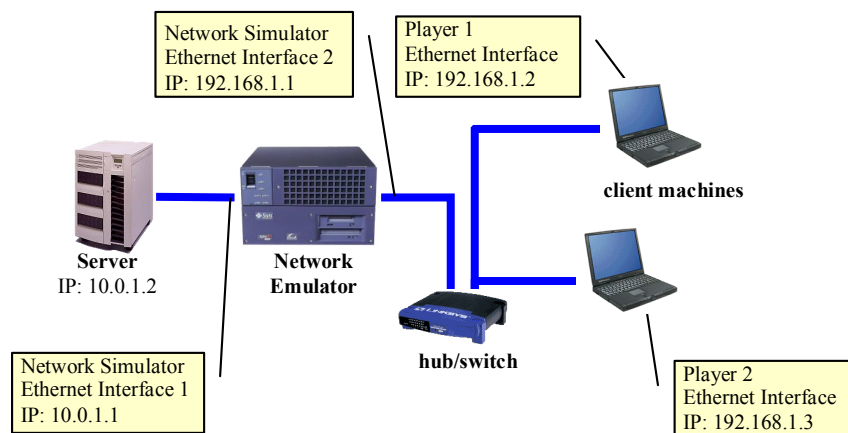


Figure 5 — A typical test bed setup

In addition to controlling the NIST Net module to emulate the pre-specified network profile, the GUI module also displays real time channel usage plots and statistics of both the uplink and downlink connections. A snapshot of the GUI is shown in Figure 6. The green curves in the figure are the pre-defined network bandwidth. The red curves are the actual real-time traffic data rate.

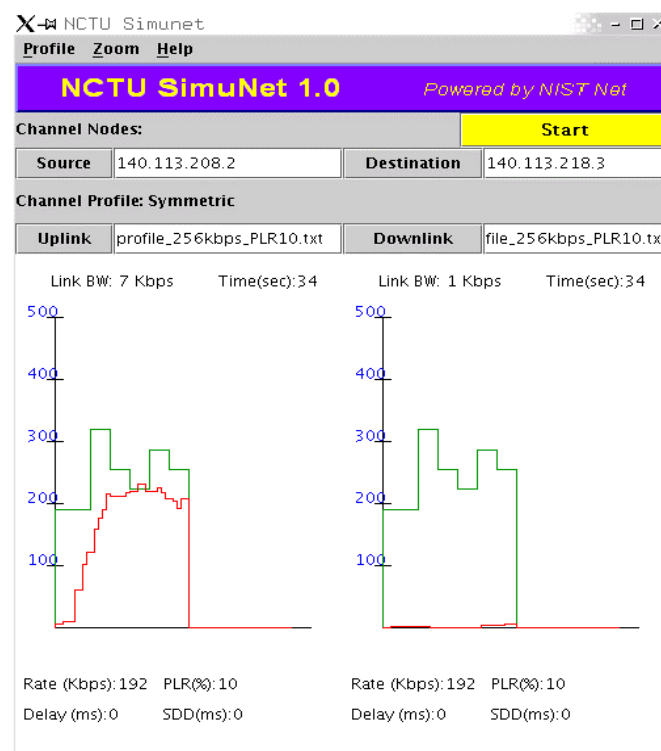


Figure 6 — The network emulator GUI

7.2 Network profile file format

The network profile is an ASCII text file that composed of two types of text lines. The first is a free-form comment line that begins with “#”. The second type of text line is a time-parameter description line. Each

time-parameter line includes six numbers: time (in second), packet loss ratio (%), channel bandwidth (in bps), packet mean delay (in msec), packet delay standard deviation (in msec), and network buffer size (specified by the number of maximum packets in the FIFO buffer). The first number “time” is the starting time when the network parameters (on the same line) take effective.

An example of a network profile is given in Figure 7.

# You can have as many lines of comments as you wish.						
#						
#	Time(sec)	PLR(%)	Bandwith(bps)	Delay(ms)	SDD	Network_Buffer_Size
	0	0	100000	20	0	80
	5	2	70000	10	0	80
	10	1	90000	10	0	80
	15	3	110000	10	0	80
	20	2	100000	30	0	80
	25	5	60000	10	0	80

Figure 7 — A sample file of network profile

7.3 Synchronization between network profiles and streaming sessions

The GUI starts the network condition control as soon as the first packet arrives in the controlled channel. This time instant is reference time zero of the network profile. When a server uses the network profile as a hint for network conditions (instead of probing the network by itself), it should know that the network emulator channel clock starts ticking when the simulator machine sees the first arrival packets in the channel. There can be more than one channel in a simulator box, each one with its own clock.

Bibliography

- [1] jRTPLib: <http://lumumba.luc.ac.be/jori/jrtplib/jrtplib.html>
- [2] NISTnet: <http://snad.ncsl.nist.gov/itg/nistnet>
- [3] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, *RTP: A Transport Protocol for Real-Time Applications*, RFC 1889, January 1996.
- [4] H. Schulzrinne, A. Rao, and R. Lanphier, *RTSP: Real Time Streaming Protocol*, RFC 2326, April 1998.
- [5] M. Handley and V. Jacobson, *SDP: Session Description Protocol*, RFC 2327, April 1998.
- [6] ISO/IEC 14496-13, *Information technology — Coding of audio-visual objects — Part 13: Intellectual Property Management and Protection (IPMP) extensions*

