中華民國資訊學會通訊

第十四卷第一期　中華民國一〇〇年三月出版

Communications of IICM

Institute of Information and Computing Machinery, Taiwan

Vol. 14, No.1, Mar. 2011

基於 Android 平台遠端即時多媒體桌面系統之設計與實作

王俊昆[1]、李茂仁[1]、陳慶儒[1]、蘇延麟[2]、羅習五[1]、郭峻因[1]、朱元三[2]、陳鵬升[1*]
[1]國立中正大學 資訊工程學系
[2]國立中正大學 電機工程學系
[*]E-mail: pschen@cs.ccu.edu.tw

摘要

近年來，嵌入式系統平台以驚人的速度蓬勃發展，並逐漸融入日常生活之中，嵌入式系統與一般電腦系統的整合與溝通顯得更加重要，遠端操控技術是其中一個非常重要的解決方法。本論文結合嵌入式系統、多媒體、與網路等相關技術，研發遠端即時多媒體桌面系統，本系統與現有遠端操控系統最大的差別在於使用者不只可以執行一般畫面更新頻率較低的應用程式（例如：文書編輯軟體），還可以執行畫面更新頻率較高的應用程式（例如：閱讀多媒體文件、觀看 Flash 動畫、查閱監視系統畫面），本系統提供使用者可隨時隨地透過支援網際網路通訊之嵌入式多媒體裝置，使用家裡、辦公室的電腦、或雲端計算中心的虛擬電腦設備，其使用的感覺就像在電腦設備旁使用一樣，讓嵌入式系統與一般電腦系統的整合與溝通更緊密。

壹、 前言

一、 簡介

近年來，嵌入式系統平台以驚人的速度蓬勃發展，並逐漸融入日常生活之中，嵌入式系統與一般電腦系統的整合與溝通就顯得更加重要，遠端操控技術是其中一個非常重要的解決方法。利用遠端操控技術，使用者所需的平台只需要簡單的運算功能，與畫面影像輸出的能力，所有運算服務皆透過網路另一端的主機執行，現今的遠端桌面系統對於畫面更新頻率較高的應用程式（例如：多媒體文件編輯軟體或是監控系統），傳輸速率會被畫面更新頻率與網路頻寬限制所影響，導致對於這些應用程式，遠端桌面技術所呈現的品質與流暢度極差，也喪失了即時性的優點。

為了解決這些問題，我們研發了遠端即時多媒體桌面系統。本系統與現有的遠端操控系統最大的差別在於使用者不只可以執行一般畫面更新頻率較低的應用程式（例如：文書編輯軟體），還可以使用畫面更新頻率較高的應用程式（例如：閱讀多媒體文件、觀看 Flash 動畫、查閱監視系統畫面），提供使用者完整且即時的遠端電腦使用環境。本系統包含了 client 與 server 兩大部分，其關鍵技術在於 server 內部具有動態影像偵測的能力，將高動態畫面擷取，另以先進的視訊壓縮技術（例如：H.264 [10,11]）壓縮處理，再傳輸至 client 端，而動態畫面在 client 端解壓縮，與其他靜態畫面重組後，再次顯示在使用者的面前。

二、 研究成果

本論文包含以下成果：

1. **遠端即時多媒體桌面系統**：我們研發了新一代的遠端即時多媒體桌面系統，重新定

義桌面上應用程式的畫面屬性，依更新頻率作為區分，將畫面分為更新頻率較低及更新頻率較高的區域，兩者在 server 端分別以不同的方式進行壓縮與傳送，在 client 端再還原成原來的畫面，解決傳統遠端桌面系統無法即時顯示多媒體畫面的問題。

2. **動態影像偵測技術**：我們提出了動態影像偵測技術，利用分析 X Window System [1,2] 裡通訊協定的內容，以極低的運算代價，分辨出動態畫面與靜態畫面。
3. **基於 Android 平台 client 端之設計與實作**：我們將 client 端實作於 Android [5,6,7]手機平台，重新規劃與設計 client 端的操作模式與顯示方式，節省 Android 平台上的運算資源，確保畫面的呈現品質。

貳、 遠端即時多媒體桌面系統

一、 系統設計

我們所研發的遠端即時多媒體桌面系統，在運作上與傳統的遠端桌面系統相互搭配與合作，目前的實作是與著名的遠端桌面軟體 VNC（Virtual Network Computing）[3,4]結合，為了能更清楚的解釋整個系統的設計與實作細節，論文之後的描述將以與 VNC 遠端桌面系統搭配為主，但透過簡易的調整，我們所研發的系統即可與其他傳統的遠端桌面系統相互搭配執行。

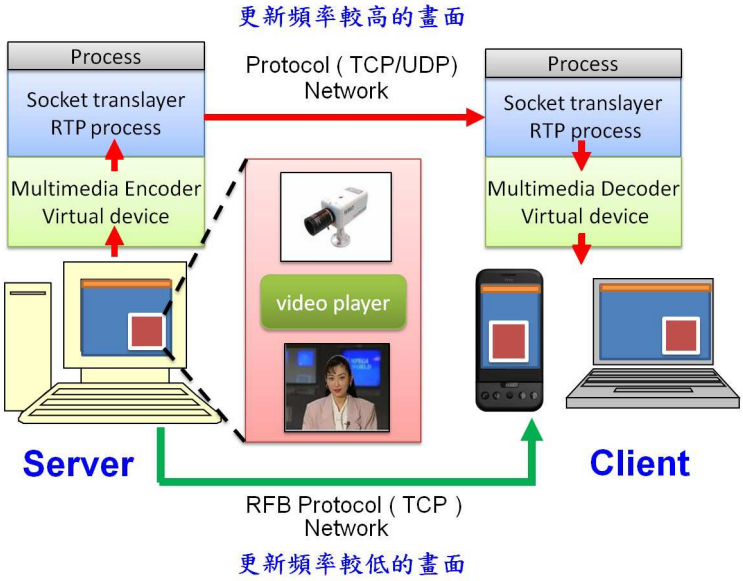

Figure 1：遠端即時多媒體桌面系統之運作概念圖。

Figure 1 顯示整個系統的運作概念圖，在 client 端部分，使用者使用 Android 系統平台，透過網路傳送指令，操控 server 端提供服務，再回傳畫面給使用者；在 server 端部分，動態影像偵測系統（Dynamic Image Detection System，以下簡稱 D.I.D.S.）會重新定義桌面上的畫面屬性，依更新頻率作為區分，更新頻率較低的畫面，利用傳統 VNC 的傳輸方式，而更新頻率較高的動態畫面被擷取出來，另以 H.264 視訊壓縮技術將其影像串流（video stream）傳輸至 client 端，解壓縮後再與 VNC client 的畫面整合，並顯示於 client 端，達到即時的影像播放。Figure 2 為系統內部之動靜態畫面偵測、擷取、與整合示意圖。

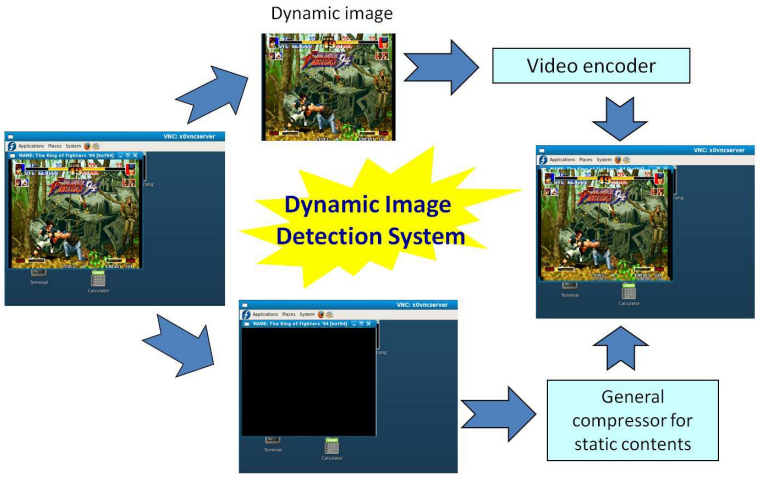

Figure 2：系統內部之動靜態畫面偵測、擷取、與整合示意圖。

二、 動態影像偵測

在偵測動態影像技術部分，目前多數的做法是採用 full search（全畫面比對找相異處，再將相異處透過 RFB protocol [12,13]傳送），此法需要較多的計算時間，同時會延遲對畫面的更新。我們的方法是在 Linux 的 X Window System 上，透過分析 X protocol 的 request，進行非 full search 的動態畫面偵測。此技術在不更動任何 X Window System 內部元件的情況下，藉由 Linux kernel 去擷取 X protocol request 中的 draw number，再存於 kernel space 的特定資料結構中，之後 user space 應用程式可藉由讀取/proc 中的檔案，得知該動態畫面的 draw number。得知動態畫面的 draw number 後，透過適當的 Xlib API，便可知該 draw panel 的座標位置與寬高大小。此時可將該資訊傳給 VNC server，告知 VNC server 該動態畫面不需要藉由 RFB protocol 傳送至 client 端做更新，而是改用 H.264 視訊壓縮技術將其影像串流（video stream）傳輸至 client 端顯示。

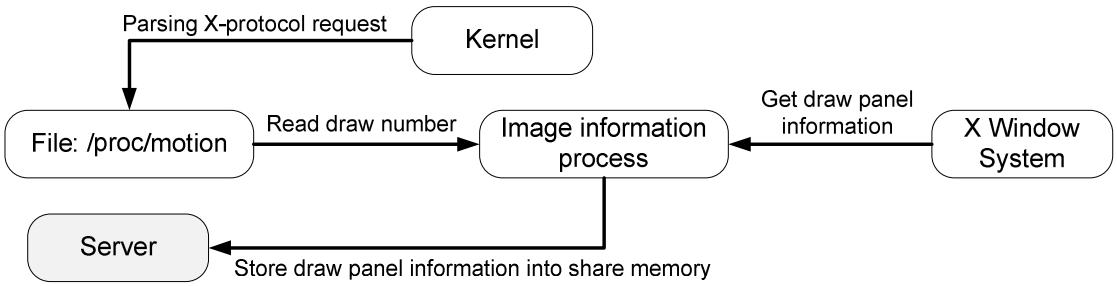

Figure 3：動態畫面偵測系統之雛型架構。

Figure 3 顯示本技術的雛型架構，從 kernel 中將 Unix domain socket 的 send socket message 攔截，再藉由 X protocol parser 擷取 draw number 並存入於/proc 中的檔案 motion 中，user space 上執行一個 Image information process 讀取 draw number，接著將相對應的 draw panel 的資訊讀出並存入 share memory，最後 server 端透過 share memory 取得 draw panel，並將畫面座標位置與寬高資訊傳給 client 端，此資訊供 client 端正確地將解壓縮後的視迅串流顯示在桌面上。

本技術在實作時，當 Image information process 透過讀取到的 draw number，跟 X Window System 索取相對應的 draw panel 的座標位置以及寬高時，如索取失敗，則 X Window System 會將此 process 的行程結束，造成 server 端無法再拿到動態畫面的座標位置以及寬高大小的資訊。為了解決這個問題，我們修改了原本的雛型架構，藉由 Monitor process 偵測行程結束時所發出的 signal，並重新產生 Image information process。Figure 4 顯示改良後的動態畫面偵測系統之架構。

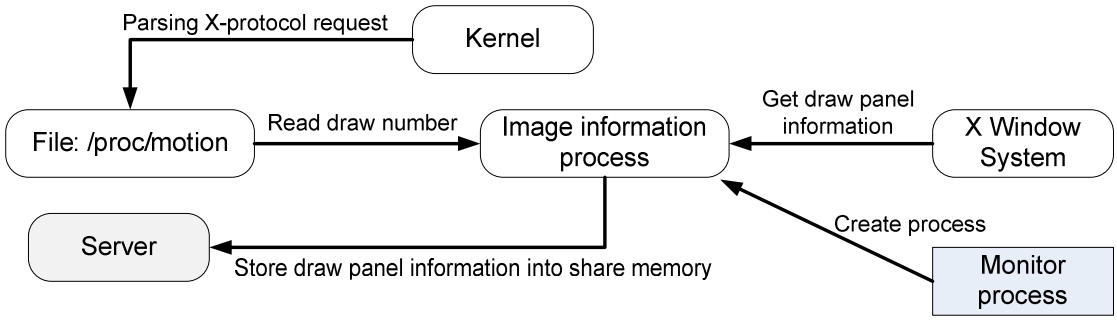

Figure 4：改良後的動態畫面偵測系統之雛形架構。

在 X Window System 裡，X client 透過遵循 X protocol 的 request 請 X server 做適當的畫面顯示，根據我們的觀察，如果執行一應用程式播放每秒 30 張 frame 的影片，則此應用程式至少會對 X server 發出 30 次 request ，適當的紀錄 request 發生的時間，則可以判斷是否為動態影像畫面。Figure 5 顯示我們用來記錄相關資訊的資料結構，該資料結構有 256 個 entries，每個 entry 有 2 個 nodes 和一個 node index，每個 node 包含了三個欄位：Draw number(4 bytes)、Jiffiess（8 bytes)、Empty flag（1 bit)，其中 Jiffies 是 Linux 核心之變數，用來記錄系統自開機以來，經過多少 tick，每發生一次 timer interrupt，則 Jiffies 的值會加一。每個 draw number 有相對應的 entry，每個 entry 裡的兩個 node 用來記錄最近兩次 request 的資訊，透過計算最近兩次 request 之 Jiffies 的差，我們就可以得知 request 發生的頻率。經由我們的觀察得知，若 frame rate 高於 10 fps（frames per second），則 VNC 遠端桌面的顯示品質就會明顯的變差，因此，我們設定 Jiffies 的差小於 0.1 秒，屬於動態影像畫面，需擷取出來，採用高階視訊壓縮技術處理；反之，則為靜態或低動態影像畫面，由傳統 VNC 的方式處理。

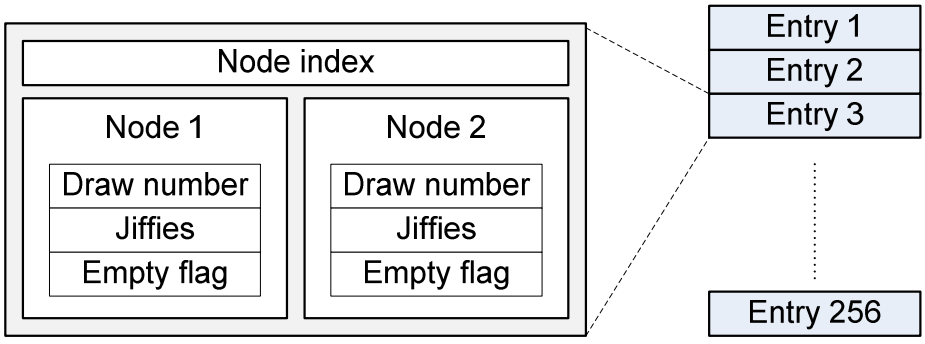

Figure 5：動態影像偵測技術之內部資料結構示意圖。

三、 server 端之設計與實作

Figure 6 顯示 server 端架構圖。server 端包含三個主要的資料處理流程：第一、網路連線管理流程，Connection manager 依據連線個數產生對應的 Connection thread 進行服務。第二、視訊壓縮管理流程，動態影像偵測的資訊透過 shared memory 機制存放在 Pool，Encoder unit 由 Pool 中取出影像並進行壓縮，最後放入 Buffer queue，再依 channel 編排管理。第三、原始 VNC server 之執行流程。此設計架構，將容許：(1) multi-connections 的功能，多個 clients 連線，提供多個使用者隨時隨地皆可操作觀看的便利性；(2) multi-channels 的功能，可偵測多動態影像、支援 camera 視訊壓縮（例如：video surveillance）；(3) multi-videos，具有使用者可隨選不同的壓縮視訊 channel 的功能。

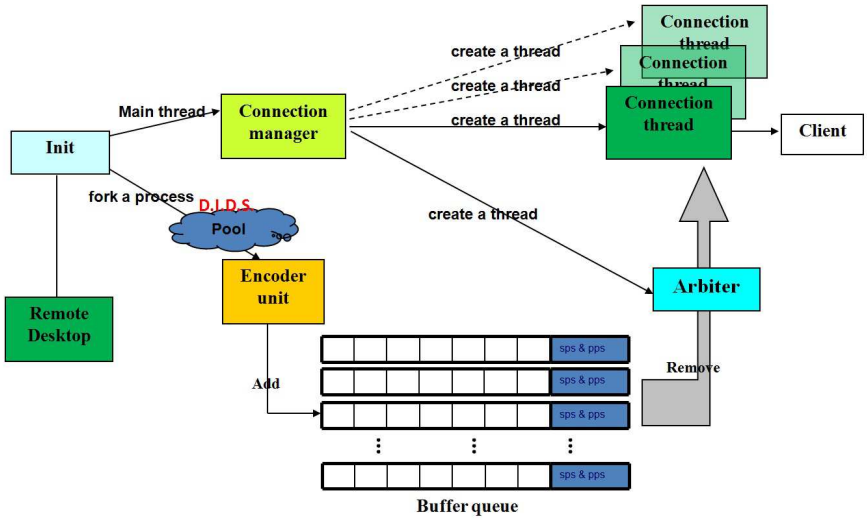

Figure 6：Server端架構圖。

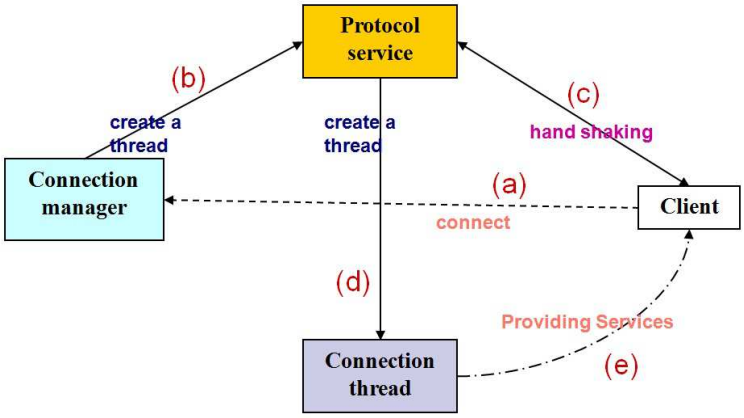

Figure 7：Connection manager與client端之連線處理流程圖。

Server 架構圖中的 Connection manger 元件，負責處理所有 client 端的連線，Figure 7 顯示連線處理時的五個步驟：(a) server 會由 Connection manger 管理所有的 client 端連線；(b)

Connection manger 會先建立新的 thread，該 thread 將執行 Protocol service 元件；(c) Protocol service 開始與 client 進行 protocol 溝通；(d)透過 Protocol service 元件得知 client 的請求後，再產生 Connection thread；(e) Connection thread 會提供給 client 一切所需的影片、視訊串流、系統資訊等服務。由以上可知，Connection manger 元件負責管理 server 端所有對外的行為，此外，為了達到良好的流量控管，我們分配給每個 client 一個特定的 port，將屬於該特定 client 所要的資料由此 port 進行傳送。

Encoder unit 主要負責動態影像的壓縮工作，透過明確的介面（interface）規範，我們將它設計成一可抽換的軟體元件，使用者可更換不同的 Encoder unit（例如：支援 H.264 或 SVC 等影像壓縮技術）以滿足不同的需求。我們目前所實作的 Encoder unit 僅支援 H.264 影像壓縮技術。Encoder unit 會將 Pool 中的動態影像進行視訊壓縮，並將 Bitstream 放入 Buffer queue 中，以 channel 區別管理，再由 Arbiter 統一取出給 Connection thread 播送視訊。當無 client 收看視訊時，Arbiter 將暫時停止，Buffer queue 中的 channel 資料便不被取走，此時 Buffer queue 中的佇列就會擺滿，一旦擺滿，則 Encoder unit 將暫停壓縮動作。因此在無 client 觀看動態影像時，此流程會全部暫停，避免浪費系統資源。

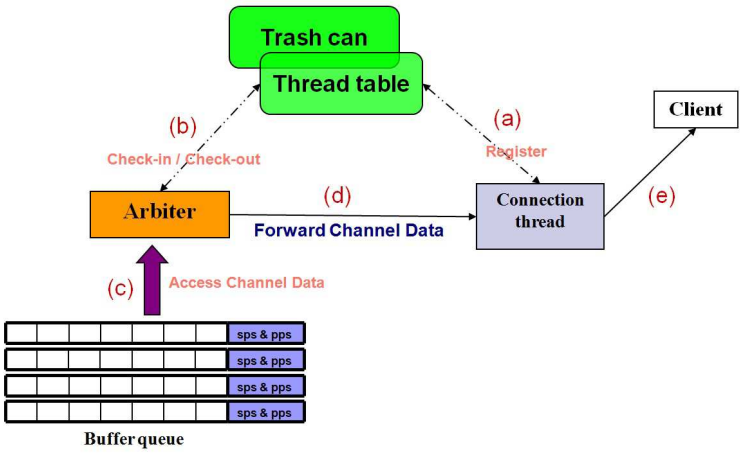

Figure 8：Arbiter之運作流程圖。

Arbiter 扮演居中協調者的角色，連結網路連線管理流程與視訊壓縮管理流程，管理所有需要視訊服務的 Connection thread，Figure 8 顯示 Arbiter 運作的五個主要步驟：(a)系統設有 Thread table 及 Trash can 的資料結構，Thread table 中記載需要服務的 Connection thread，Trash can 則是記載因中斷傳輸或結束服務的 Connection thread；(b) Arbiter 會去檢查 Thread table 及 Trash can 的資訊，進行註冊或註銷；(c)確定發送名單及影片 channel 後，Arbiter 會從 Buffer queue 中取出資料；(d) Arbiter 統一將資料傳給 Connection thread；(e)最後由 Connection thread 透過網路發送資料給 client。每個 channel 會有各自的 Thread table 和 Trash can，例如：當 Connection thread 需要收看 channel 1 時，便將自己註冊於 channel 1 的 Thread table 中，此時管理 channel 1 的 Arbiter 會依造 Thread table 的名單統一派送 data，對於新進的 Connection thread 成員，則先予以視訊初始頭檔 sps(sequence parameter set)與 pps(picture parameters set)。

四、 通訊協定設計

1. 簡介

為了讓本系統能夠有效的進行一些互動，為此制定出了符合系統需求的協定，包含了串流與畫面相關資訊的傳輸、以及建立一些連線等操作時所需的控制訊息交換。網路通訊協定（protocol）設計主軸在支援 multi-connections、multi-channels、與 multi-videos 等功能。我們將 protocol 分為兩類，分別是 control protocol 與 video protocol 的 packet format。Figure 9 為本系統的通訊協定示意圖。

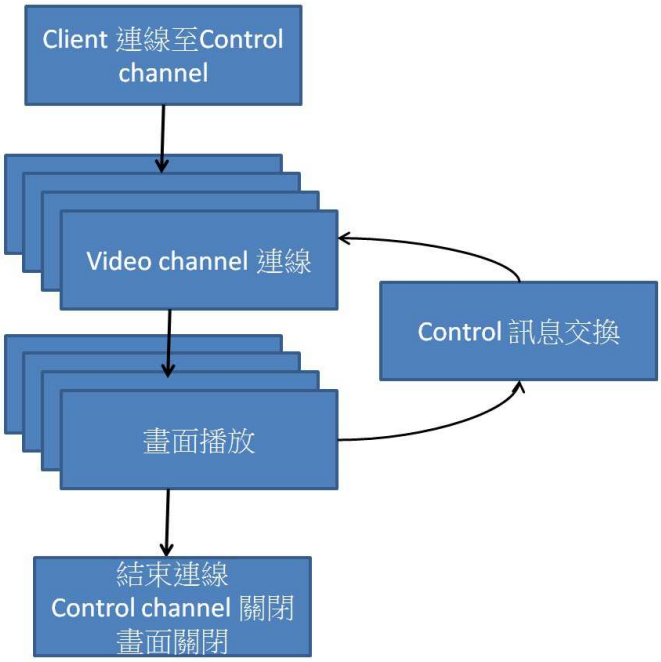

Figure 9：系統之通訊協定示意圖。

2. 名詞定義

(a) Control Message：負責進行 server 端與 client 端之間，控制資料交換與連線管理的訊息類別。

(b) Video Message：為多媒體資料相關資訊，僅 server 端至 client 端之單向的傳輸訊息類別。

3. Control Message

標頭欄位格式

Size (4 byte)	Command code (4 byte)	Content (Variable)

欄位說明：

(a) Size：告知接收端 Content 長度，單位為 byte。

(b) Command code：控制用的訊息。

(c) Content：搭配 Command 的訊息類別之後，所要傳達的參數或資料。

4. Video Message

標頭欄位格式

Series number. (4bytes)	Coded data (Variable)

欄位說明：

(a) Series number：封包傳送的序號，每經過一次傳送值就會加一，當序號達到上限時，會重新歸零計算。

(b) Coded data：畫面 XY 座標值、畫面寬高、與經過 SVC 壓縮過後的位元串流資料。

5. server 端所接收 Command code 與 Context 說明

Request 1xx：

(a) 100：請求開啟某個 Video channel 的連線，Content 為 channel 的編號。

(b) 101：請求關閉某個 Video channel 的連線，Content 為 channel 的編號。

Success 2xx：

(a) 200：成功，無需 Content。

Information retrieves 3xx：

(a) 300：client 對 server 下載可用頻寬，Content 為頻寬，單位為 kbps。

(b) 301：更改使用者觀看中某一個 channel 的解析度，Content 為 channel 編號 + 自行定義解析度(例如：0 為 QCIF、1 為 CIF…)。

(c) 302：暫停播送某一 Video channel 的串流傳送，Content 為 channel 的編號。

(d) 303：繼續某一 Video channel 的串流傳送。

6. client 端所接收 Command code 與 Context 說明

Request 1xx：

(a) 102：要求連線到指定的 Port，Content 為 Port number。

Success 2xx：

(a) 200：成功，Size 設定為零，無需 Content。

Information retrieves 3xx：

(a) 303：server 所開放的 channel 數量，Content 為畫面資訊(XY 值 + 畫面寬高)。

Error message 4xx：

(a) 400：拒絕連線，不需 Content，所以 Size 為 0。

(b) 401：server 主動發出一些即將關閉此連線的訊息至 client，Content 為原因，可由 server 自行定義說明。

7. 協定流程圖

當 client 與 server 連線之後，即為 Control channel 的連線建立完成。使用者可以藉由一些 Control Message 的交換，建立 Video channel，播放所要看的畫面。Figure 10 為本系統的通訊協定流程圖(省略 Size 參數)：

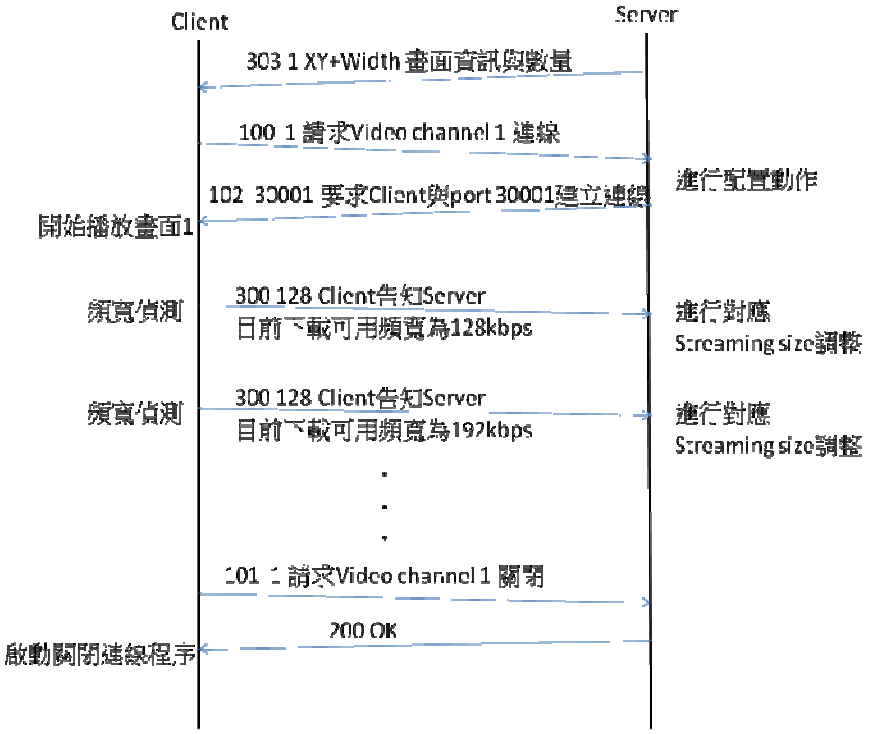

Figure 10：系統之通訊協定流程圖。

五、 client 端之設計與實作

client 端的設計以低運算複雜度、高可移植性為基本原則，整個執行流程如 Figure 11 所示，client 端開始執行後，就會建立一個新的 thread 執行傳統的遠端桌面 client 端程式，即 VNC client，原來的主 thread 會執行 Connection process 元件，Connection process 會建立新的 thread，該 thread 負責將網路收到的影像封包解壓縮，解碼後的影像再傳給 frame buffer(device node)，再由 VNC client 組合靜態畫面與動態影像後，顯示完整畫面。

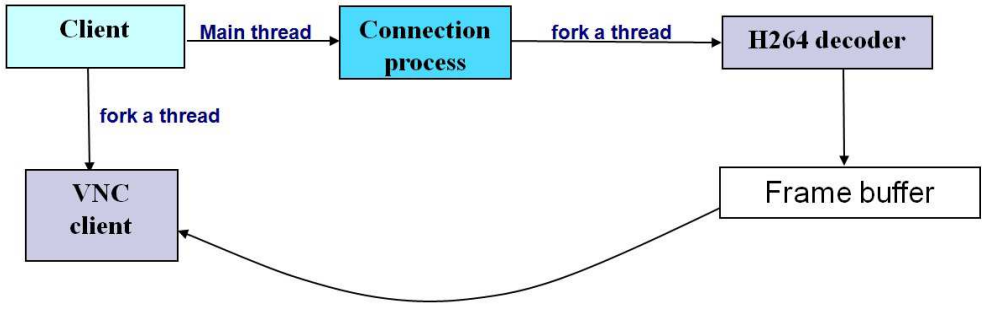

Figure 11：Client端架構圖。

參、 基於 Android 平台之 client 端設計

一、 Android 平台之 client 端的設計與實作

Figure 12：client端開啟後的設定畫面。

Figure 13：遠端電腦的畫面。

基於 Android 平台之 client 端，主要是以 VNC client 為基礎，再進行擴充與增強。client 端程式點擊進入後會要求輸入遠端電腦 IP、密碼等資料，如 Figure 12 所示。資料填妥後按下 connect 鍵就會連線至遠端桌面，如果遠端桌面有動態畫面，則需點擊動態畫面，才會切換到播放模式，即時接收由遠端電腦利用 H.264 壓縮傳送之影像串流，再將它解壓縮並顯示到螢幕上，如 Figure 14 所示。不把畫面直接顯示到如 Figure 13 的模式上，而需經過點擊才能切換到播放動態影像的主要原因，是為了要降低 CPU 的使用率，因為 H.264 之複雜度較高，解碼耗費的時間長且手機上的 CPU 效能有限，如果一面開啟 VNC client，一面解碼播放畫面，將使得系統效率低落。如此的設計模式也可以在有多個動態畫面時，每次只播放使用者想要觀看的畫面，而使得播放品質有所保證。

Figure 14：觀看動態影像時的畫面。

二、 遭遇的困難與解決方法

因為 Android 上的程式設計是以 Java 為基礎，但考量到效能問題，大多數的 decoder 都是以 C/C++撰寫，因此將 C code 寫成的 decoder 和 Android 上的 VNC 做結合是首先必須克服的問題。針對此問題，我們採用 JNI(Java Native Interface) [14]的方式，將 C 程式編譯成 library，包裝成 Java 程式可以呼叫的 interface，讓 Java 和 C 程式間能夠互相溝通，如此即可在 Android 平台上使用以 C code 寫成的 decoder 或是相關程式。

考量到手機的使用特性，我們必需要從 server 端取得相對應的座標位置和畫面大小等資訊，為方便使用者能夠直接用觸碰的方式，點選想要觀看的動態畫面，client 端必須和 server 端建立溝通連線來獲取這些訊息，但是動態畫面可能會隨時地改變位置，因此手機的 client 端也必需一直接收 server 端送來的資訊，保持更新動態畫面的資訊。此外，由於手持式裝置的 CPU 運算效能遠不如一般桌上型電腦，為了要使 Android 手機在播放動態畫面時能夠更加順暢，我們適當的設計 client 端之操作介面，讓使用者透過觸控螢幕，選擇想觀看的動態影像，並從 VNC client 模式切換至影片播放模式，讓 VNC client 轉而在背景執行，同時讓 decoder 在前景執行時使用更多 CPU 資源，可節省手機上的運算資源，確保畫面的播放品質。

肆、 實驗結果與分析

我們所使用的 server 主要硬體規格為 IBM X3400，內含兩顆 Intel Core Xeon E5405 處理器，在 Linux 作業系統下實作，kernel 版本為 2.6.23.1-42。實驗部分主要包含三大部分：動態影像偵測系統的效能分析、遠端即時多媒體桌面系統之效能分析、與 Android 平台之 client 端效能分析。

一、 動態影像偵測驗證

Figure 15 顯示受測的桌面畫面，遠端桌面上同時開啟了 VLC player、mplayer 與 xawtv 等三個應用程式，這三個應用程式同時播放著多媒體影像。我們量測原本 VNC server 傳送畫面至 VNC client 的系統負載，同時也量測加入動態影像偵測技術，將動態畫面攔截之後，系統的負載，詳細的數據如 Table 1 所示，由 Table 1 的數據可知，再加入本技術後，對其他程式並不會造成任何影響。以 VNC server 的 CPU loading 來比較，加入本技術後從原本的 CPU loading 56%減少至 33%，其原因在於動態影像的部分，已被我們的動態影像偵測所擷取，因此 VNC server 便不需要處理動態影像的部分，進而減少負擔。

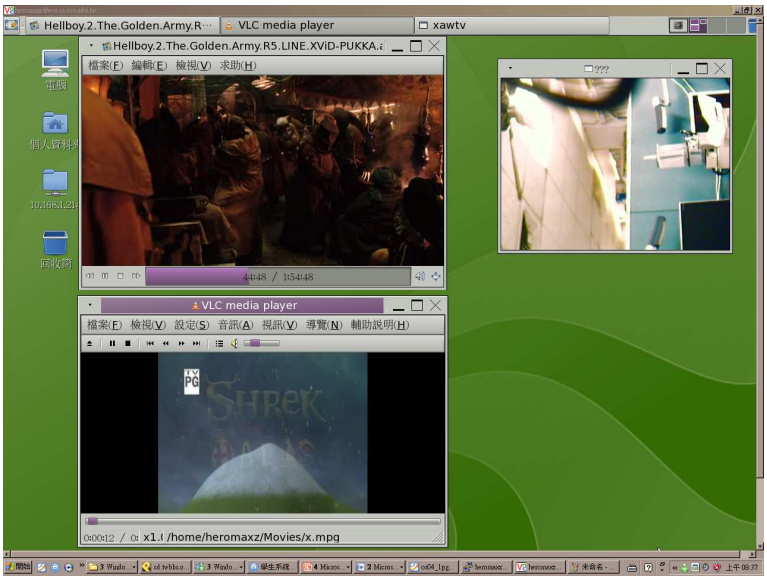

Figure 15：受測的桌面畫面。

Table 1：動態影像偵測效能比較表。

測試程式	原系統 CPU loading	加入本技術後
VNC server	56%	33%
VLC	5%	5%
mplayer	5%	5%

二、 系統效能驗證

傳統的遠端桌面系統 VNC（Virtual Network Computing）在低於 10fps 時，VNC client 畫面更新就會出現延遲現象，而決定撥放品質好壞的關鍵在於網路傳輸速率，而傳輸速率會被更新頻率與頻寬限制所影響，導致在播放影片時，無法即時播放。如使用手持式裝置（例如：Android 手機)，因網路受限於環境，往往會造成頻寬不穩定，若能在低頻寬下也能保持良好的畫面更新，則能夠保證良好的播放品質。

Table 2：頻寬限制下原 VNC 與本系統 fps 比較。

頻寬限制	原 VNC 傳輸	本系統
32 Kbytes/s	0.13 fps	22.46 fps
64 Kbytes/s	0.25 fps	28.61 fps
128 Kbytes/s	0.58 fps	28.31 fps
256 Kbytes/s	1.00 fps	28.43 fps

Table 2 為頻寬限制下原 VNC 與本系統 fps 比較，由表中可知當頻寬在 32Kbytes/s 下，原 VNC 的在傳送 CIF（352x288)畫面會降到 0.13fps，這也是為什麼在 client 端看到的畫面會品質不佳的主因；相對我們的系統仍然可以有 22.46fps 的效能呈現，這樣可以保證在頻寬不穩定或低頻寬下仍可有良好的播放品質，在其他部分的頻寬下，我們的系統平均能有 28fps 的效能，而原 VNC 則分別為 0.25fps、0.58fps、及 1.00fps。由此可見我們的系統在嚴格的頻寬限制下，仍可以有 real-time 的傳送能力，下圖 Figure 16 為 Table 2 之效能比較圖。

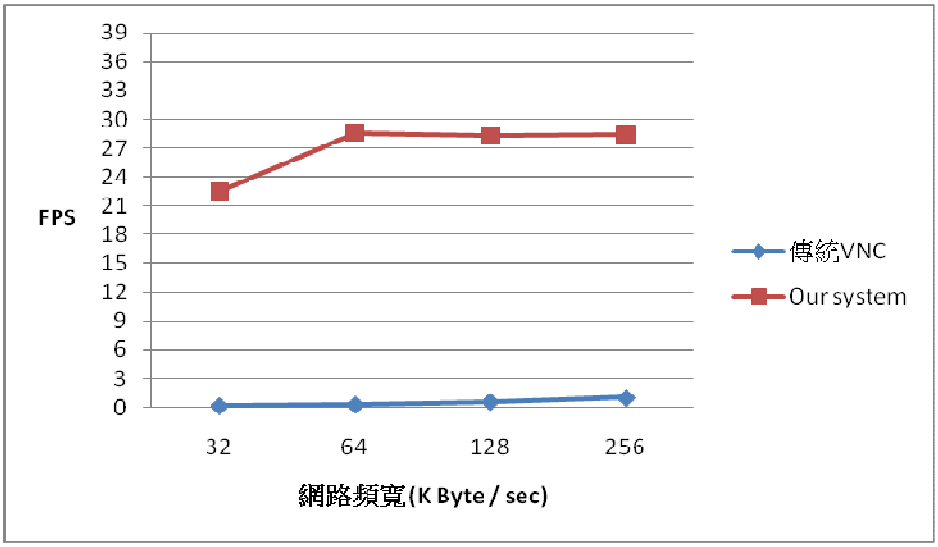

Figure 16：Table 2 之效能比較圖。

三、 Android 手機效能驗證

我們所採用之 Android 系統的主要硬體規格如下所示：

- CPU : Qualcomm QSD 8250 1 GHz
- Memory : 512MB FLASH、512MB RAM
- Network : WiFi 802.11 b/g transceiver
- Operating System : Android Mobile Technology Platform 2.1

我們實作 client 端在 Android 系統上，Table 3 為在 Android 手機平台執行的 fps 效能表。相對原本 VNC 在 128Kbytes/s 下的 0.58fps，Android 手機平台下 CIF 有 11.5fps、QCIF 則有 29fps 的效能，足以提供順暢的畫面播放品質。

Table 3：Android 手機平台 fps 效能比較。

Frame	Android
CIF (352x288)	11.5 fps
QCIF (176x144)	29 fps

伍、 結論

本論文結合嵌入式系統、多媒體、與網路等相關技術，建構一系統層級的軟體平台，使用者可隨時隨地透過支援網際網路通訊之嵌入式多媒體裝置，使用家裡、辦公室的電腦、或雲端計算中心的虛擬電腦設備，且就像在自己家裡或辦公室一樣，讓嵌入式系統與一般電腦系統的整合與溝通更緊密。

整個系統的關鍵技術在於將遠端桌面的畫面區分成動態與靜態畫面內容，動態影像偵測模組將動態畫面擷取，以 H.264 高品質影像壓縮技術壓縮，靜態畫面則經由傳統的遠端桌面傳輸模式傳送，經網路傳送到 client 端，動靜態畫面重組後即時呈現。我們將整個系統實作完成，同時也針對智慧型手機操作的特性，設計與實作了支援 Android 手機的 client 端程式，實驗結果顯示，我們的系統在頻寬限制的情況下，仍能夠提供順暢的畫面播放品質。

參考文獻

[1] Wikipedia - X Windows System. Website. Online available at http://en.wikipedia.org/wiki/X_Window_System.

[2] Robert W. Scheifler and Jim Gettys. The X window system, ACM Transactions on Graphics (TOG), 1986, Vol. 5, No. 2, 79-109.

[3] VNC. Website. Online available at http://www.realvnc.com/.

[4] T. Richardson, Q. Stafford-Fraser, K. R. Wood, and A. Hopper. Virtual Network Computing,

IEEE Internet Computing, 1998, Vol. 2, No. 1, 33-38.

[5] Android. Website. Online available at http://www.android.com/.

[6] Google. Open Source Licensing Questions - Android. Google Android Site. 2010. Online available at http://developer.android.com/guide/appendix/faq/licensingandoss.html#timeline.

[7] Spectrum Data Technologies. A Spectrum White Paper: Thoughts on Google Android, 2008. Android Developer. Website. Online available at http://developer.android.com/index.html.

[8] W. Richard Stevens, Bill Fenner and Andrew M. Rudoff. *Unix Network Programming: The Sockets Networking API*, Addison-Wesley, 2003.

[9] W. Richard Stevens. *Advanced programming in the UNIX environment*, Addison-Wesley, 1992.

[10] Joint Video Team(JVT) reference software JM 14.2.

[11] Thomas Wiegand, Gary J. Sullivan, Gisle Bjontegaard, and Ajay Luthra. Overview of the H.264 / AVC Video Coding Standard, IEEE Transactions on Circuits and Systems for Video Technology, 2003, Vol. 13, No. 7, 560-576.

[12] Tristan Richard. RFB Protocol, November 2009. Online available at http://www.realvnc.com/docs/rfbproto.pdf.

[13] S. Wenger, M. M. Hannuksela, T. Stockhammer, M. Westerlund and D. Singer. RTP Payload format for H.264, February 2005. Online available at http://tools.ietf.org/html/rfc3984.

[14] Java Native Interface. Website. Online available at http://java.sun.com/j2se/1.5.0/docs/guide/jni/index.html.