# 國立交通大學

## 網路工程研究所

## 碩 士 論 文

VoIPv6通話品質測試工具之研製

Design and Implementation of Voice Quality Testing Tool for VoIPv6 Environments

研 究 生：宋岳鑫

指導教授：林一平　博士

中 華 民 國　九十六　年　七　月

# VoIPv6通話品質測試工具之研製

# Design and Implementation of Voice Quality Testing Tool for VoIPv6 Environments


研 究 生：宋岳鑫　　Student：Yueh-Hsin Sung

指導教授：林一平　　Advisor：Yi-Bing Lin


國 立 交 通 大 學

網 路 工 程 研 究 所

碩 士 論 文

A Thesis

Submitted to Institute of Network Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

July 2007

Hsinchu, Taiwan, Republic of China

中華民國 九十六 年 七 月

# VoIPv6通話品質測試工具之研製

學生：宋岳鑫　　指導教授：林一平　博士

國立交通大學網路工程研究所碩士班

## 摘　要

VoIP (Voice over IP) 爲第三代行動通訊 IMS (IP Multimedia Subsystem) 中最重要的服務之一。在 IMS 中， VoIP 分別以 SIP (Session Initiation Protocol) 與 RTP (Real-time Transport Protocol) 協定來傳輸信令和多媒體資訊，並採用 IPv6 (IP version 6) 來提供大量的位址空間。在建置 IMS 應用服務時，需要分析器與測試工具研究相關通訊協定，並對所佈建的系統進行效能測試。本論文針對 IMS 的多媒體應用服務設計並實作 SIPv6 Analyzer ，在分析功能上提供各種通訊協定標頭的解析，並針對 SIP 對話產生 SIP 信令流程圖、播放 RTP 語音或影像串流，以及提供封包統計功能。搭配 SmartBits 硬體平台提供測試功能，具有 IPv6 通道 (Tunnel) 建立的能力，可以測試各種 IPv6 使用環境、可以發送 SIP 訊息以進行 SIP 通話建立流程，以及利用 SmartLibrary 控制 SmartBits 硬體產生多個封包串流，測試所佈建的系統在不同負載時的效能與其通話品質。

**關鍵字：** VoIP, MOS, PESQ, Voice Quality, IPv6

# Design and Implementation of Voice Quality Testing Tool for VoIPv6 Environments

Student: Yueh-Hsin Sung Advisor: Dr. Yi-Bing Lin

Institute of Network Engineering

National Chiao Tung University

## Abstract

VoIP (Voice over IP) is one of the most important services among the IMS (IP Multimedia Subsystem) services. IMS utilizes SIP (Session Initiation Protocol) and RTP (Real-time Transport Protocol) to transfer the VoIP signaling and multimedia contents. IPv6 (IP version 6) is employed in the IMS to provide large address space. This paper proposes an analysis tool referred to as SIPv6 Analyzer to investigate the IMS-related protocols during IMS service deployment and to evaluate the deployed systms. SIPv6 Analyzer dissects the protocol headers and provides SIP message flow generation and RTP audio/video stream replay. The SIPv6 Analyzer also works as a testing tool and provides IPv6 tunnel setup capbility, SIP call setup capability and multiple testing streams generated by SmartBits hardwares. Through these functions, the user can evaluate the performance and service quality of the deployed system.

**Keywords:** VoIP, MOS, PESQ, Voice Quality, IPv6

# 誌　　謝

首先誠摯地感謝指導教授林一平博士與陳懷恩博士，沒有老師的專業建議與細心指導，我無法完成此篇碩士論文。在林一平博士嚴格的指導中，我學習到了研究的方法，並從中獲得極大的助益。在陳懷恩博士的指導中，讓我獲得論文研究方向與許多專業知識。另外感謝楊偉儒博士、逄愛君博士擔任我的口試委員，並在口試中給與指正與建議，使得論文能更加完整。

感謝實驗室的同伴們，特別是：雅琳、依寰以及曉涵，謝謝你們的陪伴，讓我的研究生活更加的充實。

最後我也要感謝我的家人與我的朋友們在我研究生活中給予的鼓勵及支持。

# 目 錄

# 圖 目 錄

# 表 目 錄

# 第一章 簡介

VoIP (Voice over IP) 爲第三代行動通訊 IMS (IP Multimedia Subsystem) 中最重要的服務之一。在 IMS 中，VoIP 分別以 SIP (Session Initiation Protocol) 與 RTP (Real-time Transport Protocol) 協定來傳輸信令和多媒體資訊。此外， IMS 也採用 IPv6 (IP version 6) 來提供大量的位址空間、 QoS (Quality of Service)，以及隨插即用的功能。因此在建置 IMS 應用服務時，非常需要使用分析器來研究相關通訊協定。常見的 Ethereal 是一個開放原始碼的協定分析器，且能夠解析七百多種通訊協定標頭，但 Ethereal 並沒提供針對 IMS 分析的專屬功能。因此本論文針對 IMS 的應用服務，設計並實作出 SIPv6 Analyzer。 SIPv6 Analyzer 不但能夠解析通訊協定標頭，也能用來分析 SIP 對話、產生 SIP 信令流程圖、播放 RTP 語音或影像串流，並呈現封包統計功能，分析封包延遲的變異量。

本論文針對測試VoIP 通話品質的功能需求，搭配 SmartBits 硬體，研發 SIPv6 Analyzer 的語音品質評估模組，發送測試封包到待測系統，以評估待測系統的效能與通話品質。 SIPv6 Analyzer 的語音品質評估模組提供以下功能，使其成爲適合測試各種 VoIPv6 (Voice over IPv6) 環境之效能與通話品質之工具。 SIPv6 Analyzer 具有以下功能：

- 具有建立 IPv6 通道 (Tunnel) 之能力。
- 具有 SIP 通話建立之能力。
- 可同時產生多達 2000 個 RTP 串流。

目前已有的語音品質評估方式可分爲主觀 (Subjective) 與客觀 (Objective) 兩種。主觀的語音評估方式，目前所使用的標準爲 P.800 [1] 。藉由一個受控制的環境下，讓受測者聽取系統所播放的聲音，或是透過系統進行對話。受測者於結束後對於測試結果進行評分，將語音品質區分爲五個等級， Excellent (5) 、 Good (4) 、 Fair (3) 、 Poor (2) ，以及 Bad (1) 。藉由統計受測者所評估之分數，求得待測系統的語音品質；客觀的語音評估方式，又可分爲兩種方式，一種是分析過去主觀測試結果，經由迴歸分析，將影響語音品質的因素量化，計算各種因素的結果後得到語音品質，目前的標準爲 G.107 (E-Model) [2] ；第二種是直接對經過系統的語音之聲波進行分析，了解原本的聲波經過系統後的受損程度，評估系統之語音品質，目前的標準爲 P.862 (即 PESQ ， Perceptual Evaluation of Speech Quality) [3] 。

主觀的語音品質評估的優點，是可以直接了解使用者對於系統的語音品質滿意程度，但是進行此方式的評估，需要遵從標準所規範的測試環境，測試的過程可能相當昂貴而耗時；客觀的語音品質評估則可以迅速評估系統的語音品質。

E-Model 的評估方式中，雖然包含網路的延遲的因素，但是 E-Model 中許多影響語音品質的參數在一般 VoIP 應用中不易量測，需使用預設值設定這些參數，而且 E-Model 沒有支援所有的語音編解碼器 (CODEC) ，待測系統所使用的語音編解碼器若不在支援的清單中，便無法直接使用 E-Model 評估其語音品質；而 PESQ 是基於評估聲音訊號的受損程度計算語音品質，可以同時考量封包遺失與語音編解碼器的影響，雖然 PESQ 無法了解延遲所造成的語音品質下降，但是可以支援各種不同的語音編解碼器。因此本論文選擇 PESQ 來計算待測系統的通話品質，並提供延遲與封包遺失率做爲參考。

本論文後續的章節組織描述如下。第二章說明 SIPv6 Analyzer 的系統架構與運作範例。第三章說明 SIPv6 Analyzer 語音品質評估模組的系統架構。第四章利用 SIPv6 Analyzer 比較目前 IPv4 位址不足的解決方案之效能。第五章爲總結並提出未來方向。

# 第二章 SIPv6 Analyzer

## 2.1 SIPv6 Analyzer 系統架構

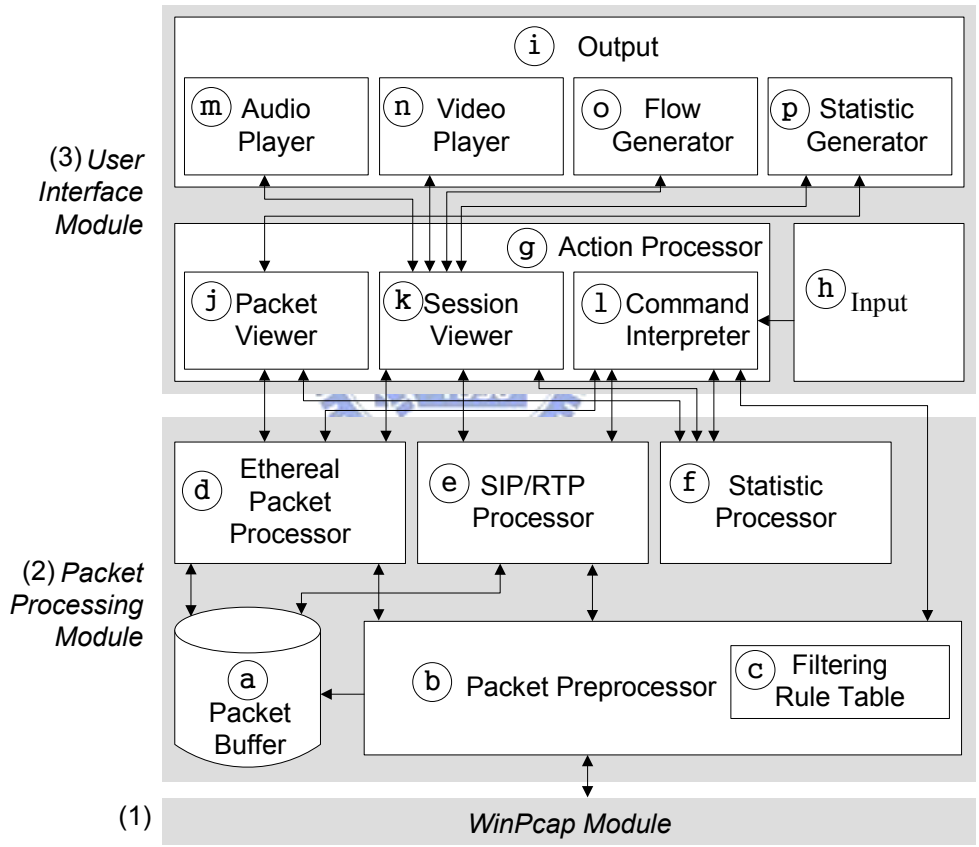



圖 2.1: SIPv6 Analyzer 系統架構圖

圖 2.1 爲 SIPv6 Analyzer 的系統架構，包含三個主要模組 (Module) 。WinPcap Module (圖 2.1 (1)) 使用開放原始碼的 WinPcap (Windows Packet Capture) 函式庫 [4] 擷取封包。 Packet Processing Module (圖 2.1 (2)) 從 WinPcap Module 取得封包、將封包分類，並且提供上層的應用程式分析系統所擷取到的封包。User Interface (UI) Module (圖 2.1 (3)) 藉由對話 (Dialog)/會議 (Session) 列表、訊息流程 (Message Flow)

、語音及影像，來展示系統所擷取到的封包之資訊。

### 2.1.1 The Packet Processing Module

Packet Processing Module 包含五個元件。當 Packet Processing Module 從 WinPcap Module 收到封包時，Packet Preprocessor (圖 2.1 ⓑ) 由封包的標頭取得 IP 位址與 通訊埠號資訊，並且將該資訊以及所收到的封包儲存進 Packet Buffer (圖 2.1 ⓐ) 。Packet Preprocessor 負責維護 Filtering Rule Table (圖 2.1 ⓒ) 並且呼叫 WinPcap Module 根據指定的過濾規則 (Filtering Rules) 過濾特定的封包。 Packet Preprocessor 將所擷取下來的封包內容，以指標 (pointer) 的方式傳遞給 Ethereal Packet Processor (圖 2.1 ⓓ) 與 SIP/RTP Processor (圖 2.1 ⓔ) 。 Ethereal Packet Processor 是使用 Ethereal 的函式庫 [5] ，來實作封包標頭資訊的解析功能。 SIP/RTP Processor 從擷取到的 IP 封包中擷取 SIP 與 RTP 的封包，並將 SIP 與 RTP 封包安排到對應的會議中。 Statistic Processor (圖 2.1 ⓖ) 紀錄收到封包的個數、 inter-arrival distribution 與 jitter 統計資料。

多數的封包分析器 (如 Ethereal 與 Sniffer) 設定過濾規則 “udp port 5060” 與 “udp port 9000” 來擷取 SIP 與 RTP 封包。然而 SIP UA (User Agent ， IMS 使用者裝備) 可能使用 TCP 傳輸 SIP 訊息。 SIP UA 也可能使用任意的通訊埠號來傳輸 SIP 訊息。因此，使用固定通訊埠的過濾規則擷取 SIP 訊息， 利用 TCP 或是彈性使用任何通訊埠號碼傳送的 SIP 訊息，將被系統忽略，而且 “錯誤的封包” 可能會被擷取下來，進行 IMS 的服務分析。

前述的問題的解決方法說明如下， SIP 訊息的第一行是 request-line 或是 status-line ，而 request-line 的最後一個字串，與 status-line 的第一個字串是使用 ASCII 字串 “SIP/2.0” 這個字串用來辨識 SIP 的版本。爲了避免誤判任何 SIP 訊息， SIP/RTP Processor 檢查 TCP 或 UDP 的承載 (Payload) ，檢查是否包含該字串。如果有的話，則可以辨識出 SIP 訊息。

同理，設定固定的過濾規則可能擷取到使用 UDP 通訊埠 9000 非 RTP 的封包，或是忽略不是使用通訊埠 9000 的 RTP 封包。此問題解決方法說明如下，因爲 RTP 的傳輸位址 (Transport Address) 藉由 SIP 訊息傳送， SIP/RTP Processor 由 SIP 訊息中 SDP [6] 所攜帶的 c 與 m 欄位，獲取正確的傳輸位址。接下來 SIP/RTP Processor 可以藉由此資訊，正確的辨認出 RTP 封包。在 RTP 封包被辨認出來後，SIP/RTP Processor 藉由 SDP 中取出的傳輸位址與 RTP 標頭資訊的 SSRC (Synchronization Source) [7] ，將其安排到相對應的 RTP 連線。

### 2.1.2 The User Interface Module

UI Module 包含三個元件， Input Component (圖 2.1 ⓗ) 接收使用者的指令並請求 Action Processor (圖 2.1 ⓖ) 執行該指令。在 Action Processor 中， Command Interpreter (圖 2.1 ①) 呼叫 Ethereal Packet Processor 、 SIP/RTP Processor 、 Statistic Processor 或是 Packet Preprocessor 執行 Input Component 下達的指令。在指令執行後， Action Processor 可由 Packet Viewer (圖 2.1 ⓙ) ，或是 Session Viewer (圖 2.1 ⓚ) ，取得封包與會議的資訊。接下來 Action Processor 呼叫 Output Component (圖 2.1 ⓘ) 將結果藉由視窗介面的多媒體工具呈現出來。例如，使用 Audio Player (圖 2.1 ⓜ) ，與 Video Player (圖 2.1 ⓝ) 播放 RTP 所攜帶的語音及影像。 Flow Generator (圖 2.1 ⓞ) 藉由封包的 IP 位址或是 SIP 訊息的標頭欄位，畫出 SIP 訊息流程 (SIP Message Flow) 。 Statistic Generator (圖 2.1 ⓟ) 使用表格、樹狀圖 (Tree-View) 或是長條圖 (Bar Chart) 呈現與封包相關的統計資料。

藉由以上的元件， UI Module 提供了以下幾個方便的功能。

- 因爲不同會議中的 SIP 訊息會互相交錯， UI Module 從 SIP/RTP Processor 獲取 SIP 對話，並將 SIP 對話列成表格 (如圖 2.6 ⓐ) 。藉由此功能，使用者很容易地觀察這些在 SIP dialods 中，經過排列的 SIP 訊息。而不是從互相交錯的訊息中進行閱讀。

- UI Module 可藉由 IP 位址或是 SIP 標頭欄位，爲每個 SIP dialog ，產生一個 SIP 訊息流程 (如圖 2.6 ⓑ) 。藉由此功能，使用者可以辨識出 SIP 信令傳輸路徑上所

經過的所有節點。

- UI Module 支援 H.263 的影像編解碼器，以及 G.711u 、 G.711a 與 GSM 三種語音編解碼器。因此使用者可以利用此功能，播放使用者所選擇 RTP 連線 (圖 2.6 ⓒ) 所攜帶的語音或影像。

## 2.2 SIP/RTP Processor

本節說明 SIPv6 Analyzer 中最主要的元件， SIP/RTP Processor 。

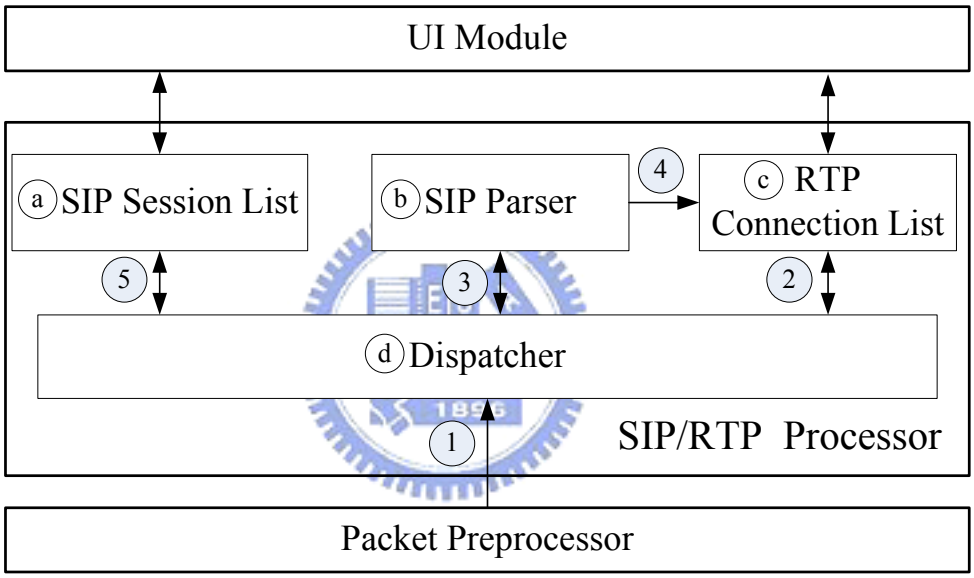



圖 2.2: SIP/RTP Processor

如圖 2.2 所示， SIP/RTP Processor 包含 SIP Session List (圖 2.2 ⓐ) 、 SIP Parser (圖 2.2 ⓑ) 、 RTP Connection List (圖 2.2 ⓒ) 與 Dispatcher (圖 2.2 ⓓ) 。

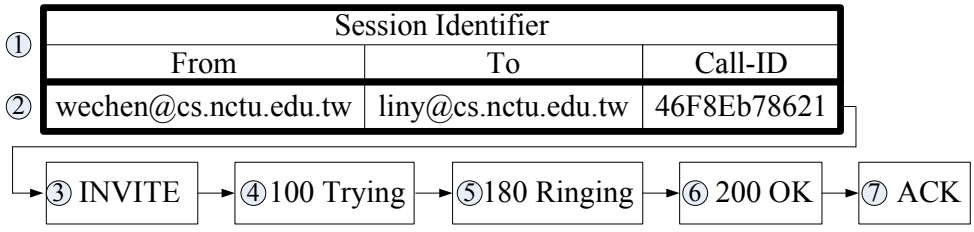



圖 2.3: SIP Session List 範例

SIP Session List 利用鏈結串列 (Linked-List) 的結構儲存 SIP 會議資訊 (參考圖 2.3 )。SIP Session List 中每筆資料包含 Session Identifier (圖 2.3 ①) ，用來標示 SIP 會議，並且使用一個鏈結串列，儲存所有與此會議關聯的封包指標。 一個 SIP 會議的 Session Identifier 包含 SIP 標頭的 From 、 To 與 Call-ID 欄位。

SIP 通話建立的交易 (Transaction) 包含一個 INVITE 訊息 (圖 2.3 ③) ，可能有零個以上的暫時 (Provisional) 訊息 (如 100 Trying 訊息，與 180 Ringing 訊息，參考圖 2.3 ④ 與 ⑤) 、一個 200 OK 訊息 (圖 2.3 ⑥) ，與一個 ACK 訊息 (圖 2.3 ⑦) 。在此交易中，INVITE 與 200 OK 訊息的 SDP ，其 c 與 m 欄位都攜帶了目的傳輸位址。當通話建立的交易完成後，通話雙方間的 RTP 連線即被建立。

| | SRC IP | DST IP | SRC Port | DST Port | SSRC | |
|---|---|---|---|---|---|---|
| ① | 3ffe:3600:1::1 | 3ffe:3600:1::2 | 9002 | 9000 | 29696 | ⑤ RTP |
| ② | 3ffe:3600:1::2 | 3ffe:3600:1::1 | 9002 | 9000 | 22607 | ⑥ RTP |
| ③ | 3ffe:3600:1::3 | 3ffe:3600:1::4 | 9002 | 9000 | 25002 | RTP |
| ④ | 3ffe:3600:1::4 | 3ffe:3600:1::3 | 9000 | 9002 | 23141 | RTP |

(a) 接收 RTP 封包後 RTP Connection List 的內容

| | SRC IP | DST IP | SRC Port | DST Port | SSRC |
|---|---|---|---|---|---|
| ① | 3ffe:3600:1::1 | 3ffe:3600:1::2 | * | 9000 | * |
| ② | 3ffe:3600:1::2 | 3ffe:3600:1::1 | * | 9000 | * |

(b) 處理 SIP 200 OK 訊息候 RTP Connection List 的內容

圖 2.4: RTP Connection List 範例

一個 RTP 會議包含兩個單向的連線， RTP Connection List (圖 2.4 (a)) 將 RTP 封包對映到其所屬的 RTP 連線。 RTP Connection List 中每個連線資料包含來源 IP 位址 (SRC IP) 、 目的 IP 位址 (DST IP) 、 來源通訊埠 (SRC Port) 、 目的通訊埠 (DST Port) 、 SSRC 欄位，與一個儲存該 RTP 連線的封包之指標的鏈結串鏈。 來源 IP 位址 、 目的 IP 位址，與目的通訊埠用來檢查一個封包是否爲 RTP 封包。利用 RTP Session List 中的全部五個欄位，可以用來將 RTP 封包分類至其所屬的連線。

在 Internet-Drafts [8] 中， RTP 的傳輸方式定義有兩種，非對稱式 (Asymmetric) 與對稱式 (Symmetric) 。圖 2.4 (a) 的第一筆與第二筆資料代表一對非對稱式的 RTP 會議，而第三筆與第四筆資料代表一對對稱式的 RTP 會議。在非對稱連線的情形

下， SIP UA 使用一個通訊埠 (如 9000) 來接受 RTP 封包，而另外使用一個通訊埠 (如 9002) 來傳送 RTP 封包。對稱式的連線的情形下， SIP UA 使用相同的通訊埠 (如 9000) 來傳送與接收 RTP 封包。

SIP Parser (圖 2.2 ⓑ) 只憑 SIP 訊息，無法判定一個 RTP 連線是否爲非對稱式或是對稱式。 因此在 RTP 連線的第一個封包被系統擷取下來之前， SIP Parser 將 RTP Connection List 的 SRC Port 欄位，標示爲 '*' 在 SRC Port 的欄位中 ( 參考圖 2.4 (b) 第一筆與第二筆資料 ) ，代表來源通訊埠號未明，並且應該從該 RTP 連線第一筆收到的 RTP 封包中取出。 同理， SIP 訊息不包含 SSRC 值， SSRC 欄位也標示爲 '*' 。在收到 RTP 連線中第一筆封包時， Dispatcher 將這些值填入 RTP Connection List 的 SRC Port 與 SSRC 欄位。

考慮以下的 SIP/RTP 封包的處理流程範例。假設發話方 (wechen@cs.nctu.edu.tw) 的 IP 位址爲 3ffe:3600:1::1 ，而受話方 (liny@csie.nctu.edu.tw) 的 IP 位址爲 3ffe:3600:1::2 。 通話雙方皆使用通訊埠 5060 收送 SIP 訊息，利用通訊埠 9000 接收 RTP 封包，通訊埠 9002 傳送 RTP 封包。在 SIP/RTP Processor 初始化後， SIP Session List (圖 2.2 ⓐ)與 RTP Connection List (圖 2.2 ⓑ) 爲空的。 INVITE 訊息的處理方式說明如下：

**步驟 1.1** (圖 2.2 ①) 在收到 INVITE 訊息時， Packet Preprocessor 將封包指標傳給 Dispatcher 。

**步驟 1.2** (圖 2.2 ②) Dispatcher 從封包的 IP 與 UDP 標頭獲得來源 IP 位址 (3ffe:3600:1::1) 、 目的 IP 位址 (3ffe:3600:1::2) 、 來源通訊埠號 (5060) ，與目的通訊埠號 (5060) 。 此資訊用來搜尋 RTP Connection List 。 Dispatcher 若未發現相符的資料，則認爲此封包並非 RTP 封包。

**步驟 1.3** (圖 2.2 ③) Dispatcher 將封包指標傳給 SIP Parser ， SIP Parser 檢驗發現此封包攜帶 SIP 訊息 (如 INVITE 訊息) 。 SIP Parser 接下來產生 Session Identifier 藉由 合併 SIP 訊息的 From (即 wechen@cs.nctu.edu.tw) 、 To (即 liny@cs.nctu.edu.tw) 與 Call-ID 標頭欄位。 爲了獲得發話端的 RTP 傳輸位址，

SIP Parser 由 SDP 的 c 欄位獲得 IP 位址 3ffe:3600:1::1 ，由 m 欄位獲得通訊埠號 9000 。此時，通話建立的交易並未完成。 SIP Parser 暫時紀錄傳輸位址，並於步驟 2.4 存入 RTP Connection List 。

**步驟 1.4** (圖 2.2 ③) SIP Parser 回傳 Session Identifier 的資訊，包含 From 、 To 與 Call-ID 欄位 (即 {wechen@cs.nctu.edu.tw} {liny@cs.nctu.edu.tw} {46F8Eb78621}) 給 Dispatcher 。

**步驟 1.5** (圖 2.2 ⑥) Dispatcher 使用此標識符 (Identifier) 搜尋 SIP Session List 。 因爲此封包屬於一個新的 SIP 會議，不會找到相符的會議。因此，一筆新的 SIP 會議資料被建立，結果如圖 2.3 ② ，而 INVITE 訊息的指標被連接至此 SIP Session 的鏈結串列 (圖 2.3 ③) 。

此時，新的 SIP Session 已經被建立，而 SIP/RTP Processor 繼續處理下一個封包指標。接下來的 SIP 暫時訊息 (即 100 Trying 訊息，與 180 Ringing 訊息) ，依照步驟 1.1 到 1.5 處理後，將被連接在 SIP Session List (圖 2.3 ④ 與 ⑤) 。 假設回應該 INVITE 訊息的 200 OK 訊息被 Packet Preprocessor 擷取下來， SIP/RTP Processor 進行以下動作。

**步驟 2.1** 與 **2.2** 同步驟 1.1 與 步驟 1.2 。

**步驟 2.3** (圖 2.2 ③) SIP Parser 偵測到此封包帶有 SIP 訊息 。 SIP Parser接著從 SDP 中的 c 欄位取得 IP 位址 3ffe:3600:1::2 ，由 m 欄位取得通訊埠號 9000 ，組成受話方的傳輸位址。

此時， SIP Parser 已經取得 RTP 連線之傳輸位址。發話方的傳輸位址 [3ffe:3600:1::1]:9000 在步驟 1.3 獲得。受話方的傳輸位址 [3ffe:3600:1::2]:9000 在步驟 2.3 獲得。

**步驟 2.4** (圖 2.2 ④) SIP Parser 將 IP 位址與通訊埠號碼填入 RTP Connection List (如圖 2.4 (b) 第一筆與第二筆資料) ， IP 位址 3ffe:3600:1::1 被填入第一筆資料的

SRC IP 欄位與第二筆資料的 DST IP 欄位。 IP 位址 3ffe:3600:1::2 被填入第二筆資料的 SRC IP 欄位與第一筆資料的 DSP IP 欄位。通訊埠 9000 與 '*' 標記，分別被填入第一筆資料與第二筆資料的 DST Port 與 SRC Port 欄位。 '*' 標記被填入第一筆與第二筆資料的 SSRC 欄位 。

**步驟 2.5** (圖 2.2 ③) SIP Parser 回傳 Session Identifier 給 Dispatcher 。

**步驟 2.6** (圖 2.2 ⑤) Dispatcher 使用 Session Identifier 搜尋 SIP Session List 。此時發現一個相符的會議 (於步驟 1.5 時建立) 。 Dispatcher 接著把 200 OK 訊息的指標連接到相符的 SIP 會議 (圖 2.3 ⑥)的鏈結串列。

此時， SIP 200 OK 訊息的處理已經完成。對映此 SIP 200 OK 訊息的 SIP ACK 訊息，依照步驟 1.1 到 1.5 ，被連接到 SIP Session List (圖 2.3 ⑦)。假設發話方開始發送 RTP 封包到受話方。第一個 RTP 封包被收到後，SIP/RTP Processor 進行下列步驟。

**步驟 3.1** 與步驟 1.1 相同。

**步驟 3.2** (圖 2.2 ②) Dispatcher 搜尋 RTP Connection List 並且找尋相符的資料 (即圖 2.4 (b) 的第一筆資料，於步驟 2.4 產生)。這筆資料中來源通訊埠爲 '*' ，同時 SSRC 的值也是 '*' 。 Dispatcher 由 RTP 封包的標頭中取出 SSRC 值 (即 22340) ，接著將 RTP Connection List 的 Src Port 與 SSRC 欄位之值，分別以 '9002' 與 '22340' 取代。被修改後的 RTP Connection List 如圖 2.4 (a) ① 。 Dispatcher 接著將封包指標連接到相符資料 (圖 2.4 (a) ⑤) 的鏈結串列。

**步驟 3.3** (圖 2.2 ②) 在 Dispatcher 成功儲存封包指標於 RTP Connection List 後， Dispatcher 繼續處理下一個封包指標。

在收到受話方送出的 RTP 封包後，SIP/RTP Processor 執行步驟 3.1 到 3.3 。在 RTP 封包被處理後，可以得到圖 2.4 (b) ② 的來源通訊埠號與 SSRC 值欄位，如圖2.4 (b) ② 所示。 RTP 封包便可連接在 RTP Connection List (圖 2.4 (a) ⑥) 。

在處理封包指標的過程中，如果 Dispatcher 無法在 RTP Connection List 找到相符的資料，並且SIP Parser 指出此封包並未攜帶 SIP 訊息，則SIP/RTP Processor 丟棄此封包指標，並且繼續處理下一個封包。

## 2.3 SIPv6 Analyzer 展示

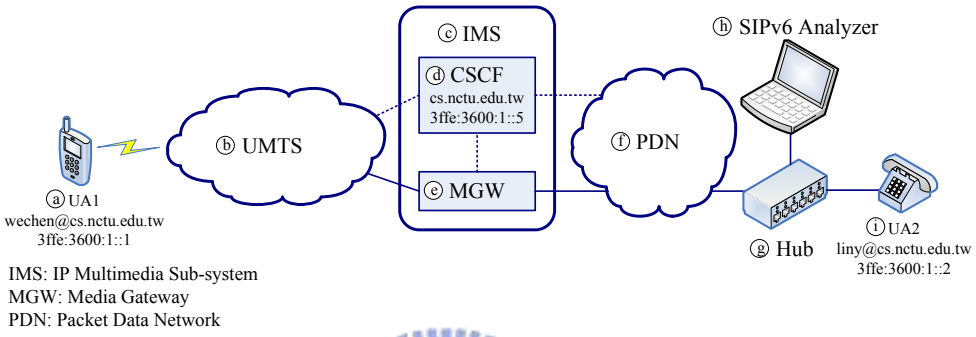



圖 2.5: IMS 環境下的 VoIP 通話範例

圖 2.5 爲展示 SIPv6 Analyzer 功能的 IMS 環境， UA1 (圖 2.5 ⓐ) 爲發話方， UA2 (圖 2.5 ⓘ) 爲受話方， IP 位址分別爲 3ffe:3600:1::1 與 3ffe:3600:1::2 。兩個 UA 使用 UMTS (Universal Mobile Telecommunications System ，圖 2.5 ⓑ) 、 IMS (圖 2.5 ⓒ) 與 PDN (Packet Data Network ，圖 2.5 ⓕ) 互相溝通。 UA1 與 UA2 分別註冊帳號 wechen@cs.nctu.edu.tw 與 liny@cs.nctu.edu.tw 於 IMS 網路中的 CSCF (Call Session Control Function ，圖 2.5 ⓓ) 在通話建立完成後， RTP 封包藉由 MGW (Media Gateway ，圖 2.5 ⓔ) 傳輸。 SIPv6 Analyzer (圖 2.5 ⓗ) 可以連接於 IMS 或是 PDN ，以分析 SIP 與 RTP 封包。在此展示中， SIPv6 Analyzer 與 UA2 藉由一個集線器 (Hub ，如圖 2.5 ⓖ) 連接於 PDN 。

SIPv6 Analyzer 透過集線器擷取所有送往 UA2 或由 UA2 送出的封包，依據步驟 1.1 到 1.5 與步驟 2.1 到 2.5 處理 SIP 訊息，並根據步驟 3.1 到 3.5 分類 RTP 封包到其所屬的會議。在擷取完成後，使用者可以分析 SIP 訊息流程與播放多媒體內容。

使用者可以藉由選擇 “Draw Message Flow” 或是 “Draw Message Flow From Head-

ers” ，要求 SIPv6 Analyzer 畫出 SIP 訊息流程。若選擇使用第一個選項 (即 Draw Message Flow) ， SIPv6 Analyzer 利用來源與目的 IP 位址畫出 SIP 訊息流程。如果選擇第二個選項 (即 Draw Message Flow from Headers) ， SIPv6 Analyzer 利用 SIP 訊息的標頭欄位 (即 SIP 的 Via 或 Route 標頭欄位) 畫出 SIP 訊息流程。圖 2.6 ⓑ 展示出使用 SIP Via 標頭欄位所畫出的 SIP 訊息流程。此訊息流程中，虛線代表 SIP 訊息傳遞路徑，虛線使用 SIP Via 標頭欄位產生。例如 SIPv6 Analyzer 所擷取的 SIP INVITE 訊息中的 Via 欄位， Via 欄位中包含兩個 URI (Uniform Resource Identifier [6]) – 3ffe:3600:1::1 與 3ffe:3600:1::5 。SIPv6 Analyzer 利用前述資訊，從 3ffe:3600:1::1 (即 UA1) 畫一個虛線到 3ffe:3600:1::5 (即 CSCF) 。藉由此功能，SIPv6 Analyzer 可以顯示 UA1 與 CSCF 之間的訊息交換，即使 SIPv6 Analyzer 並未與此兩個節點直接相連。

爲了分析擷取到的 RTP 封包，使用者可以選擇 “RTP Viewer” 子頁面 (圖 2.6 ③) ，啓用 Statistic Generator (圖 2.1 ⓟ) 以用來顯示 RTP Session List (圖 2.6 ④) 。如要播放語音或影像，使用者將該筆資料(圖 2.6 ④) 加入 Media Instance (圖 2.6 ⑦) 。接下來使用者可以點選控制按鈕 (即 Play/Hold/Stop 按鈕，如圖 2.6 ⑧) ，播放 RTP 封包所攜帶的內容。圖 2.6 ⑤ 顯示 Video Player (圖 2.1 ⓝ) 所播放的影像。藉由此功能，使用者可以評估 RTP 封包所攜帶的語音/影像的品質。

如要評估在不同 Jitter Buffer 設定下的語音/影像品質，使用者可以點選 Configuration 按鈕 (圖 2.6 ②) ，設定 Jitter Buffer 的長度 (如圖 2.6 ⑥) 。 RTP Viewer 的狀態列 (圖 2.6 ⑨) 顯示目前的 Jitter Buffer 長度 (即 100 ms) 、被丟棄的封包個數 (即 0 個) ，以及所選擇的 RTP 連線之平均 Jitter 值 (即 60.09 ms)。增長 Jitter Buffer 的長度可以減少被丟棄的封包，但會增加通訊的延遲。使用者可以使用此功能，觀察不同 Jitter Buffer 長度的影響。

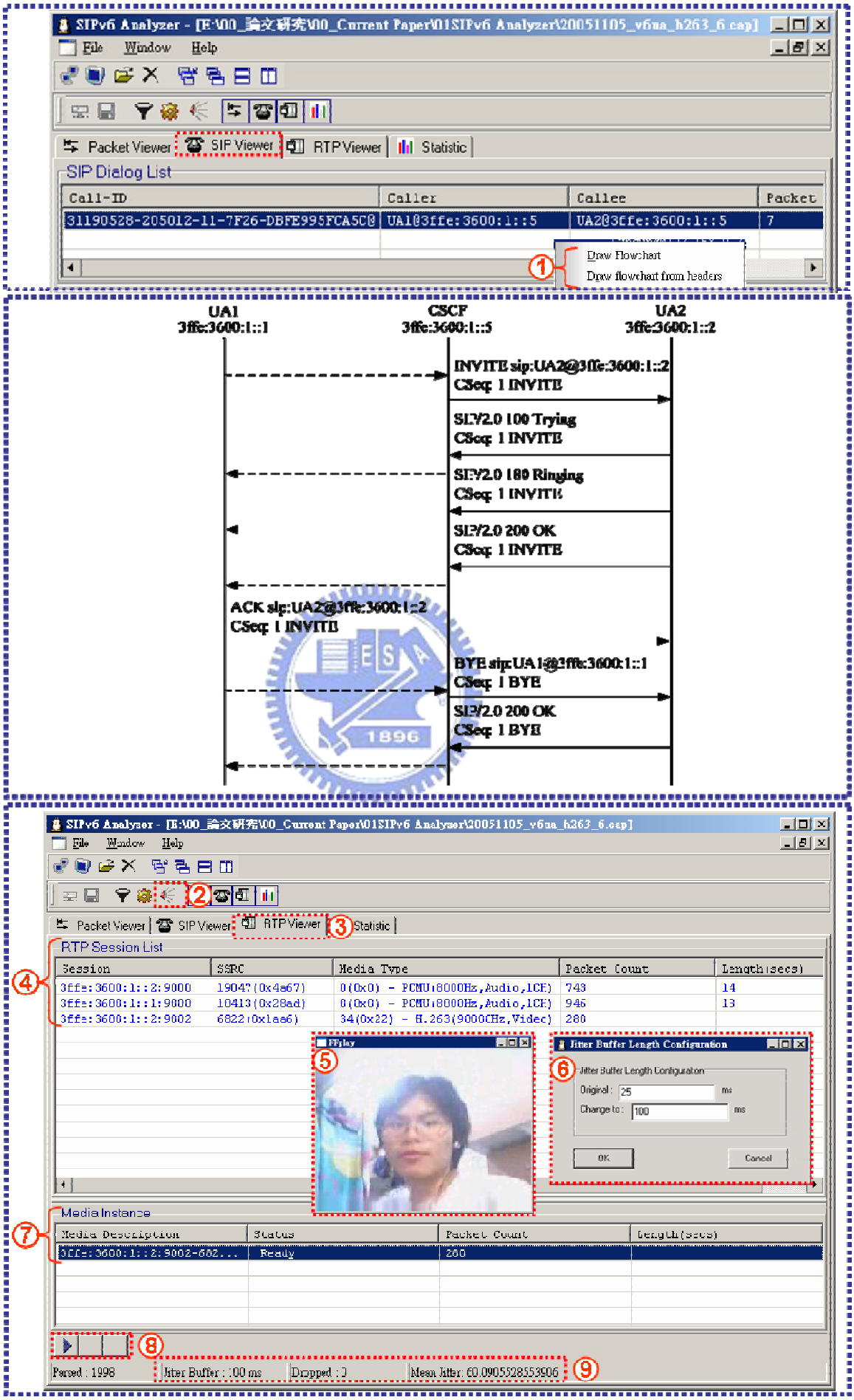



圖 2.6: SIP Viewer 、 SIP 訊息流程以及 RTP Viewer 之螢幕快照

# 第三章 SIPv6 Analyzer MOS 評估模組系統架構

## 3.1 SIPv6 Analyzer MOS 評估模組系統架構

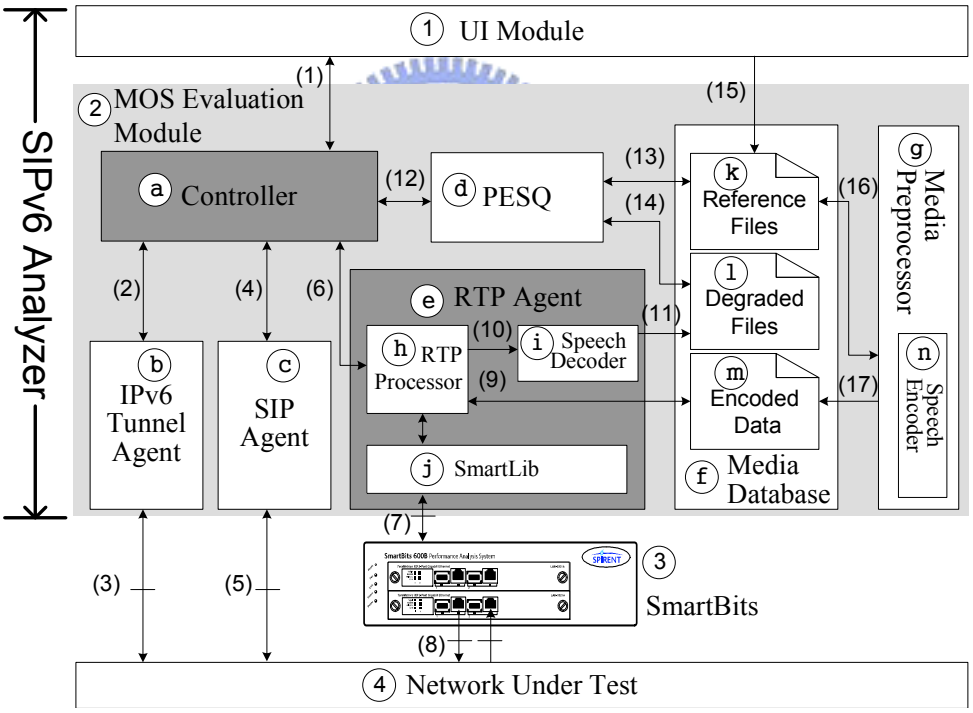



圖 3.1: SIPv6 Analyzer 系統架構圖

圖 3.1 爲 SIPv6 Analyzer 的 MOS 評估模組 (MOS Evaluation Module) 之架構圖，同時表示 SIPv6 Analzer 與待測網路 (Network Under Test ，圖 3.1 ④)的關係。 使用者介面 (User Interface Module ，圖 3.1 ①) ，爲 SIPv6 Analyzer 的使用者介面。 MOS 評估模組 (圖 3.1 ②) 爲本論文所要實作的系統，搭配 SmartBits (圖 3.1 ③) 對待測網路進行測試。

MOS 評估模組主要包含七個元件： 控制器 (Controller ，圖 3.1 ⓐ) 、 IPv6 通道代理人 (IPv6 Tunnel Agent ，圖 3.1 ⓑ) 、 SIP 代理人 (SIP Agent，圖 3.1 ⓒ) 、 PESQ (圖 3.1 ⓓ) 、 RTP 代理人 (RTP Agent，圖 3.1 ⓔ) 、 媒體資料庫 (Media Database ，圖 3.1 ⓕ) ，以及媒體前置處理器 (Media Preprocessor ，圖 3.1 ⓖ) 。接下來將介紹各個元件。

### 3.1.1 控制器

控制器 (圖 3.1 ⓐ) 負責控制 MOS 評估模組中的 IPv6 通道代理人(圖 3.1 ⓑ) 、 SIP 代理人(圖 3.1 ⓒ) 、 PESQ (圖 3.1 ⓓ) 與 RTP 代理人(圖 3.1 ⓔ) ，控制器接受 UI 所下達的指令，並將指令轉換爲控制各種元件的指令與參數。

**介面 (1)** (圖 3.1 (1)) 爲控制器與 UI Module 之間的介面，從 UI Module 傳遞的參數，共有 5 個依序爲， UAC 與 UAS 所使用的連線型態(Pure IPv6/IPv6-in-IPv4 通道/IPv6-in-IPv4 UDP 通道/IPv4) 、 UAC 與 UAS 使用的 IP 位址、測試時所使用的語音編解碼器 (G.729/iLBC/G.711u) 、 UAC 與 UAS 的 SIP URIs ，以及 SIP Server 的位址。控制器經由此介面回傳最後的結果給 UI Module 。

**介面 (2)** (圖 3.1 (2)) 爲控制器與 IPv6 通道代理人之間的介面，若 UAC 或 UAS 使用 IPv6 通道，控制器將傳遞呼叫 IPv6 通道代理人，來建立 IPv6 通道。參數包含使用何種方式建立 IPv6 通道，例如 TSP(Tunnel Setup Protocol) [9] 或 Teredo [10] ，以及 UAC 與 UAS 的 IPv4 位址。

**介面 (4)** (圖 3.1 (4)) 爲控制器與 SIP 代理人之間的介面，由控制器所傳遞的參數爲 UAC 與 UAS 的 SIP URIs 、 SIP Server 的位址，以及 UAC 與 UAS 所使用的 IP 位址。

**介面 (6)** (圖 3.1 (6)) 爲控制器與 RTP 代理人之間的介面，由控制器所傳遞的參數包含本次測試封包所使用的語音編解碼器 (G.711u) ， UAC 與 UAS 的傳輸位址， UAC 與 UAS 所使用的連線型態 (Pure IPv6/IPv6-in-IPv4 通道/IPv6-in-IPv4

UDP 通道/IPv4) ，使用 IPv6-in-IPv4 與 IPv6-in-IPv4 UDP 通道時，控制器在 UAS 的連線型態的附加參數指定 Tunnel Server 的 IPv4 位址以及 UAS 的 IPv4 位址。

**介面 (12)** (圖 3.1 (12)) ，控制器利用介面 (12) 依序將語音參考檔案與劣化品質檔案之檔案名稱輸入 PESQ ，並獲得回傳的 MOS 值。最後控制器將這次測試中得到的所有 MOS 值取平均，作爲最後輸出的 MOS 值。

### 3.1.2 媒體前置處理器與媒體資料庫

媒體前置處理器 (圖 3.1 ⓖ) 與媒體資料庫 (圖 3.1 ⓕ) ，爲兩個緊密合作的元件。媒體資料庫擁有三個資料表。 Reference Files (圖 3.1 ⓚ) 用來儲存語音參考檔，本系統所使用的語音參考檔案爲 ITU-T Recommendation P.862 [3] 所附的語音參考檔案。 Degraded Files (圖 3.1 ⓛ) 儲存每次測試後產生的劣化品質檔，與 Encoded Data (圖 3.1 ⓜ) 儲存編碼後之語音參考資料。 Reference Files 資料表會記錄每個語音參考檔案的摘要資料，摘要資料包含每個語音參考檔案的格式、取樣頻率、取樣資料大小，以及資料表更新旗標。語音參考檔的格式可以是 raw 或是 wav 檔。 raw 檔爲不含檔案標頭之語音參考檔案，使用者於輸入時必須指定其取樣頻率，與取樣資料之大小。 wav 檔爲包含檔案標頭之語音參考檔案，標頭內含有取樣頻率及取樣資料大小的資訊。資料更新旗標代表一個語音參考檔是否爲新加入的語音參考檔，當使用者從 UI Module 新增語音參考檔時， Reference Files 資料表摘要資料中會標示此語音參考檔的資料更新旗標爲 “dirty” ，代表需要進行編碼。 Speech Encoder (圖 3.1 ⓝ) 會根據此其標對語音參考檔案進行編碼，所支援的語音編解碼器有 G.729 、 iLBC 與 G.711u ， G.711u 的程式碼取自於 [11] ， iLBC 的程式碼取自於 [12] ， G.729 的程式碼取自於 [13] 。

由於每次測試時如果都要對語音參考檔案進行編碼，會比較沒有效率，因此媒體前置處理器在系統啓動時，經由以下步驟，避免每次測試都要對語音參考檔案進行編碼：

**步驟 1.1** (圖 3.1 (16)) 媒體前置處理器查詢媒體資料庫的 Reference Files 資料表，檢

查語音參考檔的摘要資料中的更新旗標。若旗標爲 “dirty” ，則媒體前置處理器將從 Reference Files 資料表，取得語音參考檔， Speech Encoder 根據摘要資料中的檔案格式、取樣頻率，以及取樣資料大小，對語音參考檔進行編碼。

**步驟 1.2** (圖 3.1 (17)) Speech Encoder 將編碼後資料存入 Encoded Data 中 。

**步驟 1.3** (圖 3.1 (16)) 媒體前置處理器更新語音參考檔的摘要資料，將更新旗標標示爲 “clean” ，代表該語音參考檔已經過編碼。

### 3.1.3 IPv6 通道代理人

在測試 IPv6 使用環境時，使用者可能是經由各種 IPv4/IPv6 轉移機制得到 IPv6 連線，爲了測試這些轉移機制的效能， SIPv6 Analyzer 需要有能力進行通道建立的流程，並藉由獲得的通道資訊設定 SmartBits 以進行測試。IPv6 通道代理人 (圖 3.1 ⓑ) 爲系統建立 IPv6 通道，使 SIPv6 Analyzer 可以透過 IPv6 通道發送 SIP 信令，進行 SIP 通話建立流程。 IPv6 通道代理人經由介面 (2) (圖 3.1 (2)) 接受控制器所傳的參數，依據參數中指定的通道建立方式 (TSP/Teredo) 以及 IPv4 位址，利用 IPv6 通道代理人整合的 Gateway6 Client [14] 或是 Miredo Teredo Client [15] ， IPv6 通道代理人經由介面 (3) (圖 3.1 (3))使用 TSP 或是 Teredo 取得 IPv6 通道。

### 3.1.4 SIP 代理人

由於測試環境中，使用者可能需要進行註冊認證後才能進行通話建立流程，或是待測系統中使用 SBC ， SBC 會修改 SIP 信令內容，決定 RTP 封包的路徑。因此本論文需要產生 SIP 信令，讓測試能順利進行。 SIP 代理人 (圖 3.1 ⓒ) 使用 SIPp [16] ， SIPp 根據輸入的 XML 檔案，決定其操作的模式爲 UAC (User Agent Client) 或是 UAS (User Agent Server) 。 XML 檔案中定義其執行的程式實體 (Instance) 執行的步驟，以及送出的 SIP 訊息內容。控制器由介面 (4) (圖 3.1 (5)) 傳送本論文所定義的兩個 XML 檔案之名稱 – 分別爲 UAC 與 UAS 所使用，以及 SIP URI 到 SIP 代理人，作爲 SIP 代理人的程式啓動參數。 SIP 代理人透過介面 (5) (圖 3.1 (5)) ，以 SIP 作爲信令交換之通訊協定。

### 3.1.5 RTP 代理人

RTP 代理人 (圖 3.1 ⓔ) 主要包含三個子元件 RTP 處理器 (RTP Processor ，圖 3.1 ⓗ) 、 Speech Decoder (圖 3.1 ⓘ) ，與 SmartLibrary (圖 3.1 ⓙ) 。 RTP 處理器接收控制器傳送的參數，透過 SmartLib 控制 SmartBits 發送 RTP 測試封包，測試結束時利用 SmartLibrary 取得測試結果，透過 Speech Decoder ，依據取得的測試結果，將預先編碼的資料 (圖 3.1 ⓜ 取出後進行解編碼，獲得劣化品質檔案。

控制器利用介面 (6) (圖 3.1 (6)) 傳遞指令與參數給 RTP 處理器，此部份同控制器的介面說明 (3.1.1) 。 此時 RTP 處理器已經得知 UAC 與 UAS 的傳輸位址，以及如何產生封包樣板 (Template) 。封包樣板爲一個位元陣列，內容爲預先設定好的網路封包標頭資訊。 RTP 處理器將 UAC 的傳輸位址，設定至封包樣板的目的傳輸位址，將 UAS 的傳輸位址設定至封包樣板的來源傳輸位址。最後將封包樣板的酬載 (Payload) 設定爲預先設定好的 RTP 封包樣板。產生完封包樣板之後， RTP 處理器透過 SmartLibrary (圖 3.1 (7)) ，設定 SmartBits 以封包樣板產生封包串流，並控制 SmartBits 發送 $N$ 個封包。所有的語音參考檔案共 $F$ 個，由於系統預設使用 P.862 所附的語音參考檔案，因此 $F$ 預設爲 20 。所有的語音參考檔案經過編碼後，共會產生 $N$ 個編碼後的資料，所以送出 $N$ 個封包，預設爲 7883 個，爲 P.862 所附的所有語音參考檔案經過編碼後產生的資料個數。在發送封包的時候，指定 SmartBits 爲每個封包加上簽章。 SmartBits 的封包簽章是 SmartBits 用來在接收埠辨認 SmartBits 所發送的封包的方式。簽章資訊包含四個欄位，封包串流編號 (Stream ID) 、該封包的序號 (Sequence) 、送出的時間郵戳，以及接收時的時間郵戳。

RTP 處理器透過 SmartLibrary 得知測試進度，在測試結束時從 SmartBits 取出 SmartBits 所接收到封包攜帶的的簽章資訊。此時 RTP 處理器藉由**步驟 5.1** 到 **5.3** 獲得封包遺失樣式 ($P$) ，封包遺失樣式爲一個紀錄封包有被接收或是未被接收的陣列，其元素值爲 1 代表有被接收， 0 代表封包遺失。

**步驟 5.1** RTP 處理器透過 SmartLibrary 詢問 SmartBits ，得知此次測試中接收到 $R$ 個封包具有簽章資訊。令 $i$ 爲 1 ，並產生封包遺失樣式 $P$ ，其長度爲 $N$ 位元 (bit) ，將其所有元素初始化爲 0 。

**步驟 5.2** 取得第 $i$ 個簽章資訊，根據其序號 $s$ 將陣列 $P$ 的第 $s$ 個元素標示爲 1 ，表示有收到序號 $s$ 的封包。

**步驟 5.3** 檢查 $i$ 是否等於 $R$ 。若不相等，則遞增 $i$ ，回到**步驟 5.2** ；若相等，則完成封包遺失樣式的產生。

RTP 處理器透過介面 (9) (圖 3.1 (9)) 取出所有使用語音編解碼器 (G.711u) 預先編碼的檔案，共 $F$ 個。此時 RTP 處理器將透過**步驟 5.4** 到 **5.5** 將取出的檔案分別存放在資料陣列 $E_f$ 中。

**步驟 5.4** 令 $f$ 爲 1 ， $n_f$ 代表第 $f$ 個預先編碼檔案內含的資料筆數，每筆資料長度爲 160 Bytes 。

**步驟 5.5** 將第 $f$ 個預先編碼檔案的資料複製到資料陣列 $E_f$ ，遞增 $f$ 。檢查 $f$ 是否等於 $F$ ，若否，則重複**步驟 5.5**；若是，則結束此步驟。

此時 RTP 處理器獲得 $F$ 個資料陣列 $(E_f)$ ，紀錄每個預先編碼檔案的資料。接下來藉由**步驟 5.6** 到 **5.9** ，將封包遺失樣式 $P$ 套用在編碼資料 $E_f$ 上，用來模擬封包遺失的情形：

**步驟 5.6** 令 $f$ 爲 1 ，代表正在產生第 $f$ 個語音參考檔案的劣化品質檔。 $n_f$ 爲陣列 $E_f$ 的長度。

**步驟 5.7** 令 $i$ 爲 1 ，將 $E_f$ 陣列的內容複製 $\lfloor \frac{N}{n_f} \rfloor$ 次，得到資料陣列 $e_f$ ，含有 $\lfloor \frac{N}{n_f} \rfloor * n_f$ 筆資料，爲套用過封包遺失樣式的編碼資料。

**步驟 5.8** 檢查封包遺失樣式 $P$ 的第 $i$ 個元素，若爲 0 則將陣列 $e_f$ 的第 $i$ 筆資料設定爲 0 ；若爲 1 則保留陣列 $e_f$ 的第 $i$ 筆資料。接下來遞增 $i$ ，並重覆此步驟直到$i$ 爲 $\lfloor \frac{N}{n_f} \rfloor * n_f$ 。

**步驟 5.9** 檢查 $f$ 是否爲 $F$ ，若不相等，則遞增 $f$ ，回到**步驟 5.7** ；若相等則結束此步驟。

此時 RTP 處理器獲得 $F$ 個資料陣列 $(e_f)$ ，透過 (圖 3.1 (10)) ， RTP 處理器將所有的 $e_f$ ，依序傳給 Speech Decoder 進行解編碼的動作，解編碼後產生劣化品質檔案。 Speech Decoder 經由 (圖 3.1 (11)) 將解編碼後的劣化品質檔案存入媒體資料庫中的 Degraded Files 資料表。最後 RTP 處理器由 (圖 3.1 (6)) 回報測試的結果，將產生的劣化品質檔案清單交給控制器。

### 3.1.6 PESQ

PESQ (Perceptual Evaluation of Speech Quality) 使用 P.862 [3] 所附的標準程式，控制器藉由介面 (12) (圖 3.1 (12)) ，輸入語音參考檔案與劣化品質檔案的名稱。 PESQ 根據輸入的語音參考檔案與劣化品質檔案的名稱，分別由介面 (13) (圖 3.1 (13)) 與介面 (14) (圖 3.1 (14)) 取得語音參考檔案與劣化品質檔案，並計算其 MOS 值。

## 3.2 驗證語音品質評估模組正確性

本論文所使用的 RTP 代理人、媒體前置處理器以及媒體資料庫，是由實驗室的詹依寰同學與作者所共同開發，主要的不同是在於本論文中所修改的 RTP 代理人，

利用 SmartLibrary 作爲控制 SmartBits 硬體平台的介面，利用 SmarBits 硬體發送測試封包，測試封包的傳送與接收的時間可以被精確的紀錄下來，並且能夠對待測系統進行壓力測試。爲了驗證語音品質評估模組的正確性，本論文設計一個實驗環境 (如圖 3.2) ，觀察語音品質評估模組所支援的三種語音編解碼器在不同封包遺失率環境下，計算其 MOS 值。並將結果與過去研究結果比較，以驗證本論文整合語音編解碼器與 RTP 代理人，及所實作的語音品質評估模組所計算的 MOS 值之正確性。

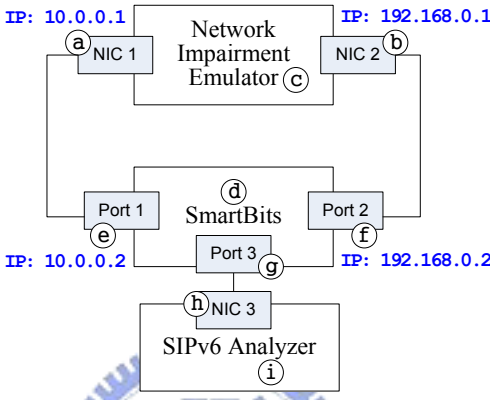


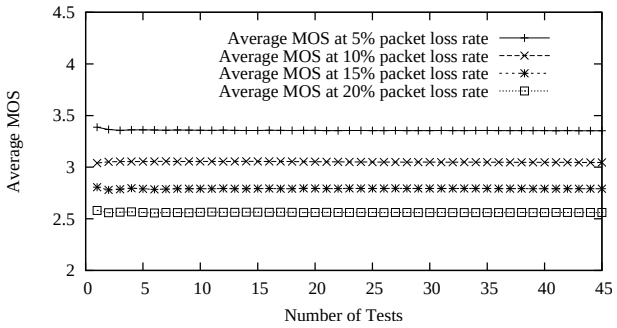

圖 3.2: 驗證語音品質評估模組正確性之實驗環境

實驗環境 (圖 3.2) 主要包含三個元件，網路模擬器 (Network Impairment Emulator，圖 3.2 ⓒ) 、 SmartBits (圖 3.2 ⓓ) ，以及 SIPv6 Analyzer (圖 3.2 ⓘ) 。 SIPv6 Analyzer 透過 NIC 3 (圖 3.2 ⓗ) 控制 SmartBits 由 Port 2 (圖 3.2 ⓕ) 發送 RTP 測試封包，由 Port 1 (圖 3.2 ⓔ) 接收封包。網路模擬器所使用的系統爲 IP Wave V.3 Network Impairment Emulator ，模擬實際網路中的封包遺失，可設定其封包遺失率，以及遺失的模式 (如：“隨機” (Random) 或 “週期式” (Periodic)) 。網路模擬器具有兩張網路卡，分別爲 NIC 1 (圖 3.2 ⓐ) 與 NIC 2 (圖 3.2 ⓑ) ， 用來轉送來自 SmartBits 的 Port 2 的 RTP 封包到 SmartBits 的 Port 1 ，網路模擬器會根據使用者在網路模擬器上設定的封包遺失率，將 NIC 2 收到的封包標示爲 “丟棄” (Dropped) 或是 “通過” (Pass) ，將標示爲通過的封包經由 NIC 1 送出，將標示爲丟棄的封包丟棄，並且使標示爲丟棄的封包佔 NIC 2 所收到的封包的比例，爲使用者所設定的封包遺失率。各元件組態與實驗流程說明如下：

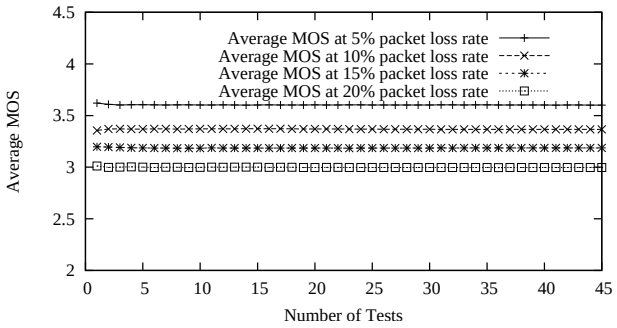
網路模擬器的 NIC 1 與 NIC 2 具有 IP 位址分別爲 10.0.0.1 與 192.168.0.1 ，設定爲不同網域。SmartBits 的 Port 1 與 Port 2 的 IP 位址分別爲 10.0.0.2 與 192.168.0.2 。 SIPv6 Analyzer 設定 SmartBits 產生一個 IPv4 的 RTP 串流，其來源傳輸位址爲 192.168.0.2:9000 目的傳輸位址爲 10.0.0.2:9000 。依據語音品質評估模組所支援的三種語音編解碼器 (G.729 、 iLBC 與 G.711u) 設定傳送的封包大小，分別爲 74 Bytes 、 92 Bytes 與 214 Bytes 。由於大部分的 SIP UA 實作中， 爲 20 ms 發送一次 RTP 封包，因此 RTP 串流的封包設定爲每 20 ms 發送一次。共進行四次實驗，分別設定網路模擬器的封包遺失率爲 5% 、 10% 、 15% 以及 20 % ，並且根據 [17] 設定網路模擬器的封包遺失模式爲隨機。

由於網路模擬器於模擬封包遺失時，使用隨機的方式標示哪些封包爲丟棄，在短時間的實驗內 SmartBits 的 Port 1 計算的封包遺失率，並不會與實驗所設定的封包遺失率完全相同。而且因爲 PESQ 根據輸入的語音參考檔案，以語音訊號所含的能量大小，判斷語音資料爲通話部份 (Speech Bursts) 或是靜音部分 (Silent Periods) ，並對通話部分的語音訊號，與劣化品質檔案中相對應的語音訊號進行計算。所以 PESQ 輸出的 MOS 值會受到封包遺失是否位於通話部分影響，如果遺失的封包攜帶的語音資料是通話部分，則對輸出的 MOS 值影響較大；反之，遺失的封包所攜帶的語音資料是靜音部分，則對輸出的 MOS 值影響較小。實驗時間若不夠長，可能因爲隨機產生的封包遺失集中在通話部分或是靜音部分，使得計算所得的 MOS 質偏低或偏高。因此實驗時間需要達一定時間以上，才能使得實驗環境中，實測的封包遺失率接近所設定的封包遺失率。而且使封包遺失可以平均出現在通話部分與靜音部分，讓這段時間內的系統計算的平均 MOS 值趨於穩定。

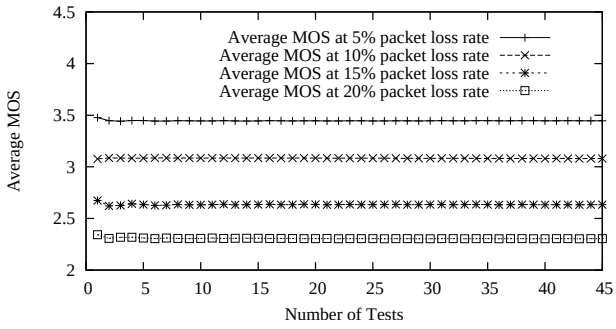
每個語音編解碼器所輸出的實驗結果如圖 3.3 ，隨著測試次數增加，平均 MOS 值逐漸趨向穩定值，直到測試次數達 45 次之後，每次得到的平均 MOS 值差異已經小於 $10^{-3}$ 。而且網路模擬器所產生的封包遺失率，與設定的封包遺失率差異值，在測試 45 次之後維持在 1% 以下。因此本論文將使用測試 45 次得到的 MOS 值，與其他研究結果進行比較。




(a) 語音編解碼器爲 G.729




(b) 語音編解碼器爲 iLBC




(c) 語音編解碼器爲 G.711u

圖 3.3: 平均 MOS 值與測試次數關係圖

本論文將三種語音編解碼器，在四種不同封包遺失率環境下的平均 MOS 值，與前人的研究結果 [18] [19] [20] 進行比較，得到表 3.1 – 3.3 。由於 [19] 的結果中， G.711 在封包遺失率爲 5 % 與 10 % 得到的 MOS 值，比 iLBC 在相同情形下得到的 MOS 值高，與本實驗與 [18] 所做的實驗結果有很大的差異。因此本論文將排除 [19] 所做的 G.711 之實驗結果。

表 3.1: 語音編解碼器之 MOS 值 (G.729)

| Loss Rate | Mean MOS (Our Work) | MOS (Relative Work) | Differences |
|---|---|---|---|
| 5% | 3.352 | 3.39 | 1.12 % |
| 10% | 3.045 | 3.09 | 1.46 % |
| 15% | 2.789 | 2.82 | 1.10 % |
| 20% | 2.561 | 2.63 | 2.62 % |

本論文所測量的三個語音編解碼器在 5% 、 10% 、 15% ，以及 20% 的環境下的 MOS 值，與 [20] 、 [19] 及 [18] 的差異值大多在 5% 以下。由於網路模擬器產生的封包

表 3.2: 語音編解碼器之 MOS 值 (iLBC)

| Loss Rate | Mean MOS (Our Work) | MOS (Relative Work) | Differences |
|---|---|---|---|
| 5% | 3.601 | 3.47 | 3.75 % |
| 10% | 3.366 | 3.25 | 3.38 % |
| 15% | 3.185 | 3.05 | 4.50 % |
| 20% | 2.997 | 2.88 | 4.06 % |

表 3.3: 語音編解碼器之 MOS 值 (G.711u)

| Loss Rate | Mean MOS (Our Work) | MOS (Relative Work) | Differences |
|---|---|---|---|
| 5% | 3.446 | 3.300 | 4.42 % |
| 10% | 3.080 | 2.930 | 5.12 % |

遺失率與設定值約有 1% 的誤差，且上述三篇論文的結果，並未說明所使用的語音參考檔案，也未說明其 MOS 值的計算進行幾次實驗，因此本論文認爲 5% 的差異在可以接受的範圍。

另外，爲了驗證封包遺失樣式的套用是否正確，本論文取得 [21] 所提供的 iLBC 參考實作，該參考實作包含 iLBC 的語音編解碼器、語音參考檔案、封包遺失樣式，以及套用封包遺失樣式後的劣化品質檔案。將該參考實作的語音參考檔案與劣化品質檔案輸入 PESQ 後得到的 MOS 值，與相同語音參考檔案及相同的封包遺失樣式的情形下，本系統所計算得到的 MOS 值做比較。於沒有封包遺失的情形下，兩者完全相同 (3.75) 。套用參考實作所附的封包遺失樣式時，兩者亦得到相同的 MOS 值 (3.384) 。由上述的實驗結果，可以驗證語音品質評估模組正確無誤。

# 第四章　實驗環境與實驗結果

在 VoIP (Voice over Internet Protocol) 應用中 IPv4 位址不足的問題，目前有 IPv6 (Internet Protocol Version 6) [22] 及 NAT (Network Address Translator) [23] 兩種解決方案。 IPv6 在使用上可分爲兩種 Pure IPv6 及 IPv6 通道，前者是網路節點全面使用 IPv6 ；後者是在 IPv6/IPv4 轉換的過程中提供 IPv6 的連線能力給 IPv4 網路的使用者。 IPv6 通道可再分爲， IPv6-in-IPv4 通道與 IPv6-in-IPv4 UDP (User Datagram Protocol) 通道，前者將 IPv6 封包封裝在 IPv4 封包內，需要 Public IPv4 位址才能建立 IPv6-in-IPv4 通道；後者將 IPv6 封包封裝在 IPv4 UDP 封包內，讓 NAT 內的使用者使用 IPv6-in-IPv4 UDP 通道，獲得 IPv6 的連線能力。

NAT 雖然解決了 IPv4 位址不足的問題，但是在 VoIP 應用中，在 NAT 下的使用者還需要 SIP/RTP 穿越 NAT 的機制。目前 VoIP 服務提供者大多使用 SBC (Session Border Controller) [24] 解決 SIP/RTP 穿越 NAT 的問題。本論文將測試 Pure IPv6 、IPv6-in-IPv4 通道、 IPv6-in-IPv4 UDP 通道，以及 SBC 四種解決方案，進行網路元件的效能分析與比較。

本論文將選出的 IPv4 位址不足解決方案，簡化其使用環境如圖 4.1 。第一種： UAC (User Agent Client ，圖 4.1 (1)) 與 UAS (User Agent Server ，圖 4.1 (2)) 都使用 Pure IPv6 解決 IPv4 位址不足的問題，通話時不需要其他元件轉送 RTP 封包， UAC 與 UAS 直接互相傳送 RTP 封包。第二種： UAC 使用 Public IPv4 ， 而 UAS 爲 NAT 下之使用者，以 SBC 解決穿越 NAT 的問題， RTP 封包經過 RTP Proxy (圖 4.1 (4)) 與 NAT 1 (圖 4.1 (7)) 。 第三種： UAC 使用 Pure IPv6 ， UAS 使用 IPv6-in-IPv4 通道， RTP 封包經過 Tunnel Server 1 (圖 4.1 (5)) 。 第四種： UAC 使用 Pure IPv6 ，

UAS 使用 IPv6-in-IPv4 UDP 通道， RTP 封包經過 Tunnel Server 2 (圖 4.1 (6)) 與 NAT 2 (圖 4.1 (8)) 。所有環境中 UAC 與 UAS 皆透過 SIP Server (圖 4.1 (3)) 繞送 SIP 信令。

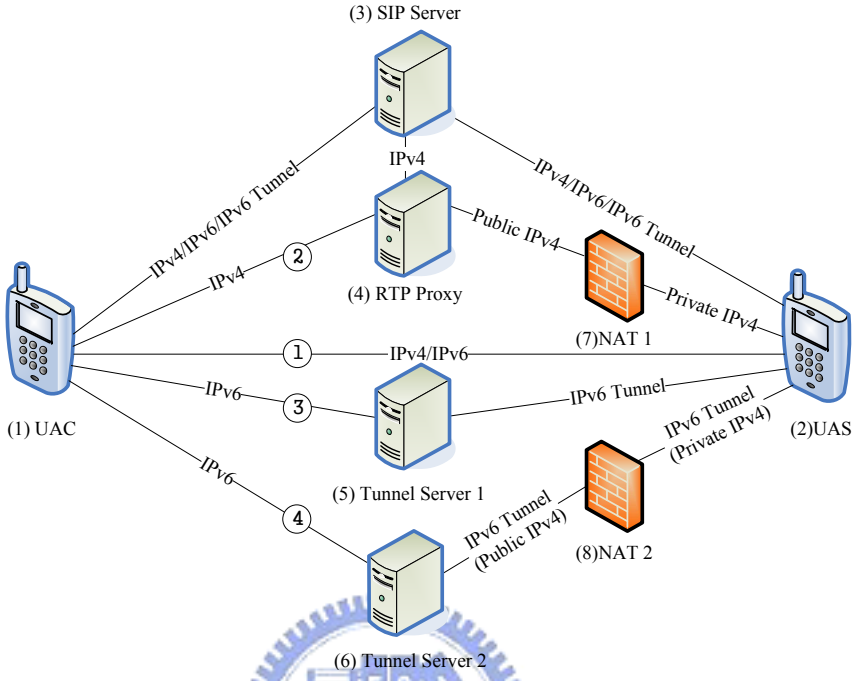



圖 4.1: VoIP 使用環境示意圖

本論文在受控制的網路環境下，設計四個實驗模擬圖 4.1 所示的四種使用環境，探討各種 IPv4 位址不足的解決方案的網路元件對通話品質的影響，最後以端到端 (end-to-end) 平均延遲、封包遺失率及 MOS 來評估其效能。

## 4.1 實驗 1 – Pure IPv6/ Public IPv4 使用環境之效能評估

爲了評估 Pure IPv6 使用環境下，在 RTP 封包不經過任何元件， UAC 與 UAS 藉由 Layer-2 Switch 連接所有設備，並直接傳送時的通話品質爲何，並以此結果作爲其他環境的比較標準，本論文設計實驗 1 的實驗環境。藉由此實驗，了解此依環境中不同語音編解碼器 (CODEC) 對於通話品質的影響，並以同時通話個數對平均延遲、封包遺失率，以及 MOS 的評估其效能。

實驗環境如圖 4.2 所示， SIP Server (圖 4.2 ⓐ) 負責繞送 SIP 信令。 Layer-2

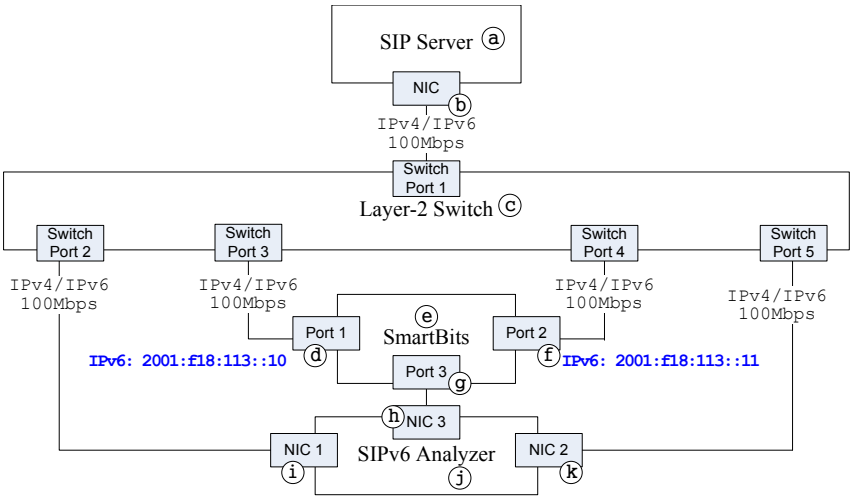



圖 4.2: Pure IPv6/Public IPv4 效能測試環境

Switch (圖 4.2 ⓒ) 將所有元建連接起來。 SmartBits (圖 4.2 ⓔ) 負責產生 RTP 的測試封包。 SIPv6 Analyzer (圖 4.2 ⓙ) 負責發送 SIP 信令以進行 SIP 通話建立流程、產生 SDP 所需攜帶的傳輸位址 (Transport Address ，參考 RFC 3550 [7]) ，並且根據收到的 SIP 訊息 SDP 攜帶的傳輸位址，設定 SmartBits 。最後由 SIPv6 Analyzer 產生實驗結果報表。本實驗以 Pure IPv6 的設定爲例，各元件的組態如下：

SIP Server 所使用的系統爲 Linux 2.6.21.3 並執行 SER [25] 0.9.6 ，而 NIC (圖 4.2 ⓑ) 具有 IPv6 位址，連接到 Layer-2 Switch (圖 4.2 ⓒ) 。SmartBits 型號爲 SMB-600B ，搭配 TeraMetrics LAN-3321A 卡板，卡板具有兩個 10/100/1000 MBps 網路埠即 Port 1 (圖 4.2 ⓓ)與 Port 2 (圖 4.2 ⓕ) 。分別模擬爲圖 4.1 中的 UAC 與 UAS 並負責收送 RTP 封包， IPv6 位址分別爲 2001:f18:113::10 與 2001:f18:113::11 。 Port 3 (圖 4.2 ⓖ) 與 SIPv6 Analyzer 的 NIC 3 (圖 4.2 ⓗ) 皆具有 IPv4 位址， SIPv6 Analyzer 透過此連線控制 SmartBits。SIPv6 Analyzer 之 NIC 1 (圖 4.2 ⓘ) 與 NIC 2 連接到 Layer-2 Switch (圖 4.2 ⓚ) 負責收送 SIP 信令以建立通話，具有 IPv6 位址。所有的網路連線皆是 100MBps Ethernet 。Public IPv4 之實驗環境與 Pure IPv6 環境相同，只有 IP 位址組態改變爲 IPv4 ，因此本節不重複描述其組態。

### 4.1.1 實驗 1 實驗流程

SIPv6 Analyzer 根據同時通話個數決定進行 SIP 通話建立流程的次數， 以 NIC 1 當作 UAC 送出 SIP INVITE 訊息， 訊息中 SDP 所攜帶的傳輸位址 ，由 SmartBits 的 Port 1 的 IPv6 位址與 UDP 通訊埠 9000 組成 (i.e., [2001:f18:113::10]:9000) 。 SIP INVITE 訊息經由 SIP Server 將 SIP 訊息繞送至 SIPv6 Analyzer 的 NIC 2 ， SIPv6 Analyzer 接著以 SmartBits 上 Port 2 的 IPv6 位址與 UDP 通訊埠 9000 產生傳輸位址 (i.e., [2001:f18:113::11]:9000) ，攜帶於 SIP 200 OK 訊息，並以 NIC 2 送出。 SIPv6 Analyzer NIC 1 收到 SIP 200 OK 訊息後回應 ACK ，完成通話建立。在每次通話建立完成後遞增 UDP 通訊埠，產生新的傳輸位址以供應下次 SIP 通話建立流程使用直到完成同時通話個數所需的次數。

成功完成 SIP 通話建立流程之後， SIPv6 Analyzer 根據 SDP 所攜帶的傳輸位址，透過 NIC 3 設定 SmartBits 的 Port 1 與 Port 2 的傳輸位址，並依據所使用語音編解碼器決定送出 RTP 封包的間隔時間。本論文所使用的三種語音編解碼器(G.729 、 iLBC 與 G.711u) ，在大部分的實作中是間隔 20 ms 送出一個 RTP 封包，因此設定送出 RTP 封包的間隔爲 20 ms ，最後以 Port 2 送出 RTP 封包 。由於 P.862 所附的 20 個語音參考檔案，編碼後共產生 7883 筆資料，因此測試時每個封包串流送出 7883 個封包，即每次測試進行時間約 157 秒。

接下來則對此系統進行 (1) 封包遺失率之測量 、 (2) 平均延遲之測量，以及 (3) MOS 之測量。由於 SmartBits 可以同時紀錄封包遺失率與封包的延遲，因此 (1) (2) 的測試可同時進行。

**封包遺失率之測量**

在此說明封包遺失率計算方式，每次實驗結果的封包遺失率計算如公式 4.1 ，SmartBits 的 Port 2 送出封包總數 $N$ ，由 SmartBits 的 Port 1 收到封包總數 $R$ 。

$$\text{Packet Loss Rate} = \frac{R}{N} * 100\% \tag{4.1}$$

尋找最大吞吐量值的方式是利用二分逼近，在給定的上限與下限之間進行搜尋，並觀察每次測試結果的封包遺失率，直到找到最大吞吐量爲止。測試前需要計算出線路可負載的通話個數上限 ($M_C$) ，並與 SmartBits 可以產生封包串流 (Packet Stream) 上限數目 (2048) 比較，並選取其最小值，作爲測試的上限值 ($U$) 。$U$ 值決定了搜尋最大吞吐量測試的搜尋上限之初始值，以及封包遺失率測試、平均延遲測試，與 MOS 測試中， SmartBits 所要模擬的同時通話個數上限。$M_C$ 由線路的頻寬，與使用語音編解碼器 $C$ 的封包串流所需的頻寬 ($B_C$) 決定。$B_C$ 計算方式爲每個封包所需傳輸的時間，乘上每秒鐘傳送封包數。$B_C$ 於 100Mbps Ethernet 的計算方式爲 (Inter-frame Gap + Preamble + Length of Ethernet Frame + CRC) * 50 。以 G.729 的測試爲例，其 $B_C$ 值爲 47200 。$M_C$ 值由公式 4.2 計算爲 2118 。

$$M_C = \lfloor \frac{100\text{MBps}}{B_C} \rfloor \tag{4.2}$$

$2118 > 2048$ 因此選 2048 爲 $U$ ，進行搜尋最大吞吐量的步驟如下：

**步驟 1** 設定上限 ($u$) 爲 2048 ，下限 ($l$) 爲 0 ，最大吞吐量($T$) 爲 0 ， SIPv6 Analyzer 將在上下限範圍中改變同時通話個數 ($S$) ，測試待測系統以求得其最大吞吐量。

**步驟 2** 設定同時通話個數 $S$ 爲 $\lceil \frac{u+l}{2} \rceil$ 。

**步驟 3** 進行通話建立流程，並進行測試。

**步驟 4** 測試結果可知是否有封包遺失，若有封包遺失則設定 $u$ 爲 $S$ ；若無封包遺失則將 $l$ 設定爲 $S$ 。

**步驟 5** 檢查 $T$ 是否與 $S$ 相同，若不同則設定 $T$ 爲 $S$ 並重複 **步驟 2** 到 **步驟 5** ；若相同則表示測試已達終止狀態。此時，最大吞吐量值應在 $T \pm 1$ 的範圍內。

**平均延遲之測量**

每個封包的端到端延遲 ($D$) 如公式 4.3 ，計算平均延遲的方式，是所有收到的封包之端到端延遲平均值。

$$D = \text{Propagation Delay} + \text{Transmission Delay} + \text{Processing Delay} \tag{4.3}$$

SmartBits 在進行測試的時候，可以設定傳送埠在每個送出的封包末端 (CRC 資訊之前)，覆蓋 18 Bytes 的簽章資料。 內含有四個欄位：封包所屬的串流編號 (Stream ID) 、 該封包的序號 (Sequence) 、該封包離開傳送埠的時間郵戳 (Timestamp) 、該封包被接收埠接收的時間郵戳。 SmartBits 接收埠在收到封包時填上其中的接收時間郵戳，其他欄位則是在送出時填上。藉由此簽章資訊， SmartBits 可以獲得每個封包的端到端延遲，統計每個串流中封包延遲時間，並產生紀錄。 SIPv6 Analyzer 在測試結束時，將 SmartBits 所產生的紀錄取出，計算每個封包串流的平均延遲如公式 4.4 。 $L_i$ 、 $d_i$ 與 $p_i$ 分別代表第 $i$ 個封包串流之平均延遲、 SmartBits 所收到第 $i$ 個封包串流之封包的端到端延遲總和，與 SmartBits 所收到第 $i$ 個封包串流之封包的總數。

$$L_i = \frac{d_i}{p_i} \tag{4.4}$$

最後系統輸出所有封包串流的平均延遲之平均值 $(L)$ ，作爲測試輸出之結果，如公式 4.5 ， $n$ 爲 SmartBits 測試時產生的封包串流數目。

$$L = \frac{\sum_{i=1}^{n} L_i}{n} \tag{4.5}$$

封包遺失與平均延遲的測試，由 1 個同時通話開始，接下來是 100 個同時通話，之後每次遞增 100 個同時通話，即 200 、 300 、 直到上限 $(U)$ ，並且紀錄每次得到的平均延遲與封包遺失率。

### MOS 值之測量

爲了解 UAC 與 UAS 間的通話品質，每次測試時有 $n$ 個同時通話， SIPv6 Analyzer 將設定 SmartBits 產生 1 個有標定簽章的封包串流，其餘 $n-1$ 封包串流做爲背

景資料流 (Background Traffic) 。藉由追蹤此一有標定簽章的封包串流中的封包，得知其每個封包接收狀態 (“已接收” 或 “未接收”) ，產生每個參考檔案所對應的劣化品質檔案 (Degraded File) ，並將每一個參考檔案，與其劣化品質檔案輸入 PESQ 評估後得到 MOS 值。最後將所有得到的 MOS 取平均後得到最後輸出的 MOS 值，此 MOS 代表實驗中所觀察的 UAC 與 UAS ，藉此了解在各種不同的背景資料流情形下， VoIP 通話的品質變化情形。

測試過程較前兩者不同，同時通話個數是由最大吞吐量 ($T$) 開始測試，可以獲得沒有封包遺失的情形下的 MOS 值，接下來測試同時通話個數超過最大吞吐量的情形，由 $\lceil \frac{T}{100} \rceil * 100$ 個同時通話開始，每次測試遞增 100 個同時通話。如此測試便可了解在不同的同時通話個數下，封包遺失率與 MOS 值的關聯性。

至此完成一種語音編解碼器的測試。接下來重複上述流程，將所有語音編解碼器測試完畢即完成 Pure IPv6 之測試。接下來根據 4.1 的設定修改各元件的 IPv4 設定，依照相同實驗流程完成 Public IPv4 之測試。

### 4.1.2 實驗 1 實驗結果

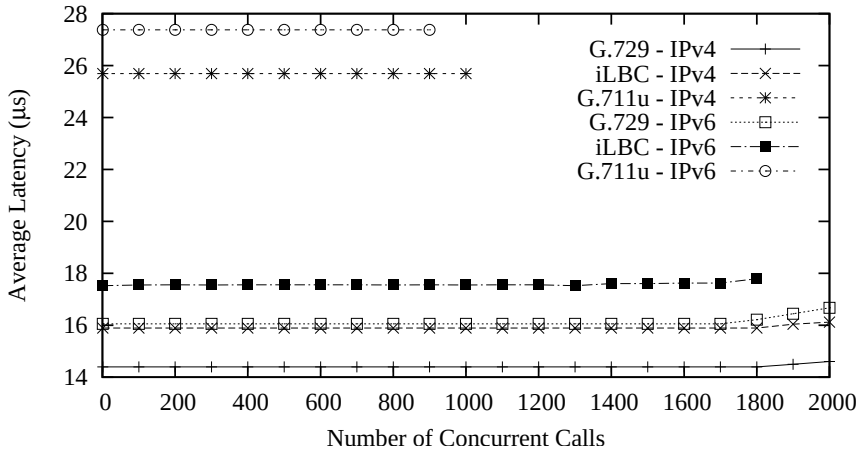



圖 4.3: Pure IPv6/Public IPv4 效能測試結果 – 同時通話個數對平均延遲之關係圖

預期此環境並不會有明顯的效能損失，同時通話個數與平均延遲的關係實驗結果如圖 4.3 ，與預測結果相符。經實驗中所使用三種編解碼器 (G.729 、 iLBC 與 G.711u)

編碼後之封包，長度分別爲 94 、 112 與 234 Bytes ，其平均延遲約爲 16 $\mu$s 、 17.5 $\mu$s 與 27.5 $\mu$s ，較長的封包，其平均延遲較長。不同的語音編解碼器的實驗結果，其平均延遲的差異值，推測主要來自於封包的傳輸延遲 (Transmission Delay) 。傳輸延遲依據 100MBps Ethernet 的理論值計算，如公式 4.6 。

$$\frac{\text{Inter-frame Gap} + \text{Preamble} + \text{Length of Ethernet Frame} + \text{CRC}}{\text{100MBps}} \tag{4.6}$$

假設處理延遲 (Processing Delay) 約略爲定值，所有封包傳遞於相同媒體，因此傳遞延遲 (Propagation Delay) 也相同。以 G.729 的封包端到端延遲爲基準，由公式 4.3 與公式 4.6 計算，得到使用 iLBC 與 G.711u 的封包的端到端延遲的理論值之差異，分別爲 1.44 $\mu$s 與 11.2 $\mu$s 。將 iLBC 與 G.711u 的 RTP 封包的平均延遲，減去使用 G.729 的 RTP 封包平均延遲，得到端到端延遲的實測值差異。比較實測值與理論值，最大的誤差列於表 4.1 。

表 4.1: 封包傳輸延遲差異值之理論值與實測值之比較表

| | Pure IPv6 | | Public IPv4 | |
|---|---|---|---|---|
| 語音編解碼器 | iLBC | G.711u | iLBC | G.711u |
| 傳輸延遲理論差異值 | 1.44 | 11.2 | 1.44 | 11.2 |
| 傳輸延遲實測最大差異值 | 1.579 | 11.325 | 1.539 | 11.295 |
| 實測與理論值之最大誤差 | 0.139 | 0.125 | 0.099 | 0.095 |

由表 4.1 可知，實測結果，之最大誤差不超過 SmartLibrary 所說明的測試精準度誤差範圍 $\pm$ 0.2 $\mu$s ，可驗證實驗結果中，使用不同語音編解碼器的 RTP 封包，其平均延遲的差異值主要來自於傳輸延遲。並可得知在實驗 1 實驗環境中， Layer-2 Switch 並不會造成顯著的效能損失，不論是平均延遲或是封包遺失。使用 G.729 、 iLBC 與 G.711u 三種語音編解碼器的通話品質，經由 PESQ 評估，分別爲 3.848 、 3.956 與 4.44 ，爲其他實驗中所能得到最佳的 MOS 值。

## 4.2 實驗 2 – SBC 使用環境之效能評估

VoIP 服務中，目前服務提供者大多使用 SBC (Session Border Controller) [24] 解決 SIP/RTP 穿越 NAT 的問題。 SBC 包含兩個元件，分別爲圖 4.1 中的 SIP Server (圖 4.1 (3)) 與 RTP Proxy (圖 4.1 (4)) 。以 SBC 解決 SIP/RTP 穿越 NAT 問題的 VoIP 使用環境中， RTP 封包會經過 NAT 與 RTP Proxy ，爲了釐清影響整理環境效能的因素，本論文分別測試 NAT 與 RTP Proxy ，最後再以整合環境測試，評估 SBC 使用環境的通話品質。

### 4.2.1 RTP Proxy 效能測試

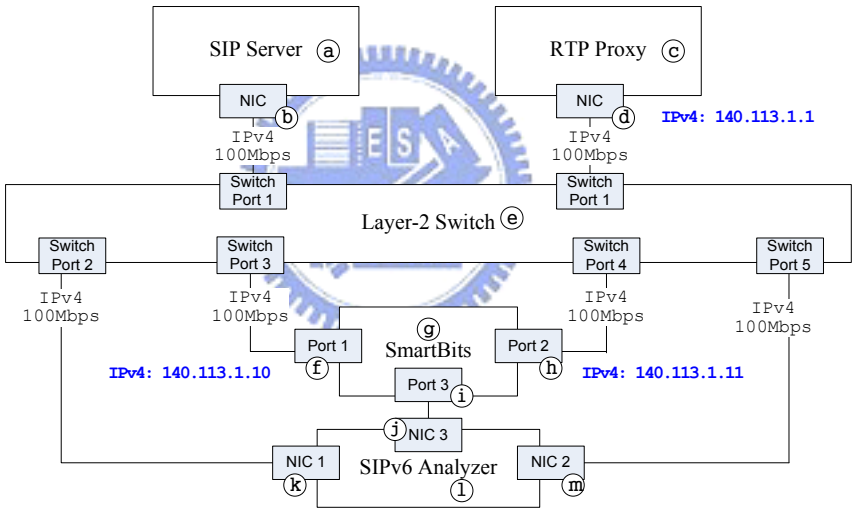



圖 4.4: RTP Proxy 效能測試環境

RTP Proxy 效能測試的環境如圖 4.4 ，由於 RTP Proxy 需要與 SIP Server 進行互動，才能設定 RTP Proxy 轉送 RTP 封包的傳輸位址，因此與實驗 1 (圖 4.2) 的環境相比，新增 RTP Proxy (圖 4.4 ⓒ) 。RTP Proxy 所使用的程式爲 Portaone RTP Proxy ，爲開放原始碼之軟體， Portaone RTP Proxy 使用 IPv4 位址 140.113.1.1 。SIP Server 具有 IPv4 位址， SIP Server 透過網路控制 RTP Proxy 。SmartBits 的 Port 1 與 Port 2 的 IPv4 位址分別爲 140.113.1.10 與 140.113.1.11 ，其餘設定同實驗 1 (圖 4.2) 。

本論文藉由檢驗 Portaone RTP Proxy 之程式碼瞭解其系統架構， Portaone RTP

Proxy 的系統架構如圖 4.5 ， Portaone RTP Proxy (圖 4.5 ⓐ) 是利用 UDP Socket (圖 4.5 ⓔ)，進行轉送 RTP 封包的動作。 Portaone RTP Proxy 主要由 RTP Forwarding Table (圖 4.5 ⓑ) 與 Polling Event Handler (圖 4.5 ⓒ) 所構成。 Polling Event Handler 定期呼叫系統的 Poll System Call (圖 4.5 ⓓ) ，檢查 RTP Forwarding Table 所維護的所有 UDP Socket 之狀態，了解其緩衝區是否已經有資料可供讀取，並回傳所有 UDP Socket 的狀態。

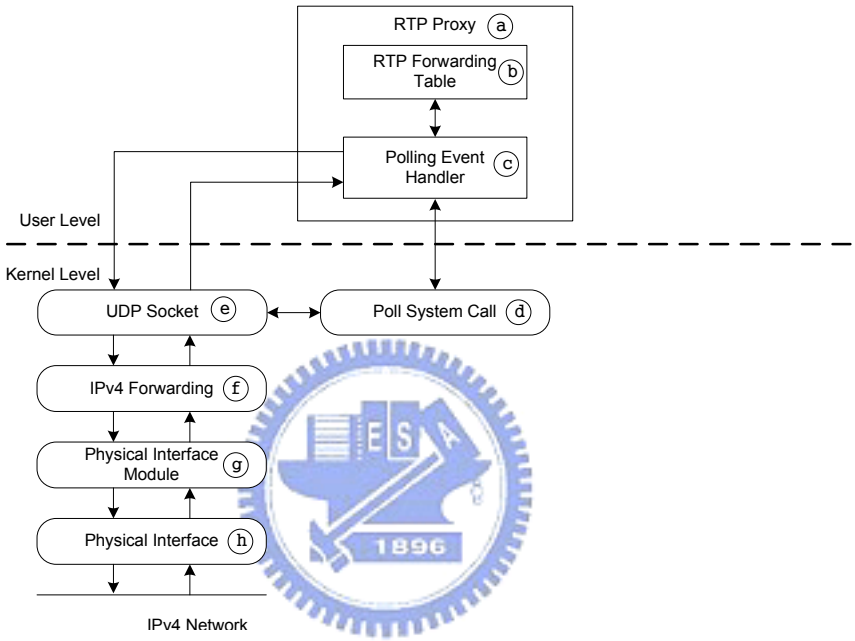



圖 4.5: Portaone RTP Proxy 系統架構

Portaone RTP Proxy 核心的運作機制爲 Polling Event Handler ，其運作流程如下：

**步驟 1** 初始化時間郵戳 $t_s = 0$ 。

**步驟 2** 取得目前系統時間，並存於時間郵戳 $t_c$ 。若 $(t_c - t_s) < 10\text{ms}$ ，則表示距離上次取得系統時間已經過了 $t_c - t_s$ ms 。讓 Portaone RTP Proxy 的程序 (Process) 進入睡眠狀態 $10 - (t_c - t_s)$ ms ，等待作業系統將程序喚醒，讓 Poll Event Handler 大約保持在 10 ms 才呼叫一次 Poll System Call 。喚醒後取得目前系統時間，存於時間郵戳 $t_s$ ；若 $(t_c - t_s) \geq 10\text{ms}$ ，將 $t_c$ 值存入 $t_s$ 。

**步驟 3** 呼叫 Poll System Call ，檢查回傳資料陣列第一筆資料。

**步驟 4** 若 UDP Socket 可供讀取，進行**步驟 6** ；反之進行**步驟 5** 。

**步驟 5** 檢查是否已經將資料陣列檢查完畢。若已經檢查完資料陣列，則回到**步驟 2** ；否則檢查陣列中下一筆資料，並進行**步驟 4** 。

**步驟 6** Polling Event Handler 已經知道 UDP Socket 的緩衝區中有資料可供讀取，讀出資料後，接著以 UDP Socket 爲鍵值 (Key) 查詢 RTP Forwarding Table ，可得知要將資料轉送至哪個傳輸位址，並以該 UDP Socket 送出。重覆此步驟直到該 UDP Socket 緩衝區內所有資料清空，回到**步驟 5** 。

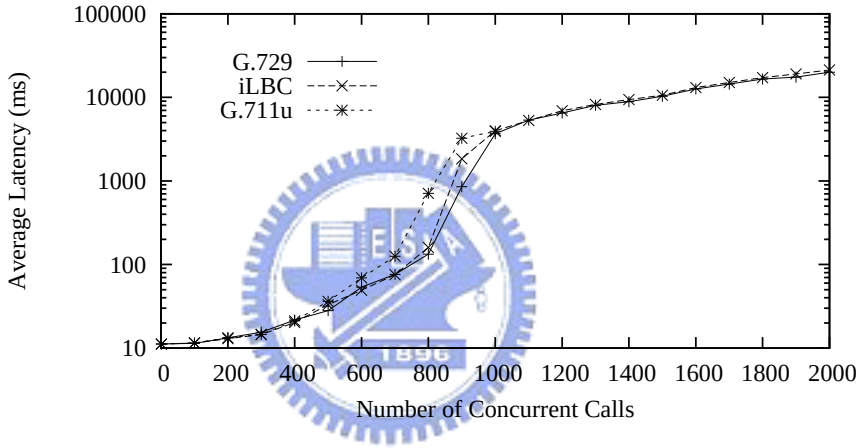
實驗流程與實驗 1 相同。但是 SIPv6 Analyzer 在傳送 SIP 訊息到 SIP Server 後，SIP Server 將修改 SIP 訊息，改變 SDP 所攜帶的傳輸位址，爲 RTP Proxy 上的傳輸位址。以第一次通話建立流程爲例， SIPv6 Analyzer 分別以 SmartBits Port 1 與 Port 2 的 IPv4 位址與 UDP 通訊埠 9000 組成傳輸位址 (140.113.1.10:9000 與 140.113.1.11:9000) 。 SIPv6 Analyzer NIC 1 送出的 SIP INVITE 訊息之 SDP ，在通過 SIP Server 時，所攜帶的傳輸位址被轉換爲 RTP Proxy 上的傳輸位址 (140.113.1.1:35000) ，同時 RTP Proxy 會在 RTP Forwarding Table (圖 4.5 ⓑ) 上新增資料，如表 4.2 的第一筆資料；同樣的方式， SIPv6 Analyzer NIC 2 送出的 SIP 200 OK 訊息之 SDP ，其攜帶的傳輸位址被轉換爲 140.113.1.1:35002 ，同時 RTP Proxy 會在 RTP Forwarding Table 上新增資料，如表 4.2 的第二筆資料。

表 4.2: RTP Forwarding Table 範例

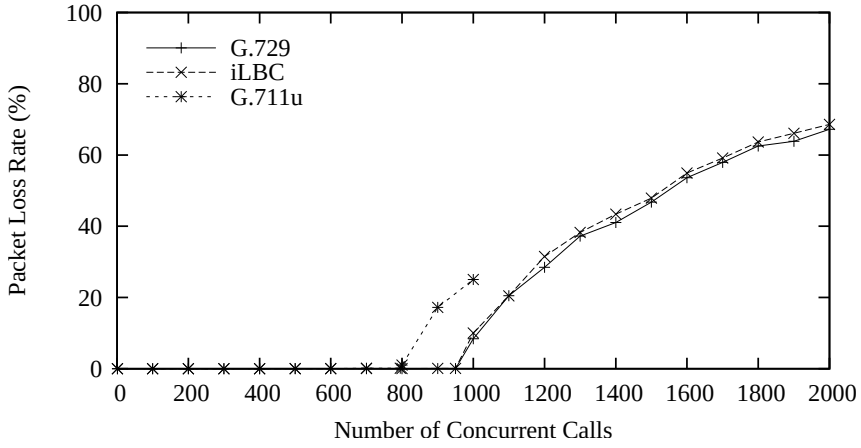
| Index | Receive From UDP Socket | Send To |
|---|---|---|
| 1 | 1 | 140.113.1.10:9000 |
| 2 | 2 | 140.113.1.11:9000 |

SIPv6 Analyzer 根據 NIC 2 收到的 SIP INVITE 訊息之 SDP ，設定 SmartBits Port 2 送出 RTP 封包，其的目的傳輸位址爲 140.113.1.1:35000 ，來源傳輸位址爲 140.113.1.11:9000 。當 RTP Proxy (圖 4.5 ⓐ) 的 Physical Interface (圖 4.5 ⓖ) 從 IPv4 網路收到 RTP 封包，經過 Physical Interface Module (圖 4.5 ⓕ) 處理後，此封包將會交給 IPv4 Forwarding ，經過判定發現此封包目的爲 RTP Proxy 上的 UDP Socket (圖 4.5

ⓔ)，系統即把此封包傳給 UDP Socket ，儲存在 Socket 緩衝區中，並標定此 Socket 緩衝區已經可供讀取。 Poll Event Handler (圖 4.5 ⓒ) 呼叫 Poll System Call (圖 4.5 ⓓ) 並解析 Poll System Call 回傳的資料陣列，得知 UDP Socket 1 的緩衝區已經有資料可供讀取。 Poll Event Handler 以 UDP Socket 查詢 RTP Forwarding Table ，符合第一筆資料，可以知道 UDP Socket 1 緩衝區資料要轉送至傳輸位址 140.113.1.10:9000 。 Poll Event Handler 便從系統的 Socket 緩衝區中資料讀出，此時需要將此資料從作業系統核心層 (Kernel Level) 複製到使用者層 (User Level) 。並根據 RTP Forwarding Table 將資料再由 Socket 送出。同理，使用 Socket 傳送資料，需要再把使用者層的資料複製到作業系統核心層。




(a) 同時通話個數對平均延遲之關係圖




(b) 同時通話個數對封包遺失率之關係圖

圖 4.6: RTP Proxy 效能測試結果

RTP Proxy 效能測試結果如圖 4.6 (a) 與 (b) 所示，使用 G.729 與 iLBC 時系統的

最大吞吐量爲 950 ， 使用 G.711u 時系統的最大吞吐量爲 800 。根據本論文檢驗其程式碼後，推測其最大吞吐量受封包大小影響之原因，是因爲 RTP Proxy 是使用 UDP Socket 的應用程式。使用者層與作業系統核心層之間的資料傳遞，需要記憶體複製的動作。需要複製的記憶體大小在某一個程度以下，所花的時間大致相同，所以 G.729 與 iLBC 測試結果的最大吞吐量的值相同。封包大小超過一定程度後，越大的封包所需的時間越長，因此造成最大吞吐量降低。

在同時通話個數到達最大吞吐量前前， RTP Proxy 的平均延遲隨著同時通話個數增加而上升，是因爲使用 Poll System Call 時系統的效能，會在 Poll System Call 所檢查的檔案描述子 (File Descriptor) 個數增加時降低。根據 [26] 及 [27] 分別在 Solaris 及 Linux 觀察到的實驗結果也發現相同情形，平均延遲可能達到數十秒，與本論文實驗結果於同時通話超過 1000 時類似。對於此問題，目前大部分系統已經提供了效能較佳的 I/O 處理機制，如 FreeBSD 系統的 kqueue [28] 、 Linux 系統的 epoll [29] ，與 Solaris 系統的 /dev/poll [26] 。

同時通話個數超過最大吞吐量時，即圖 4.6 G.711u 結果中同時通話個數大於 800 ，與 G.729 和 iLBC 結果中同時通話個數大於 1000 時。 Socket 緩衝區已經達到飽和，平均延遲爲封包在該系統的 Socket 緩衝區的等待時間，與處理時間的總和。並且平均延遲隨著同時通話個數上升，平均延遲持續上升是受到 Poll System Call 的影響，系統效能變差。

### 4.2.2 NAT 效能測試

爲了解在各種同時通話個數情形下 NAT 對於封包的平均延遲，封包遺失率的影響。本論文設計測試 NAT 效能的環境如圖 4.7 ，比較實驗 1 (圖 4.2) ，少了 SIP Server ，多了 NAT (圖 4.7 ⓔ) ， SmartBits 的 Port 2 (圖 4.7 ⓕ) 以及SIPv6 Analyzer 的 NIC 2 (圖 4.7 ⓚ) 分別連接到 NAT 的 Private Port 1 與 Private Port 2 (圖 4.7 ⓛ 和 ⓜ) 。 NAT 讓其下的使用者可以分享同一個 IPv4 位址存取 IPv4 網路。本實驗使用的 NAT 爲一般的商業產品，本論文所使用 NAT 的廠牌與型號爲 D-Link DI-804HV 與 ZyXEL

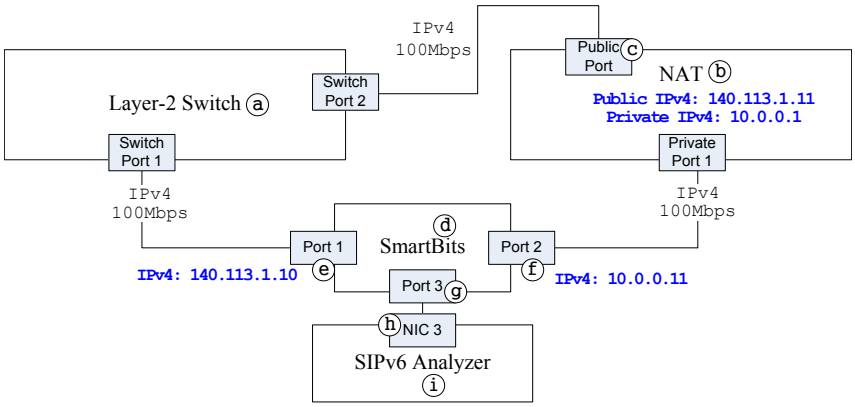


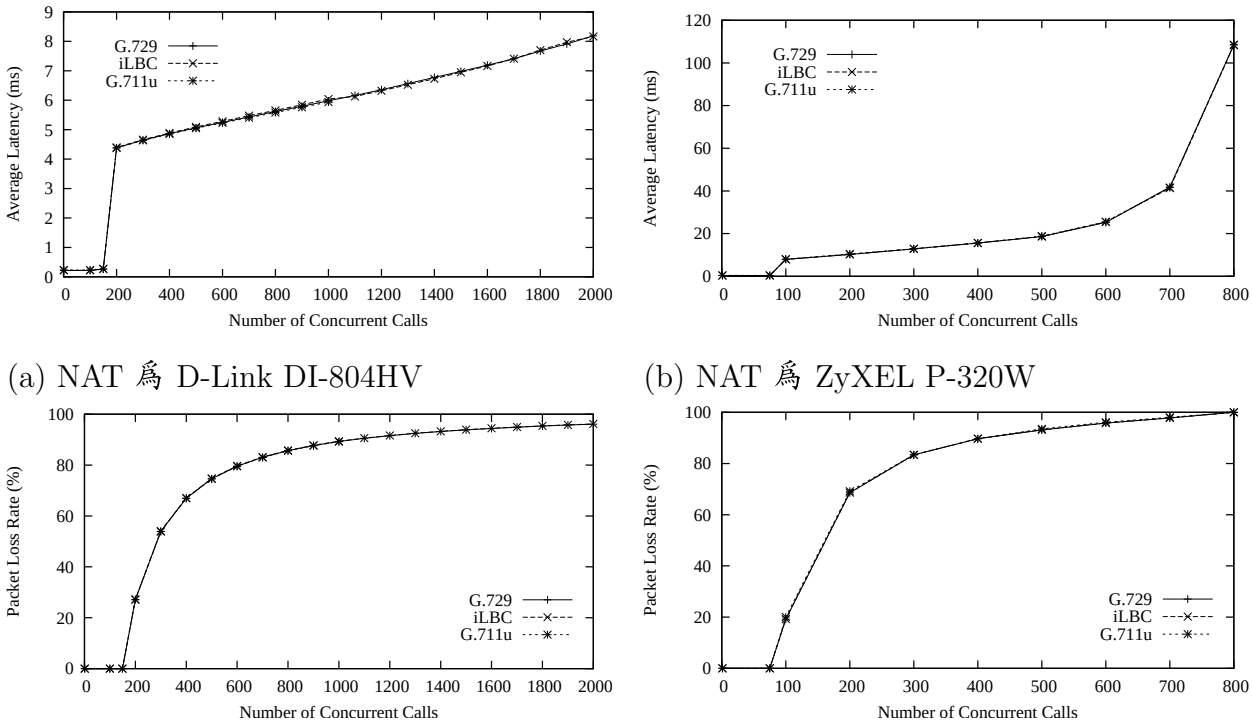

圖 4.7: NAT 效能測試環境

P-320W 。NAT 之 Public IPv4 位址爲 140.113.1.11 ， Private IPv4 位址爲 10.0.0.1 。SmartBits 的 Port 2 具有 IPv4 位址 10.0.0.11 。其餘元件組態同實驗 1 (圖 4.2) 。

實驗流程與實驗 1 (4.1.1) 不同處，在於本實驗不進行 SIP 通話建立流程， SIPv6 Analyzer 直接於 SmartBits 的 Port 2 設定封包串流，串流中的封包其來源傳輸位址爲 10.0.0.11:9000 ，目的傳輸位址爲 140.113.1.10:9000 。每次設定完一個封包串流，則遞增通訊埠以產生新的傳輸位址 (如 140.113.1.10:9001) 。

由於 NAT 是一般的商業產品，本論文無法藉由檢驗其實作方式了解其系統架構，僅能就一般系統設計的方法預測其效能與行爲。測試 D-Link DI-804HV 與 ZyXEL P-320W 兩台 NAT ，的同時通話個數對平均延遲，與封包遺失率的關係如圖 4.8 所示。

D-Link DI-804HV 的實驗結果中 (圖 4.8 (a) 和 (c))，最大吞吐量爲 150 個通話，所有的語音編解碼器測試結果的趨勢皆相同。封包大小對於平均延遲與封包遺失的影響不大，平均延遲彼此之間差距都在 0.1 ms 以下，封包遺失率的差距在 0.5 % 以下。在同時通話個數超過最大吞吐量之前，平均延遲約爲 0.215 ms ，當同時通話個數超過最大吞吐量之後，可以發現平均延遲也上升了 4.2 ms 左右。此一延遲時間爲封包在該系統的封包佇列 (queue) 的等待時間與處理時間的總和。同時通話個數超過最大吞吐量之後，封包遺失率逐漸增加，平均延遲也逐漸上升。此情形與 [30] 所說明的 “Receive Livelock” 現象類似。 Receive Livelock 會發生在一般使用中斷處理 I/O 請求的系統。此類系統中，接收封包的中斷處理函數，其優先權通常高於發送封包或是網路通訊協




(a) NAT 爲 D-Link DI-804HV

(b) NAT 爲 ZyXEL P-320W

(c) NAT 爲 D-Link DI-804HV

(d) NAT 爲 ZyXEL P-320W

圖 4.8: NAT 效能測試結果 — 同時通話個數對平均延遲及封包遺失率之關係

定堆疊 (Network Protocol Stack) 的處理函數。在系統不停收到網路介面卡收到封包的中斷請求時，由於中斷處理函數的優先權較高，因此系統發送封包或是在網路通訊協定堆疊的處理過程中，會被被接收封包的事件中斷。導致封包在發送封包處理函數，或是在網路通訊協定堆疊中的等待時間變長。系統效能下降，反映在封包平均延遲，導致平均延遲會隨著同時通話個數上升。Receive Livelock 的問題可以藉由修改系統的排程器 (Scheduler) ，或是修改驅動程式來改善。

ZyXEL P-320W 的實驗結果 (如圖 4.8 (b) 和 (d)) ，其最大吞吐量爲 75 個通話，封包大小對於平均延遲與封包遺失的影響不大。平均延遲彼此之間差距都在 0.2 ms 以下，封包遺失率的差距在 0.5 % 以下，在同時通話個數小於最大吞吐量之前，平均延遲約 0.36 ms 。與 D-Link DI-804HV 結果類似，在同時通話個數大於最大吞吐量後，平均延遲上升了約 7.7 ms ，之後平均延遲逐漸上升，封包遺失率逐漸上升，與 D-Link DI-804HV 相比，其最大吞吐量較低，造成的延遲較大。而且在同時通話數超過 800 之後，在實驗中只通過了數十個封包，封包遺失率接近 100 % 。此現象符合 [30] 所描述的 “Starvation” 。此一現象是當 Receive Livelock 的現象更嚴重時，系統會無法正常回

應。因爲此情形下的系統效能與一般情形差異太大，因此在接下來的分析當中，本論文將去除 ZyXEL P-320W 測試中同時通話數大於 800 時的實驗結果。

### 4.2.3 NAT 與 SBC 整合效能測試

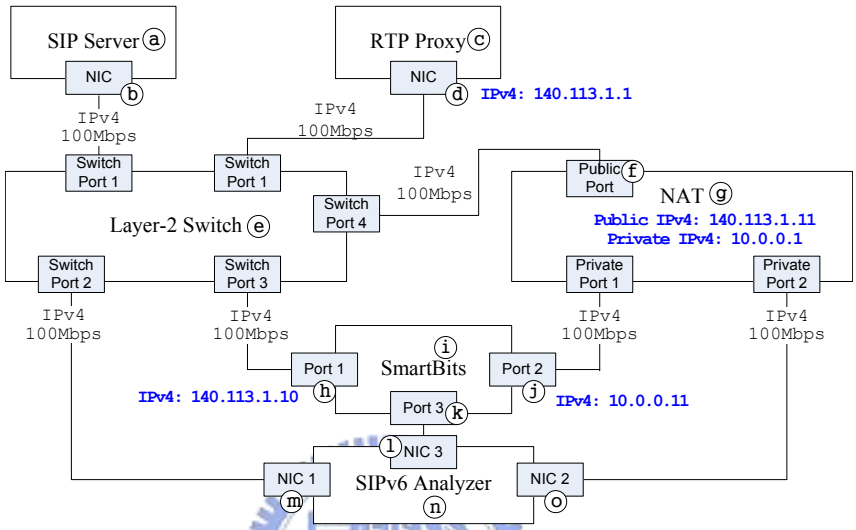
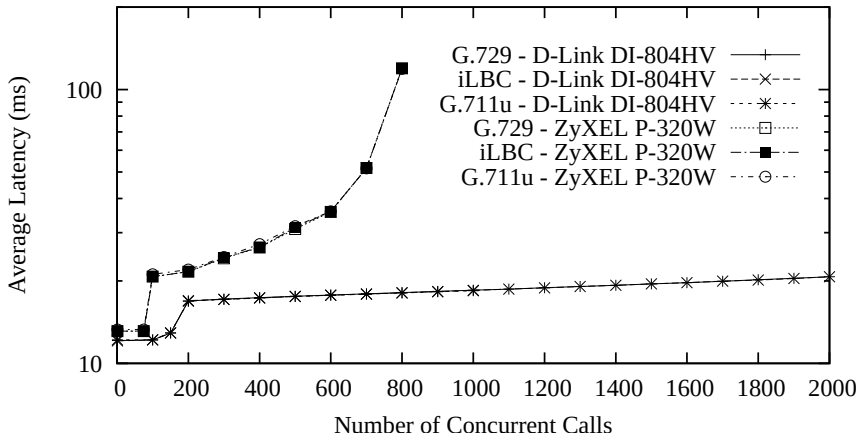



圖 4.9: NAT 與 SBC 整合效能測試環境

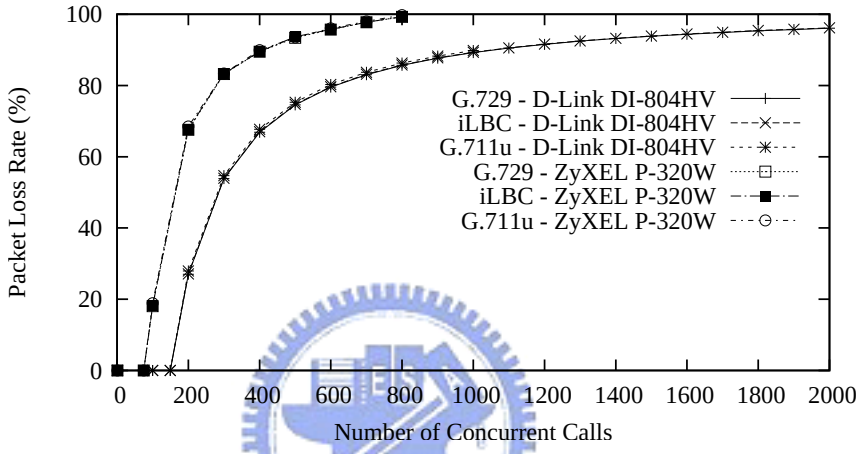
SBC 整合測試的實驗環境如圖 4.9。在了解 NAT 與 RTP Proxy 後，本論文將元件整合進行 SBC 環境的測試，以通話個數對平均延遲、封包遺失率，以及 MOS 的影響來評估使用 SBC 環境的效能。

實驗環境如圖 4.9 所示，比起 NAT 效能測試環境 (圖 4.7) 多加了 SBC ， SBC 包含兩個元件， SIP Server (圖 4.9 ⓐ) 與 RTP Proxy (圖 4.9 ⓒ) 。作爲解決 NAT 使用環境下， SIP/RTP 穿越 NAT 的解決方案，以 RTP Proxy 轉送 RTP 封包。其他元件的功能與 NAT 效能測試 (圖 4.7) 相同。本實驗的元件組態如下：

SIP Server 具有 IPv4 位址，透過網路控制 RTP Proxy ， RTP Proxy 具有 IPv4 位址 140.113.1.1 。 SmartBits 的 Port 1 與 Port 2 (圖 4.9 ⓗ 與 ⓙ) IPv4 位址分別爲 140.113.1.10 與 10.0.0.11 。SIPv6 Analyzer NIC 1 與 NIC 2 (圖 4.9 ⓚ 、 ⓜ) 具有 IPv4 位址。




(a) 同時通話個數對平均延遲之關係圖




(b) 同時通話個數對封包遺失率之關係圖

圖 4.10: NAT 與 SBC 整合效能測試結果

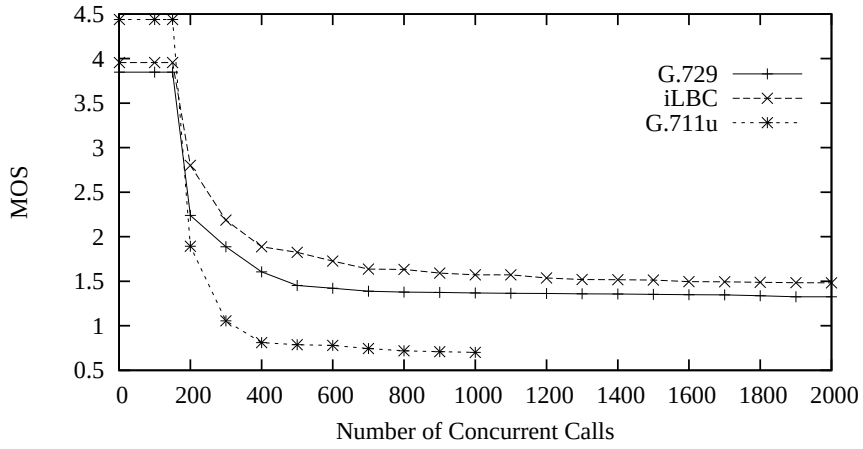
實驗流程同實驗 1 (4.1.1) ，使用 D-Link DI-804HV 爲 NAT 的測試結果如圖 4.10 (a) 和 (b) 標示爲 D-Link DI-804HV 之結果。由於 RTP 封包是由 SmartBits 的 Port 2 送往 Port 1 ，因此觀察到的最大吞吐量會與 NAT 有關，平均延遲爲兩者的加總，在同時通話個數超過 NAT 的最大吞吐量時，平均延遲的上升主要會受 NAT 影響， RTP Proxy 造成的延遲大致不變。 RTP Proxy 的平均延遲於同時通話個數爲 100 時約爲 12.8 ms ， D-Link DI-804HV 於此情形下的平均延遲約爲 0.215 ms ， 在比對圖 4.6 與圖 4.8 (a) 同時通話個數 100 之結果，可以發現在此情形的平均延遲主要來自於 RTP Proxy 。當同時通話個數到達 200 時，從圖 4.8 (a) D-Link DI-804HV 的結果發現，平均延遲上升了約 4.2 ms ，此時同時通話個數超過了 NAT 的最大吞吐量，通過 NAT 到達 RTP Proxy 的 RTP 封包，受到 NAT 的最大吞吐量限制而不再增加。因此平均延遲上升主要是因爲 NAT 所造成， RTP Proxy 造成的平均延遲仍然維持在大約 12.8 ms 。

之後平均延遲逐漸上升，亦爲NAT 造成的結果。比較封包遺失率的結果，圖 4.10 (a) 與圖 4.8 (c) 可以發現，兩者的封包遺失率相符，證實實驗結果的確是受 NAT 影響。

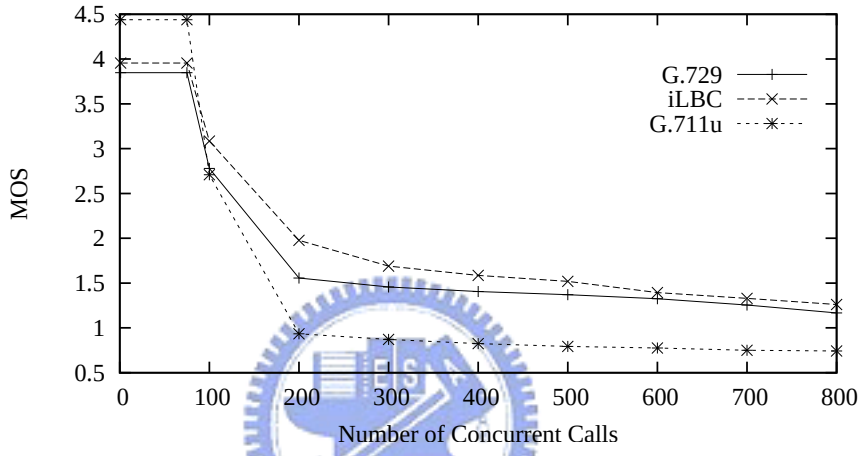
使用 ZyXEL P-320W 爲 NAT 的結果如圖 4.10 (a) 和 (b) 標示爲 ZyXEL P-320W 之結果，同樣在同時通話個數未超過 NAT 的最大吞吐量 (75) 時，在此情形下 NAT 的平均延遲約爲 0.36 ms ，平均延遲主要由 RTP Proxy 造成，約 12.8 ms 。所以在圖 4.10 (a) 可以發現， NAT 使用 ZyXEL P-320W 的平均延遲，比起 NAT 使用 D-Link DI-804HV 的平均延遲高，是因爲 ZyXEL P-320W 造成的延遲較 D-Link DI-804HV 爲高。同時通話個數達 100 時，已經超過了 ZyXEL P-320W 的最大吞吐量，平均延遲上升了 7.7 ms ，與圖 4.8 (b) 的結果相符，且隨著同時通話個數上升，平均延遲上升的趨勢也相同。比較封包遺失率的結果 (圖 4.8 (d) 與 圖 4.10 (b) ZyXEL P-320W 之結果)，趨勢亦相同。但使用 ZyXEL P-320W 的效能較差。

根據以上結果本論文推測，如果測試時的封包由 SmartBits 的 Port 1 送往 Port 2 ，由於 RTP Proxy 的最大吞吐量較高，所以通過 RTP Proxy 的同時通話個數可能超過 NAT 的最大吞吐量。因爲測試過程中會遞增同時通話個數，可以分爲三個部分來預測最後的測試結果，(1) 同時通話個數都未超過 NAT 與 RTP Proxy 的最大吞吐量。(2) 同時通話個數超過 NAT 的最大吞吐量。 (3) 同時通話個數超過 NAT 與 RTP Proxy 的最大吞吐量。首先探討 (1) ，此情形與本實驗的測試結果相同，平均延遲主要來自 RTP Proxy (12.8 ms) 。 (2) 的情形，也會與本實驗相同，因爲超過了 NAT 的最大吞吐量，所以會在 NAT 上觀察到其佇列所造成的延遲。最後是 (3) ，因爲封包會先到 RTP Proxy ， RTP Proxy 所造成的延遲 (以使用 G.711u 的測試結果爲例) ，爲 712 ms 。此時通過 RTP Proxy 到達 NAT 的封包會受到 RTP Proxy 的限制，但是 RTP Proxy 的最大吞吐量大於 NAT ，通過的同時通話個數大於 NAT 的最大吞吐量。所以會發現 NAT 所造成的延遲 (以 ZyXEL P-320W 爲例) 約爲 108 ms ，最後測量到的平均延遲應爲兩者的總和，而最後結果的最大吞吐量會受到 NAT 的限制。

NAT 使用 D-Link DI-804HV 時，最後輸出的 MOS 值如圖 4.11 (a) ， MOS 值隨著封包遺失率上升而下降。在沒有封包遺失的情形下 (同時通話個數不超過 NAT 的最




(a) NAT 爲 D-Link DI-804HV




(b) NAT 爲 ZyXEL P-320W

圖 4.11: NAT 與 SBC 整合效能測試結果 – 同時通話個數對 MOS 之關係圖

大吞吐量時) ，其 MOS 值與實驗 1 (4.1.2) 相同，使用 G.729 、 iLBC ，以及 G.711u 三種編解碼器的 MOS 分別爲 3.848 、 3.956 與 4.44 ，使用 G.711u 的 MOS 值最佳。超過最大吞吐量時受到 NAT 影響， NAT 使用 D-Link DI-804HV 的結果顯示，當同時通話個數 200 以上，經 PESQ 評估的 MOS 值都已經低於 3 (Fair) ，封包遺失率爲 27 % 以上。而 NAT 使用 ZyXEL P-320W 的結果如圖 4.11 (b) ，同時通話個數未超過最大吞吐量時， MOS 值與實驗 1 (4.1.2) 相同。在同時通話個數 100 以上時，除了使用 iLBC 仍有 3 以上的 MOS 值，使用其他語音編解碼器之 MOS 已經低於 3 (Fair) ，此時封包遺失率在 18 % 以上。從兩個不同 NAT 的測試結果發現，在封包遺失率高的時候，各種語音編解碼器的 MOS 由高到低排列分別是 iLBC 、 G.729 ，以及 G.711u 。

## 4.3 實驗 3 – IPv6-in-IPv4 通道使用環境之效能評估

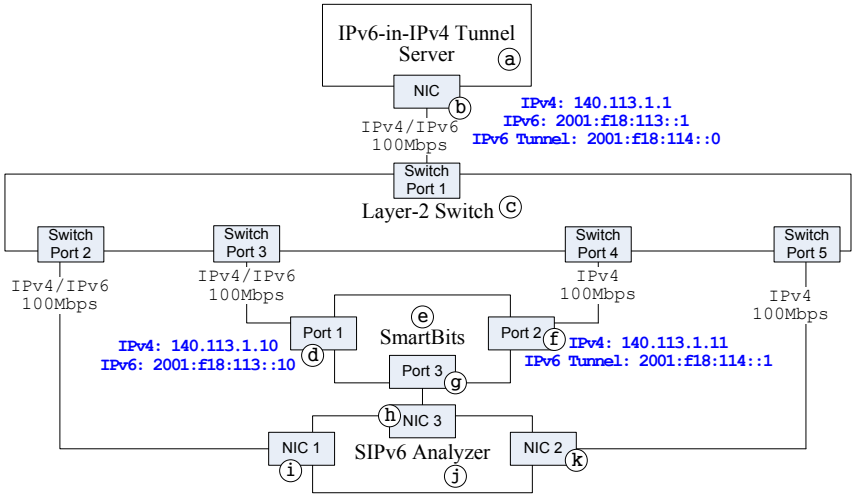



圖 4.12: IPv6-in-IPv4 通道效能測試環境

實驗 3 評估 RTP 封包經過 Tunnel Server 時，同時通話個數對平均延遲、封包遺失率與 MOS 的關係。實驗環境如圖 4.12 所示，比實驗 1 (圖 4.2) 多了 Tunnel Server (圖 4.12 ⓐ) 。 Tunnel Server 接受客戶端的通道建立之請求，建立 IPv6-in-IPv4 通道介面 (Interface) 並提供 IPv6 位址給客戶端，以提供 IPv6 連線能力給 IPv4 網路下之使用者。其餘元件的功能皆與實驗 1 相同。Tunnel Server 所使用的 IPv6-in-IPv4 通道介面爲 Linux 系統的 SIT (Simple Internet Transition) 介面，其 IPv4 位址爲 140.113.1.1 ，IPv6 位址爲 2001:f18:113::1 而 IPv6-in-IPv4 通道的位址爲 2001:f18:114::0 。 SmartBits 的 Port 2 (圖 4.12 ⓓ) 模擬爲通道端點 (Endpoint) ，其 IPv4 位址爲 140.113.1.11 ，IPv6-in-IPv4 通道的位址爲 2001:f18:114::1 ，利用 IPv4 封包搭載 IPv6 之封包。其餘元件組態與實驗 1 相同。

### 4.3.1 實驗 3 實驗流程

首先 SIPv6 Analyzer 利用 TSP (Tunnel Setup Protocol) 與 Tunnel Server 溝通，並且建立 IPv6-in-IPv4 通道介面。 IPv6-in-IPv4 通道建立完成之後， SIPv6 Analyzer 使用得到的 IPv6-in-IPv4 通道介面進行 SIP 通話建立，第一次 SIP 通話建立流程爲 UAC 與 UAS 產生的傳輸位址分別 [2001:f18:113::10]:9000 與 [2001:f18:114::1]:9000 ，並於每

次通話建立完成後遞增 UDP 通訊埠，以產生下次通話建立流程使用的傳輸位址。完成通話建立流程後 SIPv6 Analyzer 設定 SmartBits 的 Port 2 產生 IPv6-in-IPv4 通道的封包串流，封包串流的封包使用 IPv4 承載 IPv6 封包，IPv4 目的位址爲 140.113.1.10，IPv4 來源位址爲 140.113.1.11，其上層的 IPv6 RTP 封包，目的與來源傳輸位址根據 SIP 訊息中 SDP 所攜帶的傳輸位址，其餘流程同實驗 1 (4.1.1)。接下來藉由檢驗 Linux 系統的程式碼，分析 IPv6-in-IPv4 Tunnel Server 的系統架構，並以實驗環境的設定與實驗流程爲例，說明其運作方式。

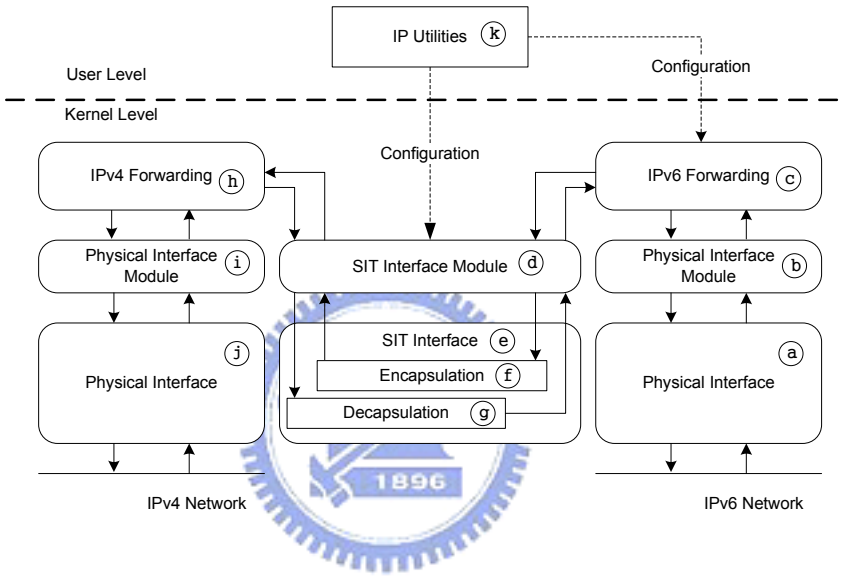



圖 4.13: IPv6-in-IPv4 Tunnel Server 架構圖

IPv6-in-IPv4 Tunnel Server 的系統架構圖如圖 4.13。Tunnel Server 根據 IPv6-in-IPv4 通道端點的 IPv4 位址，透過 IP Utilities (圖 4.13 ⓚ) 設定 SIT 介面，作業系統根據 IP Utilities 的設定，透過系統的 SIT Interface Module (圖 4.13 ⓓ) 產生出一個虛擬的網路介面卡，即 SIT 介面 (圖 4.13 ⓔ)，並分配 IPv6 位址給此介面。SIT 介面爲封包進行封裝 (Encapsulation，圖 4.13 ⓕ) 與解封裝 (Decapsulation，圖 4.13 ⓖ) 的動作。由 IPv4 Forwarding 與 IPv6 Forwarding (圖 4.13 ⓗ 與圖 4.13 ⓒ) 決定如何繞送封包。以下根據實驗環境 (圖 4.12) 的設定與實驗流程爲例，說明 Tunnel Server 運作方式：

客户端 (Client) 透過 TSP 與 Tunnel Server 交換訊息，客户端請求 Tunnel Server 所支援的通道模式 (如 IPv6-in-IPv4 通道) 與其所支援的認證方式。接下來進行客户

端的身分認證，選擇 Tunnel Server 所支援的認證方式進行認證。成功後客户端將據本端的 IPv4 位址資訊填入通道建立請求，並送給 Tunnel Server 。 Tunnel Server 從通道建立請求中發現客户端可以使用 IPv6-in-IPv4 通道，因此 Tunnel Server 使用客户端的 IPv4 位址建立 IPv6-in-IPv4 通道介面，並且修改 IPv6 路由表 (Routing Table) 使 Tunnel Server 可以正確將 IPv6 封包繞送給 IPv6-in-IPv4 通道端點。 IPv6-in-IPv4 通道設定完畢後，此時 Tunnel Server 的 IPv6 路由表簡化如表 4.3 所示。包含四個欄位， Index 、 Destination 、 Gateway ，以及 Interface ，每筆資料擁有一個唯一的 Index 值。Forwarding 機制決定如何繞送封包，首先檢查封包的目的位址是否爲本端的位址。若是本機位址，則依據 IP 標頭中的 Protocol 欄位，決定將封包交給何種處理函數，如本實驗中是交給 SIT Interface Module。若目的位址非本端位址，則以循序的方式比對封包中的目的位址與路由表中的 Destination 欄位，直到找到擁有最長字首相符 (Longest Prefix Matching) 的資料。找到相符資料後，判斷如何繞送封包，若 Gateway 欄位中有指定值，則表示系統無法直接遞送此封包，此封包將送藉由 Interface 欄位中指定的介面送出，並且由 Gateway 欄位所指定的路由器 (Router) 繞送；反之，系統直接遞送封包至 Interface 欄位指定的介面。

表 4.3: IPv6-in-IPv4 Tunnel Server IPv6 路由表範例

| Index | Destination | Gateway | Interface |
|---|---|---|---|
| 1 | any | Default | Physical Interface ⓐ |
| 2 | 2001:f18:114::/127 | — | SIT |

由 SmartBits 的 Port 2 送來的 IPv6-in-IPv4 通道封包，其 IPv4 來源位址爲 140.113.1.11 ，目的位址爲 140.113.1.1 ，標頭中 Protocol 欄位標示爲 41 。 Physical Interface Module (圖 4.13 ⓘ) 將此封包交給 IPv4 Forwarding 機制處理，由於此封包的 IPv4 目的位址爲 Tunnel Server 的 IPv4 位址， IPv4 Forwarding 機制判定爲交給本端的封包，並且根據標頭中的 Protocol 欄位得知需將此封包透過 SIT Interface Module ，交給 SIT 介面處理。 SIT 介面將此封包解封裝後交給 IPv6 Forwarding 機制。由於此封包來自 IPv6-in-IPv4 通道端點送來的 IPv6-in-IPv4 通道封包，其 IPv6 來源位址爲 2001:f18:114::1 ，其目的爲 SmartBits 的 Port 1 ， IPv6 位址爲 2001:f18:113::10 。 SIT 介面再將此封包交給 IPv6 Forwarding 機制，查詢後可知道符合第 1 筆資料，利用

Physical Interface ⓐ 送出封包，交由 IPv6 的預設路由 (Default Route) 繞送此 IPv6 封包。

### 4.3.2 實驗 3 實驗結果

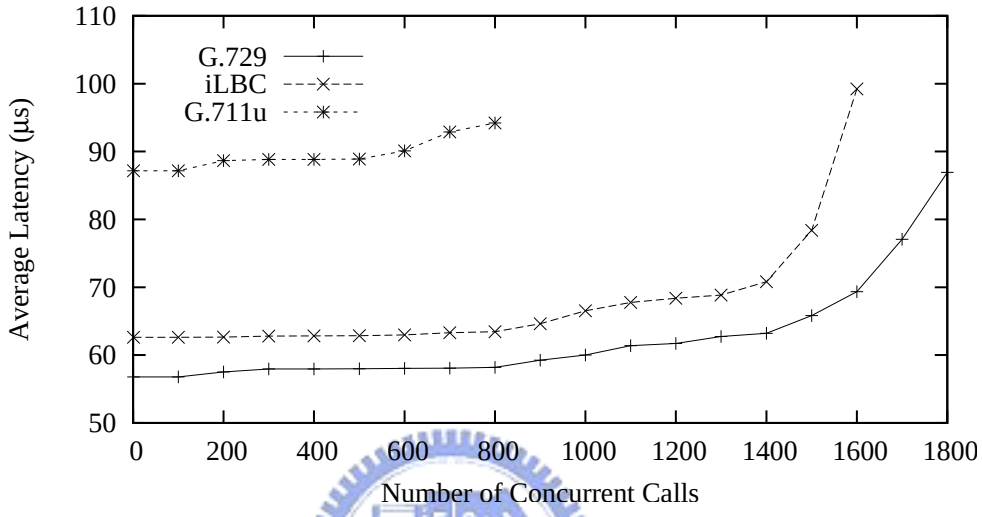



圖 4.14: IPv6-in-IPv4 通道環境效能測試結果 – 同時通話個數對平均延遲之關係圖

由於沒有封包遺失，因此經由 PESQ 評估的 MOS 值與實驗 1 (4.1.2) 相同。同時通話個數對平均延遲之關係如圖 4.14，較長的封包平均延遲較長，若與實驗 1 的 Pure IPv6 測試結果比較，不同語音編解碼器之間平均延遲的差異，比起實驗 1 來得大。由此推斷所觀察到平均延遲的差異，不僅僅是因爲傳輸延遲造成，而需考慮處理延遲所造成的差異。另外，系統受到 Receive Livelock 的影響，隨著同時通話個數增加，平均延遲逐漸上升。若與使用者層的程式 Portaone RTP Proxy 比較，由於 IPv6-in-IPv4 Tunnel Server 所有的動作都在作業系統核心層完成，而不需要將資料複製到使用者層，也不需要逐一檢查 UDP Socket 的狀態是否可供讀取，因此效能比 Portaone RTP Proxy 好，因爲 IPv6-in-IPv4 Tunnel Server 的平均延遲比 Portaone RTP Proxy 的平均延遲低。

## 4.4 實驗 4 – IPv6-in-IPv4 UDP 通道使用環境之效能評估

IPv6-in-IPv4 UDP 通道使用環境下 RTP 封包會經過 NAT 與 Tunnel Server 兩個元件，因此本實驗與實驗 2 (4.2) 相同，先行對各元件單獨進行效能測試，再進行整合環境測試。 NAT 的效能測試結果已於實驗 2 (4.2.2) 完成，因此本節不再重複。

### 4.4.1 IPv6-in-IPv4 UDP Tunnel Server 效能測試

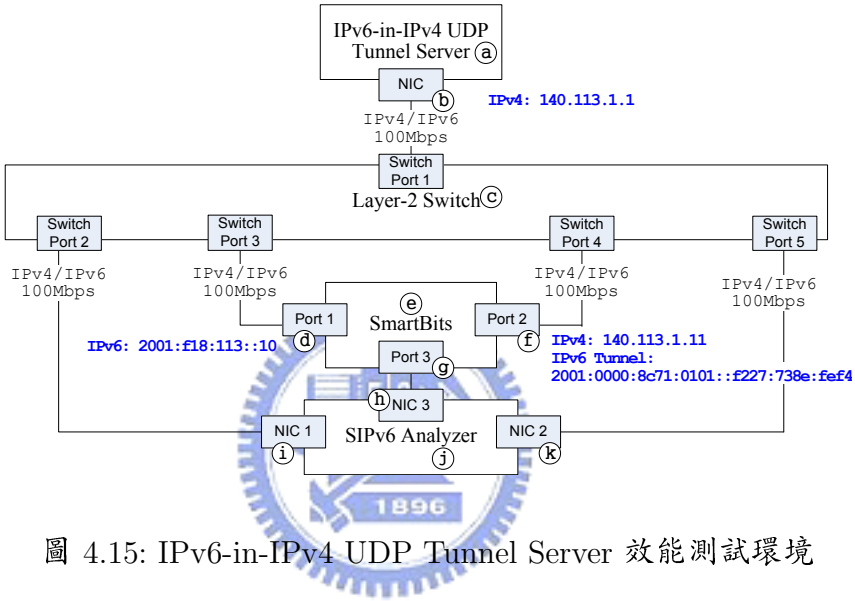



圖 4.15: IPv6-in-IPv4 UDP Tunnel Server 效能測試環境

單獨測試 Miredo Teredo Relay 效能之環境如圖 4.15 ，與實驗 1 (4.1) 不同處在於本實驗沒有 SIP Server ，而加入 Tunnel Server (圖 4.15 ⓐ) 。由於目前一般的使用者所使用的 Microsoft Windows XP 系統， 內建有 Teredo 的機制，因此本論文選擇 Teredo Server/Teredo Relay 作爲本環境下的 Tunnel Server ，並以開放原始碼的實作 Miredo Teredo Server/Relay [15] ，作爲本次測試的待測系統。 Tunnel Server 藉由 Teredo [10] 機制，爲 NAT 下之使用者提供 IPv6 之連線能力。 Tunnel Server 使用傳輸位址 140.113.1.1:3544 作爲 Teredo Relay 之通訊埠， Tunnel Server 具有 IPv6 位址，以進行 IPv6 封包的繞送。SmartBits 的 Port 2 模擬爲實際環境中的 NAT 之 Public Port ，具有 IPv4 位址 140.113.1.11 ，以 IPv4 UDP 封包搭載 IPv6 之封包，並且預先設定 IPv6 位址爲一 Teredo 位址 2001:0000:8c71:0101::f227:738e:fef4 。

實驗流程同實驗 2 NAT 效能測試之流程，但本實驗所產生是 IPv6-in-IPv4 UDP 的封包串流， SIPv6 Analyzer 直接於 SmartBits 的 Port 2 設定封包串流，封包串流中的封包使用 IPv4 UDP 承載 IPv6 之封包，其 IPv4 UDP 的來源傳輸位址爲 140.113.1.11:3544 ，目的傳輸位址爲 140.113.1.1:3544 。所承載的 IPv6 RTP 封包的來源傳輸位址爲 [2001:0000:8c71:0101::f227:738e:fef4]:9000 ，目的傳輸位址爲 [2001:f18:113::10]:9000 ，並於每次設定完一個封包串流時遞增通訊埠，以產生新的傳輸位址。接下來藉由檢驗 Miredo Teredo Relay 的程式碼，了解其系統架構，並以實驗環境與流程爲例，了解 Miredo Teredo Relay 的運作方式。

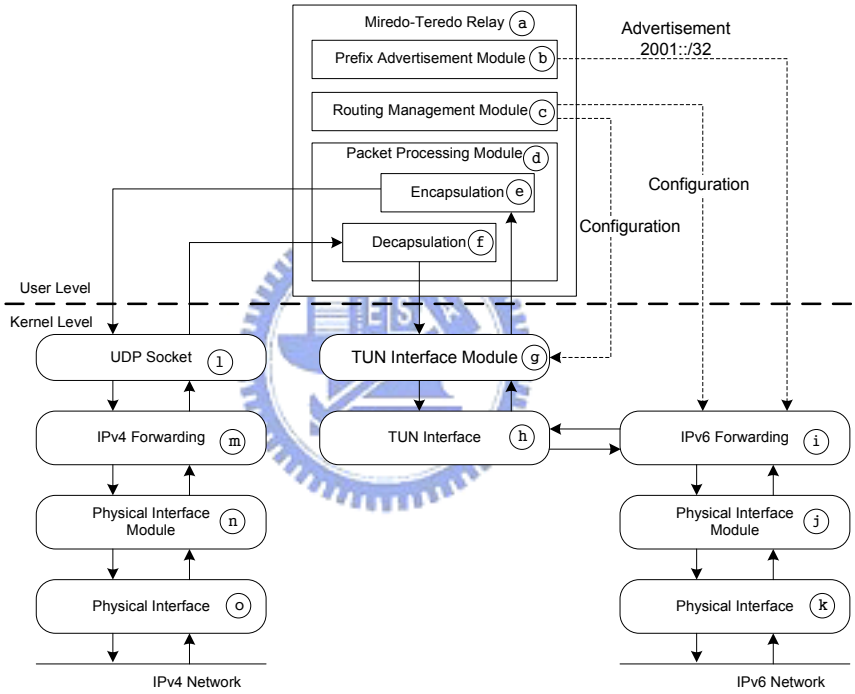



圖 4.16: Miredo Teredo Relay 系統架構圖

圖 4.16 爲 Miredo Teredo Relay 之系統架構圖， Miredo Teredo Relay 藉由 UDP Socket (圖 4.16 ⓛ) 及 TUN Interface (圖 4.16 ⓗ) 轉送封包。 TUN Interface Module (圖 4.16 ⓖ) 爲 Linux 系統上的虛擬點對點裝置 (Virtual Point-to-Point Device) 驅動程式，透過 TUN Interface Module 所產生的介面，應用程式可以從 TUN Interface Module 讀取或寫入 IP 封包，藉此實作 IPv6-in-IPv4 UDP 通道。來自 IPv6-in-IPv4 UDP 通道端點的通道封包，經過使用者層的 Packet Processing Module (圖 4.16 ⓓ) 解封裝 (圖 4.16 ⓕ) 後，藉由 TUN Interface (圖 4.16 ⓗ) 連接 IPv6 網路通訊協定堆疊，

並藉由 IPv6 Forwarding 機制將封包送往 IPv6 網路；送往 IPv6-in-IPv4 UDP 通道端點的 IPv6 封包，在 Packet Processing Module 透過 TUN Interface Module 將封包讀出後，進行封裝 (圖 4.16 ⓔ) 以 UDP Socket 送到 IPv4 網路。以下用實驗環境與流程的組態爲例，說明 Miredo Teredo Relay 的運作方式。

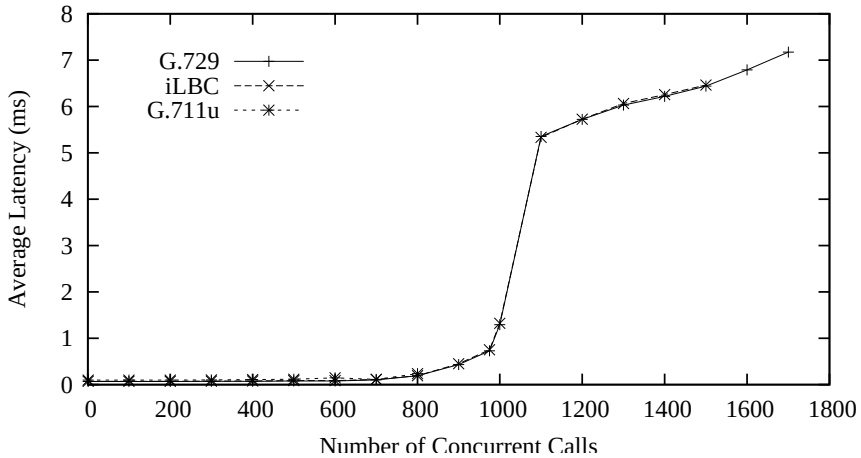
Miredo Teredo Relay 在啓動時，經由 Routing Management Module (圖 4.16 ⓒ) 設定 IPv6 路由表，與 TUN Interface Module 。將 Teredo [10] 所定義的 Teredo Service Prefix (即2001::/32) 這個網路的封包，設定爲使用 TUN Interface 送出。 IPv6 路由表如表 4.4 。同時經由 Prefix Advertisement Module (圖 4.16 ⓑ) ，向 IPv6 網路發送路由廣告 (Routing Advertisement) ，讓其他在 IPv6 網路上的節點知道如何送 IPv6 封包到 SmartBits 的 Port 2 。

表 4.4: Miredo Teredo Relay IPv6 路由表範例

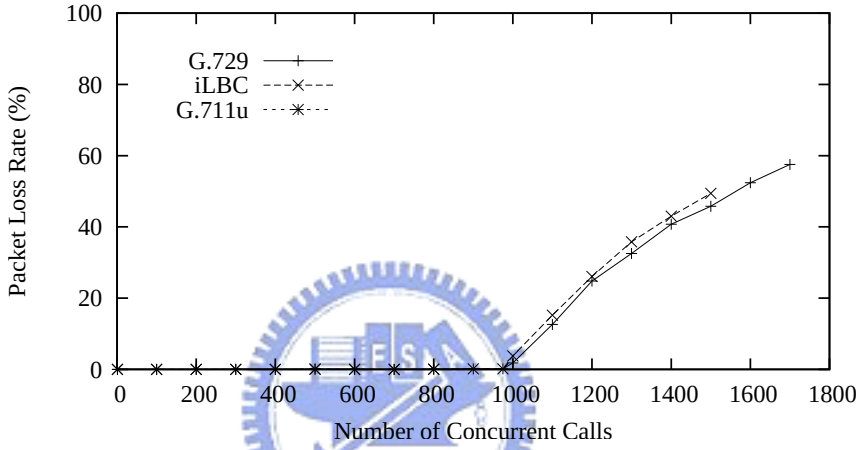
| Index | Destination | Gateway | Interface |
|---|---|---|---|
| 1 | any | Default | Physical Interface ⓚ |
| 2 | 2001::/32 | | TUN |

當 Miredo Teredo Relay 收到 SmartBits 的 Port 2 發送到 SmartBits 的 Port 1 的 IPv6-in-IPv4 UDP 通道封包時， IPv6-in-IPv4 UDP 通道封包的目的傳輸位址爲 140.113.1.1:3544 ，系統得知是給本端的 UDP Socket (圖 4.16 ①) ， Miredo Teredo Relay 的 Packet Processing Module 將封包進行解封裝後，將封包透過 TUN Interface Module 寫入 TUN Interface ，系統此時查詢 IPv6 路由表，符合路由表的第一筆資料，於是透過 Physical Interface (圖 4.16 ⓚ) 將封包送往 SmartBits 的 Port 1 。

Miredo Teredo Relay 效能測試結果如圖 4.17 ，爲同時通話個數對平均延遲與封包遺失率的關係。其中使用 G.711u 之測試結果，在同時通話個數達到 800 時，已經達到了線路可負載的上限，在這之前都不會出現封包遺失的現象。 G.729 與 iLBC 的最大吞吐量相同，因爲將作業系統核心層的記憶體資料複製到使用者層，在資料大小未超過某一程度時，所需的時間大致相同，因此 G.729 與 iLBC 所測出來的最大吞吐量相




(a) 同時通話個數對平均延遲之關係圖




(b) 同時通話個數對封包遺失率之關係圖

圖 4.17: Miredo Teredo Relay 效能測試

同。同時通話個數超過最大吞吐量之後平均延遲上升，且開始出現封包遺失。 Miredo Teredo Relay 的效能比同樣爲使用者層程式的 RTP Proxy 好，是因爲 RTP Proxy 維護多個 UDP Socket 的狀態，使用 Poll System Call 檢查 UDP Socket 的狀態， Poll System Call 使系統效能在同時通話個數上升時會有明顯的下降；反之， Miredo Teredo Relay 只需要維護一個 UDP Socket 。

而比較 Miredo Teredo Relay 與 IPv6-in-IPv4 Tunnel Server 的效能時，可以發現 IPv6-in-IPv4 Tunnel Server 的效能較佳。因爲 Miredo Teredo Relay 是一個使用者層的程式，使用 UDP Socket 收送資料皆需要將資料複製到作業系統核心層，或是從作業系統核心層複製資料到使用者層。因此使得 Miredo Teredo Relay 比 IPv6-in-IPv4 Tunnel Server 差。如果能適當將部分工作由使用者層移至作業系統核心層，則可以避免封包

資料於使用者層與作業系統核心層之間的複製，並增加系統的效能。

### 4.4.2 NAT 與 IPv6-in-IPv4 UDP Tunnel Server 整合效能測試

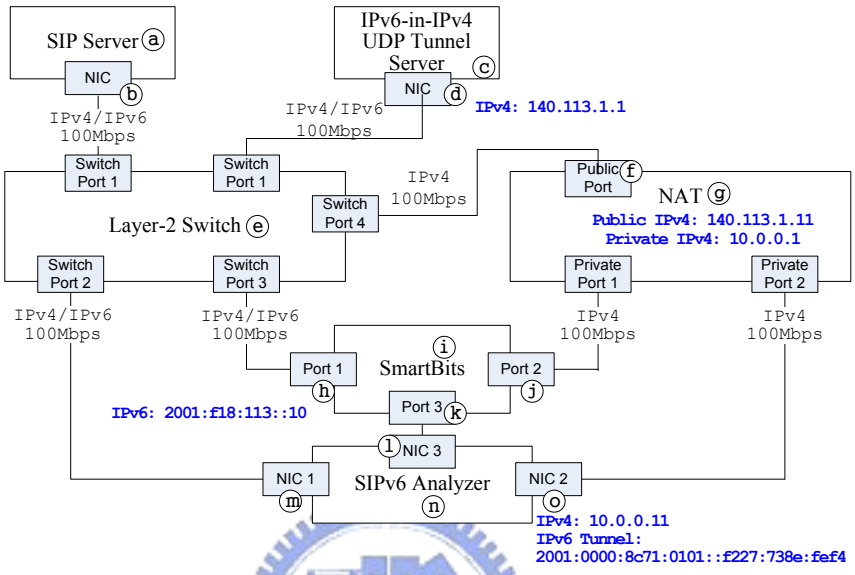
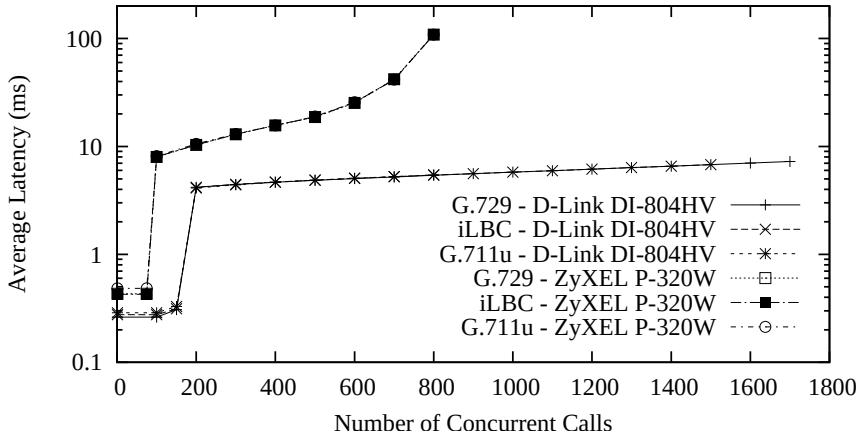



圖 4.18: NAT 與 IPv6-in-IPv4 UDP Tunnel Server 整合效能測試環境

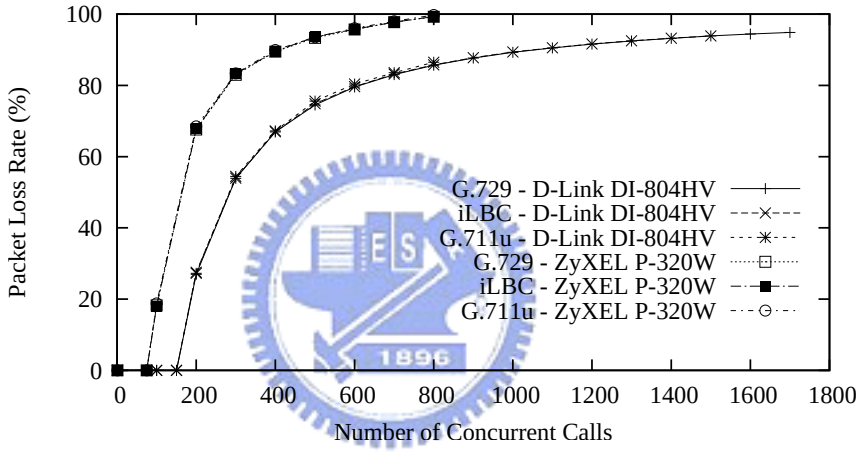
NAT 與 IPv6-in-IPv4 UDP Tunnel Server 整合效能測試環境如圖 4.18 所示，除了本環境多了 SIP Server (圖 4.18 ⓐ) 與 NAT (圖 4.18 ⓖ) 之外，其餘設定同 IPv6-in-IPv4 UDP 通道效能測試之環境 (圖 4.15)。SIPv6 Analyzer 的 Port 2 模擬爲 NAT 下的 Teredo 客户端，IPv4 位址設定爲 10.0.0.11 ，經由 Teredo 獲得的 IPv6 位址爲 2001:0000:8c71:0101::f227:738e:fef4 。NAT 的 Public IPv4 位址爲 140.113.1.11 ，Private IPv4 位址爲 10.0.0.1 。

實驗流程除了 SIPv6 Analyzer 的 NIC 2 先透過 Teredo 機制取得 IPv6 通道位址。於是 SIP 通話建立流程中，SIPv6 Analyzer 的 NIC 2 送出的 SIP 200 OK 訊息，其 SDP 所攜帶的傳輸位址由 IPv6 通道位址與 UDP 通訊埠 9000 產生 (即 [2001:0000:8c71:0101::f227:738e:fef4]:9000) 。完成通話建立流程後，SIPv6 Analyzer 於 SmartBits 的 Port 2 產生 IPv6-in-IPv4 UDP 通道的封包串流，串流中封包的 IPv4 目的傳輸位址爲 140.113.1.1:3544 ，來源傳輸位址爲 10.0.0.11:3544 ，所承載的 IPv6 RTP 封包的目的傳輸位址爲 [2001:f18:113::10]:9000 ，來源傳輸位址爲

[2001:0000:8c71:0101::f227:738e:fef4]:9000 ，接下來的實驗流程同實驗 1 (4.1.1) 。




(a) 同時通話個數對平均延遲之關係圖

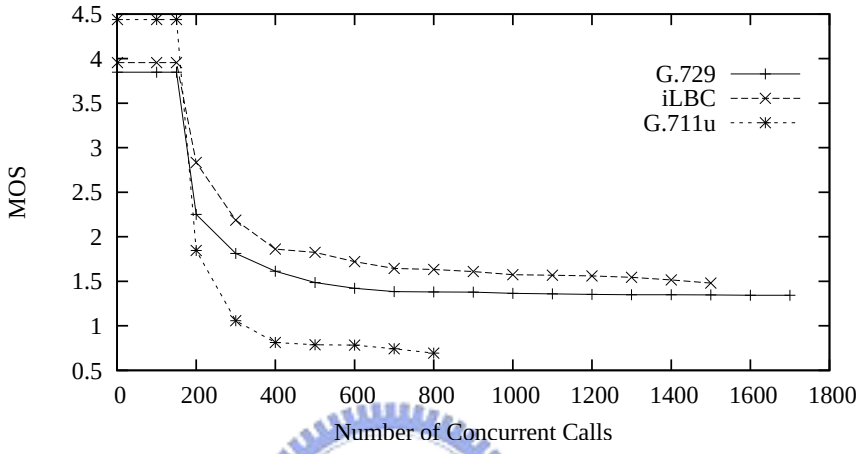



(b) 同時通話個數對封包遺失率關係圖

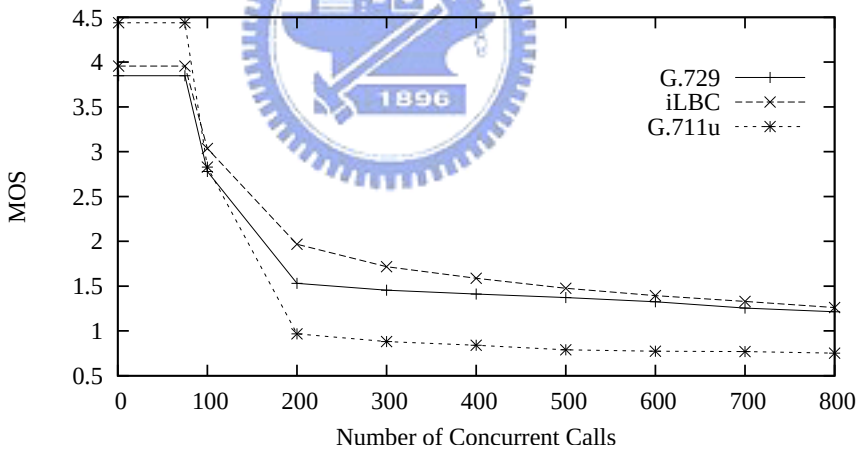
圖 4.19: NAT 與 IPv6-in-IPv4 UDP 通道整合效能測試結果

最後評估 UAS 透過 NAT 使用 Teredo 的環境的通話品質。同時通話個數對平均延遲與封包遺失率的關係如圖 4.19 (a) 和 (b) 。由於 RTP 封包是由 SmartBits 的 Port 2 傳向 Port 1 ，在同時通話個數到達 NAT 的最大吞吐量時，仍未超過 Miredo Teredo Relay 的最大吞吐量，因此封包遺失是 NAT 所造成的。Miredo Teredo Relay 在同時通話個數 200 個以下時，使用 G.729 、 iLBC ，以及 G.711u 的測試結果其平均延遲分別爲 67 $\mu$s 、 70 $\mu$s ，以及 97 $\mu$s (參考圖 4.17) 。而在同時通話個數超過最大吞吐量之前，實驗用的兩個 NAT 所造成的平均延遲分別爲 213 $\mu$s ，以及 370 $\mu$s (參考圖 4.8 (a) 與 (c)) 。比較此實驗結果，於圖 4.19 可發現，平均延遲由 NAT 及 Miredo Teredo Relay 共同貢獻。在超過 NAT 的最大吞吐量之後，通過 NAT 送往 Mireo Teredo Relay

的封包由於受限於 NAT 之最大吞吐量，同時通話個數不會再增加。實驗用的兩個 NAT 所造成的平均延遲增加分別爲 4.2 ms 與 7.7 ms (參考圖 4.8 (a) 與 (c))，比較圖 4.19 D-Link DI-804HV 的結果，同樣是在超過 150 個同時通話時，平均延遲上升了 4.2 ms ， ZyXEL P-320W 在超過 75 個同時通話時，平均延遲上升了 7.7 ms 。此一平均延遲主要是由 NAT 產生。




(a) NAT 爲 D-Link DI-804HV

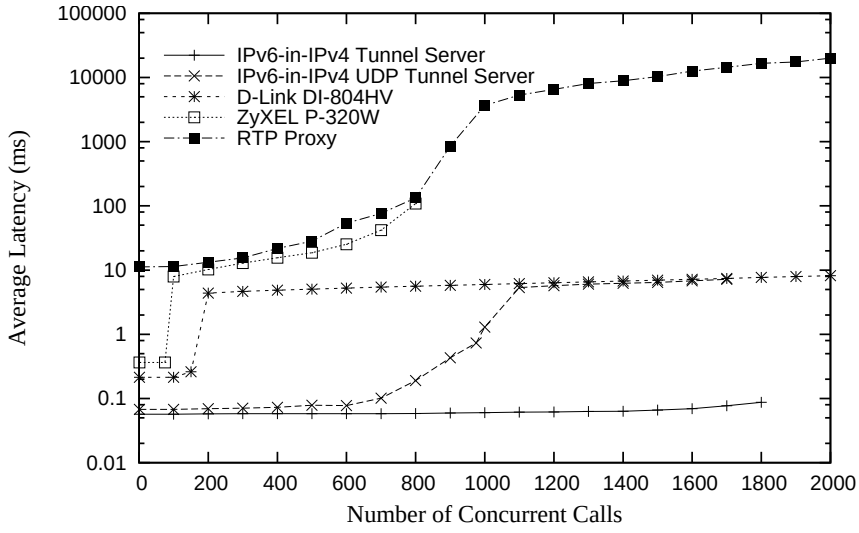



(b) NAT 爲 ZyXEL P-320W

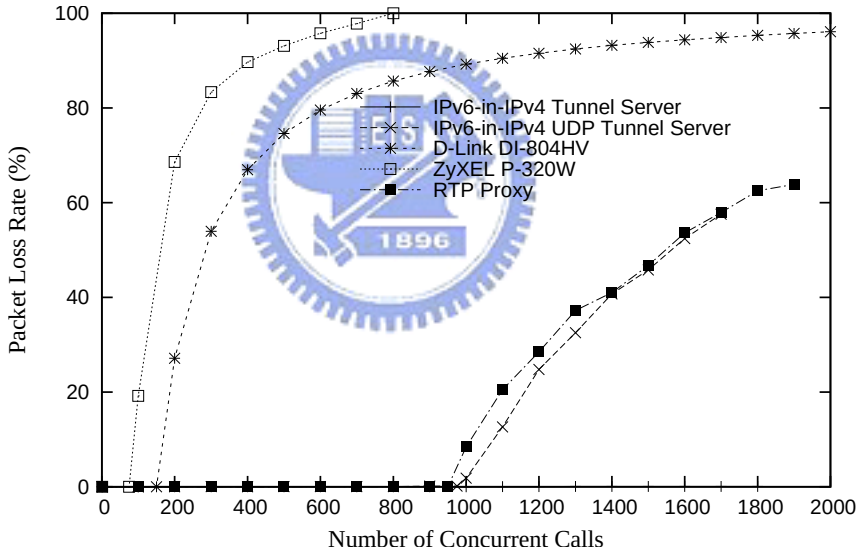
圖 4.20: IPv6-in-IPv4 UDP 通道環境效能測試結果 – 同時通話個數對 MOS 之關係圖

在本實驗環境下，由於封包遺失是由 NAT 所造成，因此同時通話個數對 MOS 之關係圖 (圖 4.20 (a) 與 (b)) ，結果與實驗 2 的 NAT 與 SBC 整合效能測試所得到的結果 (圖 4.11) 類似。

## 4.5 各元件之效能比較




(a) 同時通話個數對平均延遲之關係圖




(a) 同時通話個數對封包遺失率之關係圖

圖 4.21: 各元件效能比較

本節進行各環境中的元件進行效能的比較，取出前述四個實驗中 G.729 的測試結果所畫成圖 4.21 (a) 與 (b) ，分別是同時通話個數對平均延遲與封包遺失率的關係。從這些元件的效能可以比較 VoIP 服務中解決 IPv4 位址不足的方案的效能，IPv6-in-IPv4 Tunnel Server 所有動作在作業系統核心層完成，因爲不需核心層與使用者層之間的記憶體複製，所以其平均延遲最低，且在測試過程中也沒有封包遺失產生。 IPv6-in-IPv4 UDP Tunnel Server 平均延遲次低，爲使用者層的應用程式，效能較

IPv6-in-IPv4 Tunnel Server 差，而且測試過程中會有封包遺失。一般 NAT 也會造成效能上的損失，不論是平均延遲或是封包遺失率。最後是 RTP Proxy ，其平均延遲爲所有元件中最高的，雖然封包遺失率與 IPv6-in-IPv4 UDP Tunnel Server 接近，但是因爲要維護多個 Socket 的狀態，所以其平均延遲表現最差。由本論文的實驗結果可以發現，在目前的 IPv4 位址不足的解決方案中， IPv6 元件的效能皆比使用NAT的方案來得好。

# 第五章　結論與未來工作

本論文爲 SIPv6 Analyzer 擴充了播放 RTP 串流的語音及影像之功能，藉由此功能可以分析 IMS 的多媒體服務。搭配 Jitter Buffer 的模擬，了解在不同的 Jitter Buffer 設定之下，語音及影像的品質爲何。提供了完整的封包分析功能。

並於第三章擴充了 SIPv6 Analyzer 之架構，提供一個專門爲 VoIP 語音品質測試所設計的語音品質模組，利用傳送多個封包串流的方式，模擬多個使用者進行通話的情形，測試待測系統之效能，可以了解各元件於提供 VoIP 服務時的品質及其效能。因爲傳統的測試方式是傳送大量封包，並沒有 VoIP 通話中， RTP 封包皆是以固定時間傳送的特性。 SIPv6 Analyzer 並利用軟體提供 IPv6 通道與 SIP 信令的功能，能夠提供一般硬體較缺乏的信令溝通之功能。

利用第三章的成果，本論文於第四章使用此模組評估 VoIP 服務中，解決 IPv4 位址不足的方案的效能，以及其通話品質，並加以比較。根據第四章得到的實驗結果，VoIP 服務中使用 IPv6 作爲解決 IPv4 位址不足的方案，比起使用 NAT 解決 IPv4 位址不足並搭配 SBC 解決 SIP/RTP 穿越 NAT 的效能來得好。如果佈建 IPv6 網路考慮全面使用 Pure IPv6 ，需要花費許多經費升級，或是新增硬體及線路。然而本論文所測試的兩種 IPv4/IPv6 轉換機制，其實也能提供不錯的效能，如果能夠搭配這些轉移機制，於現階段以 IPv4 網路爲主的環境下，一方面增加 Pure IPv6 網路的佈建，同時佈建轉移機制所需的元件，利用已經運作良好的 IPv4 網路，提供服務品質不比 IPv4 網路所能提供的差的情形下，增加使用者對於使用 IPv6 的接受度，推廣 IPv6 的使用。

本論文擴充 SIPv6 Analyzer 的語音、影像播放功能，以及模擬 Jitter Buffer 的功

能後，贏得2004 年日本 IPv6 Appli-Contest 2004 實作組冠軍。由於本論文所研發的語音品質評估模組，使用 SIPp 產生 SIP 信令，未來可以將產生的信令，回饋到 SIPv6 Analyzer 中進行分析。而語音品質量測的部分，因爲本系統搭配 SmartBits 的硬體平台發送與接收封包，可以很精確的記錄封包的傳送與接收之時間，未來可以結合 E-Model 與 PESQ 評估方式，系統於測試結束後得到的平均延遲，可以輸入 E-Model，並將 PESQ 所計算的 MOS 值，回饋到 E-Model 。整合 E-Model 與 PESQ 兩種評估標準的優點，提供更完整的資訊給使用者。

# 文 獻


[1] Methods for subjective determination of transmission quality. *ITU-T Recommendation P.800*, August 1996.

[2] The E-model, a computational model for use in transmission planning. *ITU-T Recommendation G.107*, March 2005.

[3] Perceptual evaluation of speech quality (PESQ): an objective method for end-to-end speech quality assessment of narrow-band telephone networks and speech codecs. *ITU-T Recommendation P.862*, February 2001.

[4] WinPcap: the Free Packet Capture Library for Windows. "`http://winpcap.polito.it/`".

[5] Ethereal: A Network Protocol Analyzer. "`http://www.ethereal.com/`".

[6] Handley, M. and Jacobson, V. and Perkins, C. SDP: Session Description Protocol. RFC 4566, Internet Engineering Task Force, July 2006.

[7] Schulzrinne, H. and Casner, S. and Frederick, R. and Jacobson, V. RTP: A Transport Protocol for Real-Time Applications. RFC 3550, Internet Engineering Task Force, July 2003.

[8] D. Wing. Symmetric RTP/RTCP. Internet-Draft, April 2007. draft-wing-behave-symmetric-rtprtcp-03.

[9] M. Blanchet. Tunnel Setup Protocol (TSP): A Control Protocol to Setup IPv6 or IPv4. Internet-Draft, December 2002. draft-vg-ngtrans-tsp-01.

[10] C. Huitema. Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs). RFC 4380, Internet Engineering Task Force, February 2006.

[11] Software tools for speech and audio coding standardization. *ITU-T Recommendation G.191*, September 2005.

[12] Andersen, S. and Duric, A. and Astrom, H. and Hagen, R. and Kleijn, W. and Linden, J. Internet Low Bit Rate Codec (iLBC). RFC 3951, Internet Engineering Task Force, December 2004.

[13] Coding of speech at 8 kbit/s using conjugate-structure algebraic-code-excited linear prediction (CS-ACELP). *ITU-T Recommendation G.729*, January 2007.

[14] Gateway6. "`http://www.remlab.net/miredo/`".

[15] Miredo : Teredo IPv6 tunneling for Linux and BSD. "`http://www.remlab.net/miredo/`".

[16] SIPp. "`http://www.remlab.net/miredo/`".

[17] J.C. Bolot. End-to-end packet delay and loss behavior in the internet. *ACM SIGCOMM Computer Communication Review*, 23(4):289–298, 1993.

[18] Y. Amir, C. Danilov, S. Goose, D. Hedqvist, and A. Terzis. 1-800-OVERLAYS: using overlay networks to improve VoIP quality. *Proceedings of the international workshop on Network and operating systems support for digital audio and video*, pages 51–56, 2005.

[19] S. Khan, S. Duhovnikov, E. Steinbach, M. Sgroi, and W. Kellerer. Application-driven cross-layer optimization for mobile multimedia communication using a common application layer quality metric. *International Conference On Communications And Mobile Computing*, pages 213–218, 2006.

[20] L. Ding and RA Goubran. Assessment of effects of packet loss on speech quality in VoIP. *Haptic, Audio and Visual Environments and Their Applications, 2003. HAVE 2003. Proceedings. The 2nd IEEE Internatioal Workshop on*, pages 49–54, 2003.

[21] iLBCfreeware.org Project Homepage. http://www.ilbcfreeware.org/.

[22] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460, Internet Engineering Task Force, December 1998.

[23] P. Srisuresh and K. Egevang. Traditional IP Network Address Translator (Traditional NAT). RFC 3022, Internet Engineering Task Force, January 2001.

[24] G. Camarillo, R. Penfield, A. Hawrylyshenm, and M. Bhatia. Requirements from SIP (Session Initiation Protocol) Session Border Control Deployments. Internet-Draft, October 2006. draft-camarillo-sipping-sbc-funcs-05.

[25] SIP Express Router. "http://www.iptel.org/ser".

[26] B. Chapman. Polling Made Efficient in the Solaris 7 OS. "http://developers.sun.com/solaris/articles/polling_efficient.html", May 2002.

[27] M. Welsh. SandStorm I/O Core Benchmarks. "http://www.eecs.harvard.edu/~mdw/proj/seda/iocore-bench/index.html", January 2001.

[28] J. Lemon. Kqueue: A Generic and Scalable Event Notification Facility. pages 141–154, 2001.

[29] /dev/epoll Home Page. "http://www.xmailserver.org/linux-patches/nio-improve.html".

[30] J. C. Mogul and K. K. Ramakrishnan. Eliminating Receive Livelock in an Interrupt-Driven Kernel. *ACM Transactions on Computer Systems*, 15(3):217–252, 1997.