國立暨南國際大學通訊工程研究所

碩士論文

具備分散式特色之網路電話加壓測試

A Distributed Approach for VoIP Stress Testing

指導教授：吳坤熹博士

研究生：蔡政霖

中華民國九十八年 六月

致謝

兩年過去了，意味著研究所生涯即將正式的畫上句點。本論文能夠順利完成，承蒙指導教授 吳坤熹老師的悉心教導，在學術研究上給予我許多建議，並指引學生朝正確的方向前進，才使得此論文順利完成，在此學生致上最誠摯的謝意。

此外，要感謝同窗好友以及學弟妹們：文仁、嘉裕、筱婷、汎嘉、韋霖、霓雅、瑋勵、韋勳、韋立、信富，在這一、兩年間的相互砥礪及鼓勵，除了研究上的協助之外，也因為有你們，增添了許多歡笑與快樂，謝謝各位這些日子的照顧。

最後，要感謝我親愛的家人，若沒有你們的一路扶持與栽培，沒有今日的我，謹以小小的成就與你們分享。

論文名稱：具備分散式特色之網路電話加壓測試

校院系:國立暨南國際大學通訊工程研究所　　頁數: 54

畢業時間：98 年 6 月　　學位別：碩士

研究生：蔡政霖　　指導教授:吳坤熹博士

中文摘要

隨著數位科技的蓬勃發展，網際網路的使用大幅增加，進而發展出各種不同類型的網路應用服務，其中 Voice over Internet Protocol (VoIP) 是受到注目的產業之一。它相較於傳統 Public Switched Telephone Network (PSTN)，能夠大幅降低通話成本，在市場中具備競爭力。而 VoIP 使用者近幾年有增加的趨勢，為了提供用戶穩定的語音品質，進行 VoIP 伺服器的壓力測試也成為重要的議題。在本文中，我們在 Unix 作業系統下發展出免費的 DIST-V (Distributed Stress Test for VoIP)測試軟體，以分散式的環境，藉由多台測試用電腦建立數百通的會話(session)，傳送大量的 Real-time Transport Protocol (RTP)封包，進而對 VoIP 伺服器進行壓力測試，並且將測量結果轉換成 Mean Opinion Model (MOS)值，將結果提供業者選擇 VoIP 設備的評估。

關鍵詞：分散式工具(Distributed Tools)、即時傳輸協定(RTP)、會議初始協定(SIP)、壓力測試(Stress Test)、網路電話(VoIP)

Title of Thesis：A Distributed Approach for VoIP Stress Testing
Name of Institute：Graduate Institute of Communication Engineering,
National Chi Nan University Pages： 54
Graduation Time：06/2009 Degree Conferred：Master
Student Name：Cheng-Lin Tsai Advisor Name：Quincy Wu

English Abstract

In this thesis, we proposed a distributed tool for VoIP which performs the stress test of audio stream delivery, by sending Real-time Transport Protocol (RTP) packets from distributed testing nodes. An implementation on Unix was developed to illustrate how the multiple testing nodes can be utilized to synchronously launch the large volume of audio streams in a stress test. Furthermore, the testing results are represented in Mean Opinion Score (MOS) to help Internet Service Providers (ISPs) evaluating suitable VoIP equipment.

Keyword: Distributed Tools, RTP, SIP, Stress Test, VoIP

目錄

圖目錄

表目錄

1. 動機

網際網路的蓬勃發展加上通訊科技的日新月異，電話通訊的應用在網際網路也成為一項熱門的產業，其中 Voice over Internet Protocol (VoIP) 是一種透過 Internet 或其他使用 Internet Protocol (IP) 的網路來進行語音傳輸的技術，相較於傳統的 Public Switched Telephone Network (PSTN)，具有大幅節省通話及建設成本的特點，因此極具有競爭力。

語音通話最重要的是即時性以及通話品質，PSTN 是以電路交換(circuit switching)的技術進行傳輸，每通通話均擁有一條專屬頻寬而不受外界干擾。相反地 VoIP 應用在網際網路上，是以封包交換(packet switching)的技術達成，因此當使用者開始大幅成長，所衍生出來的問題是，通話過程中 VoIP 伺服器容易因無法負荷過多的線上使用者而產生封包遺失、時間延遲，進而影響到通話品質。假若有一個上千人的企業需添購一台專用的 VoIP 伺服器，機構內的使用者都必需透過它與外界進行 VoIP 通話，所選購的 VoIP 伺服器能否保持穩定的運作使通話者保有良好的通話品質，成為我們所關心的議題。

為了對伺服器進行壓力測試，以評估是否符合企業的需求，一般來說，伺服器都會採用穩定度高且較昂貴的硬體設備，為了要進行壓力測試，會購買如 SmartBits[1]、NuStreams[2][3]這類的測試設備進行測試。SmartBits 是 Spirent Communication 所開發的產品，被公認為網路產品和技術的最佳測試工具。但是 SmartBits 的價格至少是新台幣 100 萬元，而 NuStreams 價格也超過新台幣 40 萬元，相當的昂貴。儘管這類的測試設備功能相當齊全，但是 VoIP 伺服器絕大多數處理的封包為會話初始協議(Session Initial Protocol, 簡稱 SIP)[4]以及即時傳輸協議(Real-time Transport Protocol, 簡稱 RTP)[5][6]兩種封包，以現有的開放源碼即可產生兩種格式的封包模擬撥打電話

的動作；因此在價錢的考量上，若能設計一套專為 VoIP 伺服器量身訂做的測試軟體，以模擬實際通話的情形，便能開發出一套具有價格競爭力的壓力測試工具。

因此本文最主要的動機，就是為了要降低測試的成本。我們針對 VoIP 伺服器，發展出一套“免費”的分散式網路電話流量測試工具 Distributed Stress Test for VoIP ，簡稱 DIST-V；搭配數台桌上型或筆記型電腦所組成的測試用電腦，依照 VoIP 通話建立的方式，同步產生上百通的會話(session)，藉此產生大量的 RTP 音訊流(audio stream)，對 VoIP 伺服器進行壓力測試，並且計算壓力測試時的 Mean Opinion Model (MOS)給測試者做參考。當測試完畢之後，便可以將測試用電腦回歸成一般的電腦使用，因此不需另外添購 Smartbits、NuStreams 這類昂貴的設備。

接下來的章節中，第二章介紹 VoIP 所需知的相關背景，包含 SIP、RTP、會話描述協定(Session Description Protocol, 簡稱 SDP)[7]等的介紹，以及 MOS[8]、E-Model[9]的計算方式。第三章介紹 DIST-V 測試軟體所用到的 API、fping[10]等工具、以及與時間同步的問題，第四章將介紹 DIST-V 系統架構與實作成果，最後第五章為結論及未來方向。

2. 背景知識與相關研究

2.1 Session Initiataion Protocol (SIP)

會話初始協議(Session Initial Protocol, 簡稱 SIP)，是由網際網路工程任務小組(Internet Engineer Task Force, IETF)制定的媒體會議訊息通訊協定。SIP 是一種點對點(Peer to Peer)的通訊協定，透過 Uniform Resource Identifier (URI)來命名位址與使用純文字化的格式來傳遞訊息，主要是用以規範一個或多個參與者的終端設備如何建立、修改和結束會話(session)的一種標準協定。SIP 相關的網路設備也被歸類為客戶端與伺服器端，其中客戶端被稱為使用者代理(User Agent, 簡稱 UA)，它又分為使用者代理客戶端(User Agent Client, 簡稱 UAC) 以及使用者代理伺服器端(User Agent Server, 簡稱 UAS)，兩者的差別為 UAC 發起一個 SIP 的通訊要求，而 UAS 收到 SIP 通訊要求時，會回覆通訊訊息。

伺服器端分別有代理伺服器(Proxy Server)、轉址伺服器(Redirect Serve r)、註冊伺服器(Registrar Server)，其中又以代理伺服器最為重要。代理伺服器扮演類似中間人的角色，當它接到客戶端的會話請求時，將視受話端狀態，將 SIP 請求轉送至目的地。而客戶端利用 Proxy Server 建立會話後，再傳送 RTP 封包來進行語音訊通訊。另外，為了交換客戶端連線內容的負載情況和相關參數，SIP 使用了 SDP 來描述相關終端設備的特性資訊。

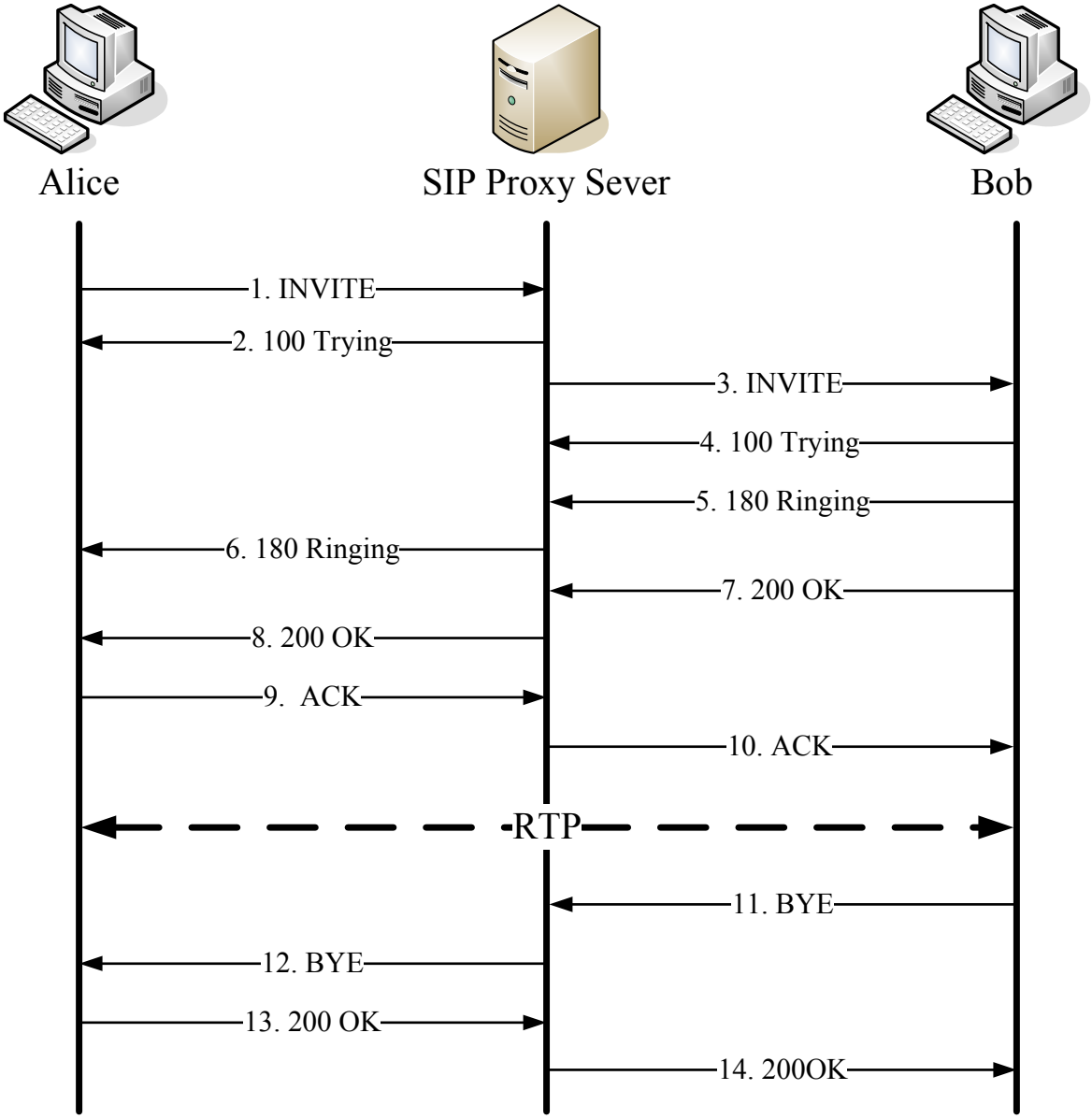

圖 1： 一般性的 SIP 語音應用架構

在圖 1 中，我們描繪出一個典型的 SIP 語音應用架構。當 Alice (UAC)要和 Bob (UAS)進行語音通訊時，其會話建立的過程如下：

1. 首先，Alice 要邀請 Bob 進行通話時，Alice 會先發出 INVITE 請求給 SIP Proxy Server。
2. SIP Proxy Server 會回送一個 100 Trying 訊息給 Alice，告知正在處理 INVITE 請求。

3. SIP Proxy Server 收到 Alice 所發出 INVITE 請求後，會轉送給 Bob。
4. Bob 收到 INVITE 請求，就會回送一個 100 Trying 訊息給 SIP Proxy Server，告知正在處理請求。
5. Bob 處理完 INVITE 請求後，會開始響鈴，並回送一個 180 Ringing 訊息給 SIP Proxy Server。
6. 當 SIP Proxy Server 收到 180 Ringing 訊息後會轉送給 Alice。
7. 當 Bob 接起話筒時，代表答應這個邀請，就會送出一個 200 OK 訊息給 SIP Proxy Server。
8. 當 SIP Proxy Server 收到此 200 OK 訊息後會轉送給 Alice。
9. Alice 收到 200 OK 訊息之後，得知對方接受邀請，於是發出 ACK 訊息給 SIP Proxy Server 作最後確認。
10. SIP Proxy Server 收到 ACK 訊息後會轉送給 Bob，當 Bob 收到最後 ACK 的確認後，表示雙方完成呼叫連線。之後 Alice 與 Bob 不需透過 SIP Proxy Server，可直接建立 RTP 流。
11. 若 Bob 要結束通話時，則發出一個 BYE 請求給 SIP Proxy Server。
12. SIP Proxy Server 收到 BYE 請求後轉送給 Alice。
13. Alice 收到 BYE 請求時，則會發出 200 OK 訊息給 SIP Proxy Server。
14. SIP Proxy Server 收到後轉送給 Bob，當 Bob 收到 200 OK 訊息後，雙方結束連線。

2.2 Real-time Transport Protocol (RTP)

即時傳輸協議(Real-time Transport Protocol, 簡稱 RTP)是為了處理具有即時特性的資料而訂定的一個通訊協定，用來支援在 IP 網路中進行傳輸即時資料，目前在網際網路廣泛用於即時多媒體串流(multimedia stream)的傳輸。RTP 最早定義在 RFC

1889[5]中，之後又在 RFC 3550[6]針對傳輸規則及演算法部份加以修訂，以處理大量串流及穿越 Network Address Translator (NAT)[11]等議題。RTP 傳輸的資訊廣泛應用於多媒體部份，例如：聲音點播 (Audio-on-Demand)、影視點播(Video-on-Demand)、網路電話(Internet Telephony)、電視會議(Video conferencing)。

RTP 表頭的大小是 12 bytes，如圖 2，它包含 RTP 的版本、Timestamp、Sequence Number 以及 Payload Type 等。Timestamp 是反應出 RTP 資料封包的第一個位元取樣的瞬間，可用來計算來回時間(round trip time)、抖動時間偏移(jitter)。Sequence Number 的初始值是隨機的，每送出一個 RTP 封包時，Sequence Number 會加一，接收端可以依據 Sequence Number 來判斷是否有遺失封包的情形。Synchronization Source (SSRC) 則用來標示每一個 RTP 串流擁有獨一無二的識別資訊。Payload Type (PT)用來定義資料為何種格式，此欄位長度為 7 bits，因此可支援 128 種不同的媒體格式，像是 PCM (A-law and μ-law)、G.729 及 GSM，列出常見的 Payload Type，舉例來說：Payload Type 為 0 則為 PCM μ-law 語音編碼，Payload Type 為 18 則為 G.729 語音編碼。為了使目的端能夠順利的將音頻信號(audio)資訊正確的解碼，接收端必需被事先通知，而這個資訊將會填入 SDP 參數中做表示。

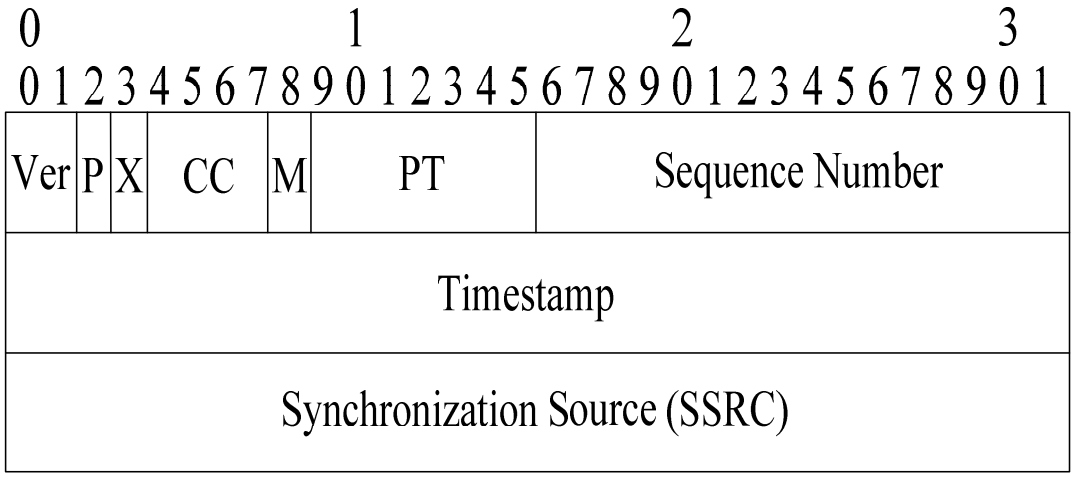

圖 2： RTP 表頭

表 1： RTP Payload Type 與 Codec 對照表

PT	Codec
0	PCMU
3	GSM
8	PCMA
18	G.729

2.3 Session Description Protocol (SDP)

為了交換會話內容的負載情況和相關參數，SIP 使用了 SDP 來描述相關終端設備的特性資訊。SDP 跟 SIP 一樣是以純文字化格式為基礎的通訊協定，用來描述多媒體會話(multimedia session)和其他形式的會話初始化協議；但是，SDP 本身並沒有指定通話建立與結束的動作，因此在 VoIP 裡，必須與其他協定(如 SIP) 合併使用。

SDP 設計的原理相當簡單且重要，它使得擁有不同媒體型態和編碼格式的通話兩端有機會進行溝通。表 2 列出一些重要的 SDP 參數：

表 2： SDP 參數

v	SDP 版本
o	連線的來源端訊息
s	會話名稱
c	連線訊息
m	媒體名稱及傳送位址
a	媒體編碼型式和相關支援能力參數

依據表格的資訊，其中參數 c 的連線訊息包括網路的型態、IP 位址的型態以及

連線的 IP 位址。參數 m 包含支援的編解碼器，媒體型態以及媒體通訊埠。當會話建立時，將會依照 SDP 所提供的資訊，提供通話兩端建立多媒體會話。

2.4 RTP Proxy

由於 IP 位址日益不足，因此現行的網際網路中充斥著 Network Address Translator (NAT)。NAT 定義在 RFC 1631[11]，它主要的功能是提供多台擁有私有(private)IP 位址的電腦使用一個公用(public) IP 位址即可連接上網際網路，以解決 IPv4 位址不夠的情形。當 VoIP 的使用者進行通話前會先使用 SIP 信令建立通話，其中“INVITE”與“200 OK”信令會夾帶 SDP 欄位。SDP 欄位中最重要的是本地端的 IP 位址與媒體通訊埠，假若建立通話的雙方皆使用公用 IP 位址，RTP 封包將依照 SDP 的資訊做點對點的傳遞，並不會造成通話上的困擾；但是當通話雙方均處在不同的 NAT 後面，SDP 則帶有本地端私有 IP 位址與媒體通訊埠，RTP 封包若依照 SDP 給予的資訊傳遞，將無法橫跨網際網路將封包傳遞到另一個 NAT 下的使用者。

因此，RTP Proxy 就扮演轉傳的角色，當 NAT 後面的使用者要進行通話前，依照 SIP 協定必須先以 INVITE”與“200 OK”訊息交換彼此的 IP 位址。SIP Proxy 收到“INVITE”與“200 OK”訊息時，則會修改其中 SDP 的“c”與“m”欄位，置換為 RTP Proxy 的 IP 位址與媒體通訊埠，將原本點對點的傳送轉變由 RTP Proxy 幫忙轉送。在通話的建立過程中，SIP Proxy Server 僅需要處理信令(signal)的部份，而 RTP Proxy Server 則需要持續處理 RTP 音訊流，直到結束通話；比起 SIP Proxy Server，RTP Proxy Server 的負荷相對較重，也是本論文主要進行壓力測試的對象。

2.5 MOS Value and E-Model

Mean Opinion Score (MOS)是 ITU-T P.800 [8]標準中所制定的「平均意見分數」主觀測試方法，評分由最差的一分到最好的五分， MOS 值的量測方式是透過多位受話

測試者聆聽些許聲音片段下，給予主觀的評分。此方法耗費成本與人力，最重要的它沒有辦法真正反應經由網路傳輸所造成的損害，並且無法評估通話延遲所帶來的影響，因此近來的趨勢是採用 ITU-T G.107[9]制定的 E-Model 作為評估的標準。

E-Model 的想法是假設語音品質的損失因素總是由網路實體層在傳遞過程中所附加的，若能靈活地加入雜訊、回音、延遲、編解碼器性能、抖動等網路損失因素的估算值，使用者主觀的體驗因素就能被全面客觀的服務品質等級所估計。E-Model 制定了一個量化參數 R，分數介於 0 到 100，公式為 **R=Ro-Is-Ie-Id+A**，其中 Ro 為訊號雜音比，Is 為語音信號同時產生的音質損害因子，Ie 為低位元(low bit rate)語音編碼處理與封包遺失所造成的音質損害因子，Id 為語音延遲造成的音質損害因子，A 為補償損害因子。R 值與 MOS 值相對應的表格如表 3，而兩者之間數值轉換的公式為：**MOS = 1+0.035*R+7*10^{-6}*R*(R-60)*(100-R)**。在 VoIP 的環境中，以 Id 與 Ie 為最主要影響 VoIP 聲音品質的參數，依據[12]我們簡化 E-Model 的公式為 R=94.2-Ie-Id，並經由其經驗法得到下列公式：$Ie=\lambda_1+\lambda_2\ln(1+\lambda_3 e)$，$\lambda_1$ 依據編解碼器的不同給予對應的參數，λ_2 與 λ_3 依據品質下降封包遺失給予對應參數，e 為封包遺失率。上述 λ 值是透過模擬不同的編解碼器與不同的遺失情形所推導出來的，而表 4 為[12][13]模擬推論出 G.711 與 G.729 兩種語音編碼對應參數。

表 3： R 值與 MOS 值對應參照表

R value	MOS value	品質
90 – 100	**4.34 – 4.50**	**極佳**
80 – 90	**4.03 -4.34**	**佳**
70 – 80	**3.60 – 4.03**	**普通**
60 -70	**3.10 – 3.60**	**差**
50 – 60	**2.58 – 3.10**	**極差**

表 4： G.711 與 G.729 品質損傷參數

Codec	λ_1	λ_2	λ_3
G.711	0	30.00	15
G.729	10	47.82	18

另外通話所造成的延遲 Id 由 [12] 中得到，Id = 0.024*d+0.11*(d-177.3)*H(d-177.3)，其中 H(x)是單位步階函數(unit step function)，而 d 為單向延遲時間(ms)。因此可以得知，得知上數兩公式的變數 e 與 d，即可推算出簡化過的 R 值，並且轉換成大眾較為熟悉的 MOS 值，這是我們進行伺服器壓力測試時，最重要的評估參考依據。

3. 相關的 API 與輔助工具

3.1 Socket Programming

Socket 是一種通訊的機制，介於網路應用程式與作業系統及網路硬體間。應用程式透過呼叫 socket 介面，發展具有 TCP/IP (Transmission Control Protocol/Internet Protocol)網路功能之應用，讓我們可以開發本地或跨越網路的 Client 和 Server 軟體。

由於 DIST-V 測試軟體必須同時運作在多台測試用電腦，因此需要一台電腦當作主控端(Master)，它負責指揮測試用電腦(Slave Sender 與 Slave Receiver)與接收測試完畢的結果，因此必須藉助 socket 通訊機制來傳送指令。以下是實作中，socket 建立連線主要用到的函式：

- `socket()`：建立 socket
- `bind()`：設定 socket 所使用的本機端 IP 位址及通訊埠
- `listen()`：設定 socket 聆聽(listen) Client 連結請求
- `accept()`：接受自 Client 的連結請求並建立 socket 連結
- `connect()`：與 Server 的 socket 進行連結
- `write()`：傳送訊息至另一端的 socket
- `read()`：接收來自另一端的 socket 傳送的訊息

要建立簡單的 Client、Server 架構，Server 端必須建立一個 socket 並且利用 `bind()` 系統呼叫設定本機端 IP 位址與通訊埠，再利用 `listen()` 系統呼叫來建立一個佇列，等候 Client 連結；而 Client 也必須建立一個 socket 並且使用 `connect()` 與 Server 連結，Server 會使用 `accept()` 接受 Client 的連結請求，並建立雙方連結後，就可以進行資料通訊。因此只要使用這些基本的函式，UAC 就可以將資料傳送給 UAS。

3.2 oSIP2、eXosip2

為了能夠實現SIP的功能，我們利用oSIP2[14][14]與eXosip2(eXtended oSIP2)[15]來開發SIP相關的應用程式。oSIP2與eXosip2是遵循GPL協議的開放程式庫，由於oSIP2沒有快速產生請求信令(signal)與回應信令的方法，所有的信令必須調用一連串的SIP message API來完成，因此衍伸出eXosip2協助使用者開發，僅需要幾個函式即可完成。

以eXosip2 開發的VoIP 電話相當的多，如Linphone、SfSipUA等。而使用eXosip2來建立一個簡單的VoIP UA也相當簡單，主要使用的函式[16][17]如下：

- `eXosip_init()`：初始化 eXosip2
- `eXosip_listen_addr()`：使用的協定與聆聽的通訊埠
- `eXosip_event_wait()`：偵測 SIP 訊息接收
- `eXosip_call_build_initial_invite()`：建立 INVITE 訊息
- `eXosip_call_send_initial_invite()`：傳送 INVITE 訊息
- `eXosip_call_build_answer()`：建立一個回應訊息
- `eXosip_call_send_answer()`：傳送所建立的回應訊息
- `eXosip_get_sdp_info()`：取出 SDP 欄位，得到會話的相關參數

使用上述的幾個函式，就可達成接收與回應 SIP 的訊息；但是使用者必須再另外針對 SIP 參數的程式碼；撰寫處理音訊功能的程式碼，才能夠完整的建立 SIP UA。

3.3 oRTP

oRTP[18][18]也是遵循 GPL 協議的開放程式庫，依據 RFC3550[6]，使用 C 語言編寫，提供了 RTP 相關的 API 函數。使用者可以在 Unix-like 或 Windows 利用 oRTP API 來建立 RTP 協定連線。實現一個簡單的 oRTP 程式所需使用到的基本函式[19]如下：

- `ortp_init ()`：初始化 oRTP

- `ortp_exit()`：結束先前使用的 oRTP
- `rtp_session_new()`：建立一個 RTP 會話
- `rtp_session_set_ssrc()`：設定 SSRC 值（在我們所設計的程式中，並不需要額外對 SSRC 做設定）
- `rtp_session_set_payload_type()`：設定所使用的 Payload Type
- `rtp_session_set_local_addr()`：設定本機端的 IP 位址及通訊埠
- `rtp_session_set_remote_addr()`：設定目的地的 IP 位址及通訊埠
- `rtp_session_recv_with_ts ()`：根據 Timestamp 從 RTP 音訊流讀取封包
- `rtp_session_send_with_ts()`：從緩衝記憶體(buffer)填入 RTP 封包並設定 Timestamp

我們使用上述的幾個函式，即可產生 RTP 封包，模擬語音傳送的情形。

3.4 fping

fping(fast ping)的操作原理如同我們熟知的 ping，它能夠在本機端發送 ICMP(Internet Control Message Protocol)[20]封包到被測試的電腦，測試彼此之間的網路是否通暢。不過 fping 比起 ping 在執行上更有效率，原因在於若要測試 N 台主機時，ping 必須循序測試每部主機，而 fping 同一時間會送出 ICMP 封包到測試的所有主機，對於有回應的電腦，則 fping 就會從測試清單中移除，假若在設定的時間內仍然沒有回應，則直接判斷為不可送達的(unreachable)。fping 的相關參數如表 5:

表 5： fping 相關參數

-v	顯示版本資訊
-u	顯示無法抵達的(unreachable)的 IP 位址
-a	顯示還存活(alive)的 IP 位址

-e	顯示 ICMP 封包來回的時間
-s	印出詳細的測試結果
-n	顯示此 IP 位址的主機名稱

其中 DIST-V 測試軟體中會以 fping < FILENAME 的語法，直接判斷檔案中所填入的 IP 位址是否存活(alive)。只要將待檢測的 IP 位址依序加入其中，便可一次偵測檔案內的所有 IP 位址。如果沒有使用 fping 快速的判別，傳統的 ping 一次只能判別一台，耗費的時間較長，不利於實際應用於大量的電腦。

3.5 Time Synchronization

為了讓所有的測試用電腦能在同一時間進行語音傳送，並且獲得 RTP 封包單向延遲時間(one way delay)，因此我們運用 ntpd(network time protocol daemon)[21]使測試用電腦進行時間同步的動作。

ntpd 是依據 NTP(Network Time Protocol)[22]所開發的一套程式，它可以用於查詢其它的 NTP 伺服器，並配置本機端的時間；ntpd 為了避免運行中的程式受到調整時間的影響，刻意不用跳躍的方式調整時間，而是以漸進式的微調調整時間。

由於 NTP 伺服器以階層式架構形成時間追溯體系，當同一個 LAN 內有多台機器需要時間的同步，每一台分別去與外界的 stratum 1 伺服器(位於階層最頂層的伺服器，直接追溯到國家標準時間)或是 stratum 2 伺服器(透過 Stratum 1 伺服器間接追溯到國家標準時間)對時，這不僅是一種浪費網路資源的作法，同時也容易因遠距離的傳遞造成時間上的誤差，因此我們選擇校內的機器作為對時。圖 3 為 Sender 1 測試用電腦與校內的 NTP 伺服器(IP 位址 163.22.20.153)對時的情形，由 refid 欄位可知，這台 NTP 伺服器是與哪一台 NTP 伺服器(IP 位址 18.26.4.105)對時；st 欄位代表校內的 NTP 伺服器為 stratum 2 伺服器；由 delay 欄位可知 Sender 1 測試用電腦與校內 NTP 伺服器的來回時間僅 1.23ms，比起與校外對時可以減少時間來回的誤差；最後在 offset

的欄位可知，與校內伺服器對時偏差為 0.512 ms，因此我們將在所有的測試用電腦安裝 ntpd 軟體，達到時間的同步。

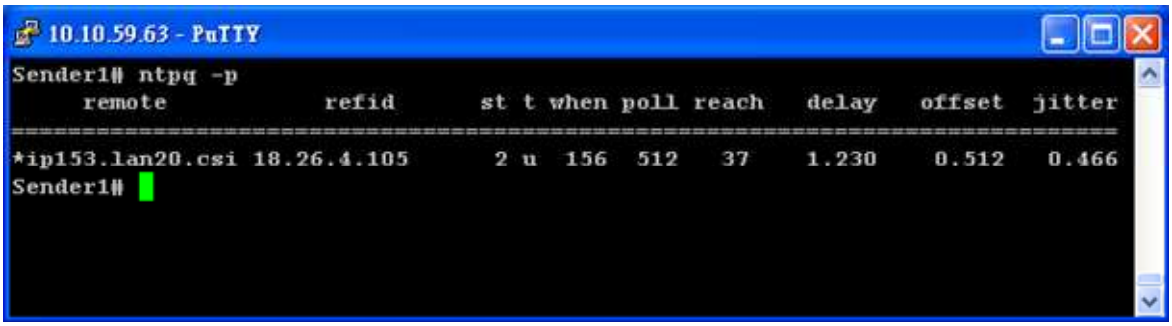

圖 3： Slave Sender1 與校內 NTP 伺服器對時

4. 系統架構與實作成果

我們自行開發的 DIST-V 測試軟體，其架構如圖 4，搭配 N 台傳送端(Slave Sender)電腦與 N 台接收端(Slave Receiver)電腦組成的測試用電腦進行測試，其中 Slave 群的作業系統為 FreeBSD 或是 Linux，並且一台 Slave Sender 電腦與一台 Slave Receiver 電腦相互對應。Master 電腦是中央控管的角色，它主要的功能在傳送指令至多台測試用電腦，並接收 Slave Receiver 回報的相關資訊，做 MOS 值的計算。由於每一對 Sender-Receiver 約可同時產生 90 通話，因此若採用分散式架構，搭配多台電腦同時產生測試流量，即可產生足夠的會話數進行壓力測試。

我們將針對負擔較吃重的 RTP Proxy Server 進行壓力測試，安裝的是 rtpproxy[23] 這套軟體，藉由產生大量的 RTP 封包對 RTP Proxy Server 進行壓力測試。以下將介紹系統架構、實作流程以及 DIST-V 與商用流量測試工具 NuStreams-600 的比較。

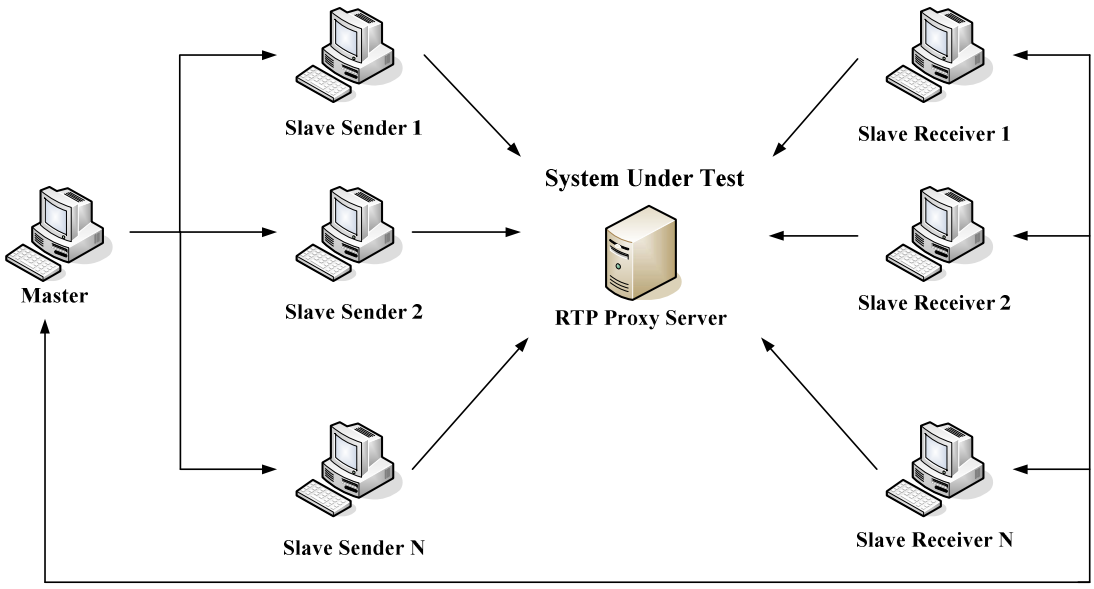

圖 4： DIST-V 架構圖

4.1 DIST-V 架構

我們搭配圖 5 DIST-V 流程圖，並分為三大主軸一一介紹，由左而右依序是，Master、Slave Sender 以及 Slave Receiver 執行流程。

首先，我們必須先執行 Slave Sender 以及 Slave Receiver 端的程式，Slave Sender 以及 Slave Receiver 將進行 socket server 端的初始化，並進入聆聽(listen)狀態，等待 Master 建立連線;啟動完 Slave Sender 與 Slave Receiver 後再執行 Master,由於 Master 扮演的是中央控管的腳色，因此會先運用 fping，判斷 Slave Sender 與 Slave Receiver 的 IP 位址有無啟動(online)。

Master 運用 fping 判斷完電腦是否啟動後，則會進行 socket client 端的初始化，並且與 Slave Sender、Slave Receiver 端一一建立 socket 連線。由於 DIST-V 是一套分散式測試軟體，需先將 Slave Sender 與 Slave Receiver 端的程式以人工方式一一啟動；為了避免有遺漏的情形，我們將採用 socket 連線的方式，判斷有無開啟對應的聆聽(listen)通訊埠，來確認有無啟動相關程式。

當確保 Slave 端的程式均啟動後，將進入建立會話的部份。Master 端會請使用者輸入所要建立的會話數，輸入完畢後，藉由 socket 函式送出，將相關指令傳送給各個 Slave Sender 與 Slave Receiver。

每一個 Slave Receiver 收到 Master 傳送的指令後，分別知道一共要建立多少個會話數以及對應的帳號，就會先行向 SIP Proxy Server 進行註冊；而每一個 Slave Sender 的也將進行帳號註冊的動作，並邀請註冊在 Slave Receiver 的 UA 進行會話。為了要對 RTP Proxy Server 進行壓力測試，因此設定 SIP Proxy 無論是否位於 NAT 後方，都須透過 RTP Proxy 進行 RTP 封包轉傳的動作，因此在 Slave Sender 發出 INVITE 邀請 Slave Receiver 進行通話時，我們藉由 `eXosip_get_sdp_info()`取得 SDP 欄位，得知 RTP Proxy Server 的 IP 位址與對應的通訊埠，並將此資訊告知 oRTP 的函式 `rtp_session_set_remote_addr()`,因此每一個 UAC 將會產生 G.711 語音編碼的 RTP 音訊流到 RTP Proxy Server 對應的通訊埠上。

當所有的 Slave Sender 與 Slave Receiver 的通話建立後，便同步進行 RTP 封包的傳送，當 Slave Sender 傳送完封包之後，將回到 socket 聆聽(listen)狀態，重新等待下次的測試指令;而每一個 Slave Receiver 將會收到由 RTP Proxy Server 轉送過來的 RTP 封包，統計收到的封包數目以及紀錄封包延遲的時間，並將這兩項數據回報給 Master,並且回到 socket 聆聽(listen)的狀態,等待 Master 端下次的測試指令;而 Master

端收到所有的 Slave Receiver 的回報後，將進行 E-Model 與 MOS 值的計算，計算完畢後程式結束。

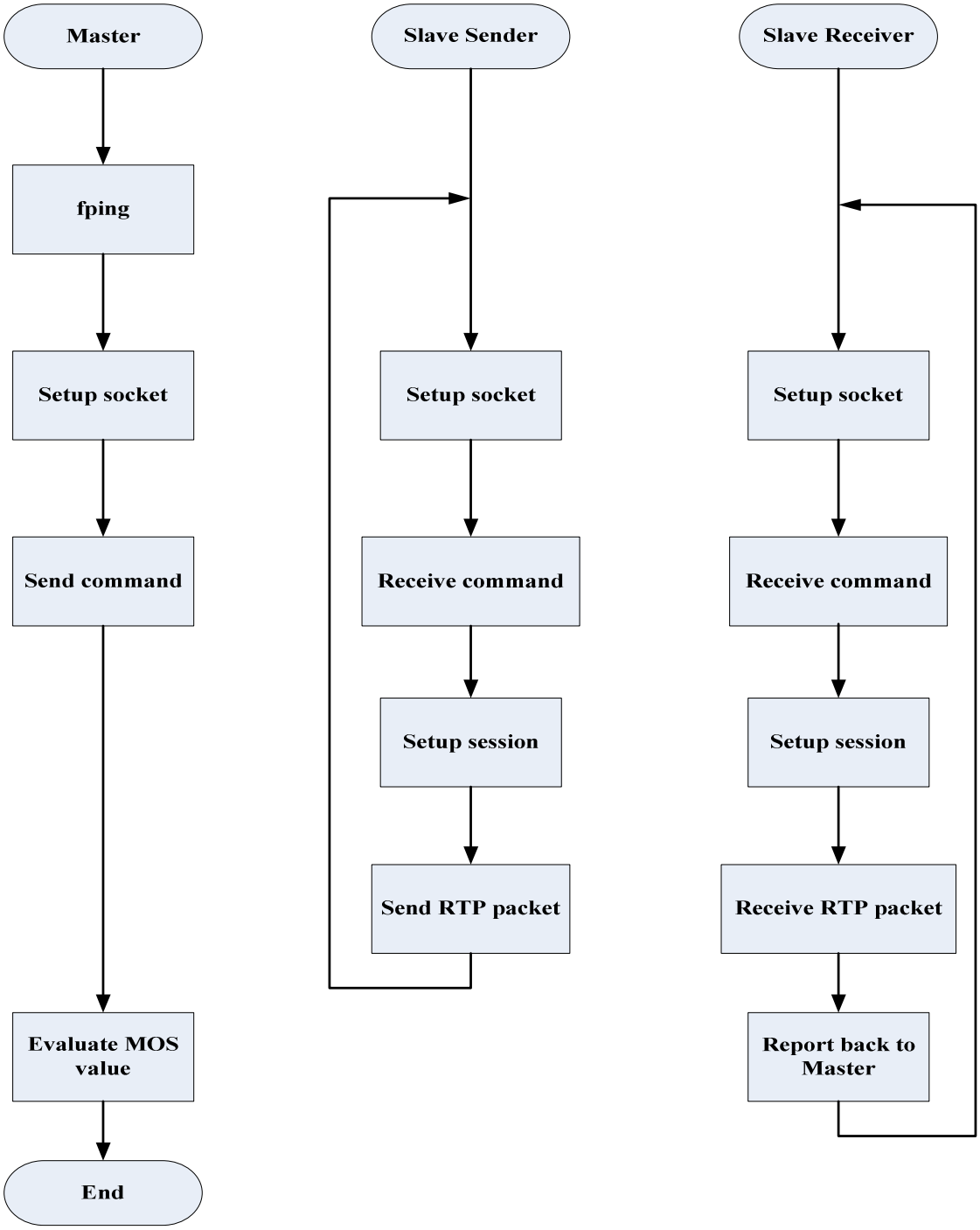

圖 5： DIST-V 流程圖

4.2 實作成果

為了展示 DIST-V 分散式的特性與可操作性，本實驗中模擬產生四個會話，利用兩個 Slave Sender 與兩個 Slave Receiver 測試用電腦進行壓力測試。圖 6 為測試架構圖，每一台電腦分別標示 IP 位址，其中 SIP Proxy Server 與待測的 RTP Proxy Server 安裝於同一台主機上，所產生四個會話將會平均分配；此外 Slave Sender 1 與 Slave Sender 2 會同步傳送 RTP 封包到 Slave Receiver 1 與 Slave Receiver 2，對 RTP Proxy Server 進行壓力測試。

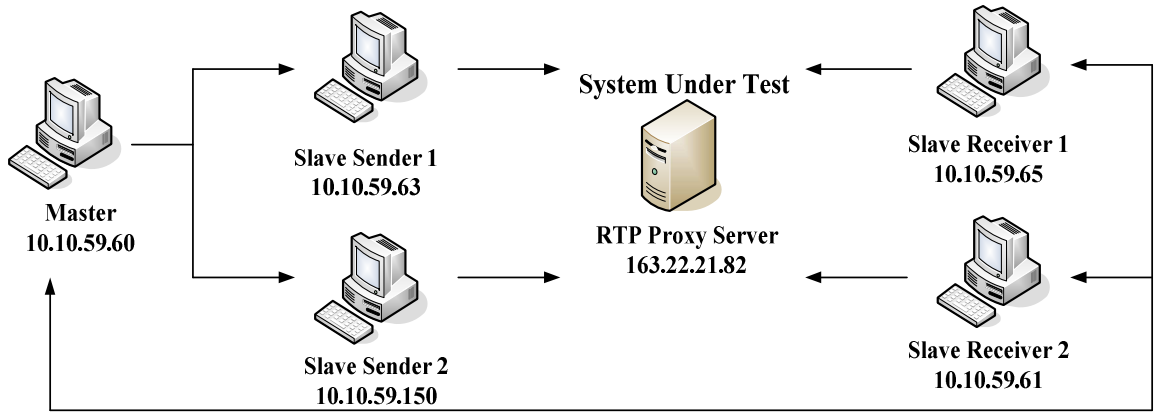

圖 6： 測試架構圖

圖 7 為 Master 執行畫面，步驟依序如下:

(1) 開始執行 master 程式。

(2) 藉由 fping 判斷 sender_ip.txt 中 Slave Sender 所對應的 IP 位址是否在線上(online)，並判斷該電腦有無執行 slave_sender 程式。

(3) 統計存放在 sender_ip.txt 裡的 Slave Sender IP 位址有幾個是（a）未啟動、(b）有啟動但是未執行 slave_sender 程式、(c）有啟動且有執行 slave_sender 程式。

(4) 再次利用 fping 判斷 receiver_ip.txt 中 Slave Receiver 所屬的 IP 位址是否在線上(online)，並判斷啟動的 Slave Receiver 電腦有無執行 slave_receiver 程式。

(5) 統計存放在 receiver_ip.txt 裡的 Slave Receiver IP 位址有幾個是（a）未啟動、(b）有啟動但是未執行 slave_receiver 程式、(c）有啟動且有執行 slave_receiver 程式。

(6) 輸入所欲建立的會話數目，並且預設二十秒後 Slave Sender 群同步傳送 RTP 流，若要更改時間請按“y”，並自行輸入時間。

(7) 鍵入更改時間後，顯示更改後當下的時間及 Slave Sender 群預定傳送 RTP 音訊流到 Slave Receiver 群的時間。

(8) 顯示各個 Slave Sender 與 Slave Receiver 建立的會話數目，並將指令分別傳到 Slave Sender 群與 Slave Receiver 群；傳送完畢後，等待相關數據回報。

(9) 收到 Slave Receiver 群回報收到的 RTP 封包數目以及平均的延遲時間，計算 E-Model 並且轉換成 MOS 值，程式結束。

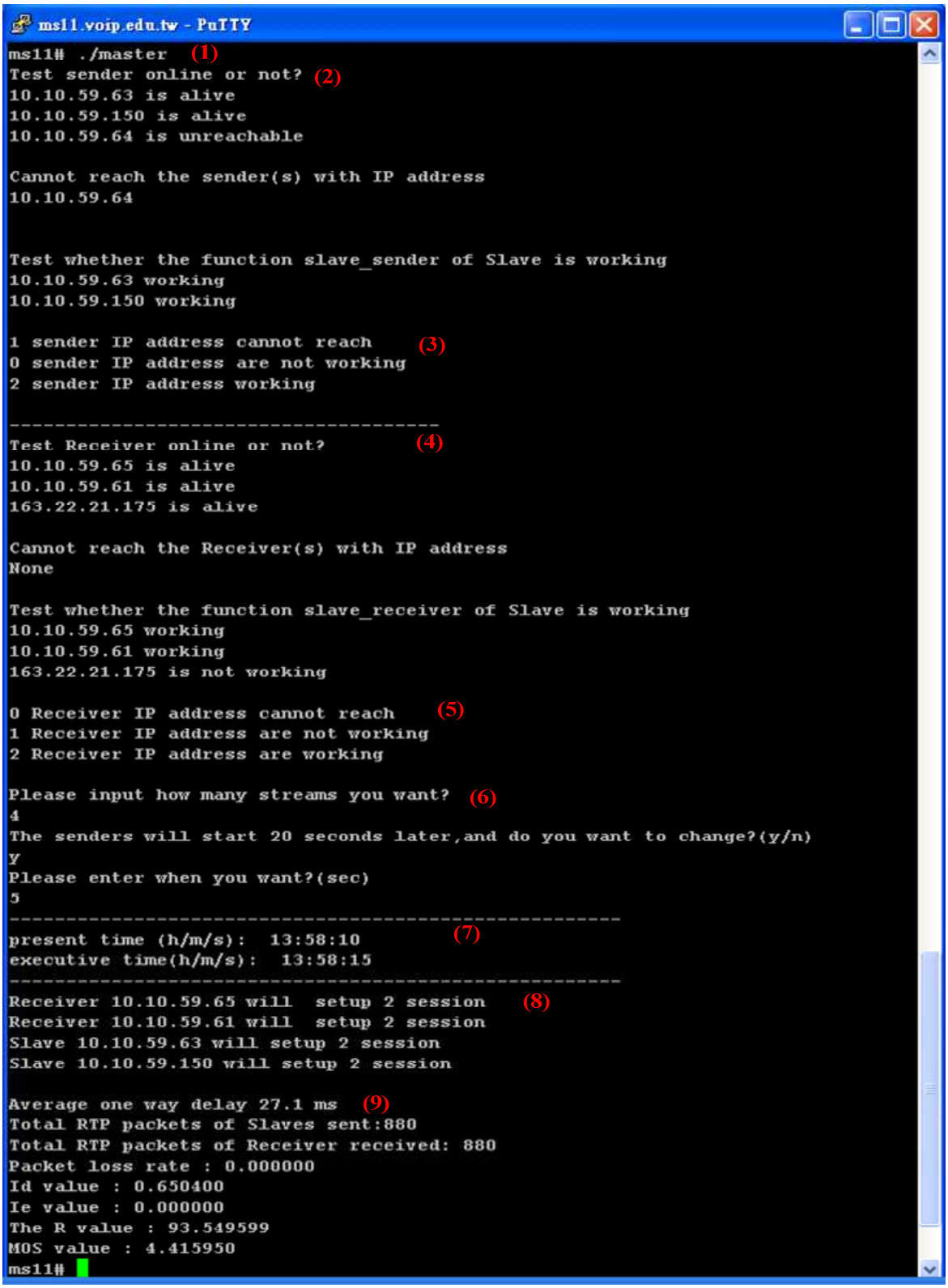

圖 7： Master 執行圖

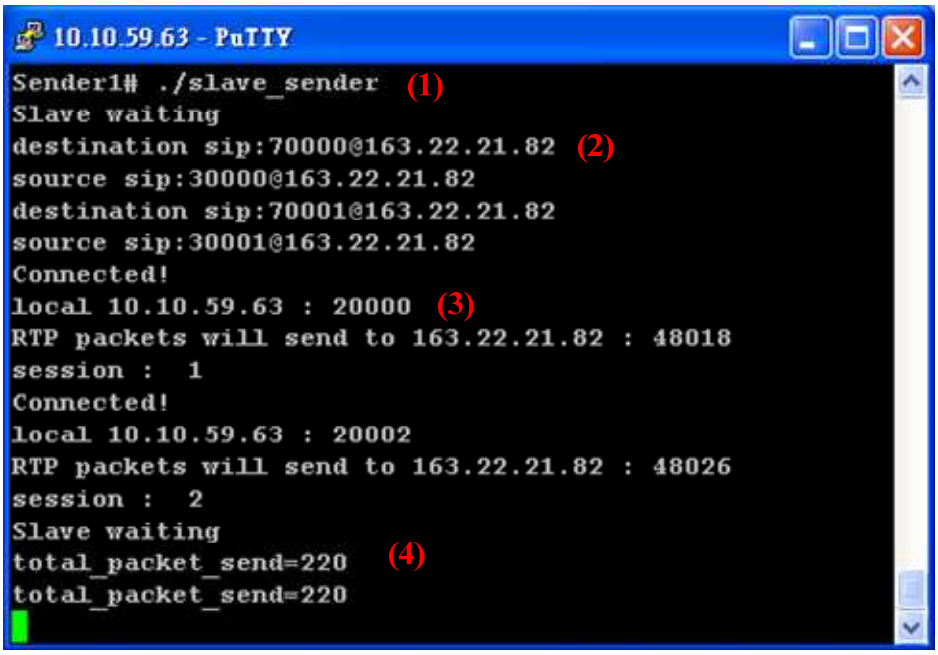

圖 8： Slave Sender 1 執行圖

圖 8 為 Slave Sender 1 的執行畫面，步驟依序如下:

(1) 開始執行 slave_sender 程式。

(2) 由於 Master 告知 Slave Sender 1 僅負責兩個會話，因此使用 30000 與 30001 兩個帳號，分別向 SIP Proxy Server(163.22.21.82)註冊，並且邀請 70000、70001 建立會話。

(3) 會話建立完畢後，顯示 30000、30001 兩個帳號分別由哪個通訊埠(20000、20002)送出 RTP 封包以及 RTP Proxy Server IP 位址(163.22.21.82)與接收的通訊埠(48018、48026)，並且等到執行時間一到(五秒後)，Slave Sender 群會同步傳送 RTP 封包至 Slave Receiver 群。

(4) 傳送完畢之後，將顯示每一個會話所傳送出的 RTP 封包，而 RTP 封包數量取決於 oRTP 函式所設定的音訊檔案大小，這可由使用者自行更改。

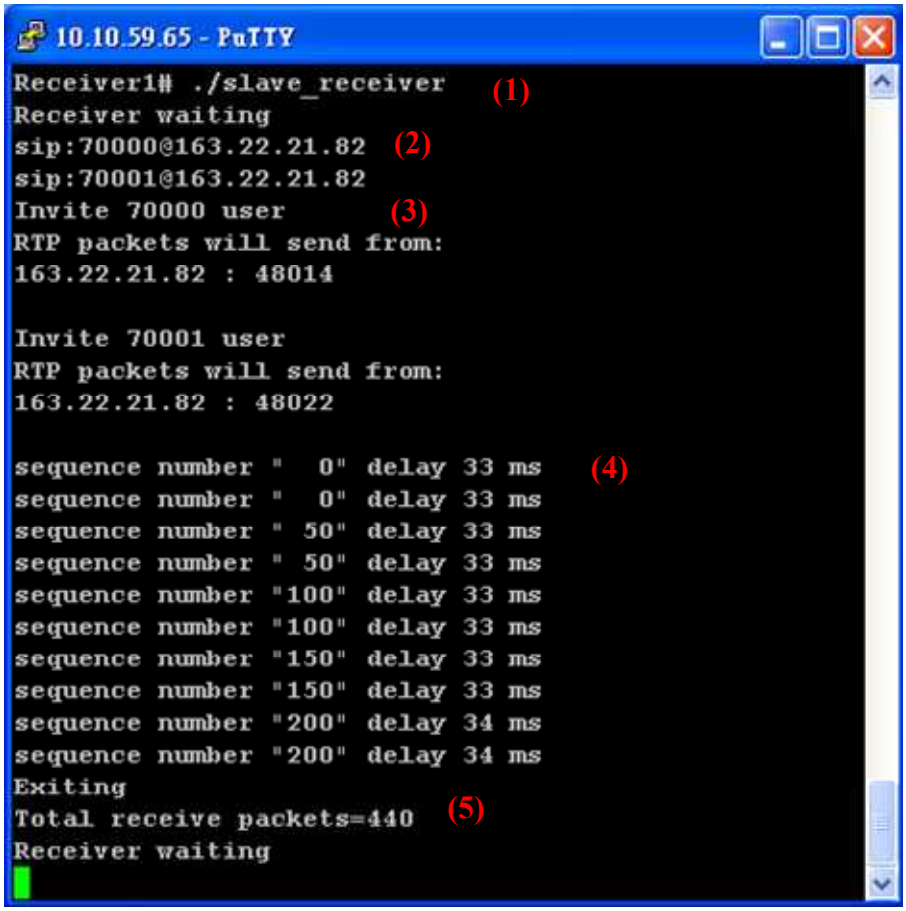

圖 9： Slave Receiver 1 執行圖

圖 9 為 Slave Receiver 1 執行畫面，步驟依序如下:

(1) 先執行 slave_receiver 程式。

(2) 收到 Master 的指令，將會以 70000、70001 兩個帳戶與 SIP Proxy Server 進行註冊，並且等待 Slave Sender 端的邀請。

(3) 當 Slave Sender 的 UAC 發出邀請後，在建立會話的過程中，將會顯示 RTP Proxy Server 的 IP 位址(163.22.21.82)與送出 RTP 的通訊埠(48014、48022)，並等待執行時間；執行時間一到，Slave Sender 群會同步傳送 RTP 封包。

(4) 每收到 sequence number 為 0, 50, 100…的 RTP 封包，記算單向延遲時間。

(5) 統計所收到的 RTP 封包個數，與封包所延遲的時間，將兩筆資料傳送給 Master，作 MOS 值的計算。

4.3 實驗數據

為了產生較多的流量,本實驗以一台 Master 以及六台 Slave Sender 以及六台 Slave Receiver 對安裝 rtpproxy 這套軟體的 RTP Proxy Server 進行測試，實驗環境的頻寬為 100Mbps,每個 session 將同步傳送長達 78 秒的 RTP 封包,而測試的 RTP Proxy Server 的規格為：CPU: Intel(R) Pentium(R) 4 CPU 3.20GHz RAM: 1GB，圖 10 為實驗的測試結果，橫軸為同步建立的會話數，縱軸為 MOS 值，我們每間隔五十通會話數做十次測試並且取平均,可發現當同時產生的會話數超過 250 通時,MOS 值開始明顯下降，當同步產生 500 通會話數時，MOS 平均值為 1.24，以表 3 來判斷語音品質，若要達到 MOS 值為普通的 3.6 分，此電腦作為 RTP Proxy Server 的上限以同步服務 200 通通話最佳。

由於我們傳送的是 G.711 語音編碼的 RTP 封包，以 E-Model 的簡化公式為 R = 94.2-Ie-Id, Ie=30 * ln(1+15e), Id = 0.024*d+0.11*(d-177.3)*H(d-177.3)。由公式可知，當延遲時間未超過 177.3 ms 時，時間延遲所帶來的損傷較小，但是封包遺失率開始逐漸攀升時，對 R 值的影響較大。由圖 11、圖 12 可知，同步建立的會話數越多，延遲時間呈現緩慢的成長，且單向延遲時間尚未超過 177 ms；而同步建立 200 通通話後，封包遺失率大幅度增加，可讓 MOS 值由 4.38 一路下滑至 1.24。因此判斷此 RTP Proxy Server 的封包遺失率是造成 MOS 值下降的主要原因。

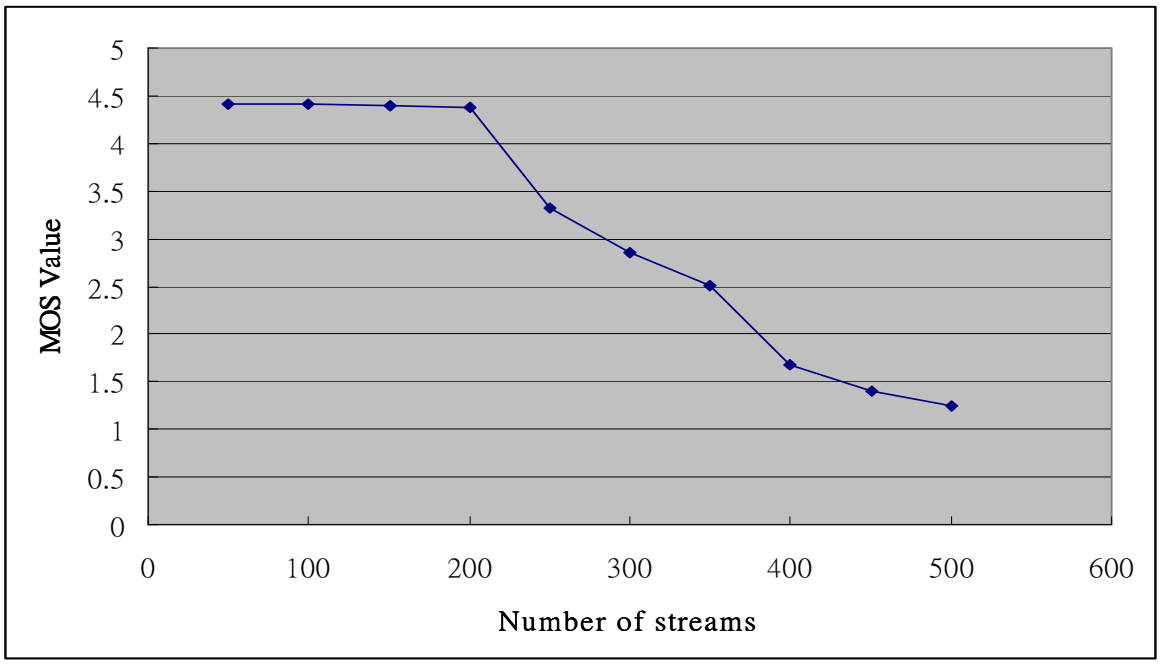

圖 10： MOS 值與會話數關係圖

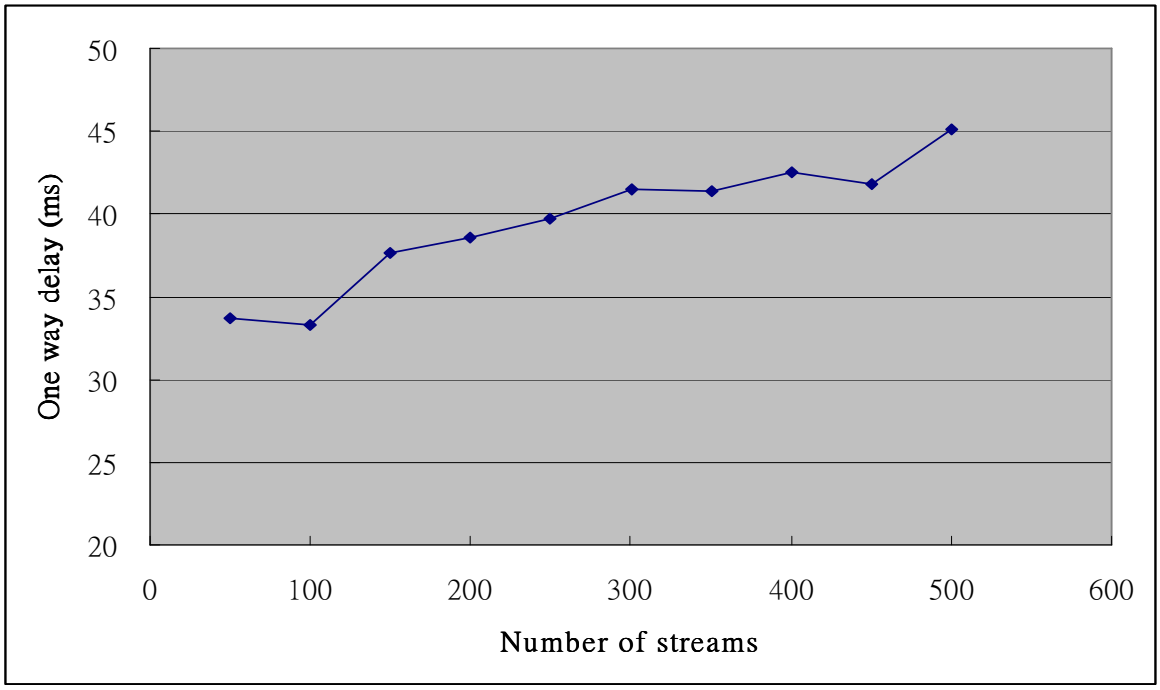

圖 11： 單向延遲時間與會話數關係圖

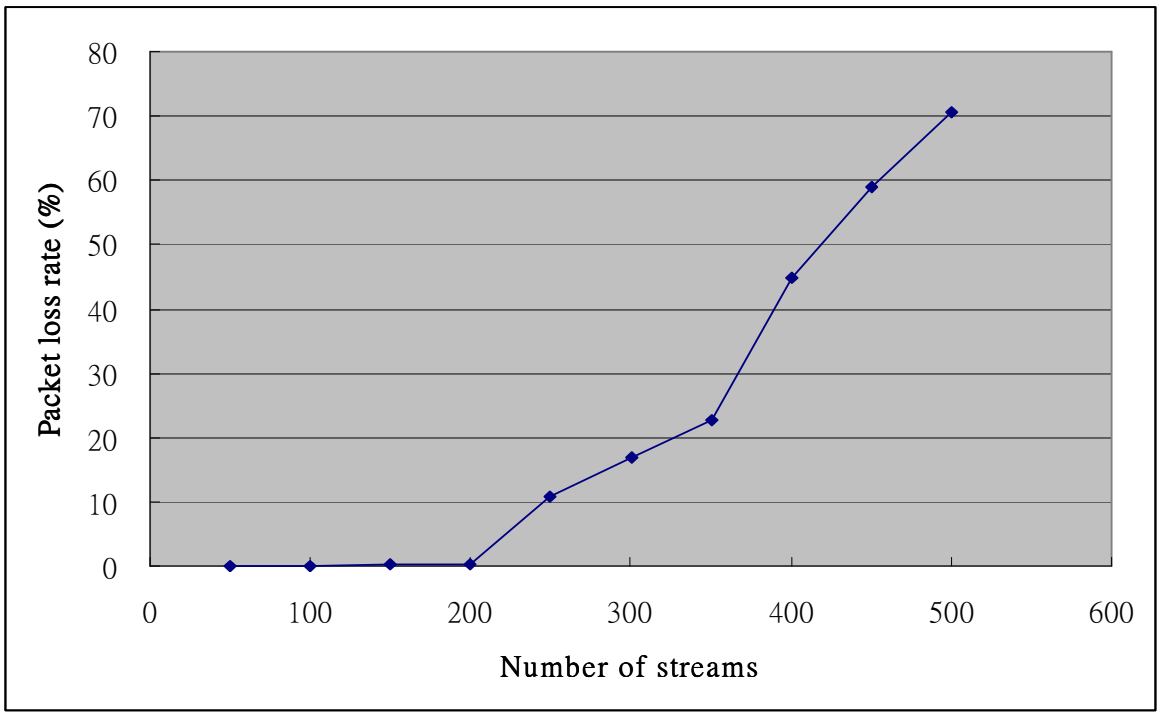

圖 12： 封包遺失率與會話數關係圖

4.4 DIST-V 與商用軟體的比較

我們拿 DIST-V 跟 Nustreams-600 來做比較，如表 6。首先，兩者差異最大的是，我們所設計的 DIST-V 為免費的測試軟體，僅需搭配桌上型或筆記型電腦在 Unix 環境進行測試，測試完畢後仍可做一般電腦使用；相較於 NuStreams-600 價格至少新台幣四十萬，能大幅降低測試成本，對使用者就具有一定的吸引力。第二點的差別在於，一組 Slave Sender 對 Slave Receiver 最多可建立 90 個會話，每秒中可產生 4500 個 G.711 語音編碼的封包，而 NuStreams-600 若要模擬產生相同大小的封包，則每秒中最多僅能產生 53418 個，則 DIST-V 搭配 12 組 Slave Sender 與 Slave Receiver 即產生相同數量，因此相對的 DIST-V 較有彈性。

另外，NuStreams-600 有眾多的統計數據，如傳送、接收的封包數目與封包大小、以及收到的 Unicast、Muticast、Broadcast 封包數目與 Collision 的數目，但是獨缺計算延遲時間。由於封包延遲時間在 VoIP 語音品質量測上相當重要，而 DIST-V 具有

計算延遲時間的功能是，可說是為了 VoIP 伺服器而設計的壓力測試軟體。再來是穩定性的差別，由於 DIST-V 需要掌控多台的 Slave 電腦，每台 Slave 電腦分別與 NTP 伺服器作對時，對時之後時間會相當的接近，但有時因網路距離的緣故，會有 0 ~ 3 ms 左右的誤差。當偏移量較大時，就會造成單向延遲時間的誤差相對增加，進一步影響 R 值的結果；另外，同時間操作多台 Slave 電腦時，當有任何一部 Slave 不如預期的運作，也會造成測試上的錯誤。例如：會話尚未全部建立完畢時，已經到了開始傳送 RTP 封包的時間，或是當每台 Slave Sender 建立超過 90 個會話數時，兩者均會造成 DIST-V 程式的錯誤；而 NuStreams-600 僅需將相關參數設定完畢後，按下執行按鈕即可長時間的測試，相對於 DIST-V 較為穩定。最後，擴充性的部份，由於 DIST-V 是我們自行開發設計的一套軟體，目前使用的是 G.711 的 RTP 封包，在未來甚至可以加入 H.264 的 RTP 封包，進行語音與影像的同步傳輸，因此擴展性也比 NuStreams-600 來的高。

表 6： DIST-V 與 NuStreams-600 的比較

	Cost	Packets/s (max)	Calculate Delay Time	Stability	Extensibility
DIST-V	Free	∞	Yes	Medium	Better
NuStreams-600	NT 400,000	53418	No	Better	Medium

5. 結論及未來方向

本篇論文中，我們在 Unix 底下實作出一個符合實際通話情形的壓力測試工具 DIST-V，它能夠產生大量會話數，藉由 RTP 封包進行 VoIP 伺服器的測試，若與 SmartBits、NuStreams 等昂貴的設備相比，能大幅節省測試成本，並且能得到 RTP 封包遺失率與單向延遲時間的測試結果代入 R 值，轉換成大家較為熟悉的 MOS 值提供測試者作為參考。

在未來的研究主題上，將會比較 rtpproxy 與 MediaProxy[24]之間的壓力測試，這兩樣軟體都是協助 RTP 封包穿越 NAT 環境，因此針對兩者進行比較，是 ISP 會重視的議題。另外壓力測試設備最重要的是長時間測試的穩定性，因此加強 DIST-V 的穩定性提供長時間的測試也是重要的發展方向。最後，可以考慮建置以 GPS 對時的 NTP 伺服器，由於 GPS 是一個可信賴的標準時間並且為 stratum 1，因此可以提升測試用電腦為 stratum 2，使得同步的時間更加準確，也間接提升 Slave 端同步執行指令時的一致性。

參考文獻

[1] SPRIRENT SmartBits, http://www.spirent.com/Solutions-Directory/Smartbits.aspx

[2] Y. H. Zhang, Z. T. Li, M. Z. Wang and L. Xiao, "A Multi-Link Aggregate IPSec Model", First International Workshop on ETCS, Volume 3, pp. 489 - 493, Wuhan, China, 7-8 March 2009.

[3] Nustream Internet, http://www.nustream.com/

[4] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M.Handley, E. Schooler, "SIP: Session Initiation Protocol ", IETF RFC 3261, June 2002.

[5] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, " RTP: A Transport Protocol for Real-Time Applications", *Network Working Group*, IETF RFC 1889, January 1996.

[6] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", *Network Working Group*, IETF RFC 3550, July 2003.

[7] M. Handley, V. Jacobson, "SDP: Session Description Protocol", IETF RFC 4566, July 2006.

[8] ITU-T "Method for subjective determination of transmission quality", ITU-T Recommendation P.800, August1996.

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

[10] fping, http://www.fping.com/

[11] K. Egevang、P. Francis, "The IP Network Address Translator (NAT)", *Network Working Group*, IETF RFC1631, May 1994.

[12] R. G. Cole, J. H. Rosenbluth, "Voice over IP Performance Monitoring", *ACM Computer Communication Review (CCR)*, vol. 31, no. 2, pp.9–24, 2001.

[13] L. Ding, R. A. Goubran, "Speech Quality Prediction in VoIP Using the Extended E-Model", in Proc. IEEE Global Telecommunications Conference (GLOBECOM), San Francisco, CA, USA, pp. 3974–3978, December 2003.

[14] The GNU oSIP library

[http://www.gnu.org/software/osip/].

[15] The eXtended osip library

[http://savannah.nongnu.org/projects/exosip].

[16] libosip Documentation

[http://www.gnu.org/software/osip/doc/html/index.html].

[17] libeXosip2 Documentation

[http://sip.antisip.com/docu/eXosip2/].

[18] oRTP, a Real-time Transport Protocol (RTP,RFC3550) library

[http://www.linphone.org/index.php/eng/code_review/ortp].

[19] oRTP API documentation

[http://ftp.twaren.net/Unix/NonGNU/linphone/ortp/docs/].

[20] J. Postel, "Internet Control Message Protocol", *Network Working Group*, IETF RFC 792, September 1981

[21] Network Time Protocol (NTP) daemon

[http://www.cis.udel.edu/~mills/ntp/html/ntpd.html].

[22] David L. Mills, “Network Time Protocol (Version 3) Specification, Dmplementation and Analysis”, RFC 1305, March 1992.

[23] rtpproxy

[http://ftp.iptel.org/pub/rtpproxy/].

[24] MediaProxy

[http://mediaproxy.ag-projects.com/].

附件

附件 A. Code of master.c

```
 //G.711 Codec //
    void e_model(float one_way_delay, float packet_loss_rate) {
    float Ie,Id,R;
    Ie = 1+15*packet_loss_rate;
    Ie = 30*log(Ie);
    Id = 0.024*one_way_delay;
    if(one_way_delay>177.3)
        Id = Id + 0.11*(one_way_delay - 177.3);


        R=94.2-Id-Ie;
        printf("Packet loss rate : %f\n",packet_loss_rate);
        printf("Id value : %f\n",Id);
        printf("Ie value : %f\n",Ie);
        printf("The R value : %f\n",R);
        if(R<=0)
            R=1;
        if(R>=100)
            R=4.5;
        i f(R>0 && R<100)
            R=1+0.035*R+R*(R-60)*(100-R)*7*0.000001;
        printf("MOS value : %f\n",R);
}
int main(void){
```

```
int account = 70000;//預設 receiver 端從 70000 開始註冊
int account1;
int sockfd;
int len;
int result_server; //判斷遠端電腦 socket_server 有無啟動
int t_conl;     //幾秒後同步執行指令
int nstream,nstream_ch;
int i=0, k=0, h=0,l=0, m=0,j=0,z;
int totalpacket =0,t_totalpacket=0, total_sq =0,n_total_sq =0;
float total_avg =0, t_total_avg=0;
float y;// 總共收到的 RTP 封包數
struct sockaddr_in address;
struct tm *tm_ptr;
time_t the_time;
char cs[150][16];
char cs1[150][16]; //可存放 150 個 Slave Sender
char cr1[150][16]; //可存放 150 個 Slave Receiver
char cr[150][16];
char *co[150]; //slave online
char *cro[150]; //Receiver online
char ch_t;
struct sockaddr_in client_address;
struct sockaddr_in server_address;
int server_sockfd, client_sockfd;
int server_len, client_len;
FILE *fp;

/* 測試 Slave Server 是否開機 */
```

```
    printf("Test slave online or not?\n");
    system("fping < sender_ip.txt");
    printf("\nCannot reach the slave(s) with IP address\n");

    fp=fopen("ping_fail.txt","r");
    while(fgets(cs1[i],40,fp)!=NULL){
        if(cs1[i][strlen(cs1[i])-1]=='\n')
            cs1[i][strlen(cs1[i])-1] = 0;
        i++;
        h++;
    }
    fclose(fp);
    if(h ==0)
        printf("None");
    for(i=0;i<h;i++)
        printf("%s\n",cs1[i]);
    printf("\n\nTest whether the function dist_slave of Slave is working\n");
    h=0;
    fp=fopen("ping_successful.txt","r");   //取出有 fping 成功的 IP 位址，判斷 slave 端程式有無啟動
    while(fgets(cs[k],40,fp)!=NULL){
        if(cs[k][strlen(cs[k])-1]=='\n')
            cs[k][strlen(cs[k])-1] = 0;
        k++;
        h++;
    }
    fclose(fp);

/* 測試 Slave Sender 有無啟動對應的 slave_sender 程式 */
    for(k=0;k<h;k++){
```

```
        sockfd = socket(AF_INET, SOCK_STREAM, 0);
        address.sin_family = AF_INET;
        address.sin_addr.s_addr = inet_addr(cs[k]);
        address.sin_port =9000;
        len = sizeof(address);
        result_server = connect(sockfd, (struct sockaddr *)&address, len);
        close(sockfd);

        if(result_server == -1){
            printf("%s is not working\n",cs[k]);
            l++;
        }
        if(result_server == 0){
            printf("%s working\n",cs[k]);
            co[m]=cs[k];
            m++;     //m 為有啟動 slave_sender 的數目
        }
    }
    printf("\n%d slave IP address cannot reach",i);
    printf("\n%d slave IP address are not working",l);
    printf("\n%d slave IP address working",m);
/*測 Slave Receiver 端的電腦有無啟動*/
    i = 0; h = 0; k = 0;l = 0;
    printf("\n\n---------------------------------------\n");
    printf("Test Receiver online or not?\n");
    system("fping < receiver_ip.txt");
    printf("\nCannot reach the Receiver(s) with IP address\n");
    fp=fopen("ping_fail.txt","r");
    while(fgets(cr1[i],40,fp)!=NULL){
```

```
        if(cr1[i][strlen(cr1[i])-1]=='\n')
            cr1[i][strlen(cr1[i])-1] = 0;
        i++;
        h++;
    }
    fclose(fp);

    if(h ==0)
        printf("None");

    for(i=0;i<h;i++)
        printf("%s\n",cr1[i]);

    printf("\n\nTest whether the function dist_receiver of Slave is working\n");

    h=0;
    fp=fopen("ping_successful.txt","r");
    while(fgets(cr[k],40,fp)!=NULL){
        if(cr[k][strlen(cr[k])-1]=='\n')
            cr[k][strlen(cr[k])-1] = 0;
        k++;
        h++;
    }
    fclose(fp);

// 測試有無啟動 Slave Receiver 端有無 slave_receiver 程式
    for(k=0;k<h;k++){
        sockfd = socket(AF_INET, SOCK_STREAM, 0);
        address.sin_family = AF_INET;
```

```
        address.sin_addr.s_addr = inet_addr(cr[k]);
        address.sin_port =9000;
        len = sizeof(address);
        result_server = connect(sockfd, (struct sockaddr *)&address, len);
        close(sockfd);

        if(result_server == -1){
            printf("%s is not working\n",cr[k]);
            l++;
        }
        if(result_server == 0){
            printf("%s working\n",cr[k]);
            cro[j]=cr[k];
            j++;
        }
    }
    printf("\n%d Receiver IP address cannot reach",i);
    printf("\n%d Receiver IP address are not working",l);
    printf("\n%d Receiver IP address are working\n",j);

/*輸入 RTP 封包傳送的時間以及會話數目*/
    printf("\nPlease input how many streams you want?\n");
    scanf(" %d",&nstream);
    printf("The slaves will start 20 seconds later,and do you want to change?(y/n)\n");
    scanf(" %c",&ch_t);

    while(ch_t!='n' && ch_t!='y'){
        printf("Please enter \'y\' or \'n\'\n");
        scanf(" %c",&ch_t);
```

```
    }
    if(ch_t=='n')
        t_conl=20;
    if(ch_t=='y'){
        printf("Please enter when you want?(sec)\n");
        scanf("%d",&t_conl);
    }
    time(&the_time);
    tm_ptr=localtime(&the_time);
    printf("------------------------------------------------------\n");
    printf("present time (h/m/s):    %02d:%02d:%02d\n",
                tm_ptr->tm_hour,tm_ptr->tm_min,tm_ptr->tm_sec);
    t_conl=t_conl+the_time;
    tm_ptr=localtime(&t_conl);
    printf("executive time(h/m/s):    %02d:%02d:%02d\n",
                tm_ptr->tm_hour,tm_ptr->tm_min,tm_ptr->tm_sec);
    printf("------------------------------------------------------\n");

/*  分配每個 Slave Sender 與 Receiver 所要建立的會話數*/
    account1 = account;
    for(i=0;i<j;i++){
        nstream_ch=nstream/m; //m is    number of slave
        l=nstream%m;
        z=m-j;
        printf("Receiver %s will    setup %d session\n",cro[i],nstream_ch);

        sockfd = socket(AF_INET, SOCK_STREAM, 0);
        address.sin_family = AF_INET;
        address.sin_addr.s_addr = inet_addr(cro[i]);
```

```
        address.sin_port =9000;
        len = sizeof(address);
        result_server = connect(sockfd, (struct sockaddr *)&address, len);
        write(sockfd, &t_conl,sizeof(int));
        write(sockfd, &nstream_ch,sizeof(int));
        write(sockfd, &account1,sizeof(int));
        account1 = account1+1000;
        close(sockfd);
    }
    for(i=0;i<m;i++){
/*      nstream_ch=nstream/m; //m is    number of slave
        l=nstream%m;
        z=m-j;
*/
        if(l == 0 && i < m)
            printf("Slave %s will setup %d session\n",cs[i],nstream_ch);
        if(l != 0 && i < m){
            if(i<l){
                nstream_ch = nstream_ch + 1;
                printf("Slave %s will setup %d session\n",cs[i],nstream_ch);
            }
            else
                printf("Slave %s will setup %d session\n",cs[i],nstream_ch);
        }
        sockfd = socket(AF_INET, SOCK_STREAM, 0);
        address.sin_family = AF_INET;
        address.sin_addr.s_addr = inet_addr(cs[i]);
        address.sin_port =9000;
        len = sizeof(address);
```

```
        result_server = connect(sockfd, (struct sockaddr *)&address, len);

        write(sockfd, &t_conl,sizeof(int));
        write(sockfd, &nstream_ch,sizeof(int));
        write(sockfd, &account,sizeof(int));
        account = account + 1000;
        close(sockfd);
    }

/* 接收 Slave Receiver 回送的 RTP 封包 */
    server_sockfd = socket(AF_INET, SOCK_STREAM, 0);
    server_address.sin_family = AF_INET;
    server_address.sin_addr.s_addr =htonl(INADDR_ANY);
    server_address.sin_port =4000;
    server_len = sizeof(server_address);
    bind(server_sockfd, (struct sockaddr *)&server_address, server_len);

    listen(server_sockfd,5);
    signal(SIGCHLD, SIG_IGN);

    for(i=0;i<j;i++){
        client_len = sizeof(client_address);
        client_sockfd = accept(server_sockfd,
            (struct sockaddr *)&client_address, &client_len);
        read(client_sockfd,&totalpacket,sizeof(int));
        read(client_sockfd,&total_avg,sizeof(float));
        read(client_sockfd,&total_sq,sizeof(int));
        t_totalpacket=totalpacket+t_totalpacket;
        t_total_avg = total_avg+t_total_avg;
```

```
        n_total_sq = total_sq+n_total_sq;
        usleep(200000);
    }
    t_total_avg = t_total_avg/n_total_sq;

    printf("\nAverage one way delay %g ms\n",t_total_avg);
    printf("Total RTP packets of Slaves sent:%d\n",nstream*220);
    printf("Total RTP packets of Receiver received: %d\n",t_totalpacket);
    y=nstream*220;
    y=1-(t_totalpacket/y);
    e_model(t_total_avg,y);
    return 0;
}
```

附件 B.　Code of slave_sender.c

```
#define SIP_BIND_PORT          5070
#define SIP_S_ACCOUNT          35000 //本機端從 3500 開始註冊
#define SIP_LOCAL_IP           "10.10.59.64" //本機端的 IP 位址
#define RTP_PORT               20000
#define ch_command             55 // ortp 指令長度
#define proxy                  "sip:163.22.21.82:5060"

    int i,m,id,j,n;
    int SIP_ACCOUNT_ADD;
    int SIP_S_ACCOUNT_ADD;
    osip_message_t              *sip = NULL;
    osip_message_t              *invite = NULL;
    osip_message_t              *ack = NULL;
    osip_message_t              *answer = NULL;
    eXosip_event_t              *event;
```

```
    char                    *d_port = NULL;
    char                    *d_address = NULL;
    char                    *s_address = NULL;
    char                    *s_port = NULL;
    char                    *username = NULL;
    sdp_message_t           *sdp = NULL;
    sdp_message_t           *s_sdp = NULL;

int initial_exosip(nstream,account){

    osip_message_t          *reg = NULL;
    int                     id;

    i = eXosip_init();

    if(i!=0){
        printf("Couldn't initialize eXosip!\n");
        return -1;
    }

    for(n=0;n<nstream;n++){

        i = eXosip_listen_addr(IPPROTO_UDP,NULL,SIP_BIND_PORT+n,AF_INET,0);
        usleep(100000);

        char d1[40] = "sip:";
        char d2[40];
        char d3[40] = "@163.22.21.82";
        char s1[40] = "sip:";
        char s2[40];

        SIP_ACCOUNT_ADD = account + n;
        sprintf(d2,"%d",SIP_ACCOUNT_ADD);
```

```
        strcat(d2,d3);
        strcat(d1,d2);
        printf("destination %s\n",d1);

        SIP_S_ACCOUNT_ADD = SIP_S_ACCOUNT + n ;
        sprintf(s2,"%d",SIP_S_ACCOUNT_ADD);
        strcat(s2,d3);
        strcat(s1,s2);
        printf("source %s\n",s1);

/* register */

        id = eXosip_register_build_initial_register(s1,proxy,NULL,1800,&reg);

        usleep(100000);
        if(id<0){
            eXosip_unlock();
            printf("Error\n");
            return -1;
        }

        eXosip_lock();
        i=eXosip_register_send_register(id, reg);
        eXosip_unlock();

        if (i!=0)
            printf("Register Error\n");

        i=eXosip_call_build_initial_invite(&invite,d1,s1,NULL,"This is a call for a conversation");

        usleep(100000);

        if(i!=0){
```

```
            printf("Error\n");
            return -1;
        }

        osip_message_set_supported(invite, "100rel");
        {
            char tmp[4096];
            char localip[128];

            eXosip_guess_localip (AF_INET, localip, 128);

            snprintf (tmp, 4096,
                      "v=0\r\n"
                      "o=josua 0 0 IN IP4 %s\r\n"
                      "s=conversation\r\n"
                      "c=IN IP4 %s\r\n"
                      "t=0 0\r\n"
                      "m=audio %d RTP/AVP 0 8 18 101\r\n"
                      "a=rtpmap:0 PCMU/8000\r\n",
                          SIP_LOCAL_IP,SIP_LOCAL_IP,RTP_PORT+2*n);

            osip_message_set_body (invite, tmp, strlen (tmp));
            osip_message_set_content_type (invite, "application/sdp");
        }

        eXosip_lock ();
        i = eXosip_call_send_initial_invite (invite);
        eXosip_unlock ();
    }
    return 0;
}

int session_setup(nstream,t_conl){
```

```
FILE *fp;
time_t the_time;
int result;
int q = 0;
char command[ch_command];

while(1){
    event = eXosip_event_wait(200,0);

    if(event == NULL){
        eXosip_quit ();
        printf("End of session\n");
        return -1;
    }

    switch(event ->type){
        case EXOSIP_CALL_REINVITE:
            printf("A new invite reveived!\n");
            break;

        case EXOSIP_CALL_PROCEEDING:
           // printf("Proceeding!\n");
            break;

        case EXOSIP_CALL_RINGING:
            printf("Ringing!\n");
            break;

        case EXOSIP_CALL_ANSWERED:
            //usleep(100000);
            printf("Connected!\n");
            s_sdp = eXosip_get_sdp_info(event->request);
```

```
            s_address = sdp_message_o_addr_get(s_sdp);
            s_port = sdp_message_m_port_get(s_sdp,0);

            printf("local %s : %s\n",s_address,s_port);

            sdp = eXosip_get_sdp_info(event->response);
            d_address = sdp_message_c_addr_get(sdp,-1,0);
            d_port = sdp_message_m_port_get(sdp,0);
            printf("RTP packets will send to %s : %s\n",d_address,d_port);

            eXosip_call_build_ack(event->did,&ack);
            eXosip_call_send_ack(event->did,ack);

            sprintf(command,"./rtpsend 1.wma %s %s %s %s
%d&\n",d_address,d_port,SIP_LOCAL_IP,s_port,t_conl);

            system(command);
            q++;
            printf("session :  %d\n",q);
            usleep(100000);

            if(q == nstream){
                //eXosip_call_terminate(event->cid,event->did); //send BYE message
                eXosip_quit();
                return -1;
            }
            break;

        case EXOSIP_CALL_ACK:
            printf("ACK received!\n");
            break;

        case EXOSIP_CALL_CLOSED:
            printf("Callee close the session!\n");
```

```
                i = eXosip_call_build_answer (event->tid, 200, &answer);

                if (i != 0){
                    printf ("This request msg is invalid!Cann't response!\n");
                    eXosip_call_send_answer (event->tid, 400, NULL);
                }
                else{
                    eXosip_call_send_answer (event->tid, 200, answer);
                    printf("Send 200 OK\n");
                    eXosip_quit ();
                    return -1;
                }
                break;
            }
        }
    }

int main(){
    int server_sockfd, client_sockfd;
    int server_len, client_len;
    int sockfd;
    int len,k=0;
    int nstream, t_conl, account;
    time_t the_time;
    FILE *fp;

    struct sockaddr_in address;
    struct sockaddr_in server_address;
    struct sockaddr_in client_address;

    system("ntpdate 10.10.59.61");

/* socket */
```

```
    server_sockfd = socket(AF_INET, SOCK_STREAM, 0);
    server_address.sin_family = AF_INET;
    server_address.sin_addr.s_addr =htonl(INADDR_ANY);
    server_address.sin_port =9000;
    server_len = sizeof(server_address);
    bind(server_sockfd, (struct sockaddr *)&server_address, server_len);

    for(;;){
        listen(server_sockfd,5 );
        printf("Slave waiting\n");
        client_len = sizeof(client_address);        //master 判斷對應的 port 有無啟動，確保 slave_sender
啟動
        client_sockfd = accept(server_sockfd,
            (struct sockaddr *)&client_address, &client_len);

        client_len = sizeof(client_address);       //接收指令
        client_sockfd = accept(server_sockfd,
            (struct sockaddr *)&client_address, &client_len);

        read(client_sockfd,&t_conl,sizeof(int));
        read(client_sockfd,&nstream,sizeof(int));
        read(client_sockfd,&account,sizeof(int));

        usleep(1000000);
        initial_exosip(nstream,account);
        session_setup(nstream,t_conl);
        eXosip_quit ();
    }
    return 0;
}
```

附件 C.　Code of slave_receiver.c

```
#define SIP_BIND_PORT              5066
#define SIP_PROXY_ADDRESS        "163.22.21.82"
#define proxy                          "sip:163.22.21.82:5060"
#define RTP_PORT                    60000
#define SIP_LOCAL_IP               "10.10.59.61" //設定 local IP address

int do_register(j,account){

    osip_message_t          *reg = NULL;
    int                         id;
    int                         i;
    int                         SIP_ACCOUNT_ADD;

    char s1[40] = "sip:";
    char s2[40];
    char s3[40] = "@163.22.21.82";

    SIP_ACCOUNT_ADD = account+ j ;
    sprintf(s2,"%d",SIP_ACCOUNT_ADD);
    strcat(s2,s3);
    strcat(s1,s2);

    printf("%s\n",s1);
    id = eXosip_register_build_initial_register(s1,proxy,NULL,1800,&reg);
    usleep(100000);
    if(id<0){
```

```
        eXosip_unlock();
        printf("Error\n");
        return -1;
    }

    eXosip_lock();
    i=eXosip_register_send_register(id, reg);
    eXosip_unlock();
    if (i!=0)
        printf("Register Error\n");

}

int recv_rtp_packet(nstream,t_conl){

    FILE *fp;
    struct tm *tm_ptr;

    char command[40]="./mrtprecv 1 ";
    char command1[21];
    sprintf(command1,"%d %d %d",RTP_PORT,nstream,t_conl);
    strcat(command,command1);
    //printf("%s\n", command);
    system(command);
    return 0;
}

int main(){
    int             i,j,n=0,m=0;
```

```
    int             nstream,account;
    char            tmp[4096];
    char            *m_port = NULL;
    char            *c_address = NULL;

    eXosip_event_t *event;
    osip_message_t *answer = NULL;
    sdp_message_t    *sdp = NULL;

    int server_sockfd, client_sockfd;
    int server_len, client_len;
    int len, t_conl;
    struct sockaddr_in address;
    struct sockaddr_in server_address;
    struct sockaddr_in client_address;

/* socket */

    server_sockfd = socket(AF_INET, SOCK_STREAM, 0);
    server_address.sin_family = AF_INET;
    server_address.sin_addr.s_addr = htonl(INADDR_ANY);
    server_address.sin_port =9000;
    server_len = sizeof(server_address);
    bind(server_sockfd, (struct sockaddr *)&server_address, server_len);

    for(;;){
        listen(server_sockfd,5);
        printf("Receiver waiting\n");
        client_len = sizeof(client_address);
```

```
client_sockfd = accept(server_sockfd,
    (struct sockaddr *)&client_address, &client_len);

client_len = sizeof(client_address);
client_sockfd = accept(server_sockfd,
    (struct sockaddr *)&client_address, &client_len);
read(client_sockfd,&t_conl,sizeof(int));
read(client_sockfd,&nstream,sizeof(int));
read(client_sockfd,&account,sizeof(int));

i = eXosip_init();

if(i!=0){
    printf("Couldn't initialize eXosip!\n");
    return -1;
}

for(j=0;j<nstream;j++){
    i = eXosip_listen_addr(IPPROTO_UDP,NULL,SIP_BIND_PORT+j,AF_INET,0);
    //usleep(100000);
    do_register(j,account);
}
for(;;){
    event = eXosip_event_wait(50,0);
    eXosip_lock ();
    eXosip_default_action (event);
    eXosip_automatic_refresh ();
    eXosip_unlock ();
```

```
if(event==NULL)
    continue;

switch(event->type){
    case EXOSIP_MESSAGE_NEW:
        printf("EXOSIP_MESSAGE_NEW!\n");
        break;

    case EXOSIP_CALL_INVITE:
        usleep(100000);
        printf("Invite %s user\n",event->request->req_uri->username);

        sdp = eXosip_get_sdp_info(event->request);
        c_address = sdp_message_c_addr_get(sdp,-1,0);
        m_port = sdp_message_m_port_get(sdp,0);

        eXosip_lock ();
        //eXosip_call_send_answer (event->tid, 180, NULL);
        eXosip_call_build_answer (event->tid, 200, &answer);

        snprintf (tmp, 4096,
                  "v=0\r\n"
                  "o=josua 0 0 IN IP4 %s\r\n"
                  "s=SIP Call\r\n"
                  "c=IN IP4 %s\r\n"
                  "t=0 0\r\n"
                  "m=audio %d RTP/AVP 0 8 18 101\r\n"
                  "a=rtpmap:0 PCMU/8000/1\r\n",
                      SIP_LOCAL_IP,SIP_LOCAL_IP,RTP_PORT+2*n);
```

```
        n++;

        osip_message_set_body (answer, tmp, strlen(tmp));
        osip_message_set_content_type (answer, "application/sdp");
        eXosip_call_send_answer (event->tid, 200, answer);
        eXosip_unlock();
        //printf("Send 200 OK\n");
        printf("RTP packets will send from:\n%s : %s\n\n",c_address,m_port);
        break;

    case EXOSIP_CALL_ACK:
        //printf("ACK received!\n");
        usleep(100000);
        m++;
        if(m == nstream){
            recv_rtp_packet(nstream,t_conl);
        }
        break;

    case EXOSIP_CALL_CLOSED:      //received BYE message
        printf("Caller close the session!\n");
        i = eXosip_call_build_answer (event->tid, 200, &answer);
        if (i != 0){
            printf ("This request msg is invalid!Cann't response!\n");
            eXosip_call_send_answer (event->tid, 400, NULL);
        }
        else{
            eXosip_call_send_answer (event->tid, 200, answer);
            printf("Send 200 OK\n");
```

```
                    return -1;
                }
                break;

            case EXOSIP_CALL_ANSWERED:
                printf("Received 200 OK\n");
                break;
              if(m == nstream)
                break;
            }
            if(m == nstream)
                break;
        }
        eXosip_register_remove(1);
        eXosip_quit();
        m =0;
        n =0;
    }
}
```