基於 Linux 平台之 SIP 網路電話動態頻寬保證及允入控制雛型實作

The Prototype Implementation of SIP Phone Dynamic Bandwidth Guarantee and Admission Control on Linux

邱家偉
國立中正大學通訊工程學系
cgw@ant.comm.ccu.edu.tw

潘仁義
國立中正大學通訊工程學系
國立中正大學電機工程學系
國立中正大學電信研究中心
jypan@comm.ccu.edu.tw

摘要

隨著網際網路飛快地成長，各式各樣的網路應用與服務也因應而生，大量的資料傳輸使得頻寬利用及分配越顯得重要，特別是即時性的資料傳輸。以現今最熱門的網路電話(Voice over IP；VoIP)為例，雖然每通語音連線所佔用的頻寬並不大，大約為 8 Kbits 到 64 Kbits 之間，但在傳輸過程中，會受到傳輸媒介的影響，如：網路壅塞、網路頻寬不足，等等，導致語音封包無法在一定的時間內抵達，並且語音封包所使用的通訊埠並不固定，所以也就無法直接使用通訊埠來判別是否為語音封包而不是一般的資料。

為了解決上述問題，我們主要針對語音傳輸所作的頻寬管理及允入控制。此機制是透過 Netfilter 擷取封包內容並且分析 Session Initiation Protocol 協定中所帶的 Session Description Protocol 資訊，如：語音壓縮種類和通訊埠，等等。若剩餘頻寬足夠則予以配置所需頻寬，此方式是藉由 Traffic Control 工具來達成頻寬配置。使用此機制可以確保使用 VoIP 時就像是使用電路交換式網路一樣，具有服務品質的保證，且對於使用者的 VoIP 軟體或硬體並不需要做任何修改或設定。

關鍵詞：階層式頻寬管理、允入控制、會議初始協定、服務品質、網路電話

Abstract

With the rise and growth of the Internet, various Internet applications and services are rapidly developed. Voice over IP (VoIP) is the popular one of these services. However, its real-time characteristics demands strict network quality. For example, each VoIP session requires bandwidth range from 8 Kbits to 64 Kbits, but the voice quality may steeply degrade with the congested network or slender bandwidth. Hence, bandwidth management plays an important role to satisfy the Quality of Service (QoS) requirement of data transmission for real-time applications such as VoIP. Nevertheless we do not know which port the data will be transmitted via, so we can not reserve the bandwidth of specific ports for VoIP transmission in advance.

We propose a bandwidth reservation and admission control mechanism implemented on Linux to solve the problem mentioned above without modification to users' VoIP equipments or software. Our mechanism captures packets with Netfilter and parses the Session Description Protocol (SDP) payload included in Session Initiation Protocol (SIP) packets to obtain more detail information, such as voice codec, port number and so on. Traffic Control tool will allocate bandwidth for this call if available bandwidth is enough. Finally, we verify our mechanism by implementing the prototype and measuring its performance on the test bed.

Keywords: Hierarchical Traffic Management, Admission Control, Session Initiation Protocol, Quality of Service, Voice over IP

一、簡介

隨著網路資訊的多樣化，單純的文字應用已不足以滿足使用者的需求，以往網際網路並沒有提供服務品質保證的方法，所以往往在傳輸的過程中會因當時的傳輸量而影響傳輸速度。當不同封包在同一時間點都要進行傳輸時，就必須搶奪頻寬資源。有些對於時間較為敏感的資料，如：即時視訊、語音或是線上即時交易等資料，對頻寬的需求及延遲的敏感度，都高於一般如：網頁瀏覽、E-mail 或 FTP 檔案傳輸等應用。然而因為缺乏適當的管理機制，造成這些資料被夾雜在非即時資料中傳輸，造成延遲或是遺失。

Session Initiation Protocol (SIP) [7]是一種用來建立、修改、終止使用者 Session 的應用層(Application Layer)控制協定，和 Hyper-Text Transfer Protocol (HTTP)及 Simple Mail Transport

Protocol (SMTP)一樣都是使用明文方式來傳輸訊息。目前 SIP 協定已經漸漸被發展 IP 電話的廠商所採用，並且有許多學術團體正在持續地做相關研究，相信是未來多媒體傳輸的主流。並且在第三代(3G)行動通訊網路中，3rd Generation Partnership Project (3GPP)組織在 2000 年決定以 SIP 做為 IP 行動網路的連線控制協定[1]。SIP 將成為自 HTTP 和 SMTP 以來最為重要的協定。

服務品質(QoS)的議題在近年來被廣泛地研究與討論，IETF 相繼提出了 Integrated Services (IntServ) [12]及 Differentiated Services (DiffServ) [13]來解決，但還是存在擴充性和實作上的問題。因此，出現以網路邊緣點為基礎的 Linux QoS 佇列機制，它可以依不同的網路需求建立各自所需的 QoS 機制，對於擴充性和實作上都具有不錯的能力。Linux 核心提供許多佇列機制，而在眾多的佇列機制中，以 Link-Sharing 架構為主的分類機制，可以用來保證每個使用者獲得一定額度的頻寬，尤其以 Hierarchical Token Bucket (HTB) [5]和 Class Based Queuing (CBQ) [2]為主要的實現機制。HTB 是最近幾年來被提出來的佇列機制，比 CBQ 更容易理解、更直覺且更快速，它可以很容易地保證每個類別的頻寬，也允許特定的類別可以突破頻寬上限，借用其它類別未使用的頻寬，也能夠劃分類別的優先順序。

為了能夠決定語音通話的建立與否以及頻寬使用的大小，我們使用了 Linux 核心中重要的組成元件 Netfilter [10]，它是 Linux 防火牆以及 IPTables [6]的實現基礎，透過此方式可在封包被轉送前做好決策，以利允入控制之實現。

本論文的架構如下：第一節為簡介；第二節則闡述使用階層式頻寬管理的架構及其優點；第三節將說明允入控制及可用度的重要性；第四節為實作系統架構的介紹說明與實測；最後為結論。

二、階層式頻寬管理

在正式進入主題之前，首先對於 Linux 核心上實現 QoS 的方式作簡單地介紹，對於其後所介紹的系統架構才會對此系統的運作流程有更清楚的體會。

2.1 Linux 封包處理流程

圖 1 顯示 Linux 核心處理封包的過程：輸入裝置→進入策略→輸入解多工（判斷是本地還是轉發）→轉發→輸出佇列→輸出裝置。進入策略（Ingress Policing）和輸出佇列（Output Queuing）是由 Linux 核心的流量控制所實現，進入策略丟棄不符合規定的封包，確保進入的各個服務封包速率的上限；輸出佇列依據配置實現封包的排隊、分類和丟棄。

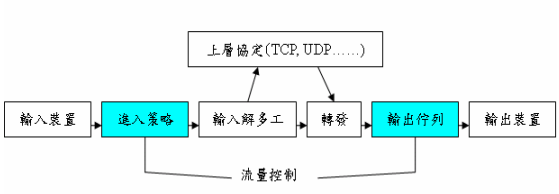


圖 1 封包的處理過程

而流量控制多半都是透過輸出佇列來實現，因每個輸出裝置在封包輸出前會先經過一個 Buffer 暫存，若沒有做特別設定，則此 Buffer 為一個先進先出(FIFO)佇列，對封包並無作任何處理則直接輸出。Linux 核心提供三種物件來對輸出佇列作規劃，分別是：佇列規則（QDisc）、類別（Class）和過濾器（Filter）。

- QDisc

QDisc 是 Queuing Discipline 的縮寫，它是理解流量控制的基礎。無論何時，核心如果需要通過某個網路介面發送封包，它都需要依照這個介面配置的 QDisc 把封包加入佇列。然後，核心會盡可能地從 QDisc 裡面取出封包，把它們交給網路介面驅動模組。最簡單的 QDisc 是 PFIFO，它不對進入的封包做任何處理，封包採用先入先出的方式通過佇列。不過，它會保存網路介面一時無法處理的封包。

- Class

某些 QDisc 可以包含一些類別，不同的類別中可以包含更深入的 QDisc，也就是階層式架構，通過這些細分的 QDisc 還可以為進入佇列的封包排程。經由設定各種類別的封包離開佇列順序，可以設置網路資料流量的優先順序。

- Filter

Filter 用於為封包分類，決定它們按照何種 QDisc 進入佇列。無論何時封包進入一個劃分子類的類別中，都需要進行分類。分類的方法可以有許多種，使用 Filter 就是其中之一。使用 Filter 分類時，核心會調用附屬於此 Class 的所有過濾條件。

2.2 Link-Sharing

Linux 核心提供的佇列規則可分為可分類的佇列規則(classful queuing disciplines)與不可分類的佇列規則(classless queuing disciplines)，可分類的佇列規則有：CBQ、HTB、PRIO 而不可分類的佇列規則有：PFIFO_FAST、Bytes FIFO (BFIFO)、Packets FIFO (PFIFO)、Random Early Detect (RED)、Stochastic Fairness Queuing (SFQ)及 Token Bucket Filter (TBF) .....，一個網路介面上如果沒有設置 QDisc，PFIFO_FAST 就是預設的 QDisc。

可分類的佇列規則是以學者 Sally Floyd 所提出鏈結共享（Link-Sharing）及資源管理機制[14]，主要目的是將鏈結資源頻寬共享分配給多種應用，使得每一種應用需求在網路壅塞時還能獲得其

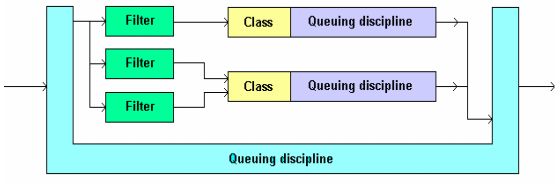


圖 2 具有流量控制之可分類佇列架構

最低鏈結資源的保證，如此鏈結就可以同時分享頻寬給各種不同類型的應用，除此以外，也可以利用優先權保障，使用一些封包排程演算法來進行管理，並保障具有較高優先權服務的應用。

所謂的可分類的佇列規則，如圖 2 所示，主要的意思是可在佇列規則中再內置相同或不同的 QDisc 及 Class 形成巢狀式或稱之為樹狀式架構，再透過過濾器來對指定的封包類型進行分類，達到流量控管的作用。

如圖 3 所示為一個以 HTB 所建構出的基本階層式樹狀架構，最上層的 Link 節點為輸出佇列的最外層的 QDisc，為最大所能輸出的頻寬，稱之為 root qdisc，再透過 Class 區分成三群使用者 Agency A、Agency B、Agency C，分別各自佔用總頻寬的 50%、40%、10%，但也不是被限死，而是給予最低所能達到的保證，也就是說當其中某群沒有使用到最低所給予的頻寬，則其他群組可以借出剩餘的頻寬，而這三群可再依其應用不同，再細分成不同的頻寬分配，完全視其所需將頻寬作完善的分配管理，最後都須透過 Filter 來將各種類型的封包導入相對應的 Class 中。

2.3 基於即時性資料之頻寬階層架構

基於上述所提及的觀念，為了能使得有限的頻寬利用率達到最好，透過 Link-Sharing 的機制與即時性資料做結合可以保證語音的品質。我們將輸出佇列的總頻寬分為兩類，分別為固定式頻寬分配與動態式頻寬分配，所謂的固定式頻寬分配是將已知的協定種類在系統啟動時就已被清楚地指定此類型封包的輸出能力，而動態式頻寬分配是此篇最主要的核心，因即時性的資料所使用的通訊埠及編碼大小都不固定，所以當有需要時才予以配置頻寬，此方式比市面上需一一指定語音通訊埠的產品還來得有效且實用，並且不是即時性資料互相搶奪頻寬資源。

對於固定式頻寬分配，可再將之分成對人與對服務來分類，以對服務來看，可將一些重要的協定給予較高的優先權以及應有的頻寬，如 icmp、ssh .....，若以對人來看，可以限制某個使用者所能使用的最大與最小頻寬，將其延伸，可再對使用者所使用的服務種類做頻寬保證或限制。圖 4 為上述所要表達意思，依網路應用所需將 root 總頻寬分成靜態與動態各佔不同的比例，除了靜態頻寬在啟動時已將所有的設定載入到核心之外，動態頻寬配置則當建立連線時才向核心要求此一頻寬，連線結束時則釋放此一頻寬，給予之後需要配置頻寬的連線之用。但頻寬資源有限，如何將這些資源發揮到淋漓盡致，是一項不可忽視的要素。在第三節中會介紹對於即時性資料如何做有效的控管，來提升網路的可用度。

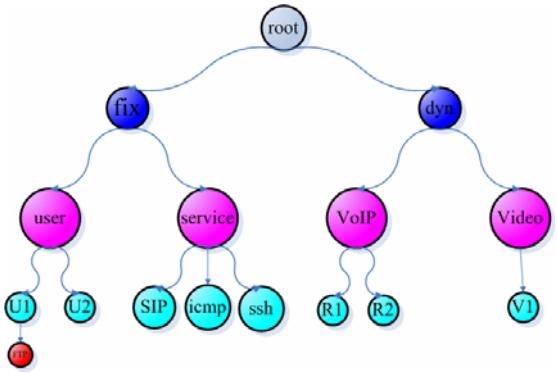


圖 4 階層式頻寬管理架構圖

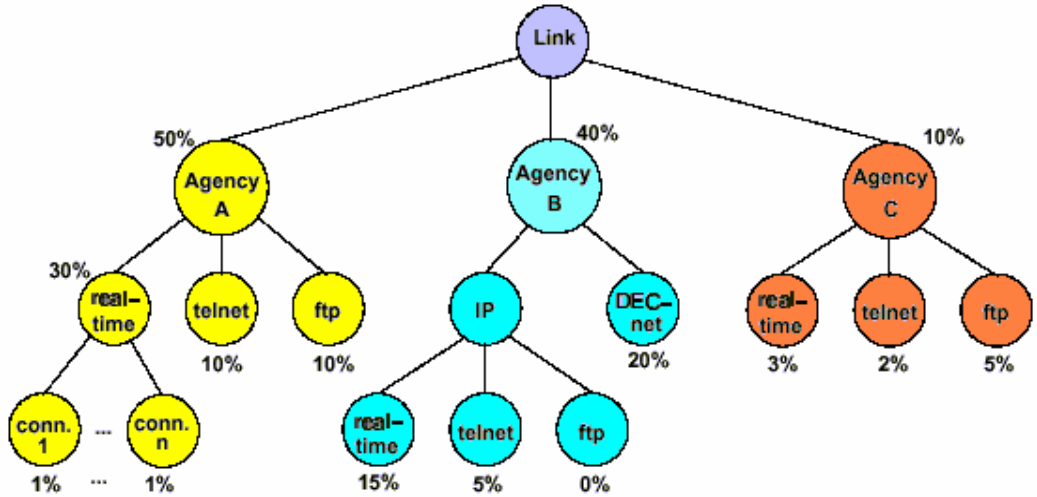


圖 3 一個階層式 Link-Sharing 的架構[14]

三、允入控制及可用度保證

3.1 允入控制

如果頻寬管理系統同意接受此連線之建立則必須給予其所提出的服務品質保障(QoS guarantee)。也就是說，網路將盡其所能使該連線的資料在傳送時能享有其服務品質。如果該頻寬管理系統認為接受此連線之建立會影響到其他已經建立的連線的服務品質，例如：頻寬資源不足，則可以拒絕此連線之建立要求。這種控制連線能否建立的機構便稱為「允入控制」(Admission control)。為了能使得分封交換網路具有電路交換網路的特性，當使用者撥通電話時，此頻道已被佔用，無法挪出給其他人使用。因此，語音封包在傳輸過程中，必須分配一個專屬頻寬，之後再透過 scheduler 來保證它的傳送速度。換句話說，相對於網路必須提供服務品質的保障給每一條連線，每一條連線在傳送資料時也應該遵守當初的約定。

為了能夠對即時性的資料給予應有的保證，必須分析封包內容才能做後續的處理，比如所需的頻寬以及所使用的通訊埠等。若是頻寬已被分配完了，則須拒絕此一連線要求，發送一封包通知對方無法建立此一連線，而不是讓使用者一直嘗試建立連線。此一方式是透過 Linux 核心所提供的功能，也是實現 Linux 防火牆與 IPTables 的主要機制－Netfilter。

Netfilter 本身在 IP 層內提供了的 5 個插入點(稱為 HOOK)，而在每個插入點上登記了一些處理函式進行處理，如可以提供 Network Address Translation (NAT)，封包過濾，甚至可以是用戶自行定義的功能功能。

- ROUTE(1)：

  對收到的封包做路由查找並判斷這個封包是需要轉送的封包還是發往本機上層的封包。

- ROUTE(2)：

  查找發出封包的路由。

- NF_IP_PRE_ROUTING：

  對所有傳入 IP 層的封包進行檢查，在這之前，有關封包的版本、長度、校驗和等正確性檢查已經完成。

- NF_IP_LOCAL_IN：

  對發往本機上層的封包進行檢查。

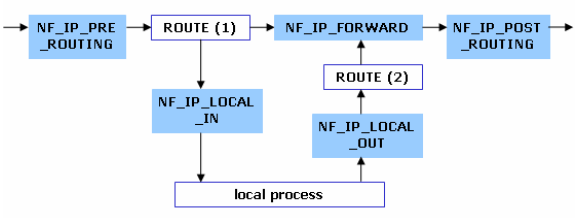


圖 5 Netfilter 主要架構

- NF_IP_FORWARD：

  檢查需要轉送的封包。

- NF_IP_POST_ROUTING：

  對所有向資料連結層傳遞的封包進行檢查，注意在此處封包的路由已經確定。

- NF_IP_LOCAL_OUT：

  本機發出的封包進行檢查，此處的路由還沒有確定，所以可以做到目的位址轉換。

這些點是已經在核心中定義好的，核心模組能夠註冊這些 HOOK 點所進行的處理，可使用 nf_register_hook() 函數指定。在封包經過這些 HOOK 函數時被呼叫，而且模組可以修改這些封包，並向 Netfilter 返回如下值：

- NF_ACCEPT：繼續正常傳輸封包
- NF_DROP：丟棄此封包，不再傳輸
- NF_STOLEN：模組接管此封包，不再傳輸此封包
- NF_QUEUE：對該封包進行排隊
- NF_REPEAT：再次呼叫該 HOOK 函數

3.2 可用度之保證

透過學者 Sally Floyd 所提出的想法[14]，不僅能提供彈性頻寬控制方案，讓不同應用網路階層獲得保障的頻寬，尤其對於即時性的資料，使得對外頻寬達到最佳使用率，如圖 6 所示，每個對外的網路透過已實作在 Intel XScale® Processor (IXP)英特爾網路處理器上的頻寬管理系統，不僅能對頻寬作有效的控管，也可以保證當地的頻寬不會受到內部或外部網路的影響，並且能使得像 Denial of Service (DoS) [3]般的頻寬資源耗損能降至最低，避免影響到其它網路應用的運行。正因有此特性，網路的可用度不會受到任何的干擾，對於已建立連線的即時性應用，能保證頻寬及延遲特性。

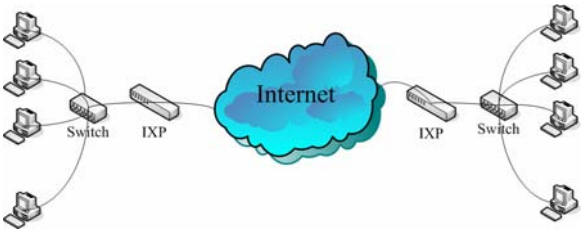


圖 6 建構頻寬管理系統拓撲

四、系統架構

由圖 7 所示之架構，將其實作在網路處理器平台之上，以期獲得快速處理封包之能力。此系統分為五大部分，SIP 模組、頻寬配置模組和 Linux QoS 是位在 kernel space [15]之中，tc 指令代理程式和 Admin User Interface (Admin UI) 是位於 user space，各自負責不同的工作，除了 Linux QoS 是

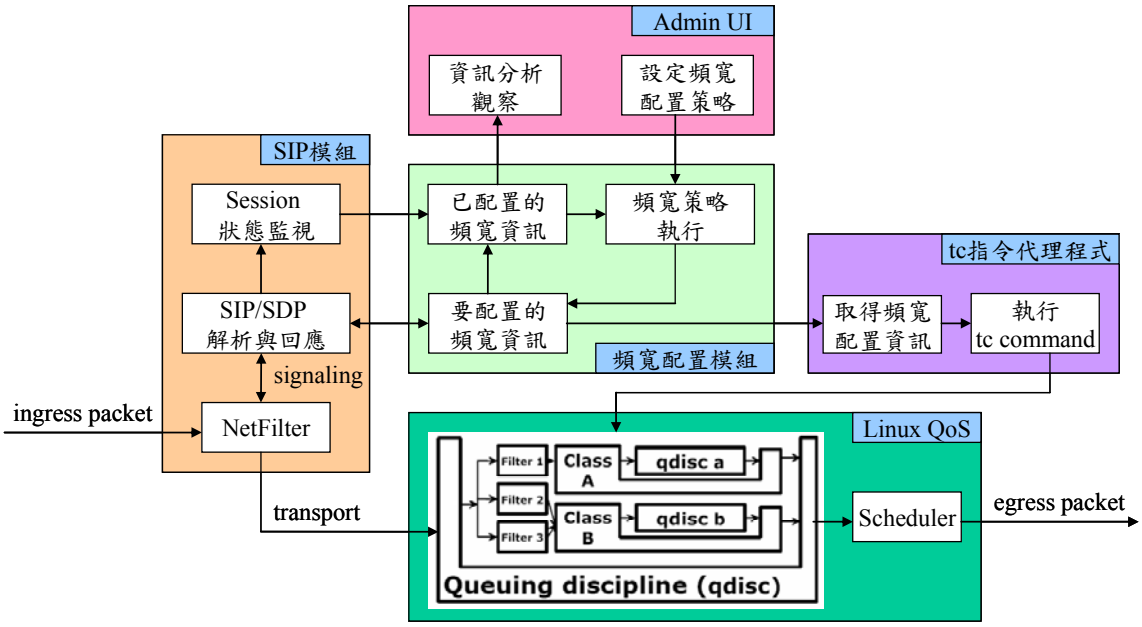


圖 7 系統架構圖

利用核心已提供之功能，其餘都是依照工作項目所實作出來的模組。下面簡單地描述整個系統運作的步驟流程。

4.1 系統方塊說明

每個方塊所負責工作內容，條列如下:

(一) SIP 模組

工作項目如圖 8 所示，主要是分析封包內容，決定是否為此通話保留頻寬，在 4.2 節中會有較詳細之過程說明。

(二) 頻寬配置模組

- 執行 Admin UI 所設定的頻寬配置策略及已配置頻寬的統計資訊的提供。
- 提供給 SIP 模組之用的頻寬要求介面及 Session 狀態資訊的取得。
- 負責與 tc 指令代理程式溝通，將要求的資訊傳遞給 tc 指令代理程式，來達到動態頻寬分配，並且也會記錄目前已要求頻寬的資訊，以利查詢目前的使用情況以及刪除指定的頻寬配置對象。

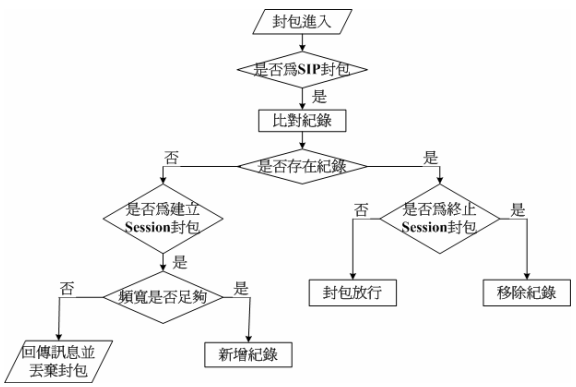


圖 8 SIP 封包運行流程

(三) tc 指令代理程式

主要負責接收頻寬配置模組所要求的資訊，然後呼叫使用者程式 Traffic Control (TC) tool [8]，產生相對應的 QDisc、Class、Filter。

(四) Linux QoS

此部份是使用 Linux 核心所提供的 QoS 機制，透過 TC tool 向核心下達頻寬配置命令。

(五) Admin UI

- 設定網路介面參數
- 設定語音最大的頻寬使用總量或是通話數以及一些固定 Port 的應用程式之頻寬大小
- 檢視目前通話數目與時間
- 查詢所分配的頻寬使用情況

4.2 運行過程

因為方塊與方塊之間有些具有依存性，因此要啟動此系統需依序載入模組與程式，首先，先將頻寬配置模組載入，再將 tc 指令代理程式執行，之後的頻寬配置模組會將之前設定好的頻寬策略向 tc 指令代理程式要求配置預設好的頻寬資訊，而 tc 指令代理程式會與核心溝通，針對不同的輸出裝置指定佇列規則、封包類別和過濾器，然後再把 SIP 模組載入，我們就可以針對語音封包進行頻寬保證的目的。

首先，當有使用者要建立語音通話時，如圖 9 所示，在 UAS 和 UAC 中會經過許多的網路裝置而在其兩個使用者的對外網路接口都已放置此篇所提出的動態頻寬配置管理器，所有封包都會經過此

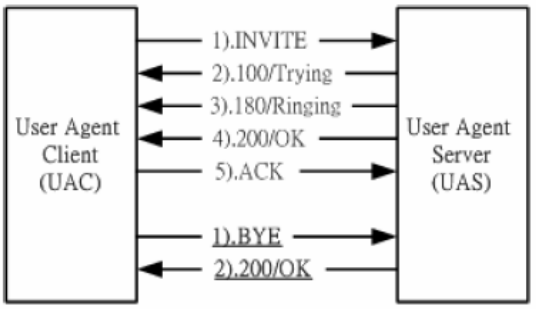


圖 9 SIP 語音通話建立與結束簡示圖

裝置，當有封包進入此裝置時，如圖 8 所示，會先呼叫向 Netfilter 所註冊的函式，主要判別封包種類，若此封包不屬於 SIP，則讓此封包直接通過，頻寬的控管交給其後的輸出佇列。若是屬於 SIP 封包且為 INVITE signaling，則分析此封包所帶的 SDP [9]內容，藉此取得連線所使用的語音編碼種類及使用的通訊埠[4]，然後查詢計算此語音壓縮需佔用多大頻寬，與頻寬配置模組詢問後，若剩餘頻寬足夠，才能向核心配置此連線所需的頻寬，否則發送封包拒絕對方。

頻寬配置成功之後，我們仍須對每條已連線的通話進行監控，依照連線中的 signaling 做後續的處理，像是偵測到通話結束的 BYE signaling，我們便可將此連線所配置的頻寬給釋放。若是屬於語音封包則判別是否屬於連線中的語音封包，若不是則將它丟棄。

4.3 使用介面設定

為了能使操作及設定簡化，此架構提供介面設定來幫助使用者操作與使用此系統，而大部份的網路設備的使用介面都是採用 Web 方式，因此本篇也以此其為介面方式，圖 10 為主要設定 QoS 策略的主要項目。

此介面設定能夠設定選擇是否要啟動 QoS 的功能，若啟用它則其後的設定才有作用。首先是設定此裝置的最大上傳與下載的頻寬，之後你可以針對靜態配置與動態配置來設定你所需之要求，靜態配置的設定可分為以使用者和應用服務種類做區分的設定，若你僅想對服務種類做設定，則以使用者為對象之細節設定則可以不用進去設定，或將其設定取消功能即可。最主要的部份為動態配置的設定，此處有兩種即時性資料的區分，目前僅實作完成 VoIP 的部份，Video 不是本機制的主要核心，但原理是相同的，故不特別說明。VoIP 的選項中的優先權應該指定給予它較高的優先權，以期能較先處理此封包，latency 也會最小。另一個選項為能夠讓你指定能接受通話的數量及頻寬，因此可以達到針對 VoIP 來保證通話的品質。

而統計資訊也是不可或缺的一項功能，不僅可以即時地監控使用頻寬的情況，更可依據此資訊來修改配置的設定，此處因透過 IXP 來作為實作平台，類似歷史紀錄或統計資訊則不是使用本機端的儲存空間，需透過 syslog 的方式紀錄到遠端 Server，再將遠端的紀錄訊息透過文字或是圖形來表示，裝置端僅存放設定檔。

4.4 實測結果

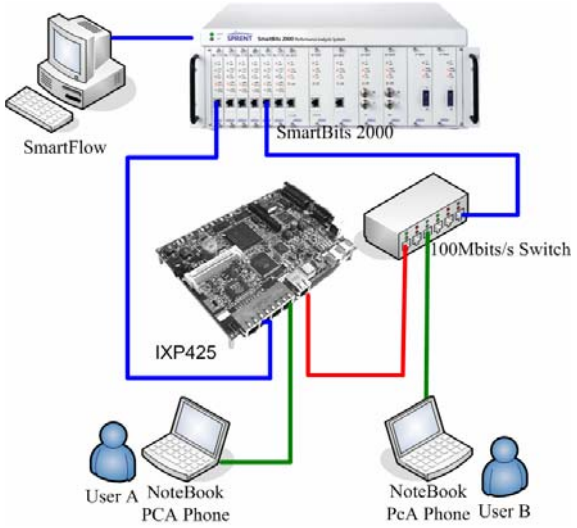


圖 11 實測架構

QoS Init Configuration			
QoS Enable		○ Disable	◉ Enable
網路實際輸出能力 (1:1)		上傳頻寬 10000 kbits/s	下載頻寬 10000 kbits/s
靜態配置 (1:2)	總頻寬	上傳頻寬 9000 kbits/s	下載頻寬 9000 kbits/s
	細節設定 : By User (1:3)	上傳頻寬 1000 kbits/s	下載頻寬 1000 kbits/s
	細節設定 : By Service (1:4)	上傳頻寬 8000 kbits/s	下載頻寬 8000 kbits/s
動態配置 (1:5)	總頻寬	上傳頻寬 1000 kbits/s	下載頻寬 1000 kbits/s
	優先權 1 (1:6)	☑ VoIP = 1000 kbits/s	☑ 連線數 10
	優先權 1 (1:7)	☐ Video= 1 kbits/s	☐ 連線數 1
預設配置 (1:9999)	無過濾符合	上傳頻寬 50 kbits/s	下載頻寬 50 kbits/s

圖 10 頻寬管理介面

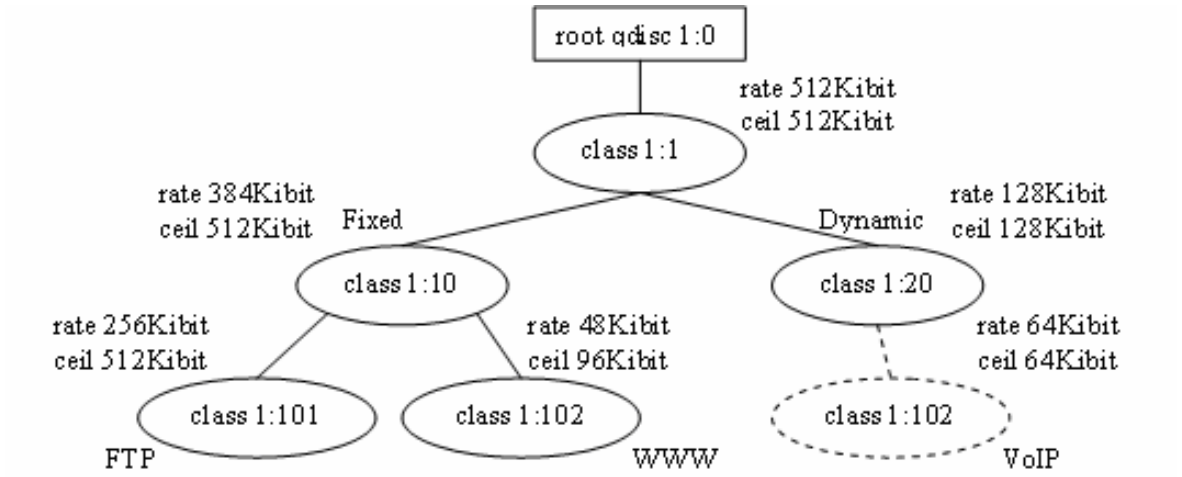


圖 12 頻寬分配表示

由以上的介紹並無法了解實際的頻寬使用情形，為了驗證此機制的可用性，在此對此系統做一個簡單的測試。

如圖 11 所示，實驗平台使用 IXP425 網路處理器作為實際的動態頻寬管理系統，透過 SmartBits® 2000 來產生總和為 100 Mbits 的 FTP 與 WWW 封包而將其包灌入 IXP425 之中，VoIP 則是透過 ITRI 所研發的 PCA SoftPhone 安裝在兩台 WinXP 系統上。

在系統啟動時，已經建立如圖 12 所示之架構分配，其所使用的速率單位 Kibit 是新版 TC 所支援，單位為 1024 bits/s ; Kbit 單位為 1000 bits/s，而舊版只有 Kbit，單位為 1024 bits/s，而圖上所示之虛線是要在通話建立時才會出現。測試時間為 700 秒，前 100 秒並無存取任何數值，僅使用 SmartBits® 2000 產生 FTP 和 WWW 流量，此作法是因為我們是透過 TC 來取得目前速率，又因 TC 是以 10 秒為間隔來計算目前的平均速率，所以在 100 秒後取到的 rate 已是穩定的數值了。

過了 200 秒後，使用 PCA 來建立 VoIP 流量，當頻寬足夠時則才分配頻寬給予此語音連線，連線建立成功後，圖 12 所示的虛線部份才會出現。由圖 13 所示，在 200 秒時，語音通話已經建立，為了能保證語音通話品質，因此需將 FTP 所借用的剩餘頻寬還給具高優先權的類別來使用，在 200 秒到 300 秒所出現的曲線，是因 TC 是以每 10 秒來計算目前的平均速率所致，並不代表實際的速率，通話時間長度為 200 秒，在 400 秒時通話結束，將此配置節點刪除，頻寬還給系統，因此 FTP 又再次地取得剩餘頻寬，而為何 WWW 取不到剩餘頻寬的原因是因其優先權相同但 quantum 會隨著 rate 的不同而異(quantum=rate/r2q)，而 quantum 是頻寬借用量的參考依據，並且它對 classid 較低的

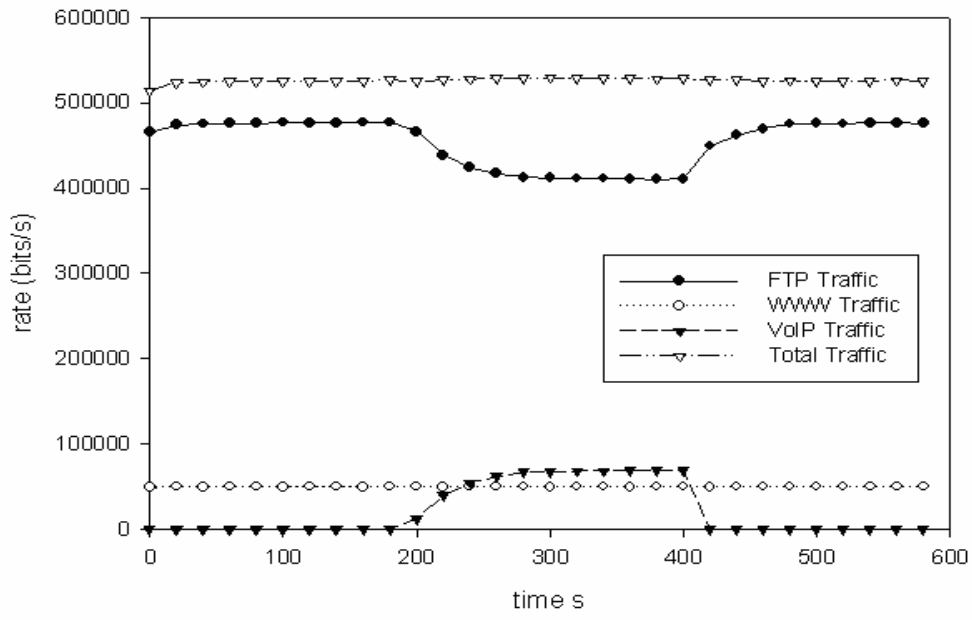


圖 13 量測結果

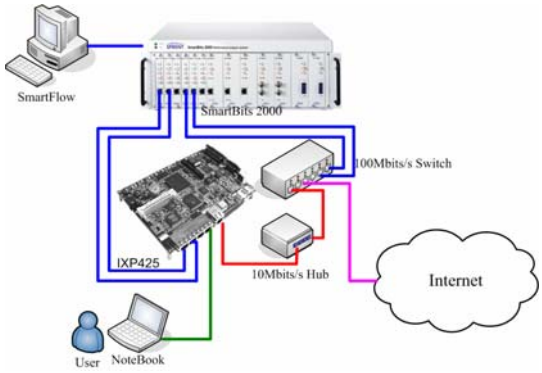


圖 14 適用性量測實驗架構

數值會先處理，因此 WWW 僅獲得保障的頻寬，而無借用到剩餘頻寬。

上述之實例為證明高優先權的類別可從低優先權類別拿回屬於自己所擁有的頻寬，為了證明此架構的優點之處，我們針對語音封包的特性來做量測比較。測試架構如圖 14 所示，詳細的測試環境如表 1 所示。

因即時性的資料對時間特別敏感，必須在一定的時間內抵達，且封包遺失率不能超過 5%，因此需量測有無此機制之封包到達時間與封包遺失率之比較。為了模擬 ADSL 的用戶連線到網際網路上僅僅只有幾 Mbits/s 的速度，在進出 Internet 前的端點接上 10 Mbits/s 的 Hub 來降速，並且利用 SmartBits® 2000 模擬同時有 10 通 VoIP 語音封包以及一個產生 50 Mbits/s 的流量經過實驗平台來測試對即時性資料的影響。由於 SmartBits® 2000 僅能夠產生 Layer 3 和 Layer 4 的封包，無法產生 Layer 7 的封包，因此這裡的十通 VoIP 並非是動態配置得來的結果，是手動指定頻寬，此處僅在量測頻寬分配後的結果，與是否是動態分配頻寬並無直接之關係。而且每通 VoIP 的 Codec 是使用最大的頻寬使用率－G.711，語音資料佔 64 Kbits/s，packetize period: 20 ms，再加上 IP、RTP、Ethernet Header 加總計算後約為 96 Kbits/s。

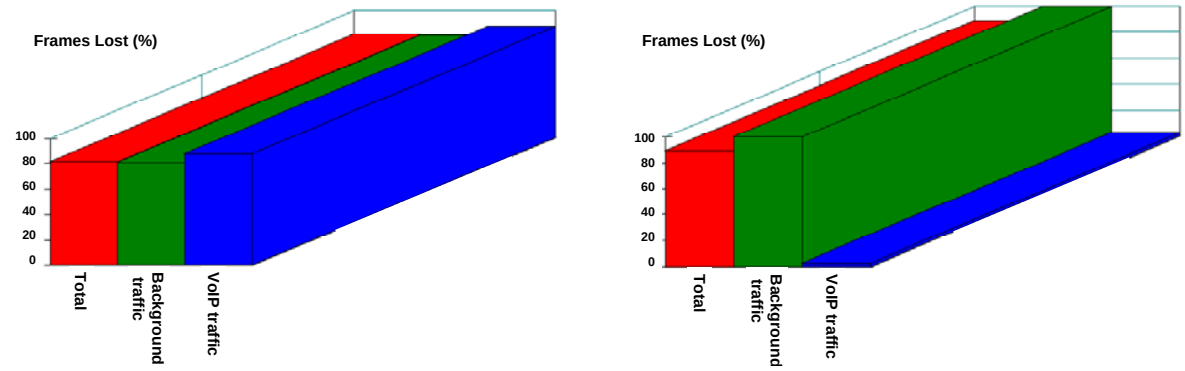
表 1 實測之相關細節

實驗平台	IXP425：LAN x 4, WAN x 1
作業系統	MontaVista® Linux® PE 3.1
附屬設備	100 Mbits Switch x 1 10 Mbits Hub x 1
測試儀器	SmartBits® 2000
測試軟體	SmartFlow™
測試內容	產生一封包長度為 1518 Bytes 速度 50 Mbits/s 之流量 產生 10 條 UDP 流量，封包長度為 218 Bytes 速度 96 Kbits/s

經由 SmartFlow™量測之數據所繪出之圖表來比較使用此機制之優點，如圖 15 所示，為 10 條 VoIP 封包和 50 Mbits/s background traffic 所顯示的封包遺失率之比較，由右圖來看採用本機制所造成的 VoIP 封包遺失率並不高，約在 4%以下，而圖 16、17、18 則顯示採用此機制的封包延遲與分佈，針對圖 18 來看，最左邊的曲線代表 50 Mbits/s background traffic，其餘的為十條 VoIP 的流量。由以上所述的四張比較圖，可以很清楚的了解到有使用此機制的結果，都符合即時性資料的特性，能夠完全在 10 ms 以內完成且延遲分佈都落在同一區域也就是說 jitter 之值不會差異過大，對於即時性資料之處理，可以說是非常地適合。

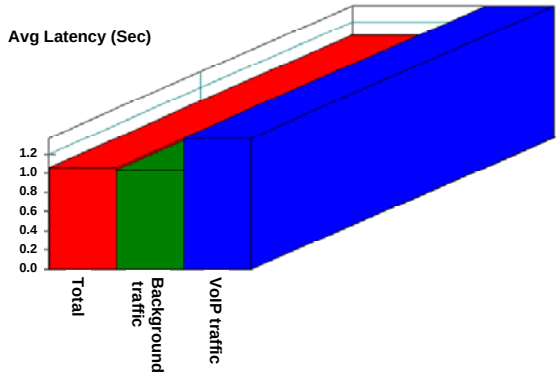
五、結論

網際網路上的應用隨著頻寬的加大而變的更豐富更有趣，但越來越多的多媒體傳輸也相對的消耗更多的頻寬。但是加大頻寬卻無法使使用者滿足，因為使用者無法獨享頻寬，而必須與許多人競爭使用頻寬資源。正因如此，網路的世界也必須有相關的控管機制，依照不同的傳輸提供所需之頻寬大小。

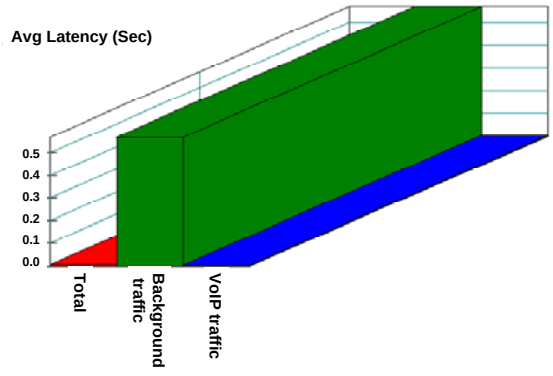


無使用任何機制　　採用本篇之機制

圖 15 封包遺失率之比較

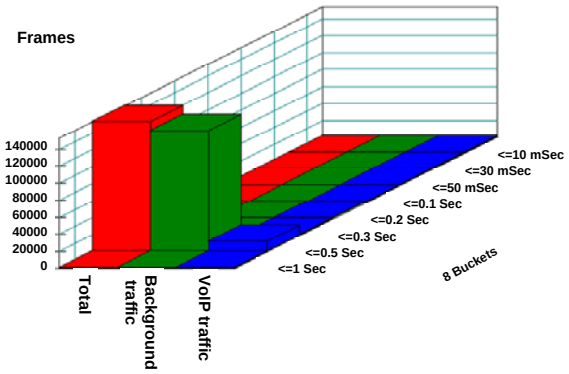


無使用任何機制

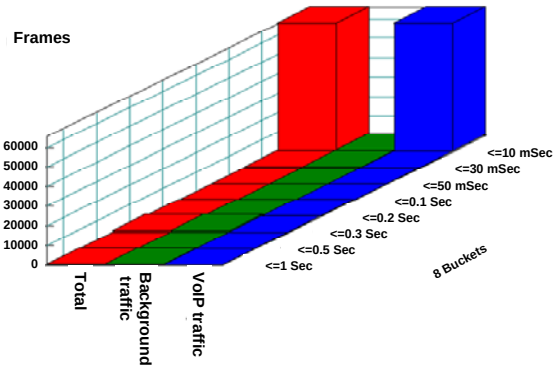


採用本篇之機制

圖 16 封包平均到達時間

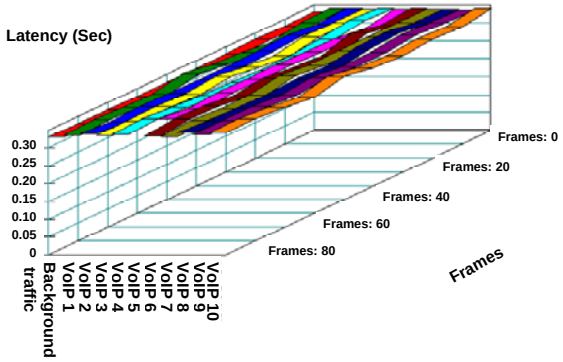


無使用任何機制

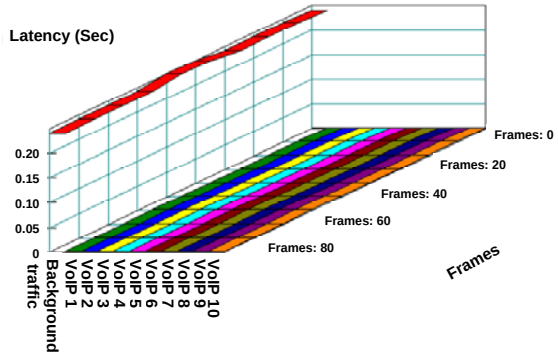


採用本篇之機制

圖17 封包到達時間分佈



無使用任何機制



採用本篇之機制

圖18 封包快照之比較

此機制的構思便是希望透過此方式將即時性的語音資料給予品質保證，使得它不會受到網路壅塞或是頻寬不足導致網路傳輸延遲過大而使得所傳輸的資料無效。我們所要實現的架構是基於 Linux 上現有的應用和方法，整合成我們所需要的功能。因此，對於程式的可信度是相當高的。

因整個實作系統是基於 Linux 的架構來實現，正因如此，此機制可以很平順的移植在任何 Linux 機器上，不是侷限網路處理器平台之上。市面上的產品都只設定已知的埠號，並無針對 VoIP 的連線來做保證，而我們實作出一個可動態分配語音頻寬和對於頻寬資源做允入控制來保證語音品質的一個頻寬資源閘道器，相信對於 VoIP 的發展會更有幫助。

誌謝

本論文由國科會計畫所補助，計畫執行編號為NSC 94-2213-E-194-042。

參考文獻

[1] 3GPP, "3rd Generation Mobile System Release 5 Specifications", Jun. 2002.

[2] Class-Based Queueing. [Online]. Available: http://www.icir.org/floyd/cbq.html

[3] Denial of Service Attacks. [Online]. Available: http://www.cert.org/tech_tips/denial_of_service.html

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

[5] HTB Home. [Online]. Available: http://luxik.cdi.cz/~devik/qos/htb/

[6] Iptables Tutorial. [Online]. Available: http://iptables-tutorial.frozentux.net/iptables-tutorial.html

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

[8] Linux Advanced Routing & Traffic Control. [Online]. Available: http://lartc.org/

[9] M. Handley and V. Jacobson, "SDP: Session Description Protocol", IETF RFC 2327, April 1998.

[10] Netfilter. [Online]. Available: http://www.netfilter.org/documentation/index.html#documentation-howto

[11] Practical QOS. [Online]. Available: http://www.opalsoft.net/qos/DS.htm

[12] R. Braden, D. Clark and S. Shenker, "Integrated Services in the Internet Architecture: an Overview", IETF RFC 1633, June 1994.

[13] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang and W. Weiss, "An Architecture for Differentiated Services", IETF RFC 2475, December 1998.

[14] S. Floyd and V. Jacobson, "Link-Sharing and Resource Management Models for Packet Networks", IEEE/ACM Transactions on Networking, Vol. 3 No. 4, pp. 365-386, August 1995.

[15] Kernel Module Programming. [Online]. Available: http://www.linuxhq.com/lkprogram.html