第五屆電腦與通信技術研討會

論　文　集

Proceedings of the Fifth Symposium on
Computer & Communication Technology, 2000

會議日期：2000 年 10 月 6、7 日

地　　點：大葉大學

主辦單位：大葉大學電機工程學系

通訊與計算機工程學系

協辦單位：教育部、國科會

經濟部工業局、工研院電通所

採透通方式達成具有 RSVP 能力之 UNIX FTP

魏全佑 苗育本 謝錫堃
成功大學電機工程系
wewe@kungsrv.ee.ncku.edu.tw
ybmiau@ybmiau.ee.ncku.edu.tw
shieh@ee.ncku.edu.tw

邱基峰 黃文祥
高雄應用科技大學電機工程系
wshwang@mail.ee.nkit.edu.tw
gary@wshlab.nkit.edu.tw

中文摘要

本論文研究目的是在RSVP-enable網路中提供一種透通性的技術，稱為 QoS 函式庫重新導向(QoS Library Redirection , QLR), 使一般的 Internet 應用程式在執行的過程中不需修改 Internet 應用程式的原始程式碼加入RSVP API 程式碼叫而能夠傳送具有 RSVP 保護的資料。本論文將以 UNIX Ftp 為例，來測試並驗證我們的方法。

1. 簡介

當前存在兩種著名的 QoS 網路架構，分別是整合式服務網路(Integrated Service, IS)和差別式服務網路(Differentiated Service, DS)，以 IS 來說，它提供聲音,影像,以及即時性資料在一致性的網路架構中傳送，它所採用的是 per-flow 分類法則，而 DS 簡化 IS 架構，採用 per-class 分類方式。我們將論文焦點擺在 IS 的資源保留協定(RSVP)[1]，並且提出一種能夠加快發展 RSVP-aware 應用程式的技術。在 RSVP-enable 的網路架構中，所有路由器及使用者端之平台必須有能力處理辨識 RSVP 的訊務信息，藉由此信息來達成資源保留的要求。目前有許多廠商及相關研究單位所發展之產品都有支援 RSVP 訊務協定，愈來愈多的廠商相繼提出支援 RSVP 協定後，整個網路可以說是 RSVP-enable 的情形，在這種環境下之 IS 架構才能實現。但是，目前發展 QoS 存在著支援 RSVP 應用程式欠缺的問題。大部分常用的網路應用程式在早期發展完成，當 QoS 之軟硬體技術發展漸趨成熟時，傳統應用程式並沒有加以跟進，造成上層應用軟體沒有與下層 QoS 技術配合。為了解決傳統 Internet 上應用程式沒有支援 QoS 的問題，有以下的解決方法：首先，最徹底的方法是重新設計並加入 RSVP API 於應用程式中，但此方式不僅大費周章，而且耗時耗力，程式設計者也須要了解 RSVP 的運作方式，第二種方式是修改原始程式碼，修改應用程式之原始程式碼並加入 RSVP API，此種方式雖能避免重新設計之困擾，但是，並非每個應用程式都能拿到原始程式碼，而且須要花費許多時間來偵錯。有鑑於此，我們提供一種透通性的技術，稱為 QoS Library Redirection(QLR)，本方法是採用函式庫重新導向的技術，並且不須修改原始程式碼，利用函式庫重新導向的方式，在應用程式執行中加入 RSVP API 程式碼，使得一般 Internet 應用程式能夠傳送具有 RSVP 訊務協定的資料，本論文採用 UNIX Ftp 應用程式來測試並驗證我們的方法。

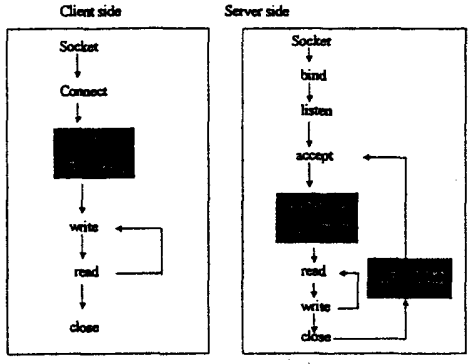


圖 2.1 RSVP-aware 應用程式

2.背景說明

2.1 撰寫 RSVP-aware 應用程式

參考圖 2.1，應用程式透過 RAPI[4]呼叫以建立起 RSVP 之連線，它運作方式是由 RAPI, RSVPD 以及 applications 三者間互相配合來完成，以下說明運作過程及如何設計 RSVP-aware 之應用程式。

1.應用程式呼叫程序介面

應用程式透過 RSVP 協定以傳遞參數並達到頻寬資源保留之目標，RSVP API 是設計給 Internet 應用程式呼叫 RSVP 協定之程式介面，它是由許多呼叫函式所構成並以函式庫的方式呈現出來。

2.溝通 RSVP daemon 之介面

RSVP API 會在應用程式與 RSVP daemon 之間建立 UNIX-domain socket 連線，透過此連線溝通彼此之間的訊息。

2.2 FTP 運作原理

FTP 應用程式在 Internet 中被廣泛地使用，它的功能是透過網路傳送的方式把檔案從一台電腦複製到另一台電腦中，採用兩條 TCP 連線的運作方式，這兩條連線分別為控制連線和資料連線。控制連線負責傳送訊務信息以及傳達命令之用，資料連線負責傳遞 Client 及 Server 間之檔案資料以及由 Server 傳送至 Client 之目錄資料，它的優點是採用 TCP 協

定可以傳送具有可靠性之資料，缺點是當網路發生壅塞時，TCP 流量控制會降低傳送速率。我們論文所提之 QLR 能夠改善網路壅塞情況下的檔案傳送速率，以提高傳送之服務品質。FTP 要能與 QLR 的機制配合時，必須了解 FTP 應用程式的 Client/Server 運作原理，以下詳細描述 FTP 運作之過程。參考圖 2.2 的三個步驟：

1.當 FTP Client 要從 FTP Server 端抓取資料時，Client 對 Server (埠號 21)發出連結的要求(步驟一所示)，如果 Server 接受這項要求，就會建立起控制連線(步驟二所示)。

2.因為要從 Server 端抓取資料，所以 Client 的使用者必須下達抓取命令，並利用控制連線傳送資訊給 Server，此資訊包含 Client 等待 Server 連結之埠號，當 Server 收到 Client 的命令以及埠號之資訊後，會向 Client 發出連結的要求，如果 Client 接受此要求，則建立起資料連線(步驟三所示)。

3.Server 端使用資料連線以傳送檔案給 Client 端。

4.每傳送完一個檔案會再建立一條新的資料連線，並結束舊的資料連線，在 Client 端等待連結之埠號值會隨著資料連線的建立而不斷地增加，每傳送完一個檔案則埠號值加一。

3.QoS 函式庫重新導向方法(QLR)

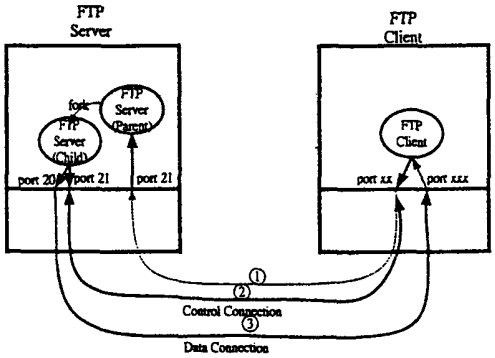


圖 2.2 Ftp 運作原理

QoS 函式庫重新導向(QLR) 是一種軟體技術，目的提供一層介面給 RSVP-unaware 的 Internet 應用程式，在執行的過程中自動加入 RSVP 訊務訊息，以使得應用程式能傳送具有 QoS 的資料，並且不必修改程式碼。我們採用函式庫重新導向方式來完成，當初發展共享函式庫的主要目的是可以節省記憶體空間，把常用的呼叫函式編輯成函式庫的形式，而每個呼叫函式只分配到固定的記憶區塊，當程式執行到呼叫函式時，把指標位址指向呼叫函式所在的記憶區塊，執行完成後再回傳回去即可。程式發展者為了避免系統函式庫功能不齊全，所以提供環境變數的設定，它可以指定使用者的共享函式庫路徑，所以使用者可以採用自己發展的共享函式庫以彌補系統函式庫功能的不足，並執行特殊的要求。函式庫重新導向是很廣泛運用的技術，發展

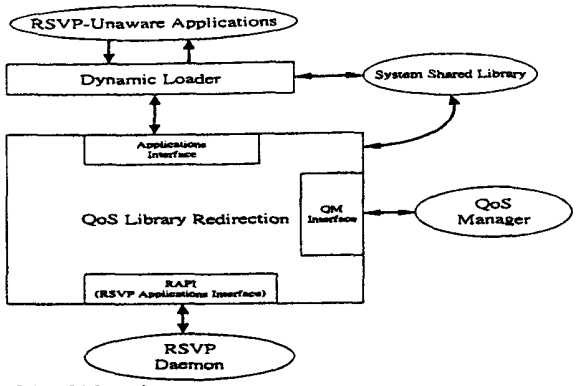


圖 3.1 QLR 模式

RSVP-enable 網路也不能缺少 RSVP API，把 RSVP API 與函式庫重新導向結合在一起，成為新的技術，我們稱之為 QLR (QoS 函式庫重新導向)。參考圖 3.1 所示。

1.應用層介面

QLR 必須要知道是何種應用程式在運作，所以這個介面要提供辨識應用程式的能力，並代替應用程式呼叫系統共享函式庫。

2.RSVP 層介面

當 QLR 辨識出何種應用程式之後，必須收集應用程式相關呼叫資訊，並把這些資訊傳給 RSVP 層介面，以作為 RAPI 呼叫的參數設定。

3.QoS 管理介面

這個介面可以設定 RSVP 頻寬保留大小，就是 IS 中所定義的 token-bucket 參數：

r token bucket rate (bytes/sec)
b bucket depth (bytes)
p peak rate (bytes/sec)
m minimum policed unit (bytes)
M maximum packet size (bytes)
R Required rate in (bytes/sec)
S Slack term (microseconds)

透過這個介面可以設定參數以得到最佳的頻寬。

4.實作與完成

4.1 QLR 架構

在上節中有描述 QLR 的模式架構，它主要目的是不修改應用程式碼的情況下，使得應用程式能傳送具有 RSVP 能力的資料，這是我們發展 QLR 的主要精神，為了實現這項觀念，我們結合作業系統的特性以及 ISI 所提供的 RSVP API 的函式庫，來達成我們的目標，並且以 UNIX Ftp 作為我們測試及驗證的應用程式。QLR 的技術是由七個動作所構成，以下將要描述這些步驟。

在描述之前，先說明我們的環境架構，基本上 QLR 的技術是運作在使用者端，所以在路由器端

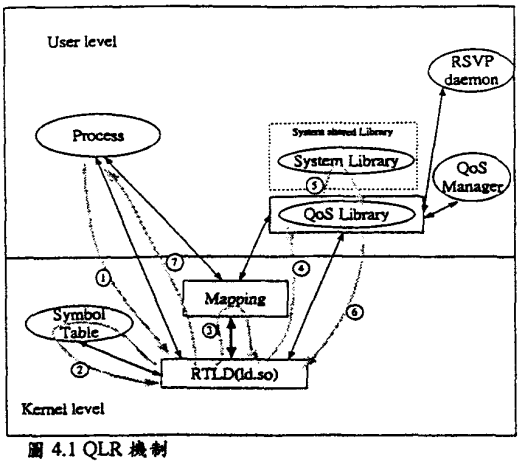


圖 4.1 QLR 機制

並不須要作任何改變，我們可以看到圖 4.1 中所示。QLR 在使用者端的運作模式，圖 4.1 顯示 user level 目前存在一個執行程序(Process)，另外在 Host 端也啟動的 RSVP daemon 以及 QoS Manager 的處理程序，負責處理 RSVP API 函式呼叫和參數設定，目前系統存在著兩種共享函式庫，一種是系統函式庫，另一種是我們自己設計的 QoS 函式庫，這個函式庫是 QLR 技術的核心部位，在上一章節中我們有提到 QLR 需要三個介面來處理各種動作，而 QoS 函式庫本身就包含這三個介面，另外，在 QoS 函式庫編譯的同時，也必須把 RSVP API 加入這個共享函式庫中，使得 RSVP API 能在 QoS 函式庫中被呼叫。以下分七個步驟來說明：

・步驟一

當使用者程序呼叫到動態共享函式庫的時候，它會向作業系統提出要求，此時作業系統會把這項要求的控制權交由(RTLD)來處理。

・步驟二

這部分有關於函式庫重新導向的運作機制，當 RTLD 接受作業系統的命令之後，RTLD 會到 Symbol Table 中查詢這個呼叫函式是否已載入記憶體，如果沒有載入，RTLD 必須到指定的檔案目錄中尋找是否有相同名稱的呼叫函式，如果發現到則將呼叫函式載入記憶體中，如果沒有找到則系統發出錯誤訊息讓使用者知道。被指定的檔案目錄中，有許多系統的共享函式庫，如果使用者想發展自己的函式庫，並且想取代掉原來系統所提供的呼叫函式，則使用者函式庫的呼叫函式名稱與系統函式庫呼叫函式名稱要相同，然後經由 LD_PRELOAD 的設定，指定使用者函式庫的檔案路徑，設定後 RTLD 會先尋找使用者函式庫，如果沒有發現再去尋找系統函式庫中是否有相同名稱的呼叫函式，然後將之載入記憶體中執行，我們利用這個機制發展自己的共享函式庫，並結合 RAPI 來作為 QLR 運作核心，這個運作核心在下一節中有詳細的說明。(LD_PRELOAD 是一般 UNIX 或 Linux 的作業系統所提供的環境變數，目的是指定使用者函式庫的檔案路徑)

・步驟三

RTLD 必須將使用者函式庫的呼叫函式記憶體空間映射到執行程序記憶體空間中，並要解決外部參考的問題。

・步驟四

RTLD 把控制權交由被呼叫的函式來執行，如果呼叫函式是屬於我們所發展的共享函式庫，則執行程序會進入 QLR 的運作機制中，如果不是，則進入一般的 Dynamic
linking 的運作機制。

・步驟五

如果進入 QLR 機制中，它必須替應用程式呼叫原來的函式，使得應用程式的運作特性不會因為進入 QLR 機制而改變。這是發展 QLR 的基本精神，並在處理過程中加入 RAPI 呼叫函式，以達成具有 RSVP 能力的應用程式。

・步驟六

當共享函式執行完畢，就把控制權交還給 RTLD 的處理程序。

・步驟七

RTLD 把控制權交還給使用者處理程序，如此完成 QLR 的整個運作程序。

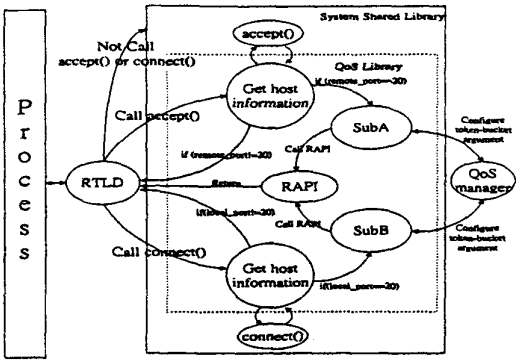
4.2 設計支援 UNIX FTP 之 QLR

為了要驗證這種方法的可行性，我們採用 UNIX FTP 應用程式來作為我們的測試。FTP 應用程式在 Internet 上被廣泛的使用，而且它具有長時間傳送資料的特性，但是今天我們所面臨的是頻寬不足的問題，雖然 FTP 的應用程式是採用 TCP 的協定，可以確保資料傳送的正確性，但有可能因為網路傳輸的瓶頸所造成的壅塞情形，而產生封包遺失的現象，當封包遺失時，因為採用的是 TCP 的協定，所以在發送端會不斷地重送封包，因此又加重了網路傳輸的負荷，封包不斷地重送的過程，降低整體網路的傳輸效能[5]，這個問題在早期的 Internet 環境中，除了增加網路驅動元件傳輸的緩衝區以外，根本無法解決，後來這幾年有許多人提出具有 QoS 架構的網路環境，並提供了許多可以應用的技術，選擇 FTP 應用程式是因為它有下列的特性：

・FTP 特性

1.Dynamic port binding, 每傳送完一個檔案，Data connection 會再重新建立，並把 port number 加一。

2.Widespread, 這種應用程式遍及全世界。因為 FTP 應用程式有上列特性，所以更能夠突顯出 QLR 的能力。為了把 FTP Client/Server 加入 QLR，設計時必



Get host information----Get local address/port and remote address/port
SubA----A subroutine which set up RSVP arguments for ftp client
SubB----A subroutine which set up RSVP arguments for ftp Server

圖 4.2 QLR 狀態圖

須要了解應用程式的特性，並且不能影響應用程式的運作方式，使應用程式仍保有它原來的特性。在這一節中，我們將說明如何辨識 FTP 應用程式以及如何加入 RSVP API。

圖 4.2 為系統核心圖，核心圖由四大要件組合而成，包含 Process, System Shared Library, QoS Library, 以及 QoS manager 所組成。

當 FTP 加入 QLR 的運作方式，將會使得 FTP 能夠建立具有 QoS 的連線，並能在傳送資料的過程中，保留適當的頻寬資源，加快傳送的速度。我們從呼叫函式的角度來說，我們設計的目標是替 FTP 資料連線作頻寬資源的保留，所以忽略建立控制連線之過程，在 Server 端透過埠號以辨別連線之種類，以資料連線來說，它的埠號為 20 (稱為 ftp-data)，因此不管在 Client 或 Server 端只須判斷埠號為 20 的連線，可得知此連線為 FTP 的資料連線，經由 FTP 程式之運作分析了解到 QLR 機制只須針對 accept()與 connect()兩個呼叫函式重新導向即可達成我們的目標，以下分別描述動作原理：

· FTP Client 端

要建立資料連線時，Client 必須等待 Server 的 connect 動作，因此 Client 會呼叫 accept()函式，在呼叫的過程中因為共享函式庫被導向到 QoS Shared Library 中，先執行 Get host information 去取得 local 位址/埠號和 remote 位址/埠號的資訊，並判斷 remote 埠號是否為 20，如果為真，表示應用程式為 FTP，接下來執行 SubA 的動作，它的工作是接收外來的資訊並設定 RAPI 所需要的參數，當參數設定完畢，接著呼叫 RAPI 函式，以傳送 PATH 或 RESV 的 message。

· FTP Server 端

必須等到控制連線建立完畢，然後 Server 等待 Client 的命令以決定資料傳送的方向，一旦接收到 Client 所下命令安令，Server 會使用 connect()函式以建立與 Client 端的資料連線，因此在 Server 端只需要針對 connect()做函式導向，並利用 Get host information 去取得 local 位址/埠號 及 remote 位址/埠號的資訊，然後判斷 local 埠號是否為 20，如果是則能夠確定應用程式為 FTP，接下來呼叫 SubB 的動作，以接收外來的資訊並設定 RAPI 所需要的參數，等到參數設定完畢，開始呼叫 RAPI，以傳送 PATH 或 RESV 的 message。

5. 實驗與結果

5.1 實驗環境架構

為了要驗證 QLR 的可行性，參考圖 5.1 所示，我們使用五台電腦架構出簡單的 IS 實驗環境，其中兩台當作 Router 來使用，Host A 與 Host B 分別為 Ftp Server 和 Ftp Client，把 QLR-1.0 的函式庫模組裝置在 Host A 與 Host B 的兩台電腦上。以我們的環境而言，所建構出來的是 RSVP-Aware 的環境，從 Host 端到 Router 端都必須執行 RSVP daemon (RSVPD)，以負責傳送 PATH 及 RESV 訊息，並做資源保留的動作及設定，如果要讓頻寬真的保留起來，還必須要在系統中加入 Traffic control 的機制，我們採用的是 SONY 研究單位所提供的 ALTQ[7]，它裡面提供了一種能夠與 RSVP 配合的 CBQ 機制，這是我們採用它的主要目的。當 RSVP 加上 CBQ 的機制時稱為 RSVP/CBQ，這樣的 RSVP 環境架構才能發揮保留頻寬的作用。我們設定 Best effort 及 RSVP 的兩個服務等級，每個等級分配到如圖 5.2 所示的頻寬，我們使用 Mgen 的應用程式以模擬網路負載情形，並採用 ttt 的應用程式監測即時流量圖形，以觀察資料傳送的行為。在 5.2 節中可以清楚的看出實驗的數據以及 QLR 的可行性，足以證明 QLR 是可行的機制，以下將會分析這些數據所代表的意義。

<實驗一>

採用 Case 1 的 CBQ 設定，並不斷地增加 Background

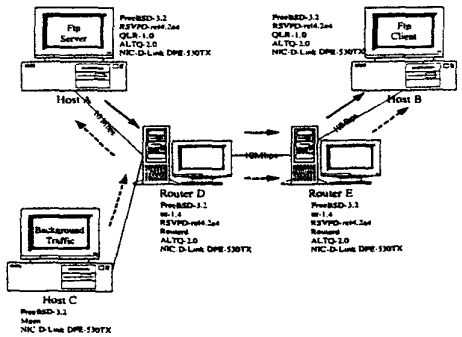


圖 5.1 實驗架構圖

traffic 流量，以模擬網路負載，並使用 UNIX FTP 應用程式，比較加入 QLR 之前的情況(RSVP-unaware UNIX FTP)與加入 QLR 之後的情況 (RSVP-aware UNIX FTP)對 20M Bytes 檔案傳送速率之影響。

<實驗二>

採用 Case 2 的 CBQ 設定，同實驗一。

<實驗三>

採用 Case 3 的 CBQ 設定，同實驗一。

5.2 實驗數據

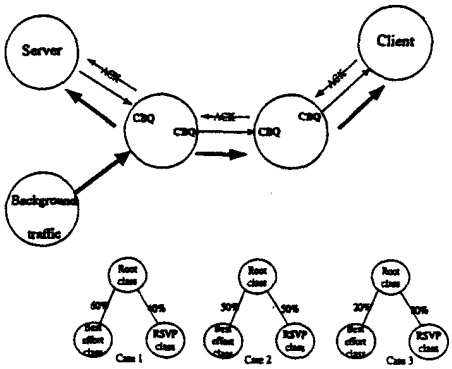


圖 5.2 實驗模式圖

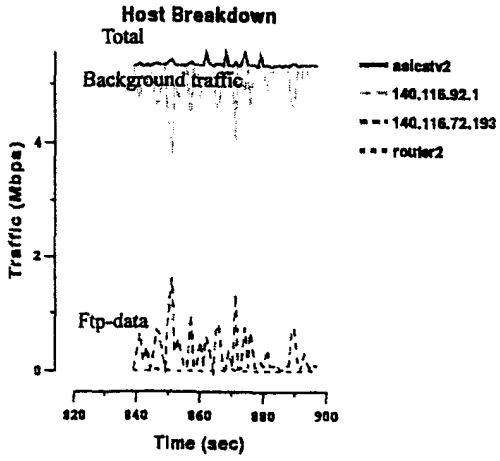


圖 5.3 加入 QLR 前的 FTP 即時流量圖 (case1)

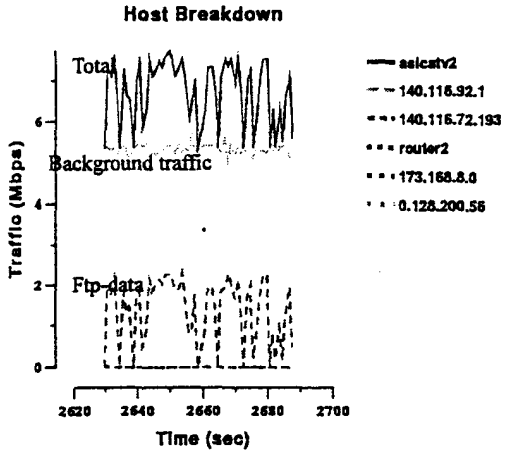


圖 5.4 加入 QLR 後的 FTP 即時流量圖 (case1)

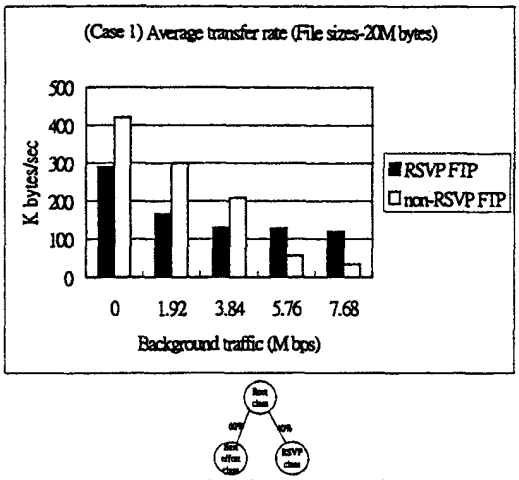


圖 5.5 網路負載與檔案平均傳送速率之關係圖(case1)

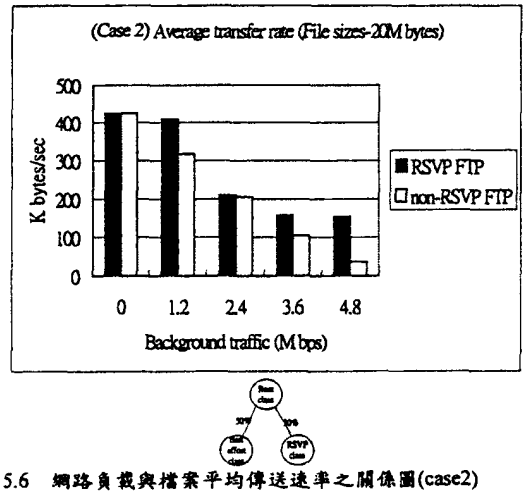


圖 5.6 網路負載與檔案平均傳送速率之關係圖(case2)

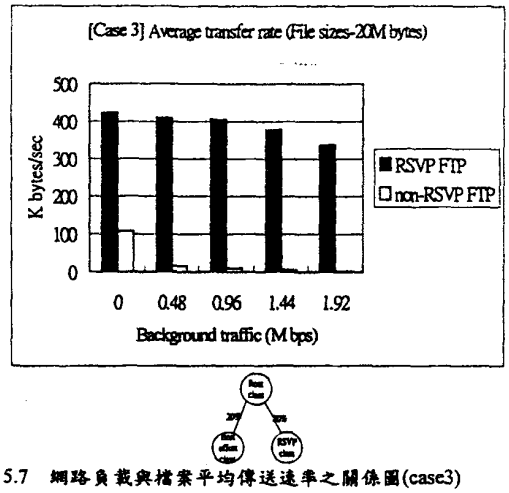


圖 5.7 網路負載與檔案平均傳送速率之關係圖(case3)

5.3 分析與驗證

我們的目的是調整 Background traffic 的參數，以觀察 RSVP-aware 的 Ftp 與 RSVP-unaware 的 Ftp 傳送速率的不同，以證明 QLR 機制的功用，從圖 5.5 數據中顯示，在網路輕載的情形下，Best effort 的傳送速率比 RSVP 等級的傳送速率還好，這是必然的情況，但是，在網路重載的情形下，明顯地看出來 RSVP 等級比 Best effort 的傳送速率好很多，所以採用 QLR 的機制，可以不經由修改程式碼的方式，使應用程式具有 RSVP 的能力，而能夠傳送 RSVP 等級的資料，並能達到預期傳送的速率。

分析與討論：

1.RSVP 設定時間(setup time)之影響

從(圖 5.5，圖 5.6，圖 5.7)觀察到網路負載大小會對 RSVP class 的檔案傳送速率有所影響，我們解釋此一現象與 RSVP 設定時間(指發送端送 PATH 訊息到接收端，接收端送 RESV 訊息到發送端所需的時間)有所關係，而檔案傳送速率的公式為

r = (File sizes) / (RSVP setup time + File transfer time)

r — 檔案傳送速率

單位 (k bytes / sec)

因此得知，檔案傳送速率會受到 RSVP 設定時間的影響，當網路負載越重，RSVP 設定時間也越長，因為 PATH 訊息與 RESV 訊息是採用 Best effort class 來傳送資料，當網路負載越重的情況下，這些訊息越不容易送到目的端，這也是採用 RSVP 不可避免的現象。

2.CBQ 所分配的頻寬

雖然 RSVP 有提供 token bucket 的參數讓我們設定傳送的速率，但是我們發現，架構在 CBQ 模式下的 RSVP 應用程式，不能夠經由調整 token bucket 的參數來決定傳送的速率，所以我們試著調整 CBQ 分配的頻寬，如同 Case 2 或 Case 3 所設定的頻寬分配，並觀察 20M bytes 的檔案傳送速率是否有所改變。我們可以觀察到(圖 5.6 與圖 5.7)所顯示的圖中，相同網路負載的 RSVP class 傳送速率有明顯地增加。

6.結論與未來工作

這篇論文提供一種透通性技術，不經由修改程式碼的方式，使傳統網路應用程式加入 RSVP 的能力，這種技術稱為 QLR。它包含應用層，QoS 管理層及 RSVP 層介面，負責代替應用程式呼叫 RSVP API 以建立具有 RSVP 能力之連線，它主要的貢獻有：

(1) 不修改應用程式的情況下，加入 RSVP API 呼叫函式。
(2) 可以透過 QoS 管理介面即時設定 token bucket 等參數。
(3) 容易擴充支援各種 QoS API。
(4) 加速發展具有 QoS 的應用程式。
(5) 以共享函式庫的形式存在，不需修改作業系統的核心部位。

設計 QLR 必須配合應用程式運作特性，FTP 是採用 TCP 協定所發展出來的應用程式，本身有流量控制與避免壅塞的處理機制，因此它具有的特性是：1.連線具有雙向性，2.以兩條連線維持整個運作機制，3.動態建立資料連線，我們把 QLR 設計成雙向保留頻寬資源架構，並配合 RSVP 動態頻寬保留特性。實驗過程採用 UNIX Ftp 應用程式來驗證 QLR 的可行性，並且經由實驗所得之數據，證明 QLR 是值得採用的技術。

未來的工作會專注於 multicast 應用程式，並朝向 Windows 作業系統發展，使得 QLR 機制能夠被廣泛地應用在網際網路的世界裡。

參考文獻：

[1] "Inside the Internet's Resource reSerVation Protocol" David Durham and Raj Yavatkar, Wiley Computer Publishing., 1999.
[2] "Quality of Service: Delivering QoS on the Internet and in Corporate Networks", Paul Ferguson Geoff Huston, Wiley Computer Publishing, 1998.
[3] Leland L. Beck, "SYSTEM SOFTWARE: an introduction to systems programming, second edition", 1990.
[4] R. Braden and D. Hoffman, "RAPI-An RSVP Application Programming Interface version 5", Internet Draft, August 11, 1998.
[5] Stevens, W., "TCP slow start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms", Internet RFC 2001, January 1997.
[6] Florissi, P.G.S.; Yemini, Y.; Florissi, D., "QoSockets: a new extension to the sockets API for end-to-end application QoS management", Integrated Network Management, 1999. Distributed Management for the Networked Millennium. Proceedings of the Sixth IFIP/IEEE International Symposium on, Page(s): 655-668, 1999.
[7] Steven Berson, "RSVP Software", http://www.isi.edu/rsvp/release.html, Last modified: 5 March 1999