即時性子網路頻寬管理系統在 Linux 核心之實作[1]
The Implementation of Rea -Time Subnet Bandwidth Management in the Linux Kerne

陳郁堂
台灣科技大學電子系
TEL:(02)2737-6420
Email:ytchen@et.ntust.edu.tw

樊正平
台灣科技大學電子系
TEL:(02)2737-6420
Email:redhap@young.url.com.tw

摘要

傳統乙太網路架構，無法對即時性資料提供傳送時間保證，以往研究利用新的硬體架構或修改網路卡驅動方式來解決這個問題，由於工程耗大，所費不貲，無法獲得市場消費者的認同。本文利用在網路裝置介面層(Device Interface)中，建立 Token Passing 機制來解決這個問題。透過 token 控制器，來管制網路上的節點，須已收到 token 才能傳送資料到網路上，可解決在乙太網路下由於封包碰撞造成傳送時間無法預測的問題。整個系統建構在 Linux 作業系統上，以軟體方式製作，因此不須更動現有硬體與驅動程式。在實際測試顯示，本系統在容錯性、穩定性，性能優越。

Abstract

The objective of this paper is to design and implement a system that can support the real-time transmission via Ethernet without changing hardwares and device drivesr. Weimplements a token -passing protocol at device interface layer to regulate the network traffic. Only after receiving a token from a subnet manger, a client can begin to transmit packets. Therefore, the packet collision can be avoided. The real-time scheduling scheme, such as Deficit Round Robin is used to coordinate the Token-Passing among clients. The whole system is implemented in Linux operating system. A performance measurement from experiments reveals our proposed system show superiority in terms of reliability, fault tolerance, and overhead.

1、緒論

由於網路的日漸發達，應用的範圍也愈來愈廣，因此有許多新的網路的技術與應用出現。多媒體網路應用是相當被看好的一種運用，例如遠端教學(distant learning)，視訊會議(Video Conference)，隨選視訊(Video On Demand, VOD)，網路影音播放器(如 Real Player)等。這些多媒體網路應用通常需要較大的頻寬，與較嚴格的時間限制，以滿足使用者對播放品質的要求。

目前乙太網路(Ethernet)為使用最普遍的區域網路架構，透過 CSMA/CD 協定來控制網路使用權，電腦偵測網路上沒有資料傳送，即可傳送封包，因此可能發生兩部以上的電腦同時傳送封包而產生碰撞。碰撞後電腦採隨機的方式決定資料重送時間，更增加了傳送時間的不可確定性，無法滿足多媒體資料的即時性需求。為解決在乙太網路上無法提供資料即時傳送的保證，許多發展網路設備的廠商以及學術團體目前正致力於即時性網路的研究，目的在提供多媒體資料傳送良好的服務品質(Quality of Service)。

最早的 Token-Bus[1]與 Token-Ring[2]架構，也可以達到即時性網路的需求。Token-Bus 協定可以應用在許多不同的硬體架構上，相對的 token 的處理會增加許多軟體花費(software overhead)。Token-Ring 是真正由硬體構成的環， 不 用 token 上記錄節點的詳細資訊，在節點多時可以比 Token-Bus 節省更多的 token 傳送與處理時間，但是硬體並不普遍。PACE[3](Priority Access Control Enabled)是 3Com 公司的一套網路資源存取協定，必須有此公司開發的 Hub，配合軟體來使用。100VG-AnyLAN[4](HP, IEEE 802 .12) 是 利 用 Round-Robin 的方式，使每一個節點在固定時間內必定能傳送出一定量的資料，不過由於需要專用的硬體，以及一個平均頻寬的協定，因此並不被普遍使用。由以上可知，若要有效的達成即時服務的需求，一般必須以硬體來配合才行。，所以如何克服硬體先天的不便，以純軟體的方式來達成即時服務的需求，就是本文所要探討的主題。

RETHER[5]為第一個提出的純軟體解決方案，以網路驅動程式來建立 Token Passin 機制，以解決封包碰撞問題。市面上有眾多網路卡，各有不同的網路驅動程式，因此無法廣為一般網路卡製造廠所接受。

本文利用架構一個類似 Token-Bus 的環境，運用 token 的機制，來確保子網路內資料傳送的服務品質(Quality of Service[6])。這個軟體模組我們稱之為 RT-SBM(Real-Time Subnet Bandwidth Manager)，目的在於保障多媒體資料傳送的即時性。無須更動網路驅動程式，僅在網路介面層建構，簡化了更換的程序，使得一般的 Linux 網路系統變成具有 Token-Passing 的能力。我們還提供了應用層的介面(API)，使用者可藉 socket 的程式來進行資源預留。為了使即時的資料可以在固定的時間經由乙太網路到達目的地，我們採用 Token-Passing 的協定架構，來避免封包的碰撞。以一 token 控制器來管理網路資源以及 token 在各節點的排程與停留時間，可透過 Round Robin 排程方式進行。

[1] 本研究由國科會 NSC 88-2213-E-011-047 補助支持

我們也預留了資源預留協定所須支援的特徵，例如服務類別(Service Type)。服務類別有 Guaranteed (GT)， Controlled Load(CL 和 Best effort(BE)三種 service。由於上述是分類與排程的問題，在本篇中不多加討論，只是說明本篇對於上層的分類及排程可以有效支援。

本文總共分為 5 節，第 2 節針對 RT-SBM 的設計限制，系統協定及排程方式做說明。第 3 節說明 RT-SBM 之建構位置、系統模組、系統運作流程、資料結構之設計、程式之流程與系統工作細節。第 4 節測量 RT-SBM 系統的軟體花費(software overhead)、穩定性、容錯性及即時性保證等性能。第 5 節對本文做個總結。

2、RT-SBM 的設計

在這個章節中，我們將敘述 RT-SBM 系統在設計上的限制，系統所使用的協定以及系統所支援的排程方式。

2.1、設計限制

RT-SBM 主要的目的是提供一個週期性的、可預期的、可預留頻寬的網路，給多媒體應用程式使用(例如影音播放器可透過網路播放)，但是不更動目前所用的乙太網路硬體。

為了達到使用網路的可預期性，我們採用 Token-Passing 的方式來控制節點的封包傳送。雖然 Token-Passing 的方法會造成一些處理上額外的花費(overhead)，但是也減少了封包的碰撞。RT-SBM 在設計上，可以由使用者來決定要不要進入即時模式，在未進入即時模式前，網路操作在 CSMA/CD 模式下。所以當有節點要透過網路播放的媒體資料時，可以要求進入即時模式，來保障播放的品質。播放結束時，可以要求切換回乙太模式。

為了避免非即時的服務在即時模式下無盡的擱置，我們預留了一些頻寬給非即時的服務。當所有非即時的服務達到最低保障頻寬時，RT-SBM Server 將不再允入節點及增加頻寬需求，直到有節點歸還頻寬為止。

TRT 是可以調整的，一般為了配合影片的播放，可以定為 1/30 秒，因為通常影片的傳送需求是每秒 30 個 frame。由於 RT-SBM Server 使用 timer 來維護 TRT，Linux 的 timer 時間單位是 10ms，因此設計上，允入的節點數會受 TRT 以及最低保障 BE 值的影響。

在 RT-SBM 中將資源預留的服務類別分為三種，分別是 Guaranteed service， Controlled Load service，Best Effort service。每一種服務類別，我們都有對應的變數來記錄使用狀況。雖然在設計上區分為三種類別，但是在將頻寬分配到節點時，所給予的是三種類別的加總頻寬（時間），再由各個節點去分配運用。這樣做的理由是因為在設計中 Best Effort 是資源可以被搶奪的服務類別，而知道頻寬分配狀況的只有 RT-SBM server。因此 RT-SBM client 必須將自己所要求的 Guaranteed service 值及 Controlled Load service 值交由 RT-SBM SERVER 做允入控制。允入成功後，Best Effort 會在不小於最小保障頻寬之下做調整。

2.2、系統協定

RT-SBM 協定分為「即時」與「乙太」兩種模式(如圖 1 所示)，每一個節點我們以它的乙太網路位址來區別。以下我們從網路未進入即時模式到進入即時模式，最後再由即時模式回到乙太模式的情況，據以說明工作的流程。為了方便下面的說明，網路假設有最大 BW_{TOTAL} (bits/sec)的傳遞能力；下面我們就開始細部的說明：

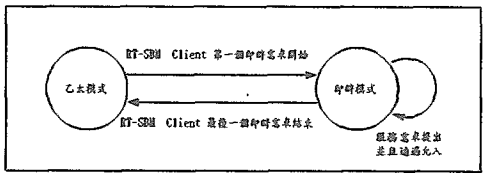

圖1. 乙太/即時模式切換圖

- **系統起動**

一開始網路是處於「乙太模式」，每個節點在開機或重新啟動後都要攔截封包輸出，並偵測 RT-SBM Server 是否存在(亦即聆聽有無「節點加入群組」廣播訊息在網路出現)，若一段時間後仍未能聆聽到此廣播訊息，解除封包輸出限制使用 CSMA/CD 協定傳送封包，並繼續聆聽是否有 RT-SBM Server 出現。

- **進入即時模式**

若有 RT-SBM Client 向 RT-SBM Server 提出「進入即時模式」需求，RT-SBM Server 接到此需求，發出「進入即時模式」廣播，所有 RT-SBM Client 收到此一廣播，執行節點加入群組動作。

- **加入節點群組**

如果 RT-SBM Server 存在，在每個 token 循環時間結束時，會廣播「節點加入群組」訊息，表示正處於「即時模式」。新節點傳送自己的乙太網路位址給 RT-SBM Server，以便加入「節點群組」。如果有兩個以上的節點同時加入，將會產生碰撞，必須等待下一次廣播訊息的來臨，以避免因訊息重送而影響其他正在進行的服務。

RT-SBM Server 在一段時間內，應該收到大部份存在此段子網路上節點的回應訊息，為了防止此時還有新節點欲加入群組，RT-SBM Server 再發送一次「節點加入群組」的廣播，若一段時間內再沒回應，則假設沒有節點欲在此時加入，RT-SBM Server 隨後產生 token，開始服務各個節點。

- **離開節點群組**

節點關機時可以向 RT-SBM Server 提出離開群組的需求，RT-SBM Server 收到後，將此節點由群組中移去。節點也可以直接關機，因為在一個 THT 後 RT-SBM Server 即將此節點刪除。

- **允入的計算方法**

RT-SBM Server 在將節點加入群組或修改頻寬前，要對節點提出的需求頻寬，利用下面式子做允入控制：

$$BW_{TEST} = BW_{BE} - BW_{REQUEST} - (Node\_Number+1) * W_{BEm}$$

上式中 BW_{BE} 指目前最大可用 Best Effort 頻寬；$BW_{REQUEST}$指節點要求的頻寬；Node_Number 是目前節點名單中的節點個數；BW_{BEm}是指 Best Effort 的最小保障頻寬。若計算後 BW_{TEST}的值小於零，則回應允入控制失敗，節點可在下一循環再次提出需求。若計算後 BW_{TEST}的值大於零，則 RT-SBM Server 回應允入控制通過，RT-SBM Server 隨後將 BW_{BE}值更新，並將此節點資訊加入節點群組，以便下一循環開始服務。

- **token 的傳遞**

RT-SBM Server 將所有的節點加入群組後，開始傳送 token。依照節點在名單中的順序，以 Round-Robin

的方式分派 token。節點收到 token 時，依照所允入的頻寬及 token 循環時間(TRT)，設定好 token 掌握時間(THT)後，開始傳送資料，THT 可由下列式子得出，其中 BW_{TOTAL} 為目前乙太網路的總頻寬。

$THT = TRT * [BW_{REQUEST} / BW_{TOTAL}]$

節點在封包傳送完或 THT 計時器 time out 時，將 token 歸還 RT-SBM Server。Server 依據節點名單，將 token 送到下個節點。下個節點收 token 時，一樣設定好 THT 後，開始傳送資料。RT-SBM Server 將 token 依序送到每個節點。節點若不傳送封包，就直接將 token 歸還 RT-SBM Server。當所有的節點都已經服務過一次後，RT-SBM Server 送出「節點加入群組」廣播訊息。若有則加入節點名單中。

- **token 遺失**

RT-SBM Server 將 token 傳送給某節點後，在經過一個 THT 後發現某節點尚未歸還 token，於是假設此節點遺失 token，由於此情形不是經常發生，所以直接從節點名單中移除。此節點會再收到 RT-SBM Server 刪除節點的通知，不論此節點是否真的當機。倘若不是當機，節點將在收到此通知後丟掉 token，等候 Server 發出「節點加入群組」的訊息時，再度加入群組。

- **token 重複**

若發生 RT-SBM Server 尚未發出 token，某節點又送回另一 token 的情況；則 Server 直接丟掉後來收到的 token 即可。在節點方面，若發生 token 重複的情形，也是直接丟掉後來收到的 token，其餘的交給 RT-SBM Server 去處理。至於可能發生的碰撞，由於發生的機率不大，而且封包會再度重送，所以我們不考慮為此一情況做特別的處理。

- **HOS 當機**

若節點未在時間內歸還 token，表示此節點當機。節點當機時，處理方式同 token 遺失。

- **RT-SBM Server 不正常關機或當機**

由於每個節點設有一計時器(Timer)，若是 token 在一個 TRT_{MAX}(此值必然大於等於目前運作的 TRT)之後沒有到達，節點假設 RT-SBM Server 已經失效(不正常關機或當機)，為了繼續使用網路，節點自行切回乙太模式。

- **回到乙太模式**

RT-SBM Server 失效以及 RT-SBM Server 正常關機時，節點都會回到乙太模式。前者是由各節點的 timer 自行偵測，後者是在關機前，會發出「回乙太模式」的廣播，來使各節點回到乙太模式。設計上即使省略廣播的情況，也不會有太大的影響，最多在一個 TRT_{MAX} 之後，各節點都會回到乙太模式。

- **頻寬需求的修改**

一個節點可能會有一個以上的需求個體產生，因此如何讓 RT-SBM Server 在 token 循環進行中可以允許節點來修改頻寬，而不是透過結束舊有的服務，再重新向 RT-SBM Server 提出需求。因此，我們必須加入修改頻寬的方法，來保障較早發生的服務不會因此停擺。

節點向 RT-SBM Server 提出的頻寬修改，在用法上我們使用絕對值而非相對值來表示增加或減少。當我們將頻寬需求送到 RT-SBM Server 的時候，在 RT-SBM Server 端必須將此值與它先前的預留相減，再將所得出的結果送到允入控制去處理，如果允入成功，則需求將更改，若不成功，則保持原先的需求。

2.3、系統排程方式

RT-SBM Server 採用 Deficit Round-Robin[7]的方式發送 token，雖然 token 中包含有 CL 及 GT 的需求值，但是一般還是使用已加總的 THT 來控制頻寬。RT-SBM Clien 在收到 token 時，可依據 token 中所包含的 CL 及 GT 需求值來輸出封包。在設計 Client 端排程時 Guaranteed Service 必須在每個回合(Round)中傳送一定數量的封包，因此必須優先使用頻寬。Controlled Load Service 可以使用 stride scheduling[8] 或 deficit round-robin 的方式來做此群組的排程。Best Effort Service 也同樣採用 round-robin 的方式來服務此群組。不同於 Guaranteed servic 的是，Best effort service 不一定在每一個循環中被服務到。

3、RT-SBM 實作

本文建構 RT-SBM 採用主從式架構。RT-SBM Server 主要負責 token 的分配與管理 segment 上所有的節點，而接受 Server 管理的所有的節點皆為 RT-SBM Client。由於採用 Client-Server 架構，所以 token 不需要儲存所有節點的資訊。所有節點的資訊都由 RT-SBM Server 以 single link list 的方式儲存，可以節省許多 token 的長度。

3.1、RT-SBM 與網路架構

首先我們必須決定在網路通信協定那一層來建構 Token-Passing 機制。由於 Linux 作業系統中可用的網路協定不只 TCP/IP，我們希望 RT-SBM 是一個可攜帶模組(Portable Module)，可以輕易在各機器間安裝，因此必須減少對核心程式(kernel)的修改。網路 TCP 與 IP 協定層因為有太多的函式相互關連，所以並不適合修改。

裝置介面(Device Interface，參見圖 2，P.8)層在 Linux 網路架構中，主要扮演著溝通各協定層及驅動程式的角色。所有上層的協定，都要透過這一層來與驅動程式溝通，因此驅動程式中的介面開啟、初始化、關閉以及封包輸入與輸出的函式，都有特定的裝置程序(Device Methods)要去對應(mapping)。所有網路驅動程式在使用前將本身的外部函式繫結(binding)到裝置程序(Device Methods)去，所以在裝置介面層以上可以不用考慮所使用的網路卡驅動程式，因為裝置介面層已經完成了繫結的動作。

由於網路卡種類太多，修改驅動程式並不是十分有利的行為。例如同樣是 D-Link 公司的產品，但是 D-Link 220ECT(ISA)的網路卡與 D-Link 500TX(PCI) 所使用的網路卡驅動程式卻不同。我們考慮在裝置介面層加入 RT-SBM 機制的原因，主要是可攜性較高，只要可以在 Linux 下正常運作的網路卡，都能直接使用。

3.2、RT-SBM 系統模組

RT-SBM 建構的主要工作在修改封包傳送協定。原有乙太網路驅動程式依 CSMA/CD 方式運作，當偵測到網路沒有信號就執行傳送。但是在 Token-Passing 的通信協定，沒有收到 token 前，不能傳送封包，這是 RT-SBM 的設計中第一個要解決的問題。

RT-SBM 系統分為 Clien 與 Server 兩部份，系統模組如圖 3(P.8)與圖 4(P.9)。RT-SBM Server 分別由 Node Manager、Token Dispatcher、Mode Changer、Message Processo r與 Ethernet Packet Sender 模組組成。

主要功能為節點頻寬管理、token 分配等。Node

Manager 的功能有加入節點、刪除節點以及修改節點的頻寬需求及允入控制。Token Dispatcher 主要功能在管理及發放 token。Mode Changer 在處理模式的切換，例如由乙太模式進入即時模式，或由即時進入乙太模式。Message Processo 與 Ethernet Packet Sender 在 Server 及 Client 都是必須的模組。Message Process o 處理進來的 RT-SBM 訊息，Ethernet Packet Sende 負責把 RT-SBM 訊息傳送出去。RT-SBM Client 由 Socket Buffer Queue Manager，Socket Buffer Throttler，Token Acker，Message Processo r與 Ethernet Packet Sender 等模組構成。主要功能為控制封包輸出以及 Socket Buffer Queue 的管理。

Socket Buffer Queue 主要功能在保存 token 未到之前在 Socket Buffer 中欲傳送的封包。Socket Buffer Throttler 負責使用 token。Token Acker 訊息的回應。Message Processo 主要在處理來自 Server 的訊息。Ethernet Packet Sender 負責傳送乙太訊息封包。在下節中我們將敘述 RT-SBM 系統實作的概要。

3.3、RT-SBM 實作

在這節中，我們將說明 RT-SBM 的功能如何實現，以及提供給使用者使用的 RT-SBM 控制程式。在第一個部份中，說明使用者如何與 Server 連線，如何做資源預留；在第二個部份中，說明在裝置介面層實作的情形。

3.3.1、使用者層

在 RT-SBM 系統下，我們提供了 Socket 介面給應用程式使用，藉此向 RT-SBM Server 提出模式切換及頻寬需求。在 Server 端的 Socket 介面負責回應 Server 的處理狀況訊息，Client 端 Socket 介面可以傳送三種命令訊息，執行我們利用 Socket API 製作的「Client」範例程式，在螢幕上將提供三個選項，即「修改頻寬」、「進入即時模式」以及「回到乙太模式」。下面我們將介紹 Socket 所傳送的 TCP/IP 封包格式，以及 Server 與 Client 端的處理。

Socket 訊息封包格式：應用程式可透過 Socket 來連接 RT-SBM Server 進行資源預留及控制。利用 Socket 所傳送的 TCP/IP 封包在 RT-SBM Server 的裝置介面層中會被 Message Processor 解碼。訊息存放在 TCP 封包的資料區中(如圖 5 所示)，其資料結構如下：

```
struct resv_msg {
    unsigned int    cl;
    unsigned int    gt;
    unsigned int    ctrl;
    char            id[26];
};
```

Type 0800	IP Header	TCP Header	cl	gt	ctrl	id

圖 5 TCP/IP 封包之控制訊息格式

其中 cl 是 Client 要求的 Control Loade 服務需求量，gt 是 guaranteed 服務需求量；Ctrl 是控制代碼，用來區別是傳送命令或預約資源。將此結構的變數填入所需數值後，以 Socket 的 write()函式即可送到 Server 端。id 主要在區別 RT-SBM 的控制訊息用，為防止一般 Socket 封包被誤認為是 RT-SBM 的控制訊息，所以設定了一組 id 給 RT-SBM 的 Socket 程式來識別。

Server 端的處理：當 Client 使用 Socket 送封包到 Server 時，Message Processor 會檢視有無控制訊息 id。如控制訊息存在且正確時，則執行控制碼所對應的程序。並將接收到的訊息原封不動的送回 Client 端，Client 端因此得知訊息已送達 Server。

Client 端的處理：當安裝好 RT-SBM 系統時，使用者可以透過「Client」範例程式向 RT-SBM Server 的「Server」範例程式提出進入即時模式的需求，在進入即時模式之前，網路一樣使用 CSMA/CD 的方式運作。當進入即時模式後，所有節點已經將自己的乙太位址(Ethernet Address)送到 RT-SBM Server 儲存。在一般的應用上，單一使用者也可以自行送出訊息來預留資源。如果與單一媒體播放程式配合，可以達到動態調整頻寬的目的。在多使用者下，可以製作一個 RT-SBM Client Deamon，每個要用網路的使用者都要向此 Deamon 提出要求，Deamon 負責記錄及合併需求，再向 RT-SBM Server 提出需求，若需求有變更時，向 Server 做預留的改變。模式的切換 Deamon 也必須考慮在內。

3.3.2、裝置介面層

Linux 的裝置介面層負責所有協定與驅動程式的溝通，主要負責封包輸入與輸出的函式，分別是 netif_rx()以及 dev_queue_xmit()(參見圖 2, P.8)。當上層協定將 Socket Buffer 送下來時，會呼叫 dev_queue_xmit()來傳送封包，我們將此函式更名為 skb_xmit()，原先的 dev_queue_xmit(函式我們用來將 socket buffer 插入 socket buffer queue 中，如此一來封包的輸出就可以受我們控制。網路卡每接收完一個封包，就會呼叫驅動程式的接收函式(例如 D-Link 500TX 的 tulip_rx())來接收封包，再將封包的資料交由 netif_rx()來處理，我們在 net_bh()可以接收到封包，如果是 RT-SBM 的訊息，就呼叫相關函式處理。為了建構 RT-SBM 協定，我們必須先制定訊息封包的格式。

- 乙太訊息封包的格式

RT-SBM 的訊息是以乙太封包(Ethernet Packet)格式傳送，如圖 6 中所示。若 type 欄為 16 進位的 0x0800 表示是一個 IP 封包

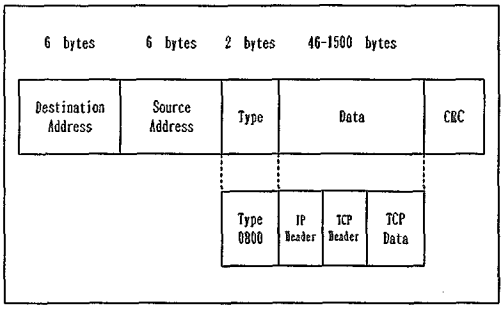

圖6. 乙太封包格式與TCP/IP封包格式

我們在 Type 欄位中新增 ETH_P_RTSBM 為 0x1234，代表 RT-SBM 所使用的訊息，並且將所有的欄位，限定在 46 個 bytes 之中，以最小封包達到減少傳送以及處理 token 之 overhea 的目的。(為避免修改 if_ether.h，ETH_P_RTSB 我們定義在 dev.c 中)

乙太訊息封包的接收與傳送

由於接收與傳送 RT-SBM 的乙太訊息封包是 RT-SBM Server 及 Client 都需要的功能，所以我們先說明 RT-SBM 協定的乙太訊息封包在裝置介面層如何接收與傳送。

Ethernet Packet Sender：在裝置介面層並沒有提供方便的乙太封包傳送方式，我們必須自行利用裝置介面層的 dev_queue_xmit()來完成此功能。只須將 socket buffer 填入必要的欄位，以及目的端的乙太網路位址，接著再呼叫傳送封包函式 dev_queue_xmit()就可以將訊息封包傳送出去。所有 RT-SBM 的訊息輸出，都是由此模組負責。

Message Processor：訊息可以傳送之後，我們必須有方法將訊息接收下來。網路卡接收一個乙太封包後，驅動程式的接收函式(例如 tulip_rx()函式)會呼叫裝置介面層的 netif_rx()，將封包存入 backlog queue，這個 queue 可儲存 300 個封包，超過時封包就會被丟棄。存入後將網路部份的 bottom half 旗號標為「未處理」，待中斷返回後，當 scheduler 發現網路有新進的封包時，會呼叫網路的 bottom half 處理程式 net_nh()來處理封包。而我們在 net_bh()中加入了訊息的判斷，如果收到的是 RT-SBM 訊息封包，就會呼叫訊息處理模組 Message Processor 來處理。

● **Server 端的處理**

RT-SBM 的 Server 主要在管理節點的異動以及分派 token 給節點，所以接下來我們將說明 Node Manager 以及 Token Dispatcher 的實作方法。

Node Manager：Node Manager 管理節點名單(Node List)的節點新增、資料修改以及節點刪除。所採用的節點名單結構是單一連結序列(Single Link List，參見圖 8)，其中 eth_addr 存放的是 Host 的乙太網路位址(Ethernet Address)；三個整數 be_serv，cl_serv，gt_serv 分別是 BE，CL，GT 三種服務型態的欄位，next 指標則指向 List 下一個節點。此 List 由 Node Manager 來管理。使用時是由一索引值來記錄下一個將握有 token 的節點位置。

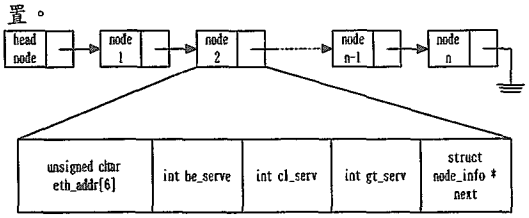

圖8. 節點名單結構圖

當 Server 分派 token 時，Node Manager 根據索引值來找出節點的位置，再將索引值加一，根據節點的位置可取得節點的乙太網路位址與頻寬需求資訊。修改某個節點

```
int admission_control(int new_req)
{
if((total_be - new_req - ((node_num + 1) *
    BE_LOWER_BOUND)) < 0 )
    return(ADC_REJECT);
else
    return(ADC_ACCEPT);
}
```

的頻寬需求，只要給 Node Manager 節點的乙太網路位址及需求值，它會先檢查節點是否存在，然後執行允入測試，如果允入成功才將頻寬更新成目前所傳入的值。

除了修改頻寬需做允入控制外，節點加入時同樣也要執行，因為有可能目前的可用頻寬已經小於最小 BE 保障頻寬。

允入控制的功能是為了保障正在服務的所有節點使用資源的權利，每個新加入的及時需求，都要經過允入控制的檢查核准後才能被接受。另外為了能向 RT-SBM Server 提出需求，以及保證 Best Effort 服務不會有無限等待的情況發生，我們保留了一定比例的頻寬給 Best Effort 服務群組使用。其中 BE_LOWER_BOUND 就是保留給 Best Effort 的頻寬，式子計算出當新加入一個節點時(node_num+1)，所有 BE 的頻寬(total_be)是否能滿足最低需求，如果不能則傳回 ADC_REJECT，可以則傳回 ADC_ACCEPT。所以在加入新的節點或節點修改需求時，都必須執行 admission_control()來決定是否能滿足此需求。區別是加入節點時 new_req 設為零。

Token Dispatcher：Token Dispatcher 主要功能為分派 token 以及節點當機時的處理。它在三種情況下會被觸發，一是「節點加入群組」時間結束，二是 Token 回返時，三是有節點當機時。Token Dispatcher 產生 token 時要向 Node Manager 取得節點的資料，填好節點乙太位址及頻寬需求後才能傳送出去。RT-SBM Server 中使用 Linux 的 time 來實作 Token Dispatcher。Linux 提供使用者一般 time 的操作有 add_timer()以及 del_timer() 兩個函式。使用 time 必須了解 timer_li s 這個結構，timer_li s 包含*next，*prev 指標，做為連結前後節點使用，由系統負責維護。另外 expires，data 及*function 必須由使用者填入所要使用的值。expires 變數是參考系統時間變數 jiffies，jiffies 負責記錄自開機後，計時器啟動以來所經過的時間，Linux 每隔 10ms 就會將 jiffie 加一，所以如果要在 1 秒後呼叫自定的 time_up() 函式，用法如下:

```
Struct timer_list my_timer;
my_timer.expires = jiffies + 100;
 //100 * 10ms = 1 sec
my_timer.data = 0; //no use
my_timer.function = time_up;
add_timer(&time_up);
```

如果需要每秒執行某工作一次，利用上面的例子，在 time_up()中加入:

```
void time_up(unsigned long data)
{
      //Insert Some Code here
      my_time.expires = jiffies + 100;
      my_timer.function = time_up;
      add_timer(&time_up);
}
```

time_up()將每次被呼叫一次。RT-SBM Server 運用上述技巧來維持系統運轉，使 TRT 被控制在所定義的範圍。另外容錯也是以 timer 來完成，例如送出 token 前可以啟動 timer，若 token 在 timer 期滿前被歸還，則執行 del_timer()取消 timer 的運作。若 token 在 timer 期滿前沒有被歸還，則假設節點已當機， timer 將呼叫 Node Manager 將此節點刪除，

● **Client 的處理**

Client 在收到 token 前不能傳送封包，但上層協定可能不定期會使用 dev_queue_xmit()來傳送封包，使用 wait 的方式等待 token 將使系統效能下降，因此我們製作了 Socket Buffer Queue 來暫存 Socket Buffer，以便在 token 來臨時讓 Socket Buffer Throttler 可以透過 Socket Buffer Queue Manager 將封包輸出。

Socket Buffer Queu：Socket Buffer Queue 只儲存

socket buffer 指標值，device 指標值以及優先權值 pri，這三個值是 dev_queue_xmit()的三個傳入參數，可以從 Queue 中取出資料後立即傳入 dev_queue_xmit()使用。Socket Buffer Queue 是一個循環佇列(cyclic queue)(參見圖 9)，有先入先出(First In First Out)的特性，在使用上我們將上述三個值用一個結構變數 skb 集合起來，方便做 Insert 及 delete 的動作。

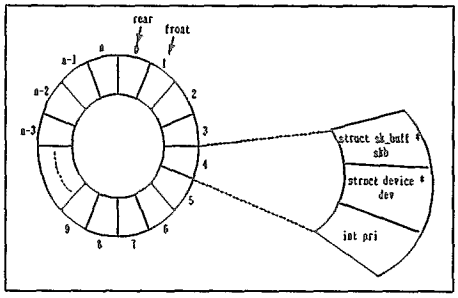

圖9. Socket Buffer Queue結構圖

製作 Socket Buffer Queue 除了解決上層協定的封包傳送問題之外，也很容易知道目前節點上有沒有封包要送。當 Queue 為空時，表示沒有封包要送。當 Queue 為滿時，封包將暫時停止插入 Socket Buffer Queue，直到 Socket Buffer Queue 有空間為止。

Socket Buffer Queue Manager：提供上層協定將結構變數 skb 插入 Socket Buffer Queue 以及 Socket Buffer Throttler 從 Socket Buffer Queue 中讀取 skb 的功能。當 Queue 為空時，傳回 Queue 已空的訊息給 Socket Buffer Throttler，然後 Socket Buffer Throttler 再通知 Token Acker 將 Token 傳回 Server。當 Queue 為滿時，暫時停止上層協定將結構變數 skb 插入 Socket Buffer Queue，並傳回 Queue 已滿的訊息給 Socket Buffer Throttler。

Socket Buffer Throttler：為了擷劫封包輸出，我們將原本裝置介面層的 dev_queue_xmit(的程式碼移到 skb_xmit()，在 dev_queue_xmit()內則加入了我們 Socket Buffer Queue 的機制。token 未到前，所有傳入 dev_queue_xmit(的 Socket Buffer 都被插入 Socket Buffer Queue 中；token 來時，Socket Buffer Throttler 才會向 Socket Buffer Queue Manager 要求讀取資料，接著再利用裝置介面層的 dev_queue_xmit()傳送出去。Socket Buffer Throttler 在三種情況下會停止封包的傳送並呼叫 Token Acker 歸還 token，一是 THT 用完，二是 THT Counter 小於零，三是 Socket Buffer Queue 為空。

4、系統性能量測

我們希望藉著實驗來出系統的性能，以及找出系統的瓶頸。由於量測的環境會影響量測的數據，所以下面的章節裡，我們首先將說明測試的配備及使用的作業系統，接著說明量測的結果。

4.1、測試平台與環境

在我們的實驗系統中，共採用了 5 台 PC，每台 PC 皆安裝了 Linux 作業系統(kernel 2.0.36)，網路的裝置介面層已全部更換成 RT-SBM 系統，系統的實際連接圖見圖 10。

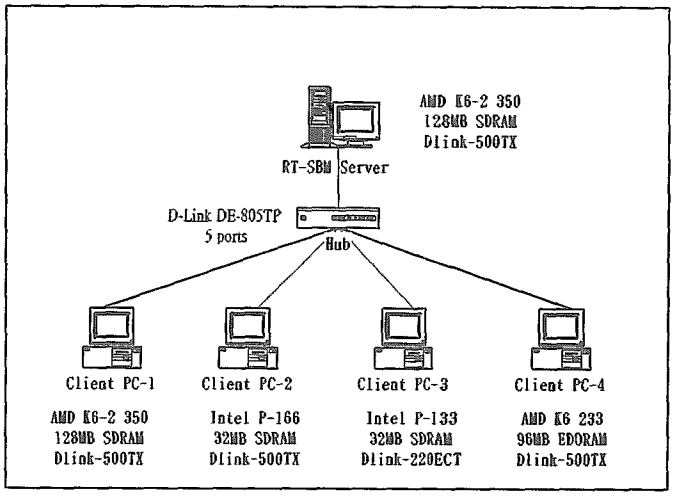

圖10 系統實際連接圖

4.2、測量結果

本篇將實驗及測試分為五個項目，分別是系統穩定性測試、容錯測試、Software Overhead 的量測、週期時間的量測以及即時性的量測。

穩定性測試：為證明所設計的 RT-SBM 系統穩定性，先以兩台 PC，一台當 Server，一台當 Client，進入即時模式後持續運轉 48 小時，依然可以正常操作。再以五台 PC，其中一台當 Server，四台當 Client 連續運行 8 小時，依然可以正常操作。所以 RT-SBM 應該沒有不當使用記憶體的行為產生，Token-Passin 的機制正常。

容錯測試：當 RT-SBM Server 進入即時模式時，我們將已允入的 PC-1 的網路線拔掉，RT-SBM Server 在 PC-1 的 THT 加一小段寬容度時間之後刪除此 PC 在節點中的資料，其它 PC 仍在即時模式下正常運作。當 PC-1 網路線插回 Hub 後，PC-1 在收到 Server 的 JOIN_GROU 訊息時，回應 JOIN_GROUP_AC 訊息，隨即加入群組中，單一節點加入的時間最大不超過一個 TRT。當 RT-SBM Server 運作中，我們試著拔掉 Server 的網路線，我們發現所有的 Client 很快的都回到乙太模式中，因為 Client 在一個 TRT_{MAX}(參見 3.3 系統協定之 RT-SBM Server 不正常關機或當機)後沒收到 token，就判定為 Server 當掉，所以都進入乙太模式。關於節點自行切回乙太模式的測量見表 3(設定 TRT_{MAX}=200ms)。由於 RT-SBM Server 不是真的當機，所以它會以為是所有的 Client 都當機，因此節點一一被刪除，直到剩下 Server 自己。所以當我們再將網路線插入 Hub 中，所有 Clien 大部份在一個 TRT 內會全部加入群組，來不及加入的也會在進入即時模式後，動態的加入群組。

網路頻寬	10Mbps	100Mbps
PC-1 測得時間	200.07 ms	200.07 ms
PC-2 測得時間	200.05 ms	200.05 ms
PC-3 測得時間	200.01 ms	200.01 ms
PC-4 測得時間	200.04 ms	200.04 ms

表 3 節點自行切回乙太模式時間

Software Overhead 的測量：為了比較網路速度對 token oeverhead 的影響，我們將測試兩種不同的速度。在此項測試中我們只允入 PC-1，在 10Mbps 及 100Mbps 的 Hub 連接下，我們測得的 software overhead 如表 4，量測的方式是在 Server 的 Token Dispatcher 中記錄 token 送出及收回的時間，為了方便量測，我們將

token 固定送給同一個 Client，而在 Client 中令其立刻將 token 歸還，不傳送任何封包，這樣就可以得到因 token 的處理而多花費的所有時間。

網路頻寬	10Mbps	100Mbps
測得時間	148.44ms	22.7ms
RETHER	Token Overhead 66.04ms	

表 4 所有因處理 token 而產生的軟體 overhead

週期時間的量測：我們將 TRT 設為 100msec，共三台 PC 運作，我們分別在兩台 Client 中的接收端量測 token 到達的時間間隔，當時的 THT 分配為 Server 為 40msec，其餘兩台 Client 皆為 30msec，測量結果如表 5。由結果可以看出在沒有資料傳送時，週期時間是很穩定的。

	最小值 (μsec)	平均值 (μsec)	最大值 (μsec)	標準偏差	
				(μsec)	(%)
PC-1	40260	40319.20	40480	39.396	0.098
PC-2	40260	40324	40400	37.245	0.092

表 5 token 到達 Client 的時間間隔

接著我們由兩台 PC 來測量傳送資料下，資料流到達接收端的間隔時間，這是為了確保在資料傳送下，週期仍然維持在一個可以接受的範圍之內。我們寫了一組測試程式，它將從傳送端透過 Socket 的方式傳送 100 組 1500bytes(15KB/S)的資料到接收端去，我們在接收端記錄收到的時間，看看偏差的範圍如何。如表 6 來看，偏差有點大，可能是 Socket 程式在取得時間上，準確性不如系統核心的裝置介面層中高的關係。

	最小值 (sec)	平均值 (sec)	最大值 (sec)	標準偏差	
				(sec)	(%)
PC-1	37520	38710.26	40065	1027.3	2.65

表 6 data 到達 Client 接收端的時間間隔

即時性保證量測：我們將實際以應用程式來測試 RT-SBM 系統對即時性的保障能力。我們在 PC-1 上架設 HTTP Server。PC-2 上安裝 MpegTV[9]播放程式，可以透過 HTT 來播放 MPEG 影片。另外 PC-3 透過 FTP 向 PC-4 下傳檔案。實驗分別在乙太模式和即時模式下進行。在 10Mbps 頻寬下(理想狀況)，使用 HTTP 傳送影片檔(MPEG-1)以及 FTP 傳送一 400Mbytes 的檔案，並記錄 RT-SBM Client 端資料的輸出量，測量的結果在表 7(P.9) 與表 8(P.9)。從表 7 可以看出，同時播放時 MPEG-1 所需的 14%頻寬被排擠到 11%，因此在乙太模式下播放時，影片經常會停頓。

表 8(P.9)是在即時模式下的量測，為了使用所有的頻寬，我們同時播放兩部影片，使用 29%的頻寬，扣除 4%的其他節點 BE 保留頻寬外，其餘的 67%全預留給 FTP 使用，測量的結果顯示 FTP 只能使用 47%的頻寬，大約有 20%的頻寬被軟體與協定 overhea 佔用，如果以乙太模式下 FTP 的結果(93%)來當做網路最大頻寬時，實際 overhea 只有 13%。除此之外，在實驗進行中，可以很明顯的從螢幕上看出，在即時模式下播放的速度很順暢，沒有停頓的現象。

乙太模式 (ethernet mode)	傳送協定	平均傳送速率	使用頻寬百分比	最大可用頻寬百分比
單獨使用頻寬 PC-1 → PC-2	HTTP	172.56 Kbytes/sec	14%	100%
單獨使用頻寬 PC-4 → PC-3	FTP	1125.26 Kbytes/sec	93%	100%
同時傳送 PC-1 → PC-2	HTTP	135.88 Kbytes/sec	11%	100%
PC-4 → PC-3	FTP	977.50 Kbytes/sec	81%	100%

表 7 乙太模式下平均傳送速率表

即時模式 (real-time mode)	傳送協定	平均傳送速率	使用頻寬百分比	預留頻寬百分比
單獨使用頻寬 PC-1 → PC-2	HTTP	171.58 Kbytes/sec	14%	14%
單獨使用頻寬 PC-4 → PC-3	FTP	232.16 Kbytes/sec	19%	20%
同時傳送 PC-1 → PC-2 PC-4 → PC-3	HTTP*2	357.78 Kbytes/sec	29%	29%
	FTP	776.12 Kbytes/sec	47%	67%

表 8 即時模式下平均傳送速率表

5、結論

傳統乙太網路架構，無法對即時性資料提供傳送時間保證。目前的即時網路多以硬體達成即時性保證因更換不易、價格偏高等問題，無法獲得市場消費者的認同。RETHE 是首先使用軟體來達成保證即時性的協定，由於實作環境在網路卡驅動程式層，對不同的網路卡必須做不同修改，所以並未獲得網路卡廠商的支持。本論文利用在網路裝置介面層(Device Interface)中，建立 Token Passin 機制來解決乙太網路下由於封包碰撞造成傳送時間無法預測的問題，既不用更新硬體也不用修改網路卡，解決了上述的問題。由量測的結果得知，透過 token 控制器，以 Deficit Round Robin 的排程方式來分配節點間傳送時間，的確可以保 障資料傳送的即時性。此外，RT-SBM 的容錯性及穩定性也比 RETHE 為佳。

參考文獻

[1] IEEE/ANSI Standard 802.4-1985, Token passing bus access method and physical layer specifications. IEEE Inc., New York, 1985.

[2] IEEE/ANSI Standard 802.5-1985, Token ring access method and physical layer specifications. IEEE Inc., New York, 1985.

[3] The 3COM Technical Journal. 3COM's PACE Technology. http://www.3com.com/nsc/501316.html

[4] The 100VG-AnyLAN concepts page http://www.iol.unh.edu/training/vganylan/teach/downld/single.html

[5] T.Chiuen and C. Venkatramani. The Design, Implementaion and Evaluation of RETHER: A Real-Time Ethernet Protocol. PhD thesis, State University of NewYork at Stony Brook, Nov 1996.

[6] Paul Ferguson, Geoff Huston. Quality of Service: Delivering QoS on the Internet and in Corporate Networks. Wiley Computer Publishing, 1998, ISBN 0-471-243582-2.

[7] M. Shreedhar and G. Vrghese. Efficient Fair Queuing Using Deficit Round -Robin. IEEE/ACM Transactions O Networking. VOL.4, NO.3, June 1996.

[8] Carl A. Waldspurger and William E. Weihl. Stride Scheduling : Deterministic Proportional-Share Resource Management. Technical Memorandum MIT/LCS/TM-528. MIT Laboratory for Computer Science, Cambridge, MA 02139. June 22, 1995.

[9] MpegTV Home Page:http://www.mpegtv.com

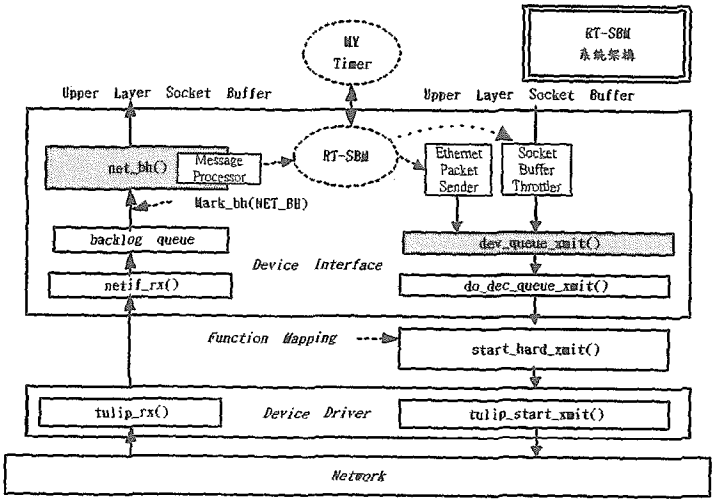

圖2. RT-SBM系統架構圖

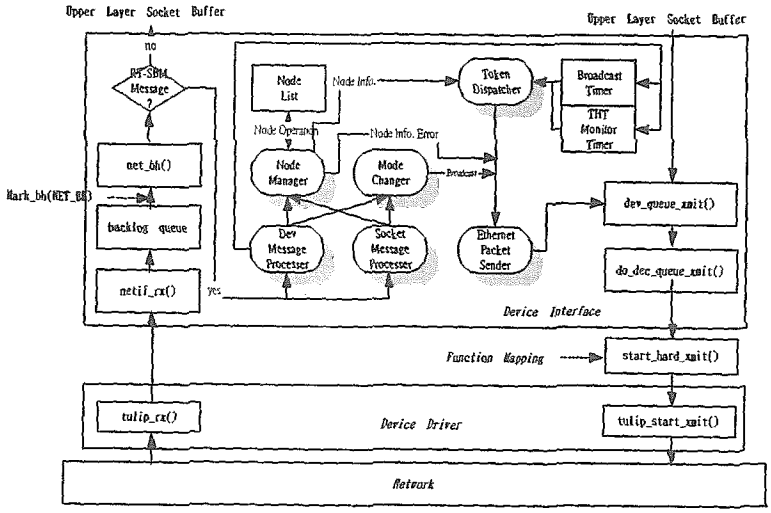

圖3 RT-SBM Server 架構方塊圖

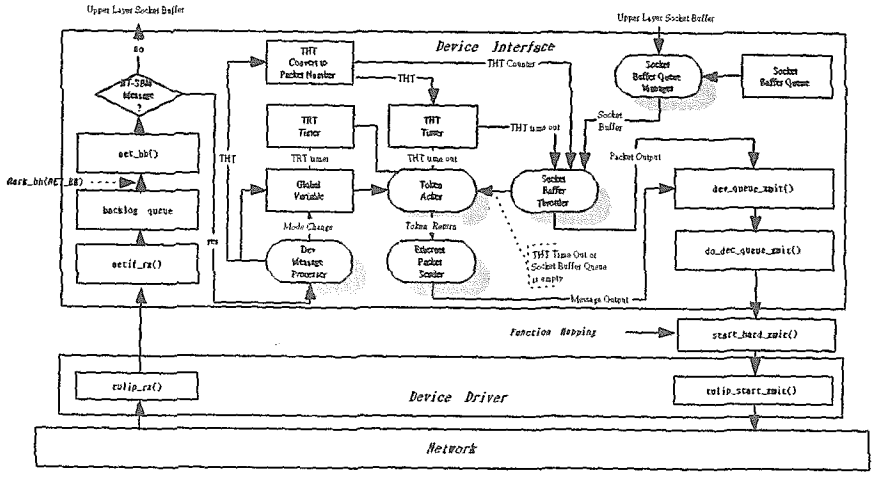

圖4 RT-SBM Client 架構方塊圖