國立中山大學電機工程學系

碩士論文

一個能夠達到最短延遲時間的 anycast 路由協定

A Minimum Delay Anycast Routing Protocol

研究生：黃偉程 撰

指導教授：李錫智 博士

中華民國 九十二 年 七 月

摘要

IPv6(Internet Protocol Version 6) [6]在位址格式中新增加了 anycast[5] 的位址。在傳送 anycast 的封包時，若多個內容相同的伺服器擁有同一個 anycast 的位址，該 anycast 封包是送到離 client 最接近的伺服器中，因為離 client 最接近的伺服器，並不代表是 delay time 最小的伺服器，因此在這篇論文中，我們發表了一個針對 anycast 封包的通訊協定。在這個通訊協定中包含了兩個子協定。第一個子協定為建立 routing table 的子協定；第二個子協定為封包傳送的子協定。在建立 routing table 的子協定中，我們提出了一個新的方法，稱之為最短延遲時間路徑(minimum delay path method)。根據最短延遲時間路徑，我們可以得到從 router 到目的端最短延遲時間的路徑，若根據此路徑來傳送封包，則可得到比傳統方法較佳的 QoS(Quality of Services)。在封包傳送的子協定中，我們提出一個最小延遲及負載平衡(minimum delay and load balancing)的方法。此方法最主要是根據目前所有伺服器的負載情況，及連接該伺服器的最短延遲時間路徑，來決定其中最適合傳送的伺服器及路徑。根據實驗的結果顯示出我們提出的方法可有效減少 end-to-end delay 及增加網路的 throughput。

Abstract

Anycast is a new communication service defined in IPv6 (Internet Protocol Version 6) [6]. An anycast message is the one that should be delivered to the 'nearest' member in a group of designated recipients. The ‘nearest’ is not always the ‘best’ member. In this paper, we propose a routing protocol for anycast message. It is composed of two subprotocols: the routing table establishment subprotocol and the packet forwarding subprotocol. In the routing table establishment subprotocol, we propose a mininum delay path method (MDP). We get a minimum delay path from router to destination by MDP. In the packet forwarding protocol, we propose a minimum delay and load balancing method (MDLB). We dispatch traffic load to a server with minimum delay and light load by MDLB. The performance has demonstrated the benefits of MDP and MDLB in reducing end-to-end delay and increasing throughput of network.

目錄

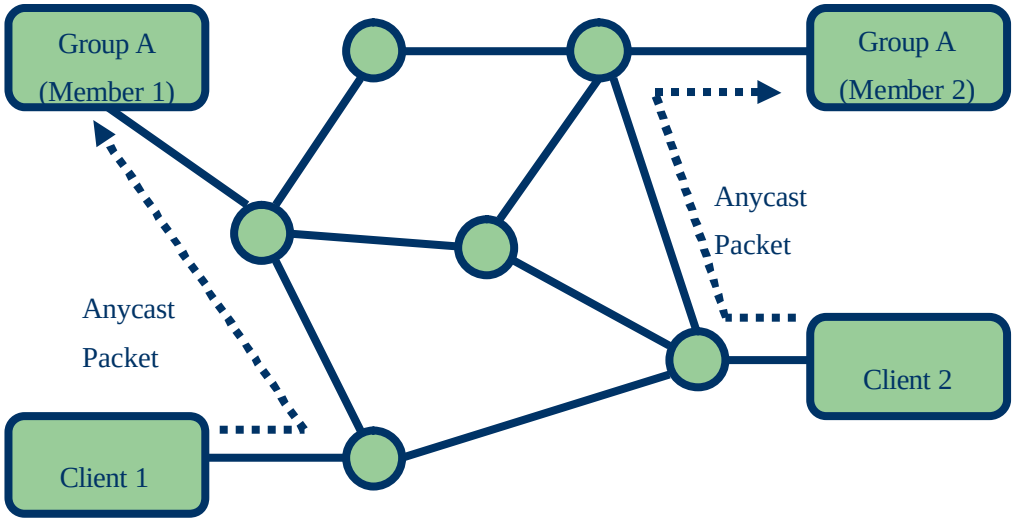
圖目錄

第一章　簡介

Anycasting[5]最初是由 Internet Engineering Task Force 所提出的，其動機主要為：當 client 對 server 提出要求，若有多個 server 可提供相同的服務, 則 client 可以不需要去在乎是那一部 server 被使用。Client 送出含有 anycast 位址的封包時，則網際網路會將該封包送至所有 anycast group member 中的一部。因此，anycasting 服務可有效地簡化了使用者在找尋最適當的 server 方面的問題，提供了使用者更佳的方法性。舉例來說，當使用者輸入：

telnet　archie.net

便可以直接連接上最接近使用者的 archie server。另一個例子，當多部內容相同的 FTP(File Transfer Protocol)伺服器使用同樣的 anycast 位址，則使用者只需連接至該 anycast 位址，便可直接連接至離使用者最接近的 FTP 伺服器。如＜圖 1.1＞所示，client 1 連接至 group A 時，會自動連接至 member 1，因為 member 1 和 client 1 的距離較 member 2 接近；同樣的，若 client 2 連接至 group A 時，則會連接至 member 2。



＜圖 1.1＞Anycasting 服務示意圖

由於愈來愈多的網路相關應用需要用到 anycast 的服務，因此在最新版的 IPv6[6]中已經將 anycast 位址加入並成為一標準。而目前在 anycasting 服務的研究上，大致可分成兩大方向：第一，在應用層(application layer)上使用 anycasting 的相關服務及管理方法[1]。第二，在網路層(network layer)上 anycasting 相關的 routing 方法[2,3,4,8]。

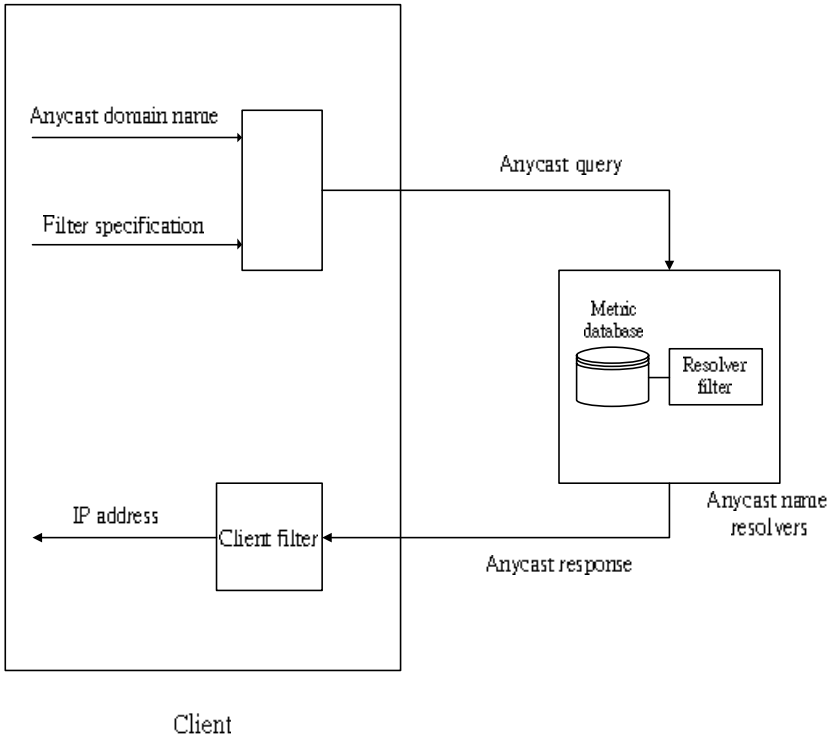
所謂應用層上的 anycasting 服務是將 anycast routing 作在應用層上。一般的作法是設置 anycast name resolvers[1]，此 anycast name resolvers 類似 domain name system(DNS)，主要功能是將 anycast domain name (ADN)轉換成 IP 位址。當 client 進行 anycasting 服務前先送 anycast query 訊息到 anycast name resolvers，anycast name resolvers 再將 anycast query 中包含的 anycast group 中最佳 member 的 IP 位址送至 client。Client 則以此

IP 位址作為目的位址，便可將封包傳送至 anycast group 中最佳的 member。

如＜圖 1.2＞所示，client 傳送到 anycast name resolvers 的 anycast query 中，需包含 anycast domain name(ADN)及選擇 anycast group member 的標準。當 anycast query 送至 anycast name resolvers 時，anycast name resolvers 根據 anycast query 中的 ADN 來找出 metric database 中所有的 anycast group member 的資料，包括：member 的 IP 位址、distance、反應時間等。再根據選擇 member 的標準，由 resolver filter 選擇其中最佳的 member。決定最佳的 member 後，將該 member 的 IP 位址放入 anycast response 中，並將 anycast response 傳送回 client。當 client 接收到 anycast response 時，由 client filter 讀出 anycast response 中的 IP 位址。接下來，若 client 要送封包至 anycast group member 時，以此 IP 位址為封包之目的位址，便可直接將封包送至 anycast name resolvers 所選擇之最佳 anycast group member。

Anycast name resolvers 主要功能為將 ADN 轉換成為 IP 位址，因此 anycast name resolvers 需要有：(1) anycast group 所有 member 的 IP 位址。(2) 所有 member 的相關資料，如：distance、反應時間(response time)、負載量等。這些資料的取得方法為：每隔一段時間 anycast name resolvers 會根據 anycast group member 的 IP 位址傳送檢測封包到所有 anycast group member，測量從 anycast name resolvers 到所有 member 的反應時間或 member 的負載量等

資訊，並記錄於 metric database **之中。**



另一方面，應用層 anycasting 服務主要的優點則為：(1) 在選擇所謂“最佳”的 anycast group member 時，可以有很大的彈性，亦即可以依據使用者不用的需求來訂定選擇的規則，例如：使用者可以選擇距離最近的 member 或是反應時間最短的 member，再將所選擇的規則放在 anycast query 中的 filter specification 中。anycast name resolvers 根據 filter specification 去找出使用者所需要的 anycast group member。 (2) 可以很容易的在現有網路上，提供 anycasting 服務。因為將 anycasting 服務放在應用層上，所以不需要更改到目前的 IP 位址、routing protocol，router 即使不支援 anycast 位址亦能執行 anycasting 服務。

我們繼續介紹網路層 anycasting 服務。所謂網路層 anycasting 服務是將 anycast routing 做在網路層上，也就是說我們必須在 IP 位址另外分配給 anycasting 服務，而且 router 必須要能支援 anycast 位址以及 anycast routing protocol。

接下來，我們簡單敘述目前有關網路層 anycasting 服務的相關文獻。

1. 在 [3] 中，計算所有 anycast group member 其權重值 (weight)，若權重值愈大，該 member 被選到的機率亦愈大。其權重值是以自 router 到 member 間的 distance 成反比來計算，如下式所示：

$$W_i = \frac{\frac{1}{D_i}}{\sum_{j=1}^{k} \frac{1}{D_j}} \qquad (1)$$

其中W_i為 member i 的權重值，D_i為 router 到 member i 的 distance，k 為 anycast group member 的個數。當得到W_i後，再根據過去該 member 之 flow 建立成功或失敗的個數來調整 weight 值。當 member 之 flow 建立失敗的次數愈多，則將該 member 的 weight 值降低，因此在該 member 中再建立新的 flow 的機率就會降低；反之，若 flow 建立成功的次數愈多，則將該 member 的 weight 值增加。因此在該 member 中再建立新的 flow 的機率就會增加。

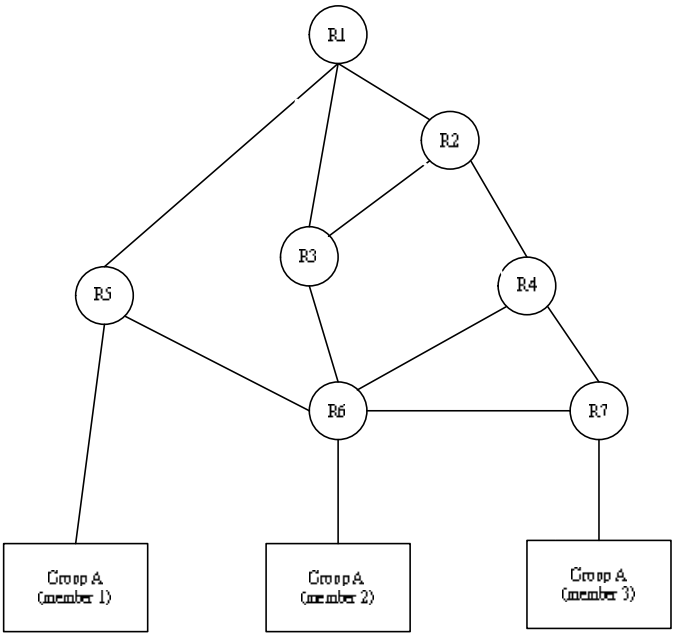
根據以上的分析，我們得知本方法乃是將 anycast 封包送至離 router 最接近的 member 機率最大，若最接近的 member 已超過負載，則變成次接近的 member 被選到的機率最大。

2. 在[4]中，該作者共提出了四種傳送 anycast 封包的方法，分別敘述如下：

(1) Shortest-Shortest Path Method(SSP)。

SSP 指的是在所有的最短路徑中選出最短的路徑來作為傳送的路徑。而所有的最短路徑指的是，從 router 到所有 anycast group member 間 distance 最小的路徑。當得到自 router 到所有 member 間的最短路徑後，選擇其中 distance 最小的路徑即為要傳送 anycast 封包的路徑。舉例來說，假設 anycast group A 共有三個 members，如＜圖 1.3＞所示。其中虛線所代表的為 R1 到 Group A 所有 members 的最短路徑，而這三條路徑又以 R1 到 member 1 的路徑為最短，因此當 R1 接收到 anycast 封包時，會先送到 R5，最後再送到

member 1。


R1
R2
R3
R4
R5
R6
R7
Group A
(member 1)
Group A
(member 2)
Group A
(member 3)

傳送的路徑。因為 MIN-D 的可傳送路徑較 SSP 為多，因此會較 SSP 不易產生 congested。但是相對的必須要額外的資訊，即 router 本身及 next hop 的 minimum distance 值，另一個缺點則是，MIN-D 所有的可傳送路徑並不一定為最短路徑。

(3) Source-Based Tree Method(SBT)。

在此，作者提出了一個理論上最理想的方法，即 SBT。以 source 作為 tree 的樹根，anycast group 的 member 為樹葉，從樹根到樹葉的路徑為兩者間的 shortest path。此方法較前兩者而言，會有更多的選擇路徑，而且這些路徑皆是從 source 到 member 的最短路徑，因此理論上而言，SBT 應該是最佳的方法。但是 SBT 卻有一個嚴重的缺失，那就是 SBT 所需要的資訊太過龐大。舉例來說，若有 N 個 sources 及 M 個 member 時，則使用 SBT 方法的 router 之 routing table 中需要有 O(NM)個 entry。相對而言，SSP 只需要 1 個 entry，而 MIN-D 需要 O(M)個 entry。

(4) Core-Based Tree Method(CBT)。

為解決 SBT routing table 太過龐大的缺點，因此作者參考[9]的方法而提出 CBT。CBT 與 SBT 最主要的差別就是 SBT 需要建立多個 tree，而 CBT 只需要建立一個 tree。CBT tree 的建立，首先在網路中選擇一個 router 作為 core，把 core 當作 CBT tree 的樹根，anycast group 的 member 為樹葉，從樹根到樹葉的路徑為兩者間的 shortest path。網路的 router

若在 tree 中, 稱為 on-CBT router;若不在 tree 中, 稱為 off-CBT router。因此在傳送 anycast 封包時，亦分成兩個部分。在 off-CBT router 部份，是使用 SSP，當 anycast 封包經過 on-CBT router 時, 再從那多條 shortest path 中選擇其中的一條。至於要從多條路徑選出一條傳送路徑的方法，如式(1)所示。距離 router 愈近的 member 有愈高的 weight 值，因此被選擇到的機會也愈高。

由上述所知，我們知道網路層 anycasting 服務需要在目前的 IP 位址另外分配給 anycast 服務，而且 router 也必須要支援 anycast 位址及 anycast routing protocol, 所以在選擇 anycast group member 時是由 router 來決定而無法由 client 來決定。這一點對使用者來說可能是優點也可能是缺點，優點就是對一般使用者而言更具有方便性，因為使用者不需決定選擇 anycast group member 的標準。相對地缺點就是缺乏選擇 anycast group member 的彈性。網路層 anycasting 服務另一個優點則是 client 在傳送資料到 anycast group member 前不需要向 anycast name resolvers 做查詢的動作, 因此可以減少傳送資料前的等待時間。而應用層的 anycasting 服務就必須要向 anycast name resolvers 查詢後才能傳送資料到 anycast group member，因此等待時間會較網路層的 anycasting 服務為長。

目前 IPv6 已經為 anycasting 服務分配了位址空間, 且 anycasting 服務亦已制定在 IPv6 中，因此我們相信未來會有愈來愈多的 router 能夠開始支援 anycasting 服務。

因為目前網路層 anycasting 服務的相關文獻在選擇 anycast group member 時，皆是以 distance 的大小作為選擇 member 的依據。然而距離使用者 distance 愈小的 member 不一定會是反應時間最短的 member，所以我們希望能夠找到一個對使用者而言是反應時間最短的 anycast group member。反應時間又包含了兩個部份，一是資料在網路上傳輸的時間，二是 member 處理資料的時間。所以我們的目標是要找到一個反應時間最快的 anycast group member。而所謂反應時間最快的 member 即為最短的網路傳輸時間加上 member 處理的時間。因此，我們提出的方法主要是針對這兩方面來進行。第一部份就是找出從 router 到 member 間傳輸時間最短的路徑，而第二部份則是要找出負載最輕的 member。

在本論文中，我們提出了一個 anycast routing protocol，作為網路層 anycasting 服務之用，並以[4]中所提的 CBT 方法為基本架構再試圖加以改良。因為我們希望能夠找到反應時間最短的 anycast group member，所以我們將 routing protocol 再分成兩部份：(1) 建立 minimum delay routing table，根據此 routing table 找出 router 到 member 間傳輸時間最短的路徑。(2)根據 member 負載資訊及傳輸時間最短的路徑來決定出最佳的 anycast group member，並將 anycast 封包傳送至該 member。

在接下來的章節中我們將更完整的介紹我們所提出的 anycast routing protocol。在第二章中，我們將簡單介紹什麼是 routing protocol 及 routing protocol 設計的要點。在第三章中，我們會提出一個建立 minimum delay routing table 的方法，來求得 router 到

member 間傳輸時間最短的路徑。第四章中，則會提出如何選擇最佳路徑及 member 的方法。而在第五章中，我們會比較相關的文獻及我們所提出的方法，模擬這兩種方法在網路上效能的比較。第六章則是對全篇論文做個總結。

第二章　我們的方法

2.1 Routing Protocol 概論

在網路中傳送資料是透過路由器(Router)來決定資料傳送的路徑，稱為 routing。而 routing protocol 就是定義 router 在執行 routing 時的相關操作。一般而言，在設計 routing protocol 時有二個必須要考慮的因素：

1. Quality of services(QoS). 任何一個 protocol 在設計時都必須要考慮到使用者在傳送資料的效率問題[7,12,13]，因此如何減少從發送端到接收端的延遲時間為本論文考慮的重點之一。
2. Overhead. 任何 protocol 在執行時皆免不了會消耗網路的資源，如：傳送控制訊息時所需要的頻寬、儲存 routing 資訊時所需要的記憶體等。一個好的 routing protocol 也必須要考慮到 overhead，使得 overhead 愈小愈好[10]。

雖然上述兩個因素皆是我們在設計一個新的 protocol 所必須要去考慮的，但是這兩項卻有可能會彼此互相衝突。舉例來說，當減少執行時的 overhead，往往會使得網路的傳輸效能變差。所以說，在設計一個新的 routing protocol 時，亦要考慮到如何使得這兩項因素達到平衡。

一般而言，router 在決定資料的傳送路徑是依據 routing table 裡的資訊。而每個 routing table 中會包含多個 entry，每個 entry 則有多個 field，這些 field 通常包括：目的位址、next hop、distance 等項

目。當 router 接收到一個封包時，會根據 routing table 來決定該封包的 next hop。因此一個 routing protocol 可以再分成兩個部分，如下所述：

1. Routing Table Establishment Subprotocol.

   此部份主要是收集交換網路 topology 及網路資源的資訊，並利用這些資訊來建立或更新 routing table。建立完成的 routing table 則作為 packet forwarding 之用。

2. Packet Forwarding Subprotocol.

   此部份主要是用在轉傳(forward)封包時使用。當 router 接收到封包時，檢查該封包內的目的位址並根據 routing table 來查出相對應的 next hop，而該封包則經由適合的 output port 來傳送至 next hop。

在接下來的部份，我們將會詳細介紹此兩部份所提出的新方法。

2.2 Routing Table Establishment Subprotocol

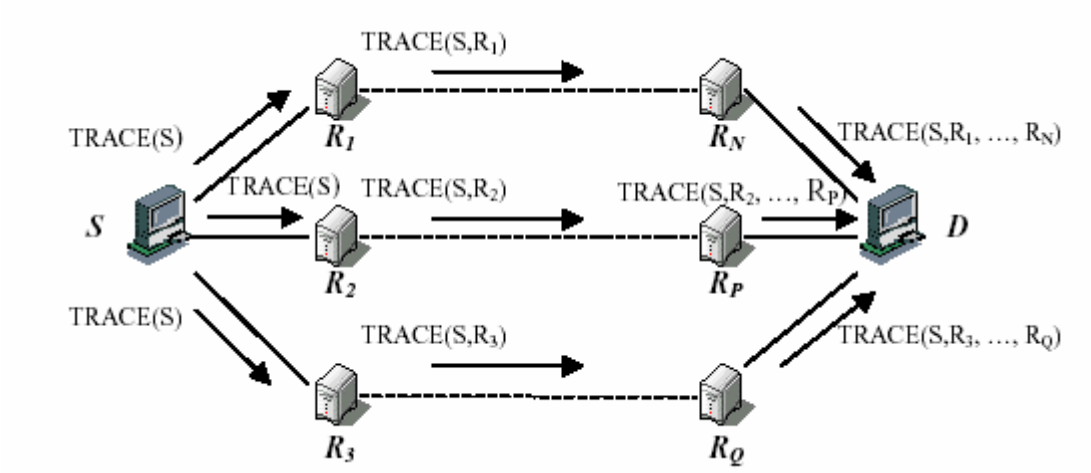
Routing Table Establishment Subprotocol 的目的就是要建立一個可供傳送 anycast 封包的 routing table，可分為下列三個部份。

1. 產生 candidate entries。
2. 由於可能會產生 loop，所以必須再從這些可能的 entry 中選擇出合格的 entry。
3. 根據合格的 entry 建立 routing table。

2.2.1 Candidate entries 的建立

在此，我們提出了一個方法稱為 minimum delay path method。也就是說我們可以產生從 router 到接收端延遲時間最小的路徑。這種方法的優點在於我們可以避免將資料傳送到擁擠(congested)的網路中，使得網路的負載能夠較為平衡，而且可以改善網路的 QoS。

首先，我們使用一個探測的訊息，稱之為TRACE訊息。並將TRACE訊息傳送到整個網路中，來找尋延遲時間最短的路徑。這個TRACE訊息包含了sequence number以及該訊息所經過的所有節點。當router D接收到多個來自router S的TRACE訊息時，則最先接收的TRACE訊息中的路徑即為從router S到router D延遲時間最小的路徑。如＜圖2.1＞所示，若router D總共接收到三個來自router S的TRACE訊息，假設TRACE(S,R1, …, RN)是三個TRACE訊息中最先接收到的訊息，則代表從router S到router D的最短延遲時間路徑為 $S \to R_1 \to \ldots \to R_N \to D$。



＜圖 2.1＞TRACE 封包之傳送圖

在此，router D **已經可以得到從** router S **到** router D **延遲時間最小的路徑了，但是這條路徑是** router S **在傳送資料時所使用的，而不是** router D **所使用的。而且網路的傳輸是雙向的，從** router S **到** router D **延遲時間最小的路徑並不代表該路徑也是從** router D **到** router S **延遲時間最小的路徑。所以說，目前** router S **尚不能得到從** router S **到** router D **延遲時間最小的路徑，因此我們必須再傳送另一種訊息，稱之為** ACK **訊息。**ACK **訊息中包含了** TRACE **訊息中所經過的路徑以及從** router S **到** router D **的延遲時間。而** ACK **訊息的傳送路徑是由** router D **沿著原** TRACE **訊息傳送的路徑之反方向送回給** router S **的，當** router S **接收到** router D **傳回的** ACK **訊息時，則** router S **可得到自** router S **到** router D **延遲時間最小的路徑。**

2.2.2 Updating Algorithm of TRACE message

因為每條 link **的延遲時間會隨著資料量的改變而改變，所以若每當延遲時間改變即送出新的** TRACE **訊息，則會產生相當可觀的**

overhead；若是將 TRACE 訊息傳送的間隔時間加長來減少 overhead，則會降低最短延遲時間的路徑的準確性。因此，我們必須想辦法在 link 資訊的準確性與控制訊息的 overhead 之間取得一個平衡點。

在此我們希望當 link 的延遲時間變動超過 threshold 時，再送出新的 TRACE 訊息以減少控制訊息的 overhead，如(2)式：

$$\frac{\left|B_{ij,T_{k+1}} - B_{ij,T_k}\right|}{B_{ij,T_k}} \geq H \qquad (2)$$

其中 B_{ij,T_k} 為在時間 T_k 時從節點 i 到節點 j 之 link 的剩餘頻寬，H 為 threshold。但是(2)式所產生的 link 更新頻率會隨著交通量多寡產生劇烈的變動，而我們為使 link 更新頻率更加穩定，因此我們動態的調整 threshold，如(3)式：

$$H_i = \begin{cases} H_{i-1} + H_{i-1} * a & if \quad U_i \geq U_u; \\ H_{i-1} - H_{i-1} * a & if \quad U_i < U_l. \end{cases} \qquad (3)$$

其中 H_i 為時間在第 i 個 interval 的 threshold，a 為介於 0 至 1 的值，U_i 為時間在第 i 個 interval 的 link 更新頻率，U_u 為更新頻率的上限，而 U_l 則為更新頻率的下限。根據(3)式，我們可以得到當 link 的更新頻率高於上限時，我們調高 threshold 來降低 link 更新頻率；反之，若 link 的更新頻率低於下限，則調低 threshold 來提高 link 更新頻率。

2.2.3 Updating Algorithm of ACK message

若當 router **一接收到** TRACE **訊息，就傳回** ACK **訊息，將產生可觀的** overhead**，另一方面若** TRACE **訊息中的路徑與延遲時間沒有變化的話，則再傳回的** ACK **訊息只會造成額外的** overhead**。所以我們必須加上一些條件來減少傳回的** ACK **訊息。**

1. **若** router D **接收自** router S **傳來的新的** TRACE **訊息中的路徑與上一個所存** TRACE **訊息的路徑不同時，如**(4)**式，代表延遲時間最短的路徑已經不同，因此必須要傳送** ACK **訊息回** router S**，通知** router S **及路徑上的所有** router **原本最短延遲時間的路徑已經改變。其中** $Path_n$ **為新接收** TRACE **訊息中的路徑；** $Path_l$ **為上一個所存** TRACE **訊息中的路徑。**

$$Path_n \neq Path_l \tag{4}$$

2. **當新接收的** TRACE **訊息中的延遲時間與上一個所存** TRACE **訊息中的延遲時間相差超過** threshold**，如**(5)**式，代表延遲時間最短之路徑雖然不變，但是該路徑延遲時間變化超過** threshold**，因此亦傳送** ACK **訊息，來通知** router S **及路徑上的所有** router **更新** routing table **中的資料。其中** D_n **為新接收** TRACE **訊息的延遲時間，** D_l **為上一個所存** TRACE **訊息的延遲時間，** T **為** threshold**。**

$$\frac{|D_n - D_l|}{D_l} \geq T \tag{5}$$

傳送 TRACE 及 ACK 訊息的演算法：

- Router S

  For i=1 to n do

  If $\frac{\left|B_{ij,T_{k+1}} - B_{ij,T_k}\right|}{B_{ij,T_k}} \geq H$ **then** send TRACE message

  endFor

- Router D

  If $Path_n \neq Path_l$ or $\frac{|D_n - D_l|}{D_l} \geq T$ **then** send ACK message

變數說明：

n : router S output port **的個數。**

B_{ij,T_k} **：時間** T_k **時從節點** i **到節點** j **之** link **的剩餘頻寬。**

H **：剩餘頻寬的** threshold**。**

$Path_n$ **：新接收** TRACE **訊息中的路徑。**

$Path_l$ **：上一個所存** TRACE **訊息中的路徑。**

D_n **：新接收** TRACE **訊息的延遲時間。**

D_l **：上一個所存** TRACE **訊息的延遲時間。**

T **：延遲時間的** threshold**。**

2.2.4 建立 candidate entries 的演算法

- Router D 接收到來自 router S 的 TRACE 訊息

  Step 1. **if** $SN_n \leqq SN_l$ **then** discard $TRACE_n$

  else goto step 2.

  Step 2. **if** $Path_n$? $Path_l$ **then** store $Path_n$, $Delay_n$ and send ACK

  else goto step 3.

  Step 3. **if** $\frac{|D_n - D_l|}{D_l} \geq T$ **then** store $Delay_n$ and send ACK

  else discard $TRACE_n$

- Router S 接收到來自 router D 的 ACK 訊息

  Step 1. Extract route field and delay field from ACK message

  Step 2. construct/update candidate entries.

變數說明：

SN_n : Sequence number of the new received TRACE message

SN_l : Sequence number of the last received TRACE message

$TRACE_n$: New received TRACE message

$TRACE_l$: Last received TRACE message

$Path_n$: Path of the new received TRACE message

$Path_l$: Path of the last received TRACE message

D_n: Delay of the new received TRACE message

D_l : Delay of the last received TRACE message

T : Threshold

2.2.5 Eligible Entries 的選擇

當我們在傳送 unicast 的封包時, 根據 routing table 中的 candidate entries 的資訊即可將該封包送到目的端。但是在傳送 anycast 封包時則非如此, 因為 anycast 封包是從多條可能路徑挑選出某一條路徑去傳送，因此有可能會產生 loop。為避免 loop 的產生，我們參考[4]所使用之方法，稱之為 core-based tree(CBT)，先在網路中將一個 router 設為 core，此 core 為 CBT 的 root，而 anycast group 中所有的 member 為 CBT 的 leaf, 從 core 到 leaf 的路徑為 3.1 節中所建立最短延遲時間的路徑。所以，整個網路的 router 可分成兩種：1)屬於 CBT 的 router 稱之為 on-CBT router。 2)不屬於 CBT 的 router 稱之為 off-CBT router。

在 off-CBT router 的 eligible entries 的選擇中，是以從該 router 到所有 member 的路徑中，選出延遲時間最短的路徑為 eligible entry。而在 on-CBT router 的 eligible entries 的選擇中，該 router 的所有 son 皆為 eligible entries。

在建立 CBT 時, 首先要先決定 CBT 的 core, 其演算法如下所述：

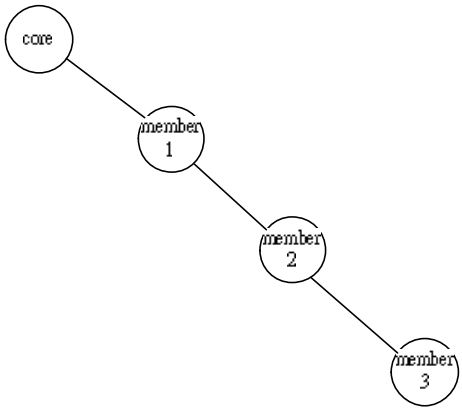
Core selection algorithm:

1. 在 group 建立之初，只有一個 member 時，將該 member 設定為 core。每隔一段時間執行下列步驟。
2. Core 啟動 timer T1 並開始做 probing 的動作，直到 timer T1 的時間截止。
3. Probing node 將 group 中所有 member 的 list 傳送至所有 neighbor node，並要求所有 neighbor node 傳回其 weight 值。
4. 所有被要求的 neighbor node 計算本身的 weight 值，並將 weight 值傳至 probing node。其中計算 weight 的公式，將於稍後詳述。
5. 當 probing node 接收到 neighbor node 傳回的 weight 值後，記錄其中最佳的 n 個 neighbor nodes。
6. 若 probing node 的 weight 值比所有 neighbor nodes 的 weight 值要低，則 goto step 11。
7. 若所有最佳的 neighbor node 皆已在 path list 中，則 goto step 11。
8. Probing node 將自己加入已拜訪過的 node list 中。
9. Probing node 選一個未拜訪過最佳的 neighbor node，作為下個 probing node。
10. 原來的 probing node 將 path list 及 group member list 送到新的 probing node，goto step 3。
11. 最後的 probing node 送出訊息到 core，而 core 即 path list

中的第一個 node，通知 core 有關 probing node 的 weight 值。

12. 將最後的 probing node 設成新的 core，並開始重覆步驟 2。

Tree delay 指的是 tree 中所有 link 的 delay 總和。在計算上述演算法中的 weight 值時，以預估的 tree delay 值作為該 node 的 weight 值。當 tree 為 linear 時，假設每條 link 的 delay 為 1，其 tree delay 等於從 core 到最遠 member 間的 link 數，如＜圖 2.2＞所示。將此 tree delay 設為預估 delay 的最小值，如(6)式所示。

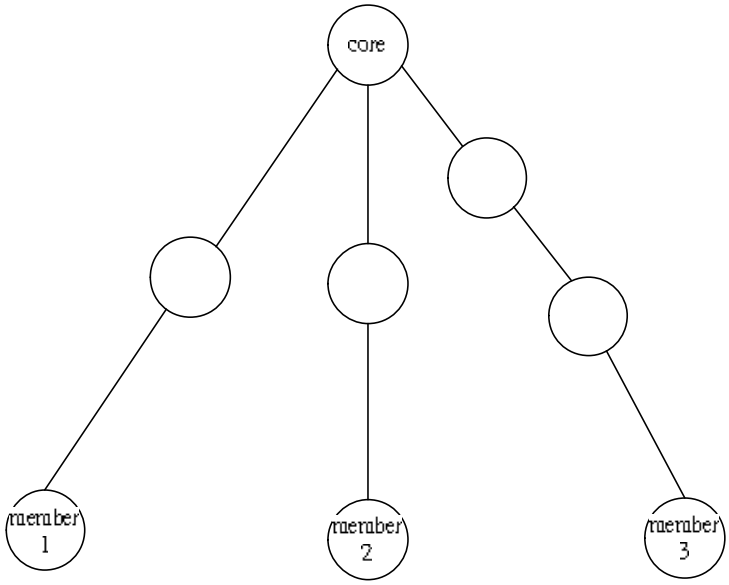


＜圖 2.2＞最小 tree delay 之樹狀圖

$$EstDelay_{\min} = \max \quad d(root,u) \quad ,u \in S \tag{6}$$

其中 $root$ 指的是樹根，而 S 為 group member 的集合(set)。$d(root,u)$ 表從 $root$ 到 u 的 delay。若從 core 到所有的 member 的路徑

彼此間完全沒有 shared link **時，如＜圖** 2.3**＞所示，其** tree delay **等於** core **到所有** member **路徑的** link **數之和。將此** tree delay **設為預估** delay **的最大值，如**(7)**式所示。**



＜圖 2.3**＞最大** tree delay **之樹狀圖**

$$EstDelay_{\max} = \sum_{u \in S} d(root, u) \qquad (7)$$

最後，我們將 $EstDelay_{\min}$ **與** $EstDelay_{\max}$ **取平均，如**(8)**式所示，其平均值為該** node **為** root **時的** wieght **值。**

$$Weight = \frac{EstDelay_{\min} + EstDelay_{\max}}{2} \qquad (8)$$

2.2.6 Routing Table 的建立

一旦 eligible entries 已經確定，我們便可以直接根據 eligible entries 來建立 routing table。因為在傳送 anycast 封包時，必須選擇一個最佳的 member，所以我們必須在 routing table 中增加一個欄位，稱之為 weight。該欄位是用在有多個 eligible entries 時選擇之用。而 weight 值的決定，我們將在下一章中有詳細的描述。另外，我們是以 minimum delay 的欄位來取代傳統的 distance 欄位。

2.3 Packet Forwarding Subprotocol

本章最主要的目的在於，當 router 接收到 anycast 封包時，應如何選擇該封包最佳的傳送路徑或是最佳的目的端。主要可分為三個步驟：

1. 在 routing table 中尋找與接收之 anycast 封包相對應之 entry。
2. 若有多個 entry，則在其中選擇最佳的一個。
3. 根據所選擇的 entry，將 anycast 封包傳送至相對應之 next hop。

2.3.1 Anycast entry 之搜尋

要在 routing table 中搜尋與接收之 anycast 封包對應之 entry 可分為下列兩個步驟：

1. 讀出封包 header 之各個欄位。如：source, destination, flow 等等。
2. 根據步驟 1 所得之欄位來比對 routing table 中之 entry 即可。

在步驟 2 **中要比對** routing table **中的** entry **與該封包是否為** flow **之封包有關。若是一條** flow **的連續封包在傳送時，**router **可以比對** cache **中該** flow **的** entry**，而不需要使用到** routing table **中的** entry**。**

2.3.2 Anycast entry **之選擇**

在找尋到對應的 entry **之後，接下來就是要在所找尋到的多個** entry **之中，選擇其中的一個。首先，我們比較每一個** entry **之** weight **值，並選擇** weight **值最大的** entry **做為我們的傳送目標。**

在此我們希望 traffic **能夠傳送至負載較輕的** member**，來達到** load balance**。另一方面，也希望** traffic **能夠利用延遲時間最短的路徑來傳送，以提高網路的** QoS**。因此** weight **值之計算公式如**(9)**式所示。**

$$W_{R,s,T_i} = \frac{1}{delay(R,s)} \times A_{s,T_i} \times b \qquad (9)$$

其中W_{R,s,T_i} **表示** router R **送到** member S **之** entry **在時間** T_i **時的** weight **值；**$delay(R,s)$ **為從** router R **到** member S **的最小延遲時間；**A_{s,T_i} **為與** member s **負載相關的參數，如**(11)**式所示。其中**L_a **為所有** member **的平均負載，如**(12)**式所示，**L_s **為** member s **的負載，因此若** member s **的負載愈輕，則**A_{s,T_i} **值愈高；反之若** member s **的負載愈重，則**A_{s,T_i} **值愈低。另外，為將**W_{R,s,T_i} **正規化，所以我們再乘上**b **，如** (10)**式所示。**

$$b = \frac{1}{\sum_{s=1}^{n}\left(A_{s,T_i} \times \frac{1}{delay(R,s)}\right)} \qquad (10)$$

$$A_{s,T_i} = A_{s,T_{i-1}} \times \frac{L_a}{L_s} \tag{11}$$

$$L_a = \frac{\sum_{s=1}^{n} L_s}{n} \tag{12}$$

在此我們再討論如何將 member **的負載資訊傳送至** CBT **中。所有** member **先將儲存負載資訊的封包傳送至** core**，當** core **接收到時，再轉傳給所有的** son**，一直重覆此步驟，直到送至** member **的** father **為止。則** on-CBT router **可得到所有** member **的負載量。**

另外，為減少傳送負載資訊所產生的 overhead**，只有當** member **的負載變化超過** threshold **時, 才傳送負載資訊的封包至** core**, 如**(13)**式所示。**

$$\frac{\left| L_{i,T_{k+1}} - L_{i,T_k} \right|}{L_{i,T_k}} \geq Z \tag{13}$$

其中 L_{i,T_k} **為** member i **在時間** T_k **時的負載，為避免** member **負載變動過大時，會傳送出大量的負載資訊封包，在此亦對** threshold Z **作動態的調整，如**(14)**式如示。**

$$Z_i = \begin{cases} Z_{i-1} + Z_{i-1} * b & if \quad V_i \geq V_u; \\ Z_{i-1} - Z_{i-1} * b & if \quad V_i < V_l. \end{cases} \tag{14}$$

其中Z_i為時間在第 i 個 interval 的 threshold，b 為介於 0 至 1 的值，V_i為時間在第 i 個 interval 的 member load 更新頻率，V_u為更新頻率的上限，而V_l則為更新頻率的下限。根據(14)式，我們可以得到當 member load 的更新頻率高於上限時，我們調高 threshold 來降低 member load 的更新頻率；反之，若 member load 的更新頻率低於下限，則調低 threshold 來提高 member load 的更新頻率。

Algorithm of updating group member load:

- Member i

  If $\frac{\left|L_{i,T_{k+1}} - L_{i,T_k}\right|}{L_{i,T_k}} \geq Z$ then send LOAD message to core.

- Core router receives LOAD message

  Forward LOAD message to sons on CBT tree.

- On-CBT router receives LOAD message

  If member is neighbor **then** diskcard LOAD message

  else forward LOAD message to sons.

變數說明：

L_{i,T_k} ：member i **在時間** T_k **時的負載。**

Z ：member **負載的** threshold。

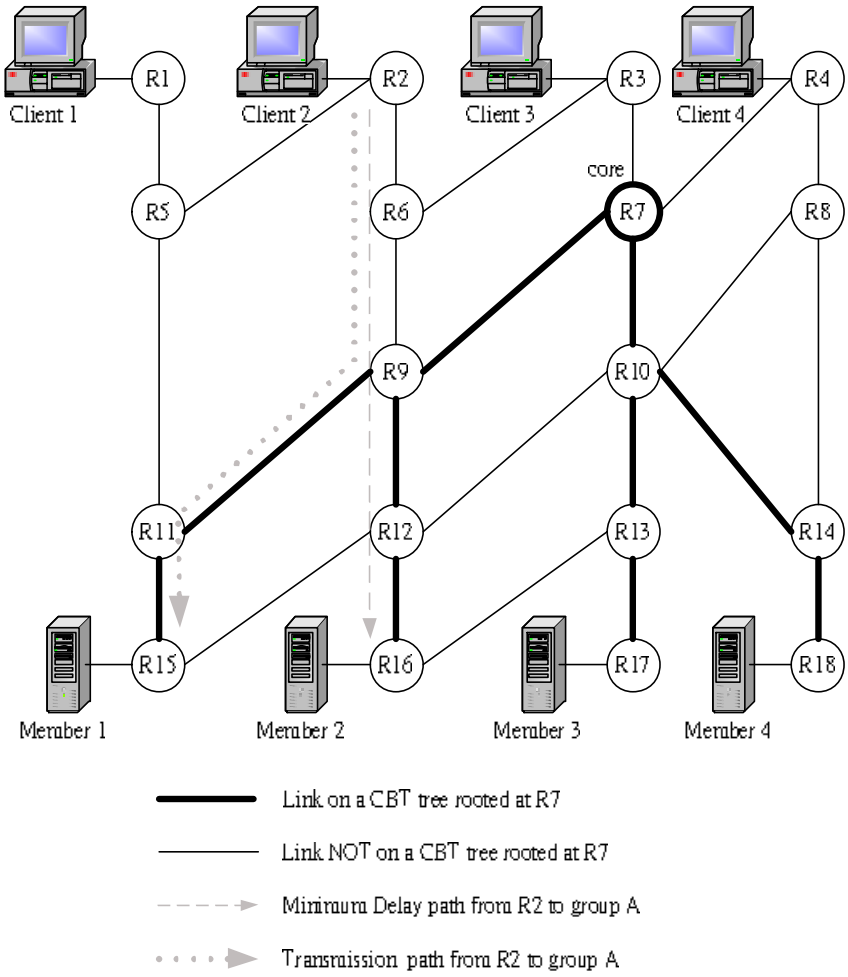
2.3.3 Anycast 封包之傳送

當選擇了一個 entry 之後，接下來只需根據該 entry 內含欄位之資訊，則 router 可以得到 next hop 及 output port 的資訊，再將 anycast 封包由 output port 傳送至所對應之 next hop 即可。

若是 flow 的第一個封包，則在該封包傳送至 next hop 後，將所選到的 entry 儲存在 cache 中，則 flow 的其餘封包再根據 cache 中所儲存之 entry 傳送。

最後我們舉一個簡單的例子來說明我們提出的方法，如＜圖 2.4＞所示。其中 client 1~client 4 為 client 端，R1　R18 為 router，member 1~member 4 為 anycast group A 的所有 member。R7 為 CBT 之 core。當 client 2 要傳送封包到 anycast group A時，會先傳送至相連接之 router，則該封包被送至 R2。在此假設 R2 到 group A 延遲時間最短的路徑為 $R2 \rightarrow R6 \rightarrow R9 \rightarrow R12 \rightarrow R16$。當 R2 接收到封包時，讀取該封包之 destination address，並查 routing table 中 destination address 為 group A 之 entry。因為 R2 是 off-CBT router，因此其 routing table 中 destination address 為 group A 之 entry 只有一項，next hop 為 R6，因此該封包從 R2 傳送至 R6。當 R6 接收到該封包時，執行與 R2 相同的動作，再將封包傳送至 R9。因為 R9 為 on-CBT router，因此當 R9 接收到封包時，根據(9)式中 weight 值的計算，來計算出 R9 所有 eligible entries 的 weight 值，在本例中 R9 共有兩個

eligible entries，分別為 $R9 \rightarrow R11$ 及 $R9 \rightarrow R12$ 兩個 entries。若 $R9 \rightarrow R11$ 此 entry 之 weight 值較高，則該封包會傳送至 R11。而 R11 只有一個 eligible entries，因此該封包會送至 R15，最後 R15 將該封包傳送至 member 1。由上述說明，在此例中 client 2 要傳送封包至 group A 時，最後會傳送到 member 1。



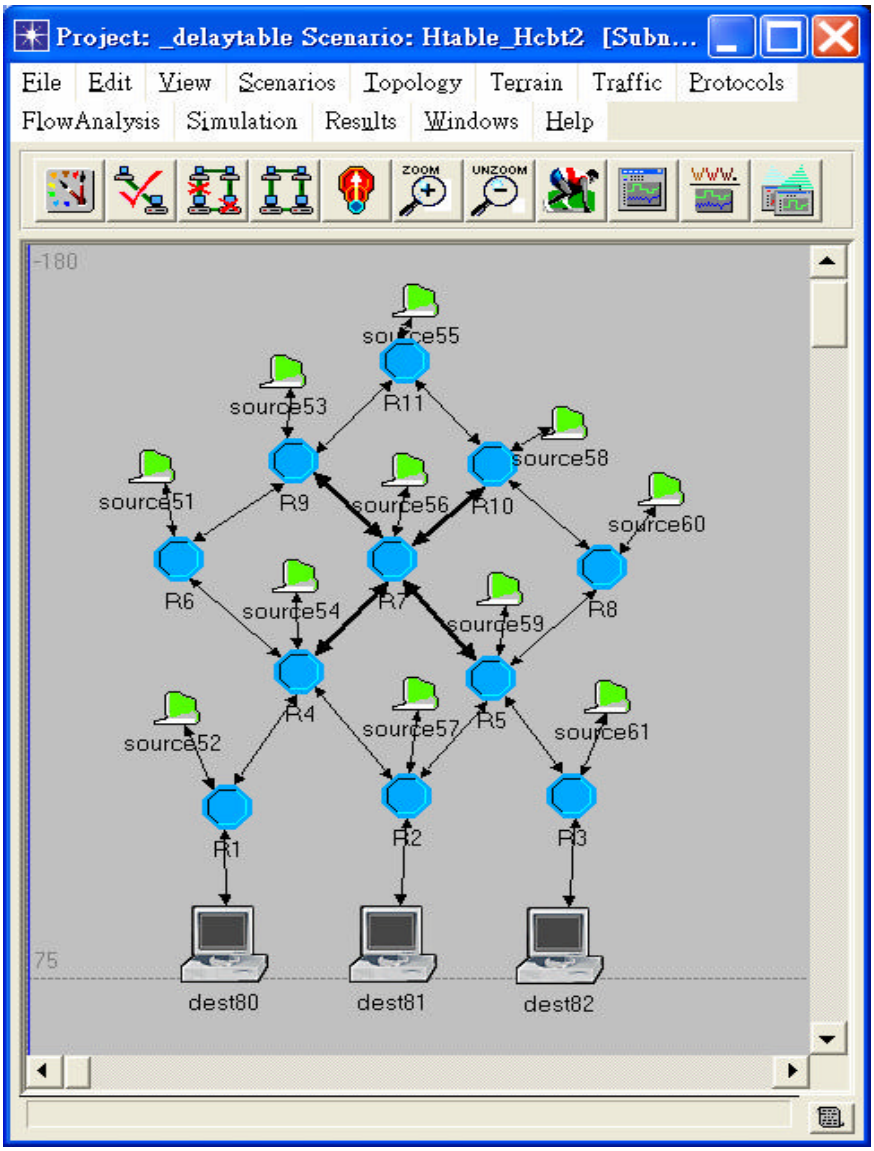
＜圖 2.4＞我們的方法示意圖

第三章　模擬結果

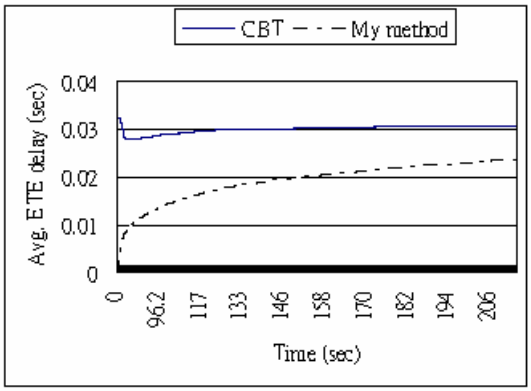
在本章將討論我們提出的方法與[4]中所提的 CBT 相互作比較。模擬時是以 OPNET[11]模擬軟體來執行，實驗共分成六個，其中實驗一與實驗二之網路 topology，如＜圖 3.1＞所示。在實驗一與實驗二的網路中共有 11 個 routers，以 28 條 link 相連接，我們假設 anycast group member 的個數為 3，為＜圖 3.1＞之 dest80~dest82。link 的頻寬分成 1Mbps 及 10Mbps 兩種，＜圖 3.1＞中線條較粗之 link 其頻寬為 10Mbps，而較細 link 其頻寬為 1Mbps，其中 source51~source61 為 client，R1~R11 為 router，dest80~dest82 為 anycast group member。實驗中的 throughput 指的是，單位時間中所有 member 接收到並處理的資料量，因為所有 member 連接的 link 頻寬皆為 1Mbps，因此在本實驗中 throughput 最大值為 3Mbps。

3.1 實驗一

在第一個實驗中，我們設定所有的 anycast group member 的處理能力皆相同，而且在傳送資料前必須先建立 connection，每條 flow 的頻寬為 100Kbps。在實驗時，隨著時間的改變，我們不斷地增加 flow 個數，並比較兩者方法間平均延遲時間的差異。根據＜圖 3.2＞的結果，我們發現我們所提出的方法其平均的延遲時間約為 0.023 秒，而 CBT 方法的平均延遲時間為 0.031 秒，所以我們的方法大約比 CBT 快上 25%左右。



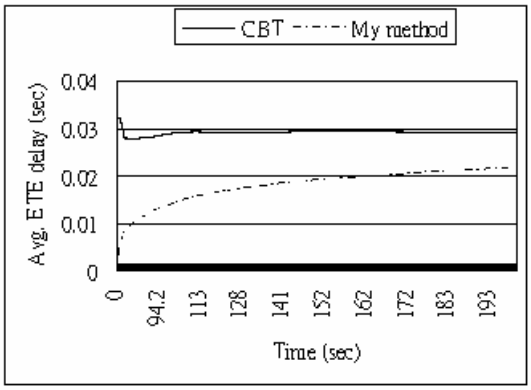
<圖 3.1>Network topology 1



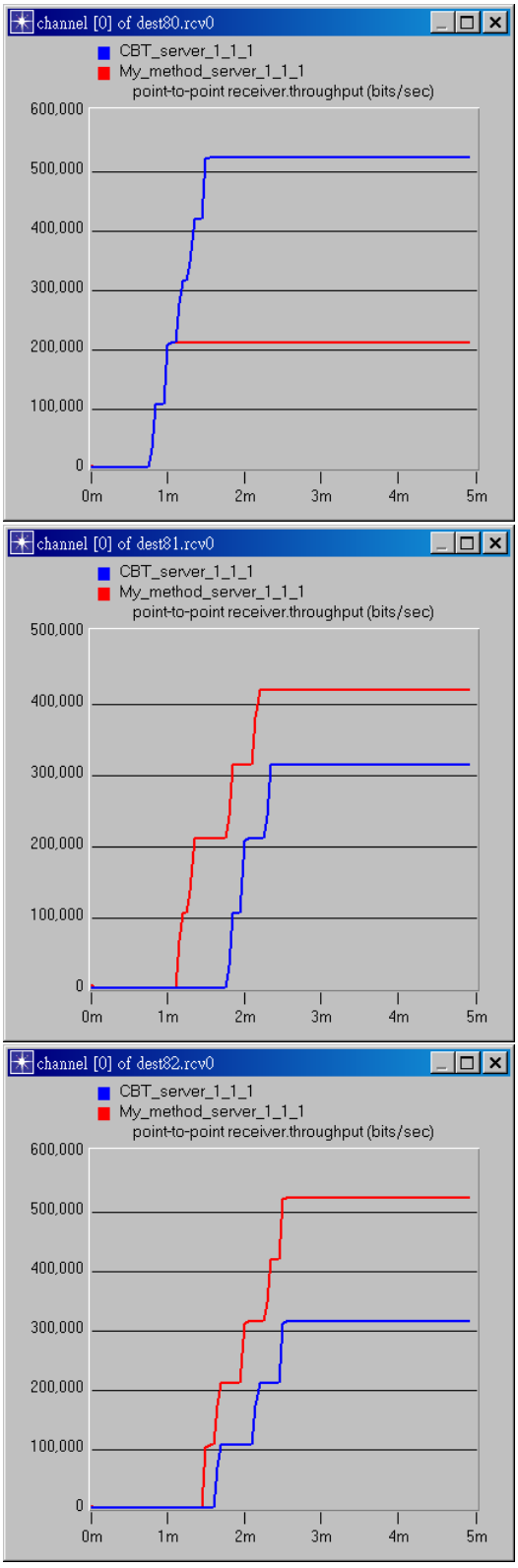
<圖 3.2>Topology 1 之平均延遲時間比較(server 能力相同)

3.2 實驗二

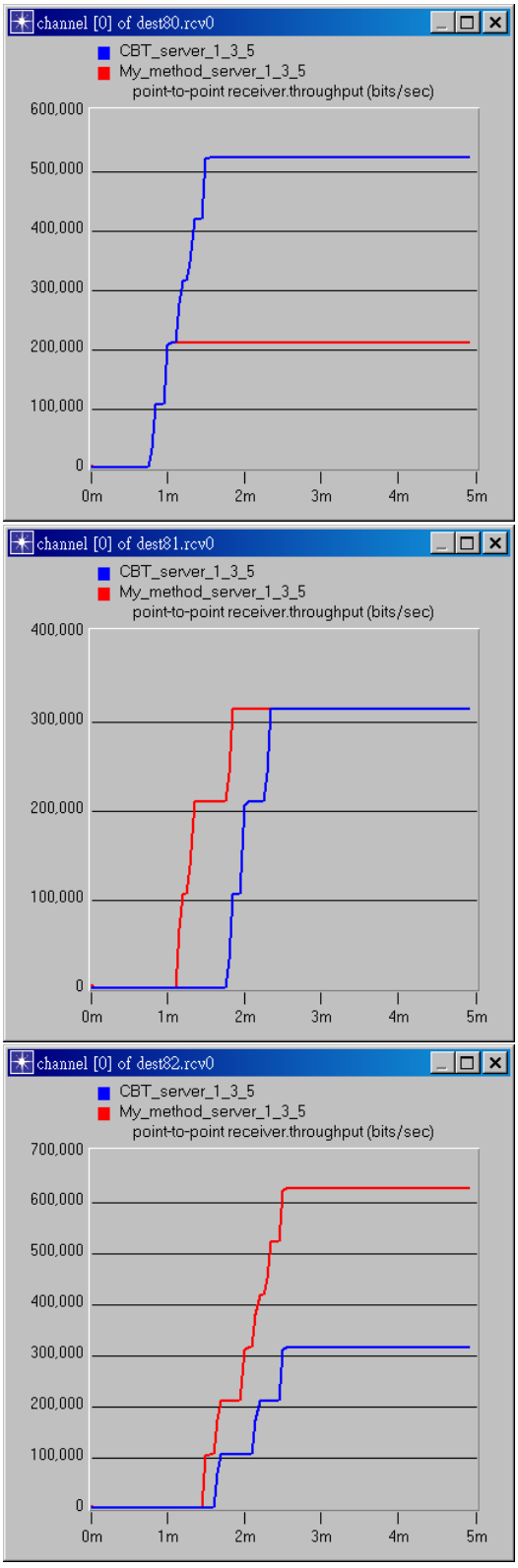
在本實驗中，我們設定所有的 anycast group member 的處理能力皆不相同，dest81 處理能力為 dest80 的 3 倍，dest82 處理能力為 dest80 的 5 倍。在傳送資料前必須先建立 connection，每條 flow 的頻寬為 100Kbps。在實驗時，隨著時間的改變，我們不斷地增加 flow 個數，並比較兩者方法間平均延遲時間的差異。根據＜圖 3.3＞的結果，我們發現我們所提出的方法其平均的延遲時間約為 0.021 秒，而 CBT 方法的平均延遲時間為 0.03 秒，所以我們的方法大約比 CBT 快上 30%左右。由於我們所提出的方法考慮到伺服器的負載，而 CBT 方法並沒有考慮到伺服器的負載，因此我們的方法能夠將 traffic 導到處理能力較高的伺服器，比較＜圖 3.4＞及＜圖 3.5＞中，我們可看出若伺服器 dest82 的處理能力提高為 dest80 的 5 倍時，其所處理的資料則有所增加，代表我們的方法會將更多的資料傳送到處理能力較高的伺服器，因此可有效減低平均的延遲時間。



＜圖 3.3＞Topology 1 之平均延遲時間比較(server 能力不同)



<圖 3.4>Topology 1 之 server throughput 比較(server 能力相同)

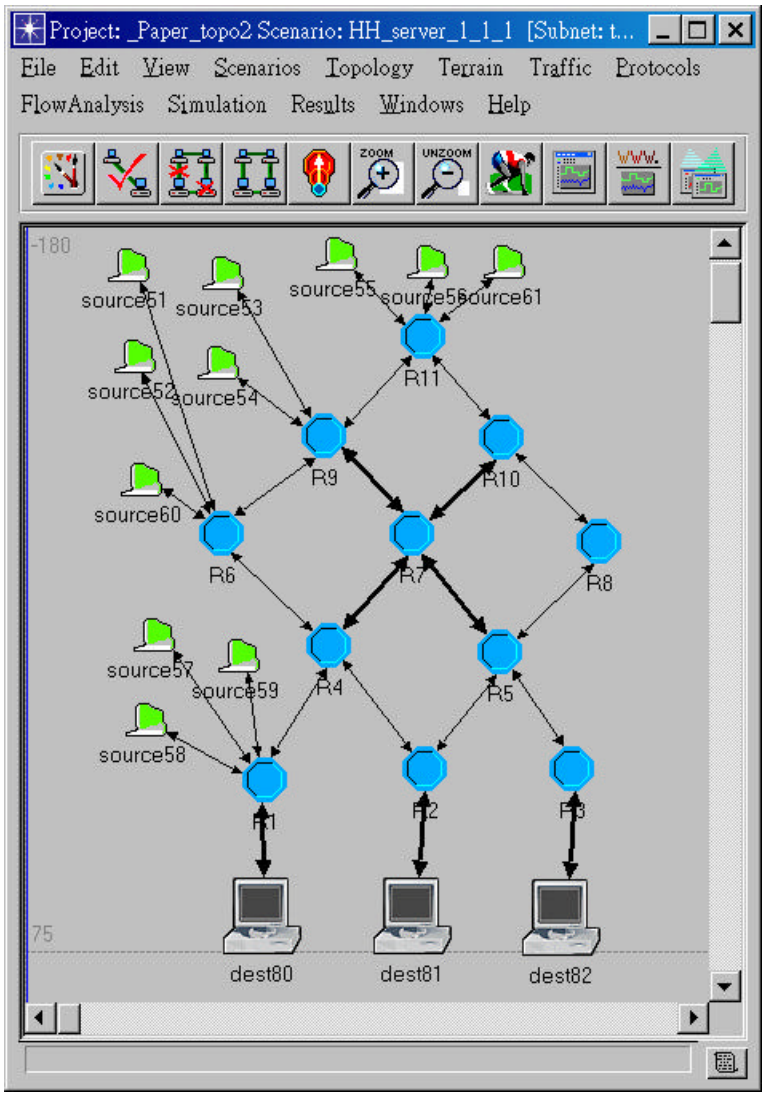


<圖 3.5>Topology 1 之 server throughput 比較(server 能力不同)

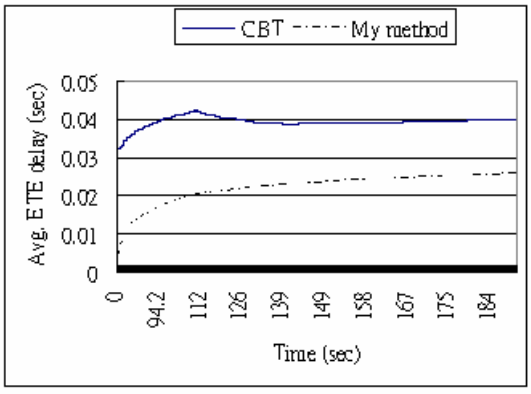
3.3 實驗三

在本實驗中，我們設定所有的 anycast group member 的處理能力皆相同，而且在傳送資料前必須先建立 connection，每條 flow 的頻寬為 100Kbps。在實驗時，隨著時間的改變，我們不斷地增加 flow 個數，並比較兩者方法間平均延遲時間的差異。本實驗的 network topology 與實驗一及實驗二相同，不同的是，我們將 client 的位置改變，假設若 client 集中於某一區域時，來比較 CBT 方法與我們所提出的方法，如＜圖 3.6＞所示。

根據＜圖 3.7＞的結果，我們發現我們所提出的方法其平均的延遲時間約為 0.026 秒，而 CBT 方法的平均延遲時間為 0.04 秒，所以我們的方法大約比 CBT 快上 35%左右。由於 CBT 方法所傳送的資料路徑為固定，而我們所提出的方法會將資料送到延遲時間最短的路徑，因此當 traffic 增加，我們的方法會將資料分散到不同的路徑，可有效減少平均的延遲時間。而 CBT 方法傳送資料的路徑依然不變，因此其平均延遲時間會較高。我們比較實驗一及實驗三的結果可發現，CBT 的平均延遲時間從 0.031 秒提高到 0.04 秒，增加了約 0.009 秒；而我們的方法是從 0.023 秒提高到 0.026 秒，只增加了 0.003 秒。



＜圖 3.6＞Network topology 2



＜圖 3.7＞Topology 2 之平均延遲時間比較(server 能力相同)

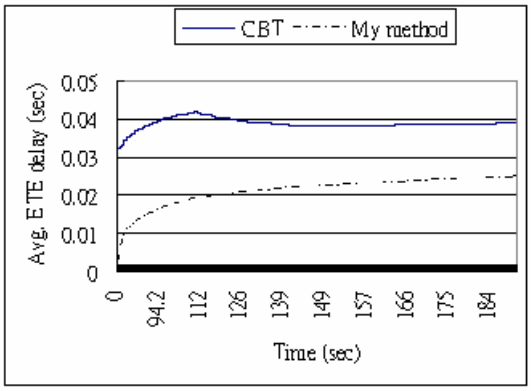
3.4 實驗四

在本實驗中，我們設定所有的 anycast group member 的處理能力皆不相同，dest81 處理能力為 dest80 的 3 倍，dest82 處理能力為 dest80 的 5 倍。在傳送資料前必須先建立 connection，每條 flow 的頻寬為 100Kbps。在實驗時，隨著時間的改變，我們不斷地增加 flow 個數，並比較兩者方法間平均延遲時間的差異。本實驗的 network topology 與實驗三相同，將 client 集中於某一區域，來比較 CBT 方法與我們所提出的方法，如＜圖 3.6＞所示。

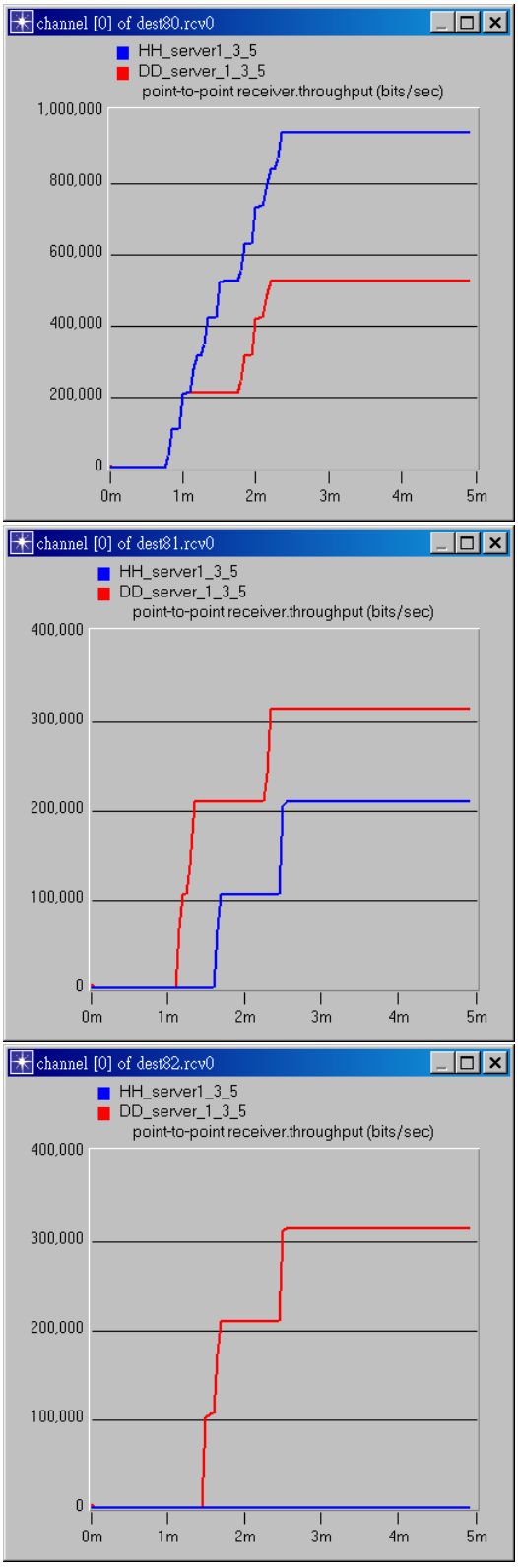
根據＜圖 3.8＞的結果，我們發現我們所提出的方法其平均的延遲時間約為 0.024 秒，而 CBT 方法的平均延遲時間為 0.039 秒，所以我們的方法大約比 CBT 快上 38%左右。由於 CBT 方法所傳送的資料路徑為固定，且並沒有考慮到伺服器的負載，而我們所提出的方法會將資料送到延遲時間最短的路徑，因此當 traffic 增加，我們的方法會將資料分散到不同的路徑，可有效減少平均的延遲時間。而 CBT 方法傳送資料的路徑依然不變，因此其平均延遲時間會較高。我們比較實驗二及實驗四的結果可發現，CBT 的平均延遲時間從 0.03 秒提高到 0.039 秒，增加了約 0.009 秒；而我們的方法是從 0.021 秒提高到 0.024 秒，只增加了 0.003 秒。

由於我們所提出的方法考慮到伺服器的負載，而 CBT 方法並沒有考慮到伺服器的負載，因此我們的方法能夠將 traffic 導到處理能力較高的伺服器，比較＜圖 3.9＞中，我們可看出若伺服器 dest82 的處理能力提高為 dest80 的 5 倍時，其所處理的資料則有所增加，代表我們的方法會將更多的資料傳送到處理能力較高的伺服器，因此可有

效減低平均的延遲時間。



＜圖 3.8＞Topology 2 之平均延遲時間比較(server 能力不同)

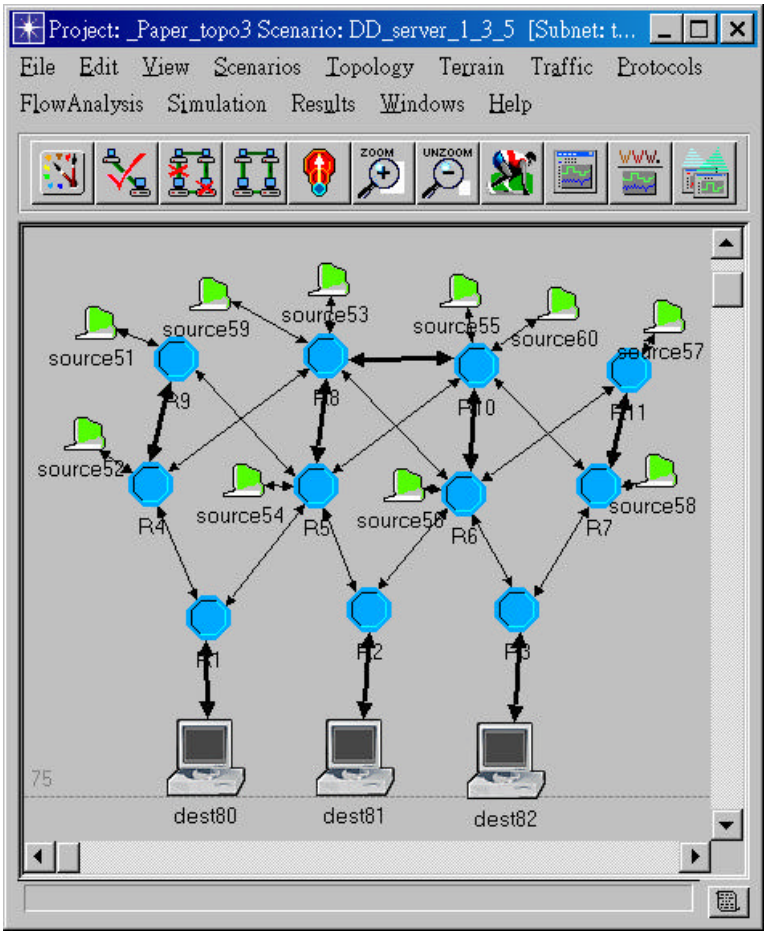


<圖 3.9>Topology 2 之 server throughput 比較(server 能力不同)

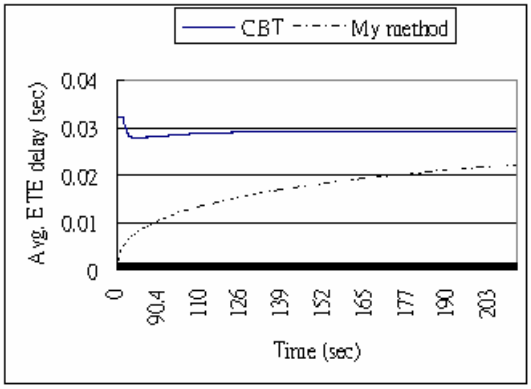
3.5 實驗五

在本實驗中，我們設定所有的 anycast group member 的處理能力皆相同，而且在傳送資料前必須先建立 connection，每條 flow 的頻寬為 100Kbps。在實驗時，隨著時間的改變，我們不斷地增加 flow 個數，並比較兩者方法間平均延遲時間的差異。本實驗的 network topology，如＜圖 3.10＞所示。

根據＜圖 3.11＞的結果，我們發現我們所提出的方法其平均的延遲時間約為 0.022 秒，而 CBT 方法的平均延遲時間為 0.03 秒，所以我們的方法大約比 CBT 快上 26%左右。由於 CBT 方法所傳送的資料路徑為固定，而我們所提出的方法會將資料送到延遲時間最短的路徑，因此當 traffic 增加，我們的方法會將資料分散到不同的路徑，可有效減少平均的延遲時間。而 CBT 方法傳送資料的路徑依然不變，因此其平均延遲時間會較高。



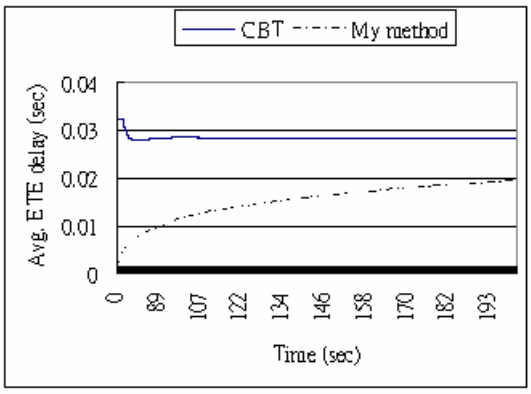
<圖 3.10>Network topology 3



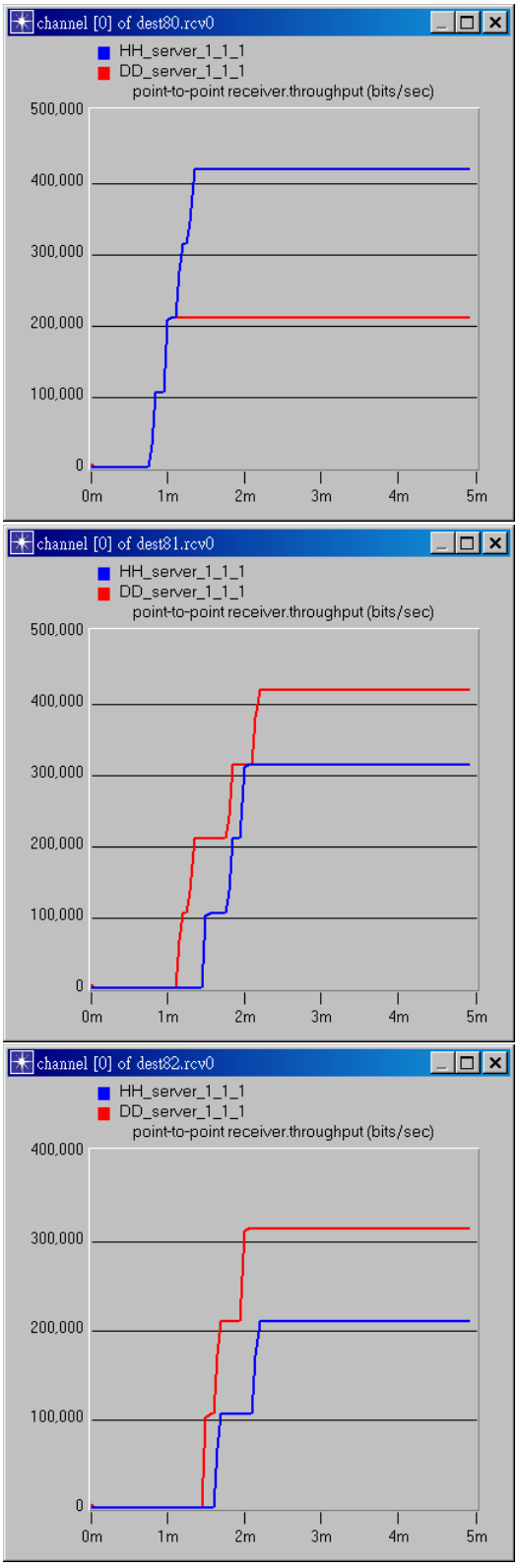
<圖 3.11>Topology 3 之平均延遲時間比較(server 能力相同)

3.6 實驗六

在本實驗中，我們設定所有的 anycast group member 的處理能力皆不相同，dest81 處理能力為 dest80 的 3 倍，dest82 處理能力為 dest80 的 5 倍。在傳送資料前必須先建立 connection，每條 flow 的頻寬為 100Kbps。在實驗時，隨著時間的改變，我們不斷地增加 flow 個數，並比較兩者方法間平均延遲時間的差異。根據＜圖 3.12＞的結果，我們發現我們所提出的方法其平均的延遲時間約為 0.02 秒，而 CBT 方法的平均延遲時間為 0.028 秒，所以我們的方法大約比 CBT 快上 29%左右。由於我們所提出的方法考慮到伺服器的負載，而 CBT 方法並沒有考慮到伺服器的負載，因此我們的方法能夠將 traffic 導到處理能力較高的伺服器，比較＜圖 3.13＞及＜圖 3.14＞中，我們可看出若伺服器 dest82 的處理能力提高為 dest80 的 5 倍時，其所處理的資料則有所增加，代表我們的方法會將更多的資料傳送到處理能力較高的伺服器，因此可有效減低平均的延遲時間。

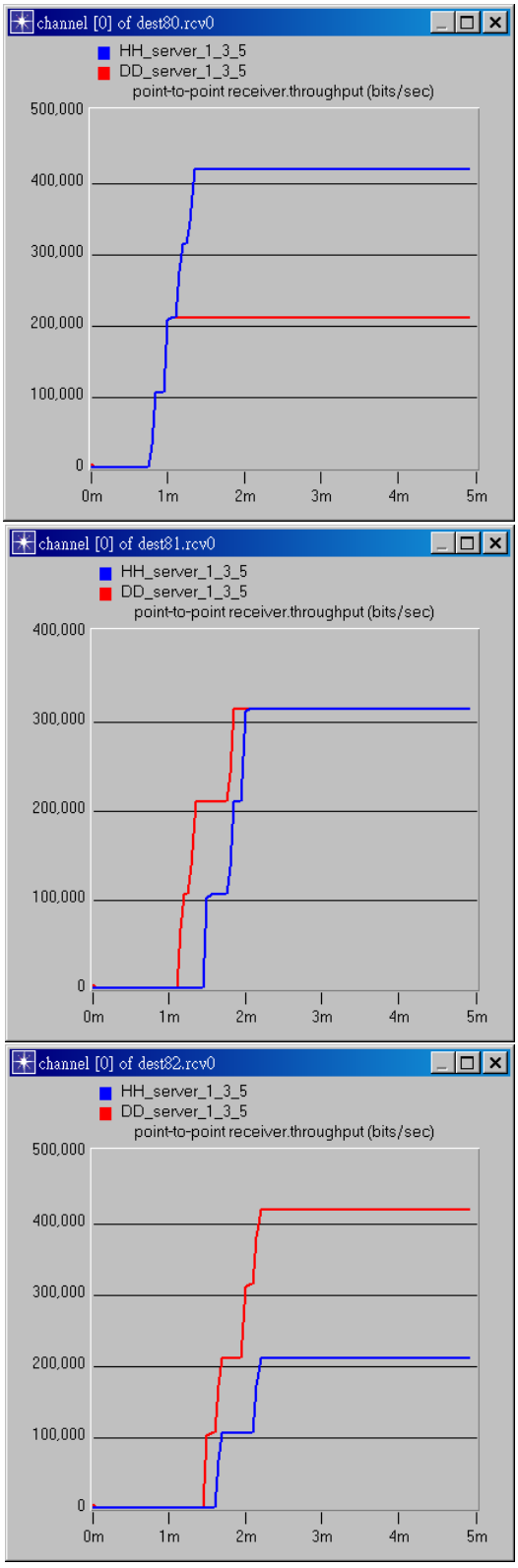


＜圖 3.12＞Topology 3 之平均延遲時間比較(server 能力不同)



<圖 3.13>Topology 3 之 server throughput 比較

(server 能力相同)



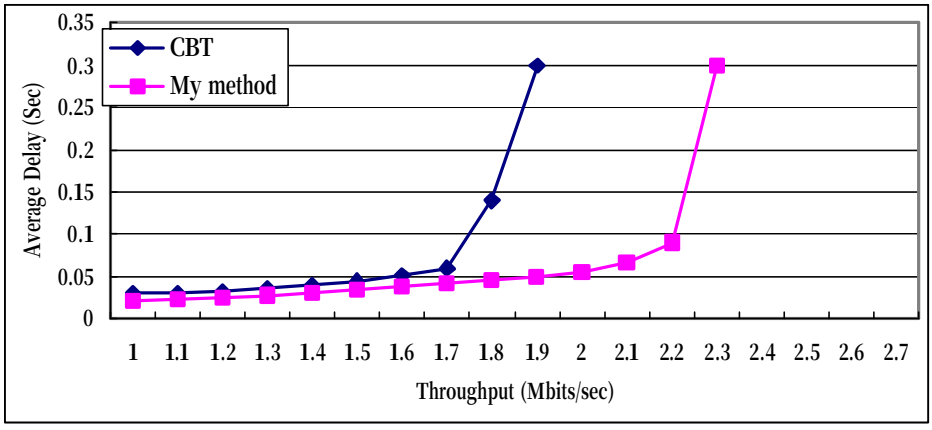
<圖 3.14>Topology 3 之 server throughput 比較

(server 能力不同)

3.7 實驗七

在本實驗中，我們不斷地增加每個 client 所傳送的 traffic，並比較兩方法間平均延遲時間的差異。根據＜圖 3.15＞的結果發現：在 traffic 較低時，我們所提出的方法平均延遲時間約為 0.02 秒，而 CBT 方法約為 0.03 秒。隨著 traffic 的增加，平均延遲時間也會慢慢增加，當 throughput 為 1.9Mbps 時，CBT 方法的平均延遲時間此時會快速的增加到約 0.3 秒左右。而我們所提出的方法，throughput 大約到 2.3Mbps 時，平均延遲時才增加到 0.3 秒。

由上述實驗之結果得知，當網路之 traffic 較低時，因為我們提出的方法是依照最短延遲時間之路徑來傳送資料，因此會較 CBT 中所使用之 distance 最短路徑有較低的 end-to-end delay。而當 traffic 增加，CBT 因為在 off-CBT router 中使用 Shortest-Shortest Path (SSP)，因此較容易產生 congested。而我們提出的方法中，若某條路徑 congested，則會找尋其他延遲時間較短的路徑來取代之，因此可以獲得較高的 throughput。



＜圖 3.15＞Topology1 之 throughput 比較

由以上的七個實驗中，我們共比較了三種不同的 network topology 及不同伺服器的處理能力，比較結果可以明顯的看出我們所提出的方法不論是平均延遲時間及 throughput 均優於 CBT 方法。

第四章　結論

在本論文中，我們提出了一個方法，最短延遲時間的 anycast routing protocol，試圖改善目前現有 anycast routing protocol 的一些缺點，包括了缺乏 anycast server 的負載情況，以及最短延遲時間路徑的使用等。根據實驗結果，我們可得知我們的方法不論在延遲時間的減少以及 throughput 的增加方面，皆較現有方法有所改善。

參考文獻

[1] E.W. Zegura, M.H. Ammar, Z. Fei, and S. Bhattacharjee,``Application-layer anycasting: a server selection architecture and use in a replicated Web service,'' IEEE Trans. Networking., vol.~8, no.~4, pp.~455-466, Aug. 2000.

[2] W. Jia, G. Xu and W. Zhao, ``Integrated fault-tolerant multicast and anycast routing algorithms,''Proc. of IEE'00, vol.~147, no.~4, pp.~266-274 , Jul. 2000.

[3] D. Xu, and W. Jia,``Distributed admission control for anycast flows with QoS requirement,'' Proc. of Distributed Computing System., vol.~8, no.~4, pp.~292-299, Apr. 2001.

[4] D. Xuan, W. Jia, W. Zhao, and H. Zhu, ``A routing protocol for anycast messages,'' IEEE Trans. Parallel and Distributed Systems., vol.~11, no.~6, pp.~571-588, Jun. 2000.

[5] C. Partridge, T. Mendez, and W. Milliken, Host Anycasting Service, RFC1546, IETF and IESG, Nov. 1993.

[6] S. Deering and R. Hinden, Internet Protocol Version 6 (IPv6) Specification, RFC 2460, IETF and IESG, Dec. 1998.

[7] H. Bettahar, A. Bouabdallah, ``A new approach for delay-constrained routing,'' Computer Communications, vol.~25, pp.~1751-1764 , Dec. 2002.

[8] M. Yamamoto, H. Miura, K. Nishimura and H. Ikeda, ``A network-supported server load balancing method: active anycast,'' IEICE Trans. Communications., vol.~E84-B, no.~6, pp.~1561-1568, Jun. 2001.

[9] D.G. Thaler and C.V. Ravishankar, ``Distributed center-location algorithms,'' IEEE Trans. Selected Areas in

Communications., vol.~15, no.~3, pp.~291-303, Apr. 1997.

[10] A. Ariza, E. Casilari and F. Sandoval, ``Strategies for updating link states in QoS router,'' Electronics Letter., vol.~36, no.~20, pp.~1749-1750, Sep. 2000.

[11] OPNET Technologies, Inc., OPNET Modeler 7.0.B,Bethesda, MD, Jun. 2000.

[12] K. Mase, A. Tsuno, Y. Toyama, and N. Karasawa,``A Web server selection algorithm using QoS measurement,''Proc. of ICC'01, vol.~8,pp.~2332-2336, Helsinki, Finland, Jun. 2001.

[13] R.L. Carter and M.E. Crovella,``Server selection using dynamic path characterization in wide-area networks,''Proc. of INFOCOM'97, vol.~3, pp.~1014-1021, Kobe, Japan, Apr. 1997.