國立交通大學

網路工程研究所

碩 士 論 文

同儕網路串流系統中頻寬與延遲感知的傳輸速率分配機制

Bandwidth and Latency Aware Send Rate Allocation in P2P Streaming System

研 究 生：黃銘祥

指導教授：曾建超　教授

中 華 民 國　九 十 九　年 八 月

同儕網路串流系統中頻寬與延遲感知的傳輸速率分配機制

Bandwidth and Latency Aware Send Rate Allocation in P2P Streaming System

研 究 生：黃銘祥　　Student：Ming-Hsiang Huang

指導教授：曾建超　　Advisor：Chien-Chao Tseng

國 立 交 通 大 學

網 路 工 程 研 究 所

碩 士 論 文

A Thesis

Submitted to Institute of Network Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

August 2010

Hsinchu, Taiwan, Republic of China

中華民國九十九年八月

同儕網路串流系統中頻寬與延遲感知的傳輸速率分配機制

研究生： 黃銘祥　　指導教授： 曾建超 教授

國立交通大學網路工程研究所

摘　要

本論文針對網狀同儕網路串流系統(Mesh P2P streaming system)，提出一套根據各個節點(Node)在系統中的貢獻來分配傳輸速率的機制，以增加每個節點的頻寬利用率、減少影像傳輸延遲以及達成高貢獻獲得高接收率的回饋目標。本論文提出的方法中，每個提供者(Provider)會先根據各個接收者(Receiver)的上傳頻寬、提供者到該接收者的連線延遲(Link latency)，以及該接收者所要求資料的延遲程度，計算該接收者對系統的貢獻，再依據不同接收者的貢獻調整壅塞控制(Congestion control)的參數，使得貢獻大的接收者在競爭頻寬時能夠從提供者分配到較多的傳輸速率。

已有研究指出，在同儕網路串流系統的各種架構中，網狀架構(Mesh)的同儕網路串流系統在各方面表現都是最好。但是網狀同儕網路系統只依靠各個節點互相隨機連線，沒有考慮各節點能對系統做出的貢獻，因此造成了以下問題：第一，大頻寬的節點不一定能收到較多資料，以致大頻寬的節點沒有足夠資料可傳輸，因而無法充分利用頻寬。第二，資料未先傳給大頻寬或離提供者較近的節點，會導致需要更長的路徑和更久的時間才能散布到整個系統，因而增加了播放延遲(Playout delay)。第三，未考慮先傳送較新的資料，導致系統中傳播的資料延遲增加，第四，上傳較多資料的節點不一定能下載較多資料，也產生貢獻回饋失衡的問題。

先前雖然有一些研究嘗試解決這些問題，但是它們通常只考慮單一條件，例如只考慮接收者的頻寬或是資料的延遲程度來決定傳輸的優先權，而沒有研究是將接收者頻寬、連線延遲和資料延遲程度一起綜合考慮。此外，先前的研究都沒有考慮節點間的頻寬對傳輸速率分配的影響，只是簡單的測量自己的頻寬，再根

據自己的頻寬，分配傳輸速率給其他節點。亦即，沒有考慮不同節點間頻寬可能不同，只是很理想化的假設，每個節點的下載頻寬都足夠承受來自任何節點的資料，這樣理想化的假設並不能套用在真正的網路中。

因此，本論文針對網狀同儕網路串流系統提出一套根據各節點貢獻來分配頻寬的機制，本論文提出的機制包含以下兩部分：(一)節點貢獻估算法:根據每個節點的上傳頻寬、連線延遲以及該節點要求的資料延遲程度，來評估該節點貢獻的計算方法。以及，(二)壅塞控制機制:探索提供者跟接收者間的頻寬，當提供者到兩個接收者的路徑有共同的壅塞連接(Shared congested link)時，貢獻大的接收者永遠能夠競爭到較多的傳輸速率。

爲了驗證本論文提出的方法，我們使用了 NS2 模擬器進行模擬，並與 Coolstreaming 以及 Prime 作比較，實驗結果顯示本論文提出的方法在大部分環境表現都比 Coolstreaming 和 Prime 好。

關鍵詞：同儕網路串流系統、網狀架構、壅塞控制、傳輸速率分配

Bandwidth and Latency Aware Send Rate Allocation in P2P Streaming System

Student: Ming-Hsiang Huang Advisor: Dr. Chien-Chao Tseng

Institute of Network Engineering
College of Computer Science
National Chiao Tung University

Abstract

In this thesis, we proposed a send rate allocation mechanism in mesh P2P streaming system that can increase the bandwidth utilizations, reduce the playout delays, and augment the rewards of nodes' contributions in the system. The underlying idea of the proposed mechanism is that a provider allocates its send rate to a receiver according to the contribution of the receiver. A provider will first estimate the contribution of each of its receivers according to the receiver's upload bandwidth, the link delay to the receiver, and the delay of the contents requested by the receiver. Then, the provider will adjust the congestion control parameters of each receiver according to the contribution of the receiver, and thus allocate higher send rates to the receivers with larger contributions.

Previous research shows that mesh architectures outperforms others in P2P streaming. However peers in previous mesh P2P streaming systems just randomly connects to and requests data from one another, without considering how much contribution a peer can make for the system. The neglect of peers' contributions in P2P streaming may cause the following problems: first, peers with large bandwidth may not have enough data, and thus can not utilize their bandwidth to transmit data to more peers. Second, providers do not send data to those peers with high bandwidths or low link delays from the providers, so that it takes more time, and possibly longer paths, to disseminate data to all peers across the P2P streaming

system. Third, providers do not send new data with higher priorities, and thus more old data swarm in the system. Fourth, peers that contribute more upload bandwidth do not receive comparable rewards.

Some researchers have tried to resolve some of the above problems. However they consider either peer upload bandwidths or content delays only, but no both at the same time. Furthermore, the upload bandwidth they considered is the bandwidth measured by a provider. However, the bandwidth between a provider and a receiver depends on the routing path between the two peers, not just the upload bandwidth of the provider. In addition, the link latency between a provider and a receiver also contributes the playout delays. Therefore, in our design, the contribution of a receiver peer depends on three factors, that is, the upload bandwidth of the peer, the link latency between the receiver and its provider, and the delay of the content requested by the receiver.

The send rate allocation mechanism we propose contains two parts: a contribution estimation method and a congestion control mechanism. In our design, each peer executes the contribution estimation method to determine the contributions of its neighbors periodically when it receives the information of the neighbors. A provider then uses the congestion control mechanism to allocate send rates for requesting peers in accordance of the contribution of the peers. With the congestion control, a provider will increase the send rate to a peer when the provider receives all acknowledgements from the peer, and decrease send rate when the provider detects packet losses. However, a peer with a larger contribution will increase more but decrease less than the peers with smaller contributions. As a consequence, the congestion control mechanism can explore the available bandwidth among peers, and when the paths from a provider to several peers share a congested link, the provider will allocate more send rates to the peers with larger contributions.

We conduct simulations with NS2 simulator to evaluate our send rate allocation proposal. The simulation results show that the congestion control can indeed allocate more bandwidth to the peers that contribute more. Furthermore, we

compare our allocation method outperforms with Coolstreaming and Prime in terms of the average peer receive rates and the source-to-peer transmission delays. The results show that in most environments, peers in our proposal can have higher receive rates with lower source-to-peer transmission delays compared with the ones in Coolstreaming and Prime.

Keywords: P2P streaming system, mesh structure, congestion control, send rate allocation

誌　謝

我要感謝許多人的幫助，讓我能夠完成這篇論文。首先我要感謝我的指導教授曾建超老師，不斷的提醒我論文該注意的地方，以及研究中常犯的錯誤。感謝各位口試委員寶貴的建議，讓我看到了做研究時未曾注意到的部分。

我要感謝協助我完成論文的博班學長姐。感謝王豪，雖然我的想法變來變去，還是每個禮拜都花時間和我討論。感謝學姊在內外務一大堆的時候，還抽空幫我修改論文。感謝承遠幫我減輕了 D-Link 的許多工作，讓我可以更專心的爲畢業努力。

我要感謝實驗室的各位學長，同學和學弟妹們。感謝承運在我拼命寫模擬的時候常常到實驗室來陪我，感謝貴笠和 JJ 在我獨自奮鬥時給我的鼓勵，感謝大樑哥爲管理實驗室所做的努力。感謝竣晨，蛋捲和坤穎，跑出去玩的時候還沒有把我給忘記，感謝小毛總是用一貫的熱情來對待所有人，感謝阿盧一直讓我們有新話題可以聊。感謝碩一的各位學弟妹們，明辰、阿彪、貞慧、沅春、圈圈和家明，很高興認識你們。

最後，我要感謝我的高中同學偲帆，感謝你贊助的強大 Computing power，讓我的模擬能夠以外星人達不到的速度飛快的跑完。

目　錄

圖目錄

表目錄

第一章　緒論

1.1　研究動機

同儕網路是近年來興起的網路分享模式，最大的特色是每個使用者除了下載資料之外，同時也會上傳資料給其他使用者。由於每個使用者都會貢獻自己的上傳頻寬，因此最初的分享者並不需要具備非常大的上傳頻寬，就可以分享資料給大量的使用者，這種優點使得同儕網路被廣泛的應用於檔案分享或多媒體串流等大資料量的分享上。

同儕網路也被應用於即時的多媒體串流，例如運動賽事的即時轉播等用途。但，相較於檔案分享，即時多媒體串流傳輸的資料是有時間性的，若影像資料未能在播放時間(Playout time)之前送達，就會造成影像播放的延遲(delay)或跳躍(skip)，因而降低影像的品質。所以為了使影像資料能夠在播放時間之前送達，同儕網路串流系統於是發展出了有別於檔案分享系統的架構和排程方法。

先前研究所提出的同儕網路串流系統架構大概可以分為三類[1]，分別為樹狀架構、多樹架構，以及網狀架構。樹狀架構中，所有節點(node)會互相連線形成一個樹狀結構，來源節點位於樹的根部(root)，並將串流經由樹狀結構散布到所有節點。多樹架構中，節點會形成多個樹狀架構，影像被分割成數個子串流(sub-stream)並各自經由不同的樹散播，且每個節點會根據自己想收到的影像品質決定要加入多少棵樹。網狀架構中，每個節點互相隨機連線，並且互相交換自己尚未收到的影像片段。先前也有研究對同儕網路串流系統建模並指出了理論上的效能上限[2]。

雖然先前已有研究指出網狀架構在各方面表現都是最好[3]，但是網狀架構仍然有許多能夠改進的地方。在網狀架構中，每個節點都只考慮如何增進自己的利益，而沒有考慮如何增進整體系統的效能，因此造成了低頻寬利用率、高影像延遲，以及速率分配的不公平等問題。先前雖然已經有許多針對這些問題的研究，但是它們通常只考慮單一的條件，例如只考慮節點的上傳頻寬或只考慮資料的延遲程度來決定傳輸的優先權，而沒有一種方法將節點的上傳頻寬、連線延遲，以及資料的延遲程度一起綜合考慮。

另外一個問題是，先前研究都沒有深入探討速率分配的方法。爲了增進系統效能，根據不同節點的能力來分配傳輸速率是很直覺的方法，但是由於節點之間的頻寬會受到路由路徑上連接的頻寬影響，因此提供者無法預先得知自己可以分配多少頻寬給接收者，甚至可能出現多個接收者只能有一兩個能夠收到完整的速率的情況。但是先前研究都只是假設兩個節點間的頻寬只會受限於提供者的上傳頻寬，所以就直接測量節點的上傳頻寬，再根據接收者的能力把上傳頻寬分給不同接收者，而沒有去考慮更深入的問題。

由於網狀同儕網路串流系統仍然存在以上所述的問題，因此本論文希望能夠改善解決這些問題。

1.2 研究目的

本論文的研究目的即是爲了解決上一節所述的兩個問題。首先，本論文會根據節點的上傳頻寬、連線延遲，以及它所要求的資料的延遲程度來估計該節點能對系統做出的貢獻，接著本論文會提出一套傳輸速率分配的機制，能夠充分利用節點間的頻寬，且當提供者到多個接收者的路由路徑上出現共同的頻寬瓶頸時，貢獻較大的接收者自動會接收到較高的速率。

1.3 章節簡介

本論文的節編排與內容簡介如下：

第一章：緒論，簡介本論文的研究動機與預期達到的目標。

第二章：背景知識介紹，簡介本論文的主題與使用到的相關知識，包括各種同儕網路架構的介紹以及該架構所面臨的問題。

第三章：相關研究，簡介之前發表過的有關本論文主題的研究資料。

第四章：說明本論文所提出方法的運作原理和流程。

第五章：效能分析與結果，針對本論文提出的方法，在不同的環境下進行模擬，並分析其效能。

第六章：結論與未來工作，總結本論文研究結果，和未來可繼續研究的方向。

第二章　背景知識介紹

2.1　同儕網路串流系統

同儕網路串流系統的特色是每個使用者都必須貢獻自己的上傳頻寬，所以一開始的分享者不需要很大的頻寬就可以將影像分享給大量使用者，因此具有建置成本低的優點。但是相對的，同儕網路串流系統必須處理許多節點間的訊息與資料交換，必須能夠容忍不斷有使用者加入和離開，還必須考慮到影像傳輸的時間性問題。因此，先前研究也發展出了不同的架構來處理這些問題，不同架構也都各自有它的優點和缺點，這些都會在本章加以介紹。

但是不論是何種架構，基本的組成都是差不多的。一個同儕網路串流系統會包含一個或多個來源節點(Source node)，一個啓動伺服器(Bootstrap server)，以及許多一般使用者的節點。來源節點是最初的影像資料提供者，而啓動伺服器負責提供一些已在系統中的節點位址給新加入的節點。每個節點都會有一部分其它節點的名單，也會不斷的和其他節點交換訊息，至於交換的對象以及訊息就因架構而有所不同。互相交換訊息的節點稱做鄰居(Neighbor/Partner)，而當一個節點實際開始提供資料給另一個節點時，它們的關係就成爲提供者和接收者。

以下就開始介紹不同的同儕網路串流系統的架構，以及它們的優缺點。

2.2　同儕網路串流系統的架構

目前的同儕網路串流系統主要有樹狀架構、多樹架構，以及網狀架構幾種[1]，以下分別對這幾種架構做介紹。

2.2.1　樹狀同儕網路串流系統

樹狀架構是網路廣播最直覺的架構，因此也被應用在同儕網路串流中。在樹狀架構中，所有節點會互相連接形成一個樹狀的結構，來源節點位於樹的根部，並將影像串流經由樹狀結構散布到所有節點，如圖 1。而由於使用者會不斷加入和離開，所以先前關於樹狀架構同儕網路串流系統的研究[4]主要的焦點都是如何在這種情況下維持樹狀的結構。

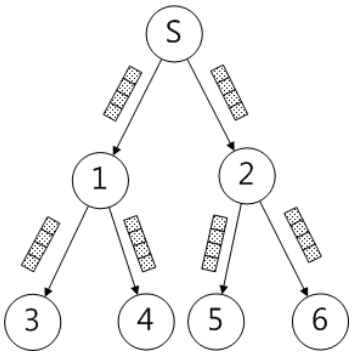


圖 1　樹狀架構同儕網路串流系統示意圖

在樹狀架構中，一個節點收到資料後就馬上傳給在它之下的節點，因此具有影像延遲小的優點，但是樹狀架構仍然有以下的缺點：第一，當一個節點離開時，在它底下的所有節點都會因爲失去資料來源因而造成影像中斷，因此不適用於節點加入和離開很頻繁的環境。第二，末端節點(Leaf node)由於不會傳資料給其他人，因此它們的上傳頻寬無法被充分利用。

2.2.2　多樹同儕網路串流系統

爲了改善樹狀架構的缺點，於是之後又發展出了多樹架構，多樹架構的特點是影像被分割成數個子串流，所有的節點形成多個樹狀結構，且每個子串流都經由不同的樹狀結構傳送，如圖 2。由於在一棵樹的末端節點會在其他棵樹是中間節點(Internal node)，因此解決了樹狀架構無法利用末端節點頻寬的問題。此外，影像使用 Multiple description coding (MDC)[5]分割成數個子串流，所以遺失少數的子串流並不會對影像品質造成太大的影響，也使多樹架構比較能適應使用者頻繁上下線的環境。另外 MDC 也讓使用者可以根據自己想收到的品質決定要加入多少棵樹。[6]是先前關於多樹架構的研究。

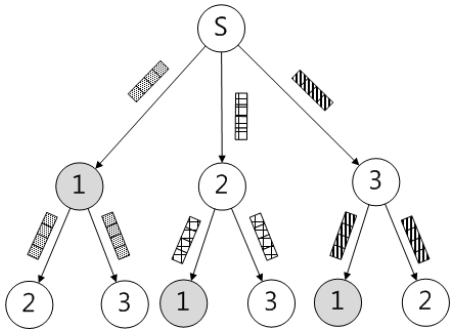


圖 2　多樹架構同儕網路串流系統示意圖

但是，爲了降低一個節點下線造成的影響，所以多樹架構中，一個節點只會

在一棵樹中是中間節點，在使用者上下線頻繁的環境中，維持這樣的結構將會十分困難。另外，雖然一個節點只在一棵樹中是中間節點，但若是比較高層的節點下線，仍然會影響到很多使用者。

2.2.3 網狀同儕網路串流系統

跟樹狀架構不同，網狀架構的同儕網路串流系統並沒有一個固定的結構，在網狀架構中，每個節點都會隨機跟其他節點連線，影片被切成許多固定大小的片段(Chunk)，每個節點都會告訴其他節點自己有哪些片段，並且跟其他節點要求自己沒有的片段，如圖 3 所示。由於沒有固定的結構，網狀架構很能夠適應使用者頻繁上下線的環境。先前關於網狀同儕網路串流系統的研究有[7][8]。另外， [3]比較了網狀架構和多樹架構的表現，它得到的結論是網狀架構在各方面表現都比多樹架構好。

但是在網狀架構中，每個節點都要先知道其他節點有哪些片段，才能跟其他節點要求自己沒有的片段，其他節點才會把片段傳給自己，這樣冗長的步驟使得影像的播放延遲變大，另外，由於每個片段都必須先要求才會收到，也使得訊息交換數量變多。

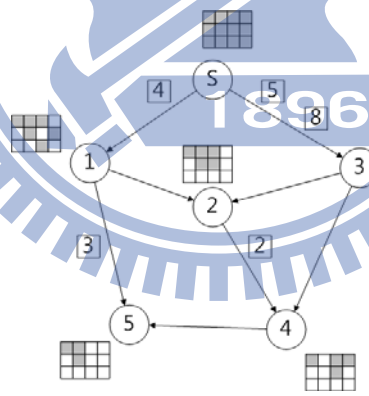


圖 3　網狀同儕網路串流系統示意圖

爲了改善這些問題，Coolstreaming[9]提出了子串流式的網狀同儕網路串流系統。Coolstreaming 將片段依照編號分成不同群組，每個群組是一個子串流，如圖 4。

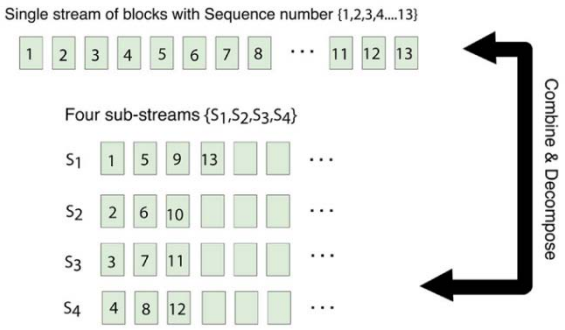


圖 4　片段分組成子串流

節點會互相告訴對方自己有哪些子串流，每個節點再互相要求自己沒有的子串流，如圖 5。而當一個節點收到另一個節點的要求，就會不斷的將屬於該子串流的片段傳給該節點，不需要每個片段都要求一次，因此減少了訊息交換量以及影像的延遲。

之後爲了說明清楚，我們先前互相交換片段的網狀同儕網路系統爲片段式網狀同儕網路系統，稱交換子串流的爲子串流式網狀同儕網路系統。

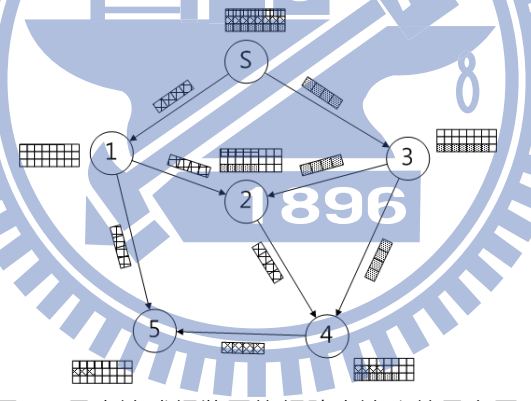


圖 5　子串流式網狀同儕網路串流系統示意圖

第三章　相關研究

3.1　同儕網路串流遭遇的問題

在同儕網路串流系統中，每個節點都是以自己的利益爲出發點做各種選擇，而沒有考慮整體系統的效能，因此會造成以下問題：

1. 大頻寬的節點不一定能收到較多資料，以致大頻寬的節點沒有足夠資料可傳輸，因而無法充分利用頻寬。
2. 資料未先傳給大頻寬或離提供者較近的節點，導致需要更長的路徑和更久的時間才能散布到整個系統，因而增加了播放延遲。
3. 未考慮先傳送較新的片段，導致系統中流傳的片段大部分是比較舊的。
4. 上傳較多資料的節點不一定能下載較多資料，造成公平性的問題。

表 1 描述了問題 2 和問題 3 產生的原因和改進方法。

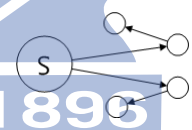
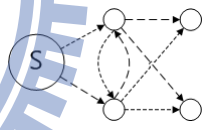
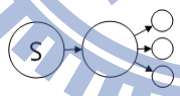
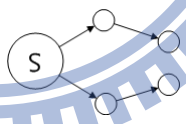
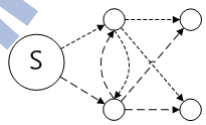
	Bandwidth	Latency	Content delay
Don't care			
Consider			

表 1　考不考慮頻寬、連線延遲、資料新舊所造成的差異

所以根據節點的頻寬、連線延遲，以及要求的資料的新舊來分配傳輸優先度，是可以得到較好的表現的。過去有許多研究也在這個方向努力，針對不同節點給予不同的傳輸速率，期望能提高系統的效能。但是這些研究大部分的目標都是爲了鼓勵使用者提供更多上傳頻寬，所以上傳越多的節點就可以收到越多資料以做爲回報，因此這些研究通常都只有考慮節點的上傳頻寬來分配傳輸速率，而未考慮連線延遲以及資料的新舊這兩點。PRIME[10]雖然提出了優先傳輸最新片段的概念，但是卻未考慮節點的上傳頻寬與連線延遲，因此仍然有改進空間。

此外，先前關於同儕網路串流系統的研究還有一個共同的問題。先前研究在

處理速率分配問題時，都是假設每個節點可以測量出自己的上傳頻寬，而只要分配給每個接收者的傳輸速率總和不超過自己的上傳頻寬就沒有問題。但是實際上每個節點間的頻寬會受到它們之間的路由路徑的頻寬影響，而不是只受限於提供者的上傳頻寬，因此只用提供者的上傳頻寬來當作速率分配的依據是會有問題的。

以圖 6 爲例，連接 C 的頻寬無法承載一條子串流，連接 D 的頻寬無法承載兩條子串流，因此就算提供者的上傳頻寬足夠，接收者 4 仍然無法收到完整的子串流，而接收者 1 跟接收者 2 只有一人可以收到完整的子串流。但是先前研究都是假設壅塞只會發生在連接 A，而沒有考慮壅塞發生在其他連接的可能性。[15] 提出了一套偵測共同壅塞連接的方法。

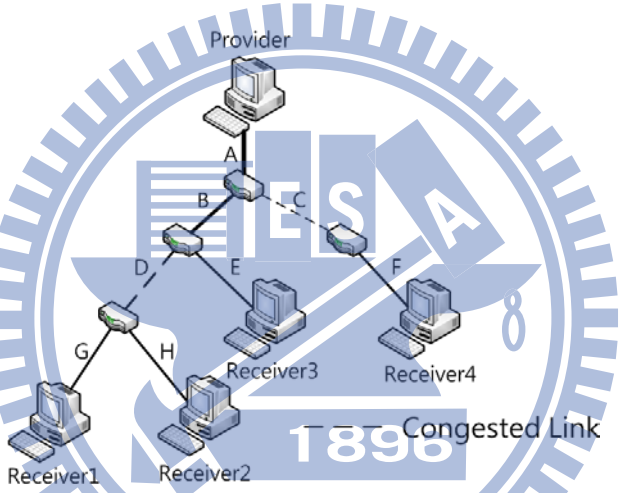


圖 6　速率分配的問題

3.2　先前相關研究

本節會介紹先前同儕網路系統的相關研究，和它們不足的地方。

3.2.1　Coolstreaming

Coolstreaming[9]是一個網狀同儕網路串流系統，並首次提出了子串流式網狀架構的概念。在 Coolstreaming 中，一個節點收到其他節點的要求後，會無條件的接受，並將子串流傳給該要求者，所以一個節點可能會有超過自己能負擔的數量的接收者，因此，接收者們會不斷的確認自己接收的情況，若發現自己的提供者無法負擔自己的要求時，就會再尋找另一個提供者。由於 Coolstreaming 使用 TCP 來傳輸串流，因此當一個節點的上傳頻寬不足時，它的接收者們就會因爲傳

輸速率不足而導致子串流的延遲不斷增加，所以每個接收者會去確認自己收到的子串流的延遲程度，當超過某個範圍時就會重新尋找一個提供者。

節點 A 會根據以下兩個不等式來判斷是否該更換提供者：

$$\max\{|H_{S_i,A} - H_{S_j,p}| : i \leq K\} < T_s$$
$$\max\{H_{S_i,q} : i \leq K, q \in \text{parnters}\} - H_{S_j,p} < T_p$$

其中$H_{Si,A}$是節點A收到的子串流S_i的最新的片段的時間(Timestamp)，$H_{Sj,p}$是子串流S_j的提供者收到的S_j的最新片段的時間。第一個式子是用來檢查節點A收到的子串流的延遲程度，若節點A的子串流S_i比它的任一個提供者的子串流落後超過T_s，就代表S_i的提供者可能沒有足夠頻寬將S_i傳給自己。第二個式子是用來檢查A的提供者收到的子串流的延遲程度，若節點A的子串流S_j提供者收到的子串流S_j比節點A的任何一個鄰居的任何一個子串流落後超過T_p，就代表S_j的提供者沒有收到足夠速率的Sj。兩個不等式只要有一個不成立，節點A就會從鄰居之中找一個符合上述兩個不等式的節點當新的提供者。但是因爲更換提供者太過頻繁也會降低系統效能，因此Coolstreaming限制更換提供者的周期爲T_A，也就是每個T_A的時間內只能更換一次提供者。

Coolstreaming 並沒有任何根據節點的頻寬或延遲來決定優先順序的機制，一切都只靠節點間的隨機要求與更換提供者，因此它的效能其實還有進步的空間。

3.2.2 PRIME

PRIME[10]指出了片段式網狀同儕網路串流系統中的兩個效能瓶頸(Performance bottleneck)，分別爲頻寬瓶頸(Bandwidth bottleneck)與內容瓶頸(Content bottleneck)。頻寬瓶頸產生的原因與每個節點平均的分支度(Degree)和頻寬的比例有關，一個節點的入分支度(In-degree)代表它向多少個節點要資料，出分支度(Out-degree)代表它傳資料給多少節點。提供者每一條連線能上傳的速率爲

$$\frac{\text{outbw}_p}{\text{outdeg}_p}$$

而接收者每一條連線能下載的速率爲

$$\frac{\text{inbw}_r}{\text{indeg}_r}$$

若是

$$\frac{\text{outbw}_p}{\text{outdeg}_p} > \frac{\text{inbw}_r}{\text{indeg}_r}$$

則代表接收者的下載速率無法承受提供者的上傳速率，因此無法充分利用提供者的上傳頻寬，反之，則無法充分利用接收者的下載頻寬。所以 PRIME 主張每個節點要根據它的上傳和下載頻寬調整入分支度和出分支度，使得

$$\frac{\text{outbw}_p}{\text{outdeg}_p} = \frac{\text{inbw}_r}{\text{indeg}_r} = \text{bwpf}$$

也就是說，每條連線的上傳速率和下載速率都等於 bwpf 時，才能最大化頻寬的利用率。

而內容瓶頸產生的原因是，一個節點因爲在某些時刻沒有其他節點需要的片段，所以沒辦法提供片段給其他節點，導致它的上傳頻寬無法被充分利用。爲了提高頻寬利用率，PRIME 首先提出了一個系統整體性的資料傳輸模式，能夠使內容瓶頸發生的機率降到最低，接著再提出一套要求片段的排程方法，每個節點都按照這個方法來要求片段，就能使整個系統達到理想中的資料傳輸模式。

PRIME 所提出傳輸模式的概念是，最新的片段必須先快速的散布到系統中的不同節點，使得每個節點都有一部分的最新片段，接著節點再互相交換自己所沒有的片段。在這個概念下，形成的結構就如圖 7。

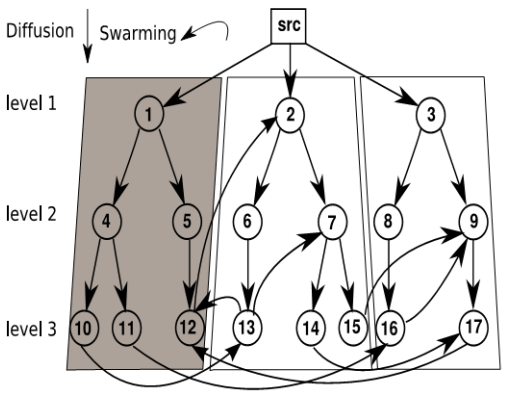


圖 7　PRIME 提出的片段傳輸模式

在 PRIME 提出的傳輸模式中，一個片段在系統中傳播會經過兩個階段。首先，新的片段從來源節點產生後，每個節點會盡力的將它散布到系統中，因此新的片段會先沿著幾個不同的樹狀結構散布到不同節點，這個階段稱爲 Diffusion phase，而這些樹狀結構則稱爲 Diffusion tree，在圖 7 中以直線表示。當不同的新片段都散布到系統中不同的的節點後，每個節點再互相交換自己沒有的片段，這個階段稱爲 Swarming phase，在圖 7 中以曲線表示。由於較上層的節點都已經將自己的頻寬用在 Diffusion phase，因此在 Swarming phase 中，負責傳輸的節點通常是位在 Diffusion tree 最底層的節點。

提出了這樣的傳輸模式後，PRIME 接著提出每個節點要求資料的排程方法，使得系統達到理想中的傳輸模式。而排程方法其實用一兩句話就可說完，就是每個節點每次要求片段時，都先要求一部份最新的片段，再要求那些自己快要播放到，而還沒收到的片段。

PRIME 的方法看似十分完美，但是仍然有可以改進的空間。首先，在 Diffusion phase 時，如果能考慮節點的頻寬，優先將片段傳給頻寬大的節點，將可使片段散布的速度更快，每個片段在 Diffusion phase 的時間變短，相信能使內容瓶頸發生的機率更加降低。第二，由於 PRIME 主張每條連線的傳輸速率都必須等於 bwpf，因此若將影像切割成 bit-rate 爲 bwpf 的數個子串流，然後每個節點都優先傳送最新的子串流，相信也能達到減低頻寬瓶頸與內容瓶頸的目的，而且因爲不必每個片段都要求，訊息交換量也可以更加降低。第三，每個節點在決定自己的入分支度和出分支度時，都是根據自己測量得知的上傳和下載頻寬，但是前面提過，節點間的頻寬並不一定只受限於提供者的上傳頻寬或接收者的下載頻寬，因此只根據測量得知的頻寬決定分支度，可能並不適用於所有的環境。

3.2.3 Enabling Contribution Awareness in an Overlay Broadcasting System

Sung YWE 等[11]針對多樹架構中提出一套根據節點上傳速率分配優先權的機制。首先它規定每個節點能獲得的下載速率取決於該節點願意付出的上傳速率，能下載的速率和願意付出的上傳的速率關係式爲

$$r_i = \frac{1}{t} * f_i + \frac{t-1}{t} * \sum_i \frac{f_i}{N}$$

其中r_i是節點i能獲得的下載速率，f_i是節點i願意付出的的上傳速率，N是系統的總節點數，t是稅率。這個式子所代表的意義是，系統規定了稅率爲t，所以付出了f的上傳速率，能獲得的下載速率爲f/t，表示在式子的第一項。但是稅率大於 1 時，一定會有一些多出來的頻寬未被使用，這些頻寬應該平均分配到每個節點上，就是式子的第二項的意義。每個節點都會對其他節點誠實的告知自己的f_i。r_i稱爲節點i的Entitled bandwidth，每個節點會根據r_i來決定自己可收到多少子串流，而加入多少棵樹，這些樹稱爲Entitled tree，而這些節點稱爲Entitled node。

但是若有些節點的 Entitled bandwidth 大於影像的 bit-rate，就代表會有些頻寬未被利用(因爲節點最多只會下載到跟影像的 bit-rate 一樣快)，這些未被利用的頻寬稱爲 Excess bandwidth。爲了使 Excess bandwidth 也能被充分的利用，Entitled bandwidth 小於影像的 bit-rate 的節點會不斷嘗試加入更多棵樹，以獲得更多的下載頻寬，這些節點稱爲 Excess node。而爲了保障 Entitled node 的權益，Excess node 的優先權是比 Entitled node 小的，因此只有一棵樹中有多餘的頻寬時，才會分給 Excess node，而 Excess node 也會不斷調整嘗試的頻率，當它在一棵樹中一直沒辦法獲得 Excess bandwidth，它就會不斷降低對該棵樹的嘗試頻率。

以上是這篇論文提出的方法。但是它只有考慮以節點的頻寬來分配傳輸優先權，而沒有考慮到連線延遲或要求的子串流的延遲。另外，在頻寬分配的部分，只考慮節點的上傳頻寬來判斷可傳給多少接收者，當接收者數量太多就只分配給優先權高的，所以也是沒有考慮到節點間的頻寬可能不只受限於提供者上傳頻寬的事實。

第四章　Bandwidth and Latency Aware Send Rate Allocation in P2P Streaming System

4.1　簡介

本論文針對子串流式網狀同儕網路串流系統(Sub-stream based mesh P2P streaming system)，提出一套提供者根據接收者的貢獻分配給接收者的傳輸速率之機制，以提升系統的頻寬利用率和傳輸速率分配的公平性。本論文的應用爲即時的影像串流，例如運動賽事的即時轉播。本論文的方法中，提供者會無條件接受任何接收者的要求，而當提供者的上傳頻寬無法滿足所有的接收者時，提供者會把傳輸速率分配到貢獻最大的幾個接收者上，因爲每個節點會定期偵測是否從提供者收到足夠的速率，所以貢獻不夠大而沒有收到足夠速率的接收者在一段時間後就會改選擇其它的提供者。至於各節點如何選擇提供者以及偵測是否該更換提供者，並非本論文重點，因此這部分是沿用 Coolstreaming 的方法。

本論文提出的機制稱爲 Bandwidth and Latency Aware Speed Allocation (BLASA)，BLASA 包含以下兩部分：

1. Bandwidth and Latency Aware Contribution Estimation (BLACE): 一個提供者根據各個接收者的上傳頻寬、提供者到該接收者的連線延遲，以及該接收者要求的子串流延遲程度評估該接收者對系統貢獻度的方法。
2. Contribution Biased Congestion Control (CBCC): 一套壅塞控制的機制，提供者根據每個接收者的貢獻度調整壅塞控制的參數，使貢獻較大的接收者能夠從提供者得到較大的傳輸速率。

爲了提升網狀同儕網路串流系統的傳輸效率，我們希望每個節點都能將自己收到的資料盡快的散布給更多人，而在一個提供者的頻寬有限的時候，它就必須將自己有限的頻寬用在夠好的接收者上，也就是可以把提供者送來的資料快速的散布出去的接收者。好的接收者應該有下列條件：

- 上傳頻寬大
- 和提供者之間的連線延遲小

- 和提供者要求的子串流是比較新的

但是這些條件不一定會全部滿足，所以本論文提出了一個綜合這些條件計算不同子串流的各個接收者貢獻度的方法。提供者計算出所有接收者的貢獻度，再根據各個接收者的貢獻度來分配對該接收者的傳輸速率。

至於如何達到要求的速率分配，本論文提出了一套壅塞控制的機制，由提供者根據各個接收者的貢獻調整壅塞控制的參數，當多個接收者互相競爭同一條連線的頻寬時，貢獻最大的接收者能夠獲得最大的傳輸速率。

先前有關同儕網路串流系統的研究，都沒有深入探討如何達到要求的速率分配，，只是很簡單的假設每個節點都能測量自己的上傳頻寬，再根據測量出的上傳頻寬來分配對各個接收者的傳輸速率。這樣理想化的假設應用在真實的網路環境會有一些問題，因爲真實的網路中，每個節點間的頻寬會受到節點間的路由路徑(Routing path)的頻寬影響，提供者和接收者之間的頻寬可能無法負荷提供者分配的傳輸速率，甚至也可能出現兩個接收者共用一壅塞連接，所以只有其中一個接收者能獲得分配的傳輸速率的情況。這些問題在先前研究中都沒有被討論過。

因此，本論文提出用壅塞控制機制來分配傳輸速率的概念，本論文提出的壅塞控制機制會自動探索提供者和接收者間的可用頻寬，且當網路壅塞發生以致無法讓所有接收者都收到完整的串流速率時，傳輸速率會自動分配到貢獻最大的幾個接收者上。

4.2 Bandwidth and Latency Aware Contribution Estimation

本論文所提出的貢獻計算方法是站在提供者的立場，考慮它的各個不同子串流的接收者可以將它傳送過去的子串流再送給多少人。所以就算是同一個節點，對不同子串流的貢獻還是會受到它到不同子串流的提供者的連線延遲，以及提供者收到這些子串流的延遲程度的影響，而有不同的計算結果。但是雖然計算結果不同，其實背後考慮的都是同一件事，也就是「這個接收者能把我送過去的子串流再送給多少人」。本論文提出的計算方法需要每個節點都知道它的接收者的上傳頻寬，以及它到接收者的連線延遲，此外，每個節點還需要知道一些整個系統的資訊，例如總節點數和每個節點平均的上傳頻寬，所以本節最後會說明如何獲得這些資訊。

本節所使用的符號意義都註明於表 2。

表示符號	意義
$\mathbf{con_i(X)}$	節點 X 真實的貢獻
$\mathbf{econ_i(X)}$	節點 X 被估計出的貢獻
T	子串流的延遲門檻，延遲超過這個值，節點就會另外尋找提供者
S	來源節點
N	系統目前總節點數
$\bar{\mathbf{B}}$	系統目前節點平均上傳頻寬
L	系統目前節點間平均連線延遲
$\mathbf{O_c(t)}$	片段 c 由一節點送出後經過 t 秒後，當下有多少 Outstanding 的片段 c
$\mathbf{A_c(t)}$	片段 c 由一節點送出後經過 t 秒，總共有多少片段 c 被收下
$\boldsymbol{\lambda}$	節點間連線延遲指數分布的參數，$\lambda = 1/L$
$\boldsymbol{\rho}$	一個 Outstanding chunk 被收下後，可淨增加多少 outstanding chunk
$\mathbf{S_N}$	子串流數量，是系統預設的參數
$\mathbf{S_R}$	一個子串流的 Bit rate，是系統預設的參數
$\mathbf{B(X)}$	節點 X 的上傳頻寬
$\mathbf{A(t)}$	一般化後的 $A_c(t)$
$\mathbf{O(t)}$	一般化後的 $O_c(t)$

表 2　4.2 節用到的所有符號解釋

4.2.1　貢獻的定義

綜合前面的敘述，我們很直覺的定義一個接收者對一個子串流的貢獻度為：「直接或間接由這個接收者收到這個子串流的節點數」。節點 X 對子串流 i 的貢獻度以$\mathrm{con_i(X)}$表示。如圖 8 的情況，$\mathrm{con_2(W)} = 3$，因為 X、Y 及 Z 都經由 W 收到子串流 2。

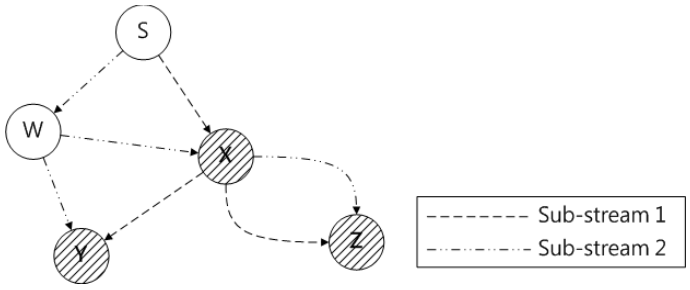


圖 8　節點貢獻的定義

4.2.2　貢獻計算的概念

雖然我們已經定出了貢獻的意義，但是一個節點會把子串流散布給多少節點

並沒辦法事先知道，所以本論文提出了一套根據節點的上傳頻寬、連線延遲，以及要求的子串流的延遲程度來估計該節點能做出的貢獻的方法，估計出的節點 X 對子串流 i 的貢獻以$econ_i(X)$表示。

提供者會比較所有子串流的所有接收者的貢獻，把有限的頻寬分給貢獻最大的幾個接收者。但是由於貢獻的計算概念是每個子串流都一樣的，且每個子串流在系統中傳播的模式都很相似，所以下面的討論都只針對單一子串流，除非有必要才會強調特定的子串流。

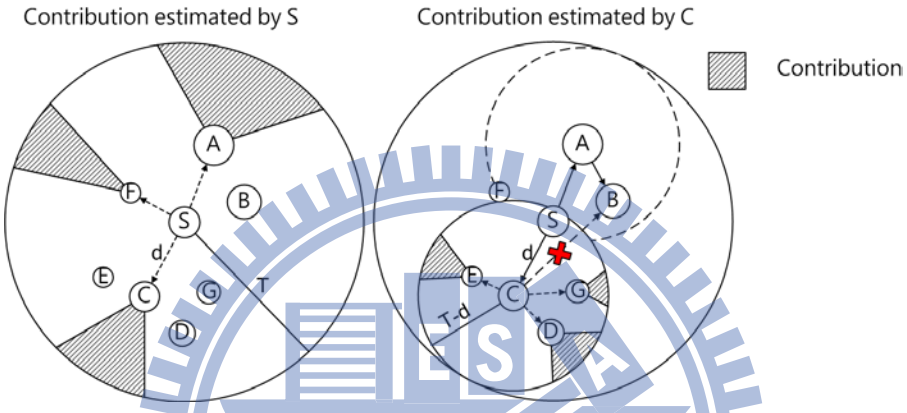


圖 9　估計節點貢獻的概念

圖 9 示意了本論文貢獻計算方法的概念。我們定義 T 是子串流延遲的門檻值，若一個節點的子串流延遲超過這個值，代表它的提供者沒有足夠頻寬傳送它，因此節點就會去尋找另一個提供者。而一個節點對一個子串流的貢獻就是在這個子串流的延遲到達 T 之前，這個節點能夠把這個子串流散布給多少人，所以貢獻會受到提供者收到子串流時的延遲程度、提供者到接收者的連線延遲，以及接收者的上傳頻寬影響。以圖 9 爲例，節點 S 是來源節點，S 在估計節點 A、C、F 的貢獻時，會根據自己到 A、C、F 的連線延遲，判斷它們收到自己傳送的子串流時，延遲各是多少，接著 S 再考慮 A、C、F 的上傳頻寬，來估計出它們在子串流的延遲達到 T 前可以將子串流散布給多少人。接著節點 C 要估計節點 D、E、G 的貢獻時，也是根據相同的概念，唯一不同的是 C 收到的子串流已經有了延遲，所以它在判斷節點 D、E、G 收到子串流的延遲時，除了考慮 D、E、G 到自己的連線延遲，還要考慮自己收到這個子串流時已經有的延遲。

雖然我們的貢獻計算方法需要每個節點都誠實回報自己的頻寬，但是我們相信可以很容易將[12][13]的方法應用到我們的系統中。

4.2.3 貢獻計算的方法

和上一小節一樣，本小節的討論也只針對單一子串流，除非有必要才會特別強調不同的子串流。

了解了貢獻計算的概念之後，剩下的問題就是如何根據接收者的上傳頻寬與連線延遲來估計出圖 9 中斜線部分的大小。我們再用一張圖更清楚的描述這個問題。

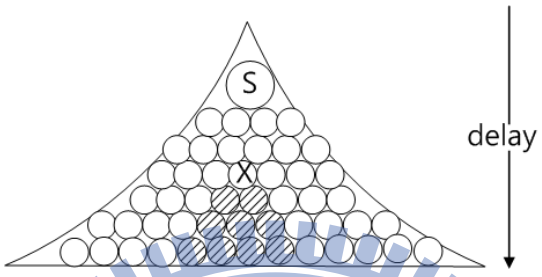


圖 10　節點對系統的貢獻示意圖

圖 10 是一個子串流由來源送出，散布到全部的節點的過程，節點 X 的貢獻就是圖中的斜線區域，也就是我們要估計的目標。在估計斜線部分的大小之前，我們必須要先知道一些系統的資訊，並利用這些資訊，來推測系統散布子串流的能力，也就是一個子串流在過了一段時間後，會被系統散布給多少人，知道了系統的這項能力後，才有辦法去估計斜線部分的大小。需要知道的系統資訊列於表 3，本節最後會說明如何獲得這些資訊。

表示符號	意義
N	系統中總節點數
$\bar{B}$	系統中節點平均頻寬
L	系統中節點間的平均連線延遲

表 3　每個節點都需要知道的系統資訊

接著開始說明如何利用這些資訊來推測系統的能力。先前章節有提過，子串流其實也是由片段組成，我們考慮一個片段 c 剛由一個節點送出時，由於該節點到它的接收者有連線延遲，所以片段 c 不會馬上到達接收者，我們稱這一些還在半路的片段爲 Outstanding chunk，稱已經到達目的地的片段爲 Arrived chunk。當片段 c 到達接收者後，接收者會再將片段 c 送給下一步的接收者，也就產生更多的 Outstanding chunk。過程如圖 11 所示。

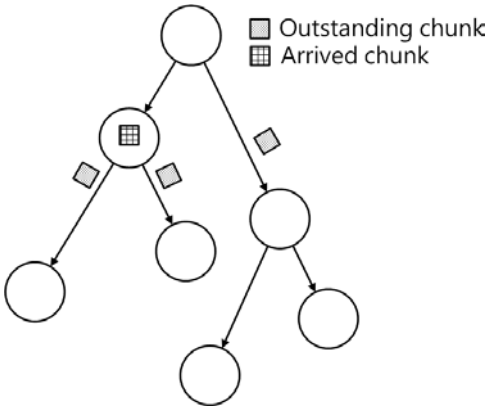


圖 11　Outstanding chunk 和 arrived chunk

我們定義 $O_c(t)$ 爲片段 c 由節點送出經過 t 秒後，產生的 Outstanding chunk 的數量，$A_c(t)$ 爲片段 c 由節點送出經過 t 秒後，產生的 Arrived chunk 數量，也就是 t 秒後收到片段 c 的節點數。注意 $O_c(t)$ 是片段 c 被送出經過 t 秒後，當下正在系統中流竄的 Outstanding chunk 數，而 $A_c(t)$ 是片段 c 被送出經過 t 秒後，收到片段 c 的節點數。

一個 Outstanding chunk c 平均要經過 L 秒後才會變成 Arrived chunk，所以平均而言，系統中所有的 Outstanding chunk c 在經過 Δt 秒後，其中會有 $\Delta t/L$ 的比例到達目的地而變成 Arrived chunk，也就是

$$A_c(t+\Delta t) = A_c(t) + O_c(t)\lambda\Delta t \tag{1}$$

其中 $\lambda = 1/L$。

此外，Outstanding chunk 變成 Arrived chunk 後，收到它的接收者會再把它傳給其他人，因而產生更多 Outstanding chunk。我們定義 ρ 表示一個 Outstanding chunk 被收到後，淨增加的 Outstanding chunk 數。ρ 的值會受到系統中節點的平均頻寬影響，如果平均一個節點可以把子串流傳給三個人，就代表一個 Outstanding chunk 被收到會產生三個 Outstanding chunk，淨增加了兩個 Outstanding chunk，於是 $\rho = 2$。我們假設每個節點會平均的把頻寬用來傳每一個子串流，因此

$$\rho = \frac{\overline{B}}{S_N \cdot S_R} - 1 \tag{2}$$

其中 S_N 是子串流的數量，S_R 是一個子串流的 bit rate。我們可以發現若平均頻

寬不夠，ρ 可能爲負值。現在我們可以列出 $O_c(t)$ 和 ρ 的關係：

$$O_c(t+\Delta t) = O_c(t) + \rho \cdot O_c(t)\lambda\Delta t \tag{3}$$

再加上初始條件：

$$O_c(0) = 1 \tag{4}$$

$$A_c(0) = 0 \tag{5}$$

根據(1)(2)(3)(4)(5)，我們就可以解出 $O_c(t)$ 和$A_c(t)$ 的答案：

$$O_c(t) = e^{\lambda\rho t} \tag{6}$$

$$A_c(t) = \frac{1}{\rho}e^{\lambda\rho t} - \frac{1}{\rho} \tag{7}$$

此外，這兩個函數其實適用於任何片段，而不是只有片段 c，因此可以再一般化成 $O(t)$ 和 $A(t)$。其中 $A(t)$ 是一片段被送出後，在 t 秒內可以收到這個片段的節點數，也可以理解成一子串流被送出後，在延遲增加 t 秒之前，可收到這個子串流的節點數。這個函數就可以用來計算節點 X 的貢獻。

根據以上的推導，節點 X 收到一子串流後經過 t 秒，可以將該子串流散布給這麼多節點：

$$\frac{B(X)}{S_N \cdot S_R}A(t) \tag{8}$$

但 X 的貢獻是在子串流延遲達到 T 之前，能夠將子串流散播過去的節點數。假設節點 X 收到的子串流已有 d_X 的延遲，則 X 對該子串流的貢獻就可以用下式表示：

$$\frac{B(X)}{S_N \cdot S_R}A(T - d_x) \tag{9}$$

當 X 的提供者要計算 X 對一個子串流的貢獻時，它會考慮自己收到子串流時已有的延遲，加上它跟 X 之間的連線延遲，來推測 X 收到子串流的延遲 d_X，再跟 X 的上傳頻寬 $B(X)$ 一起代入(9)，就能計算出 X 對這個子串流的貢獻。

4.2.4 系統資訊的取得

本論文的方法中，每個節點會定期根據它的接收者回應的 ACK 來測量出自己的有效頻寬，再告訴它的提供者。另外，節點也會根據接收者回的 ACK 來測量接收者和自己之間的 RTT。

在 Sung YWE 等 [11]裡提出了一個讓每個節點能夠獲得系統資訊的方法，它提出的作法是每個節點都收集以自己爲 root 的 sub-tree 的資訊，回報給它的提供者，如此來源節點就可得知整個系統的資訊，來源節點再把這些資訊散布給整個系統。本論文也使用相似的概念，每個節點都收集在自己底下 sub-tree 的所有節點的個數、總共的上傳頻寬和連線延遲，再回報給提供者，提供者就可以得知系統總共的節點數、上傳頻寬和連線延遲，來源節點計算出平均後再散布給整個系統。

4.3 Contribution Biased Congestion Control

4.3.1 以壅塞控制來分配傳輸速率

在第二章中提過，節點間的頻寬會受到路由路徑影響，而不是只受限於提供者的上傳頻寬，所以爲了在這種條件下達到理想的速率分配，本論文提出以壅塞控制來分配傳輸速率的概念。本論文的壅塞控制機制可以自動探索提供者跟接收者之間可用的頻寬，且當多個接收者競爭同一條連接的頻寬時，貢獻最大的幾個接收者能夠獲得最大的傳輸速率。

目前絕大多數的壅塞控制演算法都使用 Acknowledgement 的機制來偵測封包遺失，並使用 Congestion window (Cwin)來控制傳輸速率。所謂的 Acknowledgement (簡稱 Ack)指的是接收端收到一個封包後，會回應一個 Ack 訊息通知傳送端自己還沒收到的封包的序號，所以若傳送端收到多個相同的 Ack，就代表該序號的封包可能已經遺失。另外，傳送端若太久沒有收到接收端回應的 Ack，也會斷定是封包遺失，這個情況稱爲 Timeout，傳送端會使用一個 Timer 判斷是否 Timeout。而 Cwin 的目的是限制已送出但還未被 Ack 的資料數量，也就是說，傳送端在收到接收端回應的 Ack 之前，最多只能送出不超過 Cwin 大小的資料量，因此 Cwin 就相當於是一個 RTT (Round trip time)能夠送出的資料量，而調整 Cwin 的大小就能改變傳送端的傳送速率。絕大多數的壅塞控制演算法都是根據收到的 Ack 來調整 Cwin 的大小：若發生封包遺失，就減少 Cwin 的大小；若將 Cwin 用完後，沒有發生封包遺失，就增加 Cwin 的大小。根據每次增加和減

少 Cwin 的方法，壅塞控制可分爲 AIAD、AIMD、MIAD 和 MIMD 四類，在[14]中有這四類壅塞控制方法更詳細的特性比較。

本論文提出的壅塞控制機制稱爲 Contribution biased congestion control (CBCC)，CBCC 是每條連線獨自進行(Per-flow based)，基於 AIAD (Additive increase, additive decrease)的壅塞控制，當用完 Cwin 且沒有封包遺失時，會將 Cwin 的大小增加一固定的值，而當封包遺失時，Cwin 的大小會減少一固定的值。本論文的方法中，提供者會根據接收者的貢獻調整每次增加和減少的值，貢獻大的接收者增加較多，減少較少，貢獻少的接收者則反之。如此一來，當兩個接收者競爭頻寬時，情形就如圖 12 所示。

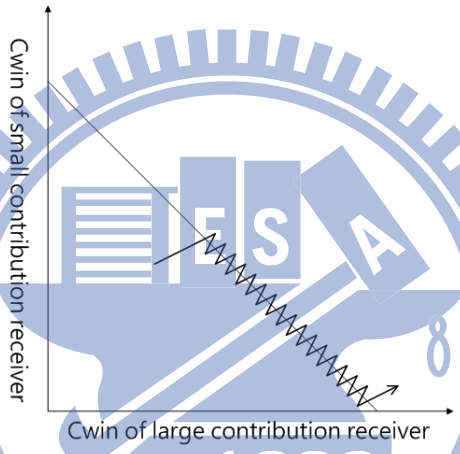


圖 12　兩個接收者競爭頻寬的情形

當沒有封包遺失時，貢獻較大的接收者 Cwin 增加的速度較快。直到兩個接收者的 Cwin 大小總和超過連接能負荷的傳輸速率時，兩個接收者同時會產生封包遺失因而減少 Cwin 的大小，此時貢獻較小的接收者減少較多，如此反覆動作就會使貢獻較大的接收者慢慢獲得大部份的傳輸速率。這個概念也可套用於多個接收者的情況。

但是並不是說多個接收者競爭頻寬時只有一個接收者能獲得大部分速率，由於一個子串流的 bit rate 是固定的，提供者提供給接收者的傳輸速率不會超過這個值，因此當連接的頻寬足夠傳輸兩個以上的子串流時，並不會發生所有頻寬都被一個接收者搶去的情況。所以總結來說，若有多個接收者競爭一個只能傳輸 m 個子串流的連接的頻寬，CBCC 可以讓貢獻最大的 m 個接收者收到完整速率的子串流。

至於如何分配每次 Cwin 增加和減少的值，我們目前採用簡單的等差方式，貢獻最大的接收者每次增加的值最多，減少的值最少，而貢獻次大的接收者每次增加的值就減少一個差距，減少的值則增加一個差距，貢獻更小的接收者也是相同道理。貢獻最大的接收者每次增加／減少的值，以及其他接收者跟它的差距都是預先定義好的常數。接收者 R_i 每次 Cwin 增加／減少的值分別以 $Inc(R_i)$ 及 $Dec(R_i)$ 表示。

更正式的說，提供者使用 CBCC 控制對接收者 R_i 的傳輸速率的流程如以下的 Pseudo code：

```
Set size of cwin to InitCwin
Send segments not exceed cwin
loop
begin
    Wait for ack arrive or timeout
    if duplicate acks or timeout
    begin
        Retransmit segments immediately
        Decrease size of cwin by Dec(Ri)
    end
    else if all segments in fully used cwin are acked
    begin
        Increase size of cwin by Inc(Ri)
    end
    Send segments not exceed cwin
end
```

且提供者會不斷的根據交換到的訊息來重新計算 R_i 的貢獻，並更新 $Inc(R_i)$ 和 $Dec(R_i)$。

4.3.2 去除 RTT 造成的不公平

CBCC 和大部分的壅塞控制方法一樣，都是每個 RTT 就增加或減少 Cwin 的大小，因此不論貢獻大小，和提供者之間的 RTT 較小的接收者增加的頻率會比較大，因而有可能獲得較大的傳輸速率，這並不是我們想要的結果。所以我們會根據 RTT 來調整每次增加的值，以消除這種效應。

我們先考慮如何在RTT不相同的情況下，每個接收者都分到相同速率。在RTT不相同的情況下要達到相同速率，Cwin必須跟RTT成正比，因爲Cwin就是一個RTT可以傳輸的資料量。但是實際上Cwin增加的頻率卻跟RTT成反比，因此爲了使Cwin跟RTT成正比，每次增加的量必須跟RTT^2成正比。在[16]中，其實也提出了類似的結論。

因此在 CBCC 中，爲了使每個接收者分到的傳輸速率只跟它的貢獻有關，接收者 R_i 每次增加的值必須調整爲

$$\left(R \cdot RTT(R_i)\right)^2 \cdot Inc(R_i) \tag{10}$$

其中 $RTT(R_i)$ 是 R_i 和它的接收者間的 RTT，R 是預先定義且每個接收者都相同的常數，用來避免因爲 RTT 的值太小導致 Cwin 增加太慢的情況。

4.3.3 去除提高速率導致封包遺失率升高引發的影響

除了 RTT 會對 CBCC 的效果造成影響外，封包遺失率是另一項會減低 CBCC 效果的因素。當發生壅塞時，原本傳輸速率較快的接收者勢必會遺失較多封包，會造成以下兩個結果：第一，傳輸速率較快的接收者 Cwin 會被減低較多次，使提供者無法拉開貢獻大的接收者和貢獻小的接收者之間的傳輸速率差距；第二，如果按照 TCP 的做法，ACK 只回報未收到的封包中序號最小的，當遺失封包的數量多，就必須花好幾個 RTT 才能全部重新傳送完畢，也會使得本來傳輸速率較快的接收者要花較多時間才能回復。

爲了解決第一個問題，在 CBCC 中，提供者並不是每得知一個封包遺失就減少接收者的 Cwin 一次。提供者會先判斷這次遺失的封包是否跟上一次遺失的封包屬於同一「批」，如果是，則不需要再減少 Cwin。具體的流程是：提供者會確認遺失的封包是在什麼時候傳出的，如果是在上一次減少 Cwin 的時間之前，則代表這個封包遺失的原因是之前的 Cwin 太大，但是因爲 Cwin 已經減少過了，所以不需要再減少一次。如果遺失的封包傳出的時間是在上一次減少 Cwin 的時間之後，代表減少後的 Cwin 仍然太大，所以還是有封包遺失，則提供者才會再次減少接收者的 Cwin。

爲了解決第二個問題，我們讓一個 ACK 可以回報多個封包遺失。當接收者從提供者收到一個封包後，它會跟提供者要求序號在這個封包之後的資料，以及

序號在這個封包之前，但是卻還沒收到的資料，所以不需要等一個封包重送完才知道下一個遺失的封包是哪一個。

圖 13 是本小節所述方法的運作例子。

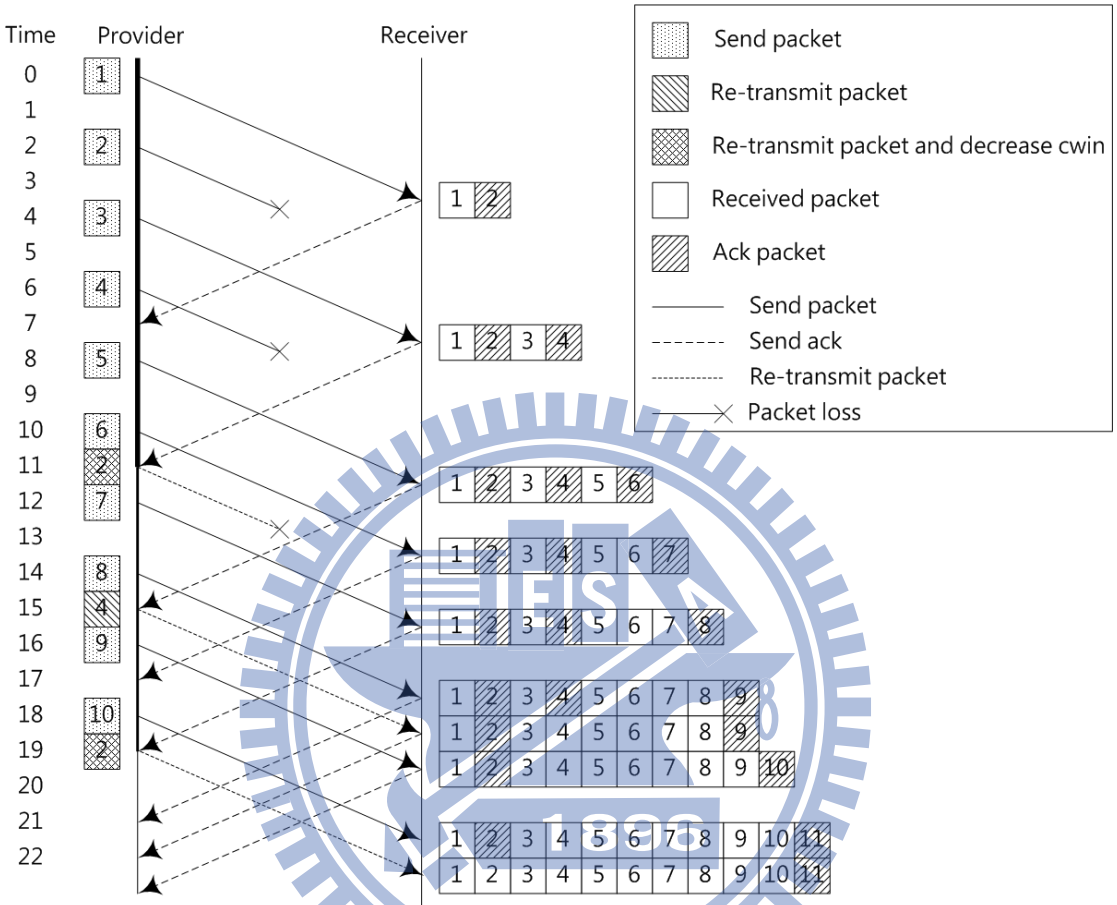


圖 13　CBCC 中處理封包重送以及判斷何時該減少 Cwin 的例子

在第 11 秒時提供者由 ACK 得知了封包 2 遺失，於是重送封包 2，並減低接收者的 Cwin。在第 15 秒時提供者得知了封包 4 也遺失了，於是重送封包 4，但是此時並不會減少 Cwin，因爲封包 4 上次送出的時間(第 6 秒)比上次減少 Cwin 的時間(第 11 秒)還早。另外，雖然第 15 秒收到的 ACK 中也有封包 2，但是接收者並不會再重送一次封包 2，因爲這個 ACK 是回應封包 5，而封包 5 送出的時間(第 8 秒)比封包二最後送出的時間(第 11 秒)還早，所以這個 ACK 中有封包 2，並不代表最後送出的封包 2 已經遺失了。直到第 19 秒，提供者才確認重送的封包 2 又遺失了，此時因爲上次送出封包 2 的時間(第 11 秒)大於等於上次減少 Cwin 的時間(第 11 秒)，因此提供者判斷 Cwin 仍然太大，所以還是有封包遺失，因此提供者再次重送封包 2，也再次減低接收者的 Cwin。

4.3.4 CBCC 帶來的額外效益

CBCC 除了可以讓一個提供者妥善的分配它到各個接收者的傳輸速率，還有另一個額外好處：不同提供者到不同接收者的路徑中，若有共同的壅塞連接，貢獻大的接收者仍然有較大機率可以獲得較大的傳輸速率。如圖 14 的情況，提供者 1 到接收者 1_3 和提供者 2 到接收者 2_3 的路徑有共同壅塞連接，此時接收者 1_3 和接收者 2_3 也會通過像圖 12 的過程競爭頻寬，雖然提供者 1 和提供者 2 爲它們設定的壅塞控制參數並不一定會符合它們貢獻的大小，但是我們還是可以說貢獻較大的接收者在競爭頻寬中勝出的機率較大。

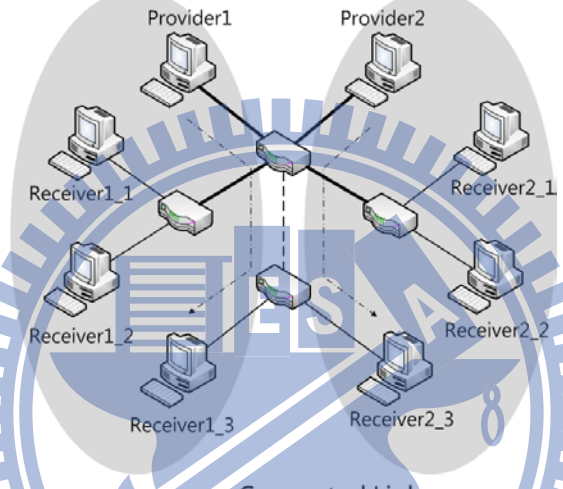


圖 14　不同提供者和不同接收者共用相同壅塞連接

第五章　模擬實驗結果與討論

5.1　簡介

本論文探討的是同儕網路串流系統，由於同儕網路串流系統中動輒是數千至數十萬個節點，因此以實際網路系統來測試效能是有困難的。目前同儕網路的研究測量效能的方法主要有兩種，第一是使用模擬器進行實驗，第二是使用 PlanetLab[17]的環境進行實驗。僅有少數研究，例如 Coolstreaming[9]，有實做出實際的系統並量測效能。

PlanetLab 是一個全球性的實驗平台，包含在世界各地的幾千台電腦，可讓研究者使用真正的網路環境進行實驗。但是由於不了解申請程序，再加上本論文還在初步發展階段，因此尚不考慮使用 PlanetLab 進行實驗。

目前針對同儕網路的模擬器選擇有很多，例如 P2PSim、OverSim[18]等等，S. Naicken 等[19]對現存大部分的同儕網路的模擬器做了比較。但是經過多方的比較之後，我們發現大部分的同儕網路的模擬器都只針對像是 Distributed hash table (DHT)或是 Gossip based 之類的搜尋演算法有提供比較便利的介面，而少有針對同儕網路串流應用而特別設計的模擬器。

因此，最後我們還是選擇使用學術界十分知名的 NS2 模擬器[20]。NS2 最早是來自 1989 年的 Real Network Simulator，一直以來 NS 都在吸收全世界各地研究者的支持，經過多年的發展之後，NS2 成爲學術研究上最具影響及代表性的網路模擬器。

NS2 模擬器中使用了 TCL 和 C++兩種語言，其中 C++主要用來實做新的模組，用來處理通訊協定或資料，而 TCL 主要用來設定模擬的網路環境，例如節點數和連線頻寬等。之所以要這樣設計，是因爲通訊協定需要處理大量的資料和控制訊息，此時處理的速度是很重要的，因此使用編譯式的 C++，以獲得較快的處理速度。而模擬的網路的環境設定常常會需要調整，所以使用直譯式的 TCL，以免每次調整網路環境都需要重新編譯。

以下會報告本論文提出的方法的模擬實驗結果。本實驗分爲兩部分，首先本實驗會先測試 CBCC 的效果，確認是否可以使貢獻大的節點分到較多速率，接著本實驗測試 CBCC 結合 BLACE 後，應用在同儕網路串流系統上的效果，並與 Coolstreaming 的效能做比較。

5.2 CBCC 的效能測試

5.2.1 模擬環境設定

在 CBCC 的測試中，我們先使用較簡單的網路拓樸，如圖 15 所示。我們讓每個節點都只向來源節點要求資料，而不互相要求資料，接著設定來源節點的上傳頻寬，讓它不足以傳送串流給所有節點，並在程式中直接設定每個節點的貢獻值，使得編號較小的節點有較大的貢獻值，再來確認編號較小的節點是否都能收到較多的資料。

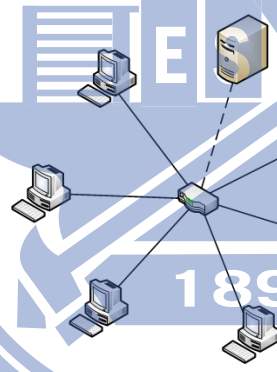


圖 15 CBCC 模擬所使用的網路拓樸

此外，我們測試了幾個不同的情境。首先，我們將所有節點對來源節點的 RTT 都設爲 0.4 秒，測試編號較小的節點是否收到較多資料，接著，我們將編號 1 到 6 的節點 RTT 分別設爲 0.60、0.40、0.28、0.18、0.12、0.08 秒，並測試有考慮 RTT 調整每次 Cwin 增加的值，和不考慮 RTT 的情況下，效果有什麼不同。本部分的參數設定都列於表 4。

CBCC 模擬實驗的參數設定	
子串流數量	1
影像 bit-rate	384kbps
片段大小	4.8KByte
片段長度	0.1 秒

模擬時間	1000 秒	
總片段數	10000	
模擬次數	100 次	
節點數量	2, 3, 4, 5, 6	
節點 RTT	情境 1	皆爲 0.4 秒
	情境 2	節點 1~6 分別爲 0.6, 0.4, 0.28, 0.18, 0.12, 0.08 秒
來源節點頻寬	600kbps, 1000kbps, 1500kbps	

表 4　CBCC 模擬實驗的參數設定

5.2.2　實驗結果與分析

經過實驗，我們發現 CBCC 可以讓貢獻大的節點獲得較多的傳輸速率，但是 CBCC 的效果會受到上傳頻寬的大小，以及競爭頻寬的節點數的影響，而有不同的表現。我們的實驗觀察了在各種條件下，每個節點各自收到多少片段，以及接收速率隨時間的變化。

首先，先看每個節點 RTT 都相同的情況。由圖 16、圖 17、圖 18 可看出，貢獻較大(編號較小)的節點的確收到較多的片段，但是貢獻大的節點並不一定能收到完整的速率，當來源節點上傳頻寬小時，貢獻大的節點還是會被貢獻小的節點搶走一點接收速率，如圖 16 的情況，Peer1 並沒有收到完整的 10000 個片段。

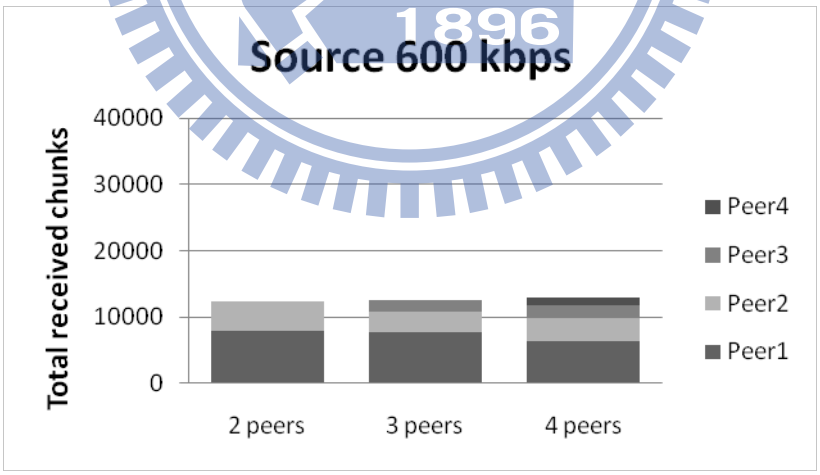


圖 16　相同 RTT，上傳頻寬 600kbps，各個節點接收到的片段數

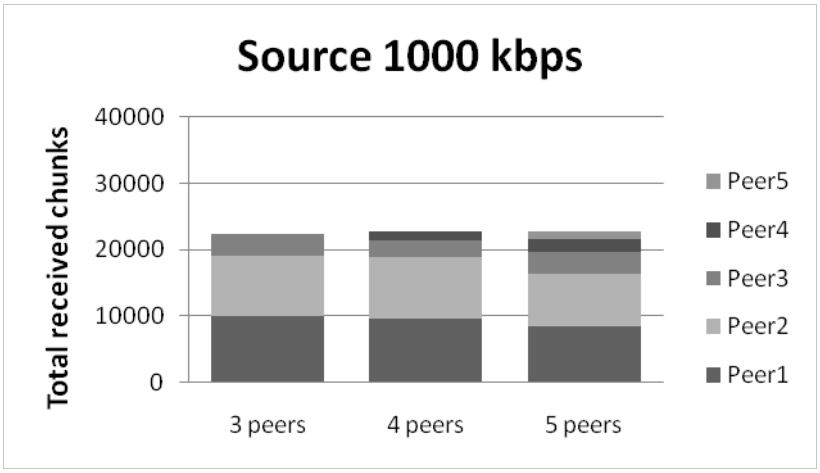


圖 17　相同 RTT，上傳頻寬 1000kbps，各個節點接收到的片段數

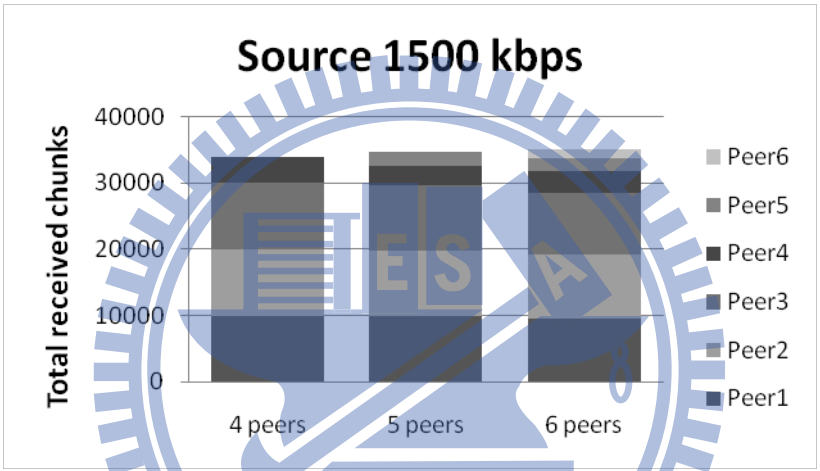


圖 18　相同 RTT，上傳頻寬 1500kbps，各個節點接收到的片段數

但是隨著來源節點的頻寬變大，貢獻最大的節點受到的影響也越來越小，如圖 18，當來源節點可傳送三個子串流時，Peer1 收到的片段數就比較不會被競爭頻寬的節點數影響，都可以收到將近 10000 個片段，但是 Peer3 在競爭節點數多的時候，接收速率還是會被貢獻小的節點搶走一些。

接下來再看各個節點接收速率隨時間的變化。

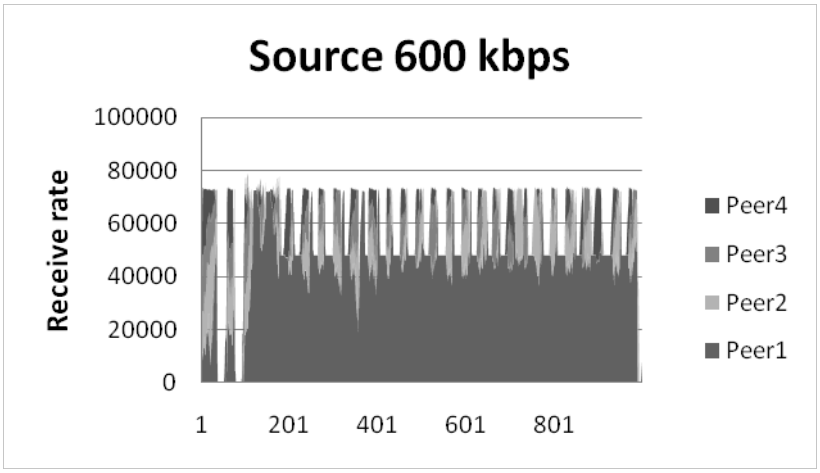


圖 19　相同 RTT，上傳頻寬 600kbps，各個節點接收速率的變化

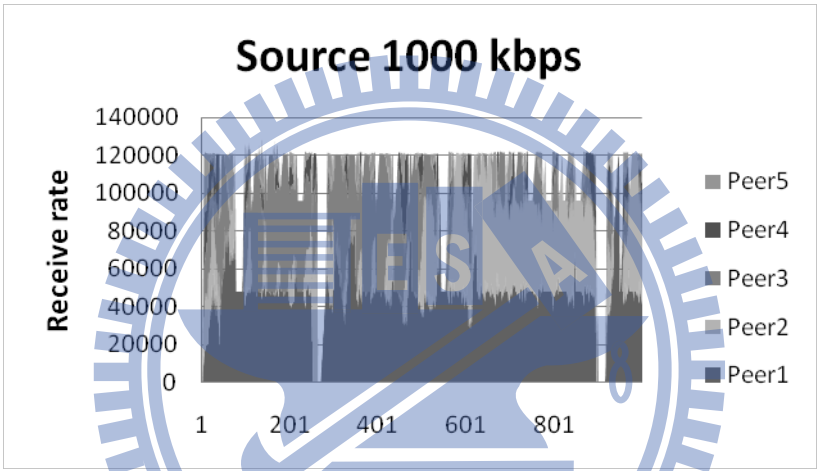


圖 20　相同 RTT，上傳頻寬 1000kbps，各個節點接收速率的變化

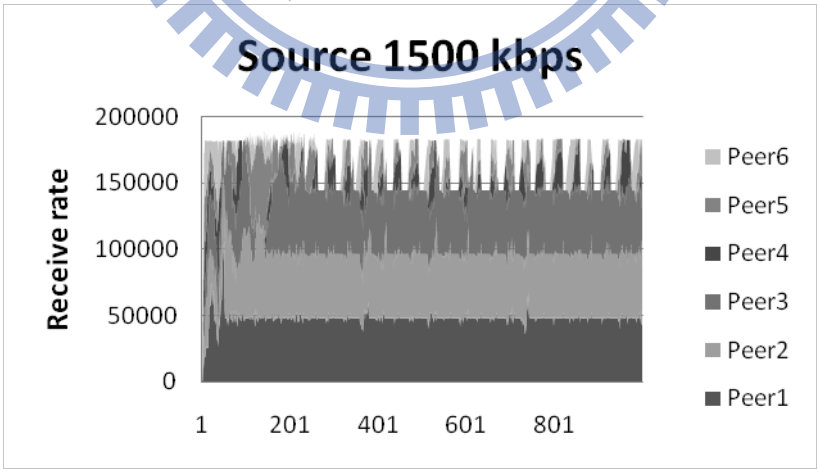


圖 21　相同 RTT，上傳頻寬 1500kbps，各個節點接收速率的變化

從圖 19、圖 20、圖 21 也可看出，當來源節點上傳頻寬大時，貢獻較大的節點的接收速率是比較穩定的。圖上端的振動產生的原因是，貢獻小的節點無法得

到足夠的接收速率，因此不斷的更換提供者，但是我們限制一個節點在更換提供者後，必須至少過 30 秒才能再次和舊的提供者要求資料，這個限制本來是爲了防止接收者更換提供者後，又跟相同的提供者要求子串流。但是這個部分的實驗，提供者只有來源節點一個，所以導致貢獻小的節點在放棄舊的提供者後，有一段時間找不到提供者，因此下載速率變爲 0 的情況。

接著來看不同 RTT 的情況下，考慮 RTT 調整每次 Cwin 增加的值，以及不考慮 RTT，兩者的表現差異。

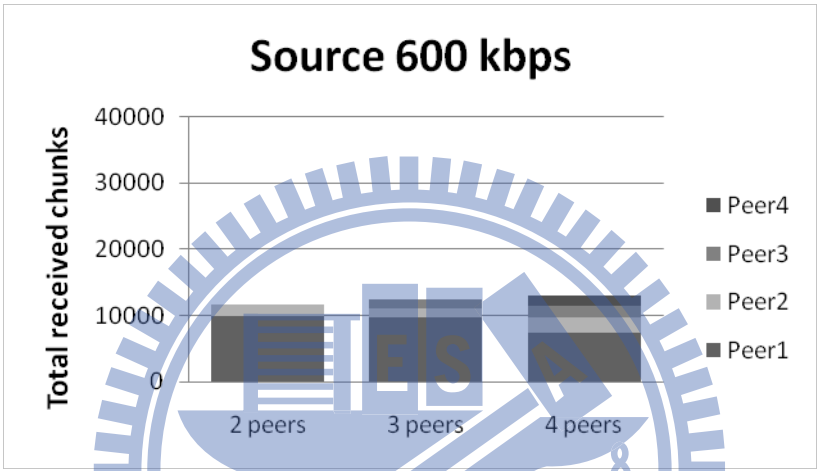


圖 22　考慮 RTT，上傳頻寬 600kbps，各個節點接收到的片段數

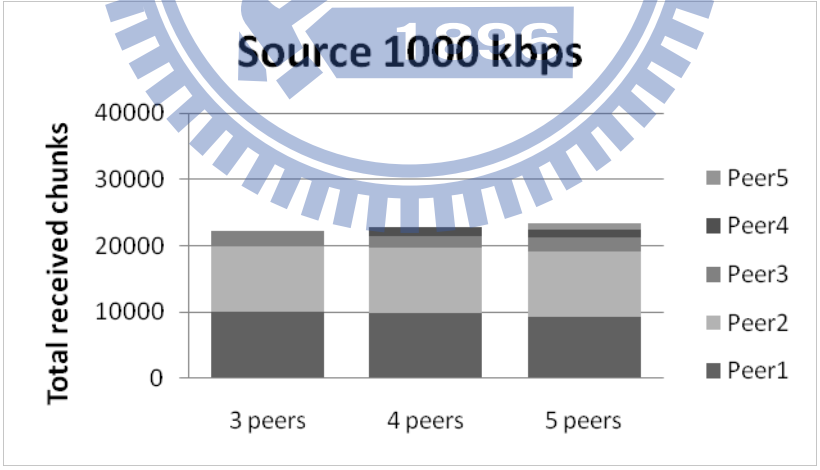


圖 23　考慮 RTT，上傳頻寬 1000kbps，各個節點接收到的片段數

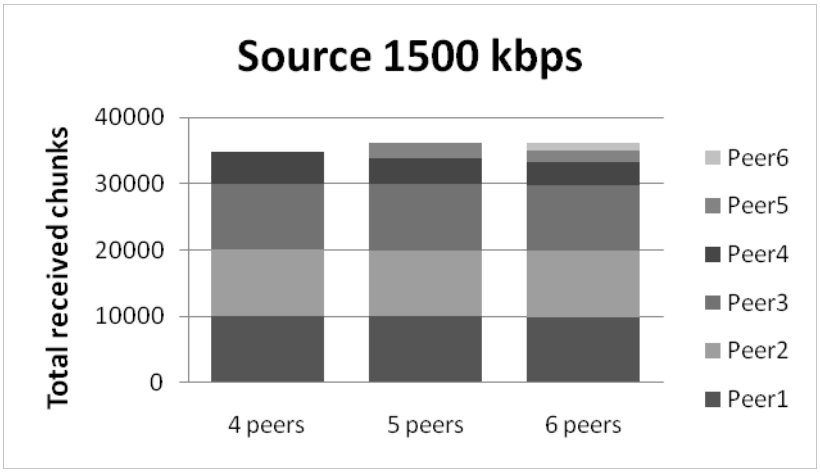


圖 24　考慮 RTT，上傳頻寬 1500kbps，各個節點接收到的片段數

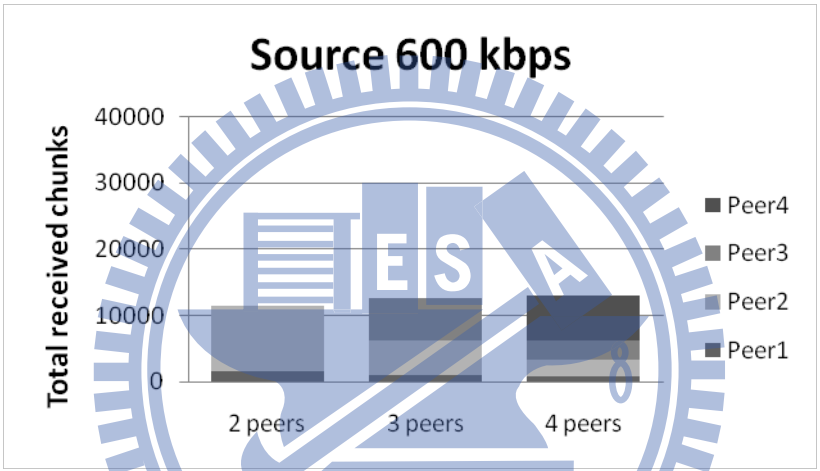


圖 25　不考慮 RTT，上傳頻寬 600kbps，各個節點接收到的片段數

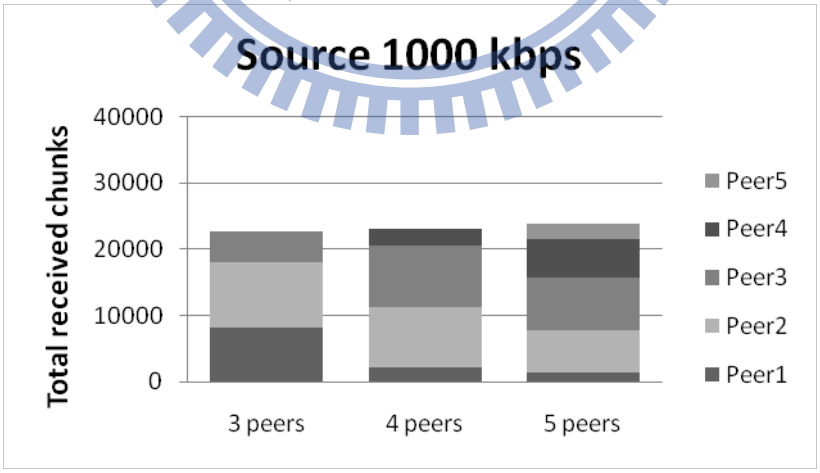


圖 26　不考慮 RTT，上傳頻寬 1000kbps，各個節點接收到的片段數

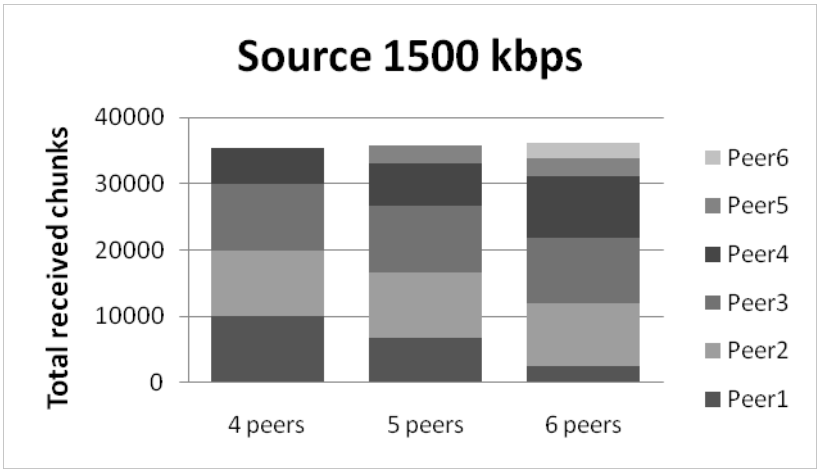


圖 27　不考慮 RTT，上傳頻寬 1500kbps，各個節點接收到的片段數

由圖 22 到圖 27，我們可以很明顯的發現，考慮 RTT 調整 Cwin 增加的值，可以使 RTT 不同時，仍然是貢獻大的節點接收到較多片段。若沒有考慮 RTT 來調整 Cwin 增加的值，則每個節點接收到的片段數會受到它的貢獻以及 RTT 的影響，以致貢獻大的節點並不一定能收到較多的片段。

5.3　BLACE 與 CBCC 的結合效能測試

在這一節中，我們會將 BLACE 與 CBCC 結合，測試它的效能，並與 Coolstreaming 和 Prime 做比較。在我們的方法中，每個節點會使用 BLACE 估計接收者的貢獻，並使用 CBCC 根據不同接收者的貢獻來分配到它們的傳輸速率。

5.3.1　模擬環境設定

在這部分的實驗，我們使用了比較複雜的網路拓樸，如圖 28 所示。

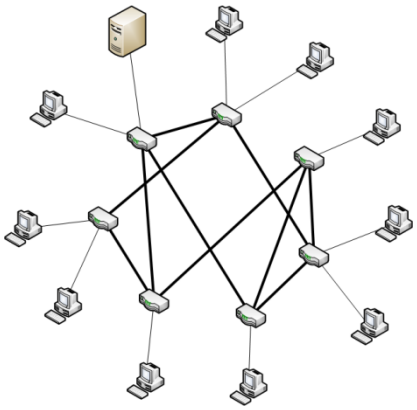


圖 28　CBCC 與 BLACE 結合測試使用的網路拓樸

中間的路由器並不會執行同儕網路的程式，而只是負責封包繞送。路由器彼此之間有隨機的連接，稱爲 Core link，具有較大的頻寬。同儕網路的節點都經過一條連接隨機連到路由器上，該連接稱爲 Edge link。這邊設定的網路環境是，每個節點都有足夠的下載頻寬，但是不一定有足夠的上傳頻寬，因此我們設定 Edge link的不同方向有不同的頻寬，路由器到節點的方向(下載)，頻寬一律是100Mbps，而節點到路由器的方向(上傳)，頻寬則隨機分布，且根據不同的測試環境有不同的分布。

我們會測試在不同網路環境下，本論文的方法和 Coolstreaming 以及 Prime 的表現。我們測試的不同網路環境包括來源節點的上傳頻寬不同、一般節點的平均上傳頻寬與分布不同，以及節點加入和離開的頻率不同。

我們會測試使用 CBCC 分別根據節點的頻寬、節點要求的子串流的延遲，以及節點使用 BLACE 估計出的貢獻來分配傳輸速率，這三種不同的速率分配方式的效能差異，並且與 Coolstreaming 以及 Prime 做比較。

本部分實驗的詳細參數設定列於表 5。

CBCC 與 BLACE 結合測試的實驗參數設定	
子串流數量 (Prime 不需要)	8
影像 bit-rate	384kbps
片段大小	4.8KByte
片段長度	0.1 秒
模擬時間	3600 秒
節點個數	1000
路由器個數	100
Core link 頻寬	100M-1000M 之間平均分布
Core link 延遲	1ms 到 100ms 之間平均分布
Edge link 頻寬	0-600K, 100-700K, 200-800K, 300-900K 之間平均分布
Edge link 延遲	1ms 到 100ms 之間平均分布
來源節點上傳頻寬	1.6M, 2.4M, 3.2M, 4.0M, 4.8M
節點加入離開頻率	每 10 秒 0, 1, 2, 3, 4 個節點

表 5　CBCC 與 BLACE 結合測試的實驗參數設定

5.3.2　實驗結果與分析

首先我們先測試沒有節點上下線的情況。在這個部分裡，我們設定來源節點的上傳頻寬爲 4.0Mbps，測試一般節點頻寬分布分別爲 0-600kbps、100-700kbps、200-800kbps、300-900kbps 時，各種方法的效能，以下是測試的結果。

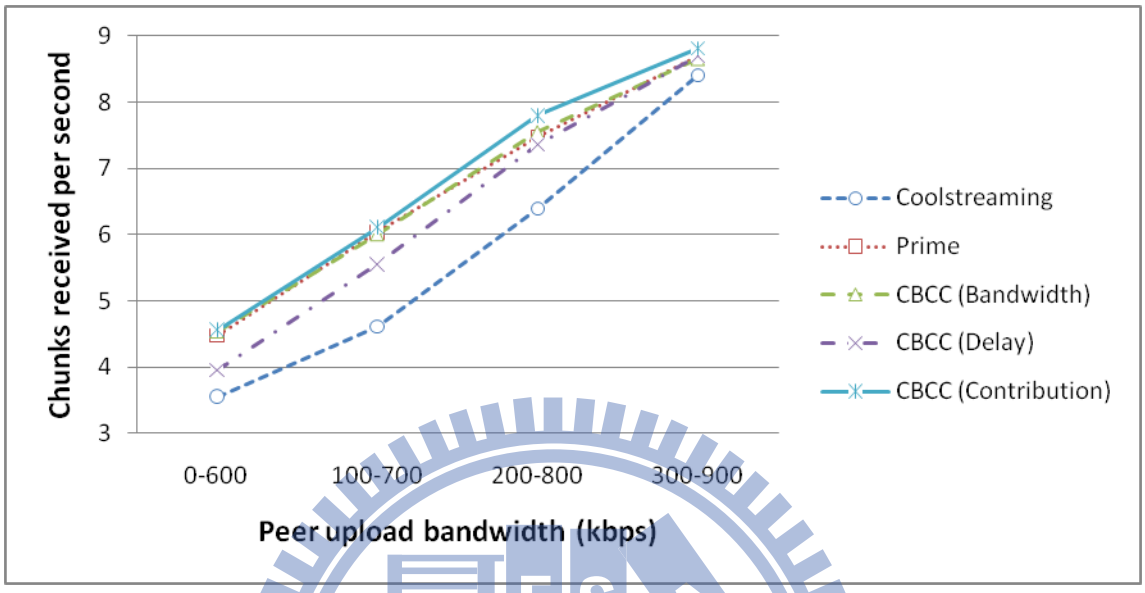


圖 29　沒有節點上下線，不同上傳頻寬分布下節點平均的接收速率

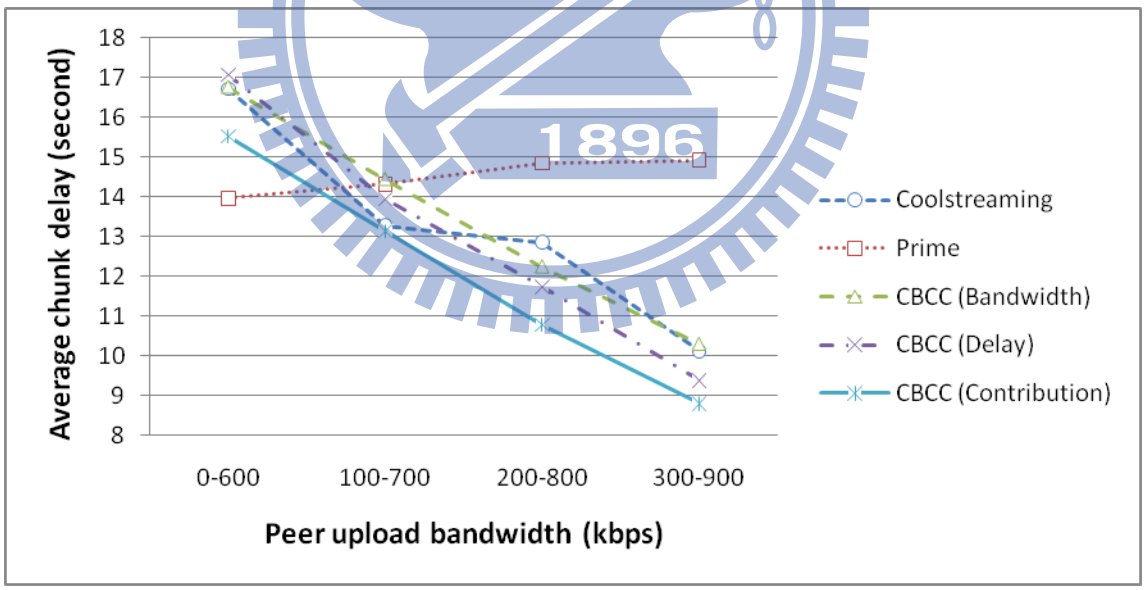


圖 30　沒有節點上下線，不同上傳頻寬分布下節點收到片段的平均延遲

從以上圖表可看出，除了頻寬分布在 0-600 kbps 時，Prime 收到的片段延遲是最小的之外，不管是在節點平均接收速率以及接收到的片段延遲方面，使用 CBCC 根據 BLACE 估計出的貢獻來分配頻寬，得到的效能都是最好的。

我們發現除了 Prime 之外，其他 方法的片段延遲都受到節點的上傳頻寬分布很大的影響。我們推測這是因爲，除了 Prime 之外的方法都是子串流式的架構，我們設定子串流式的架構中，除非子串流的延遲超過 20 秒，否則節點不會更換提供者，因此在節點上傳頻寬小的環境中，子串流傳輸的速度慢，造成延遲變大，但是不會超過 20 秒；而越大的頻寬可讓子串流傳輸越快，因而使得延遲降低。而 Prime 是片段式的架構，雖然節點上傳頻寬小的環境也會讓片段傳輸變慢，但是因爲片段式的架構中，節點必須要求它想要的片段，而超過播放時間的片段節點就不會再要求，因此就沒有很舊的片段來拉高片段延遲的平均，使得在節點上傳頻寬小的環境中，Prime 的平均片段延遲顯得比較低。

另外，我們發現節點上傳頻寬越來越大時，CBCC (bandwidth)和 CBCC (delay)在節點接收速率的表現越來越接近，我們推測是因爲節點平均上傳頻寬較小時，子串流在系統中容易傳到頻寬不足的節點，導致沒辦法繼續散布下去，因此優先考慮節點的頻寬可以避免這種情形。但是節點平均上傳頻寬提高後，這種情形自然就比較不會發生了。因爲子串流傳遞中斷的情形變少的關係，所以不管怎麼傳，終究是會傳到所有的節點，因此不管根據頻寬還是延遲來分配傳輸速率，反映在節點的接收速率上的結果都是差不多的。此時能夠改進的部分就是減低子串流散布到所有節點所花的時間，這時候考慮子串流的延遲來分配傳輸優先權就能得到比較好的效果。

另外值得一提的是，BLACE 估計貢獻時，會考慮系統的平均上傳頻寬，在平均上傳頻寬小時，節點的上傳頻寬在貢獻中佔的比重會增加，而在平均上傳頻寬大時，就換成是延遲會佔比較大的比重。

接著我們測試節點上下線頻率不同時，各種方法的效能。這個部分我們設定來源節點的上傳頻寬爲 4.0Mbps，一般節點的上傳頻寬平均分布於 300kbps 到 900kbps 之間。

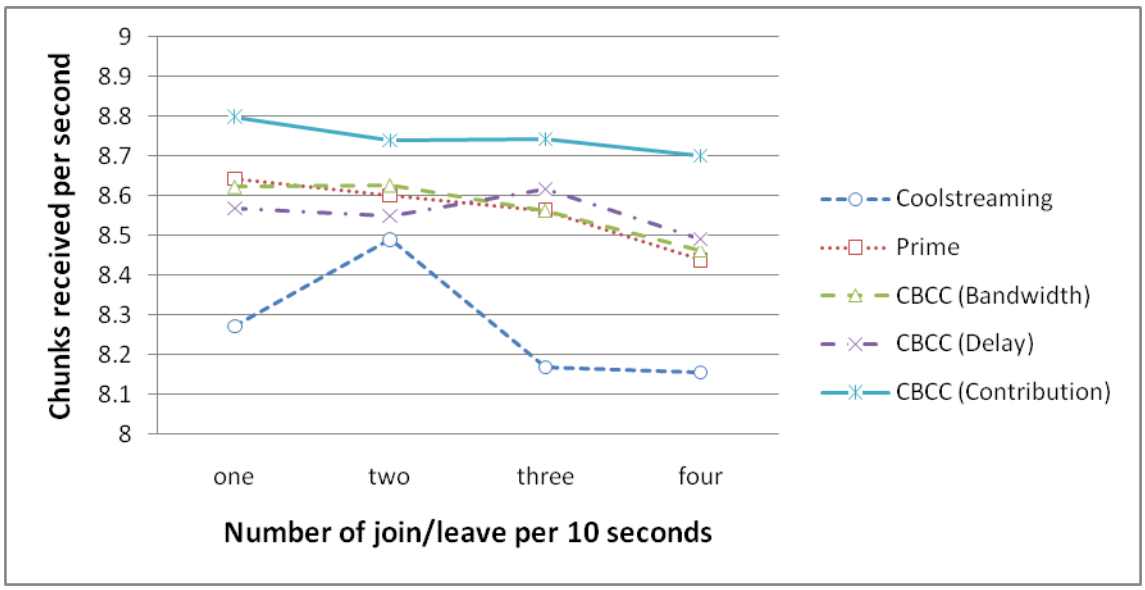


圖 31　上下線頻率不同時，節點平均的接收速率

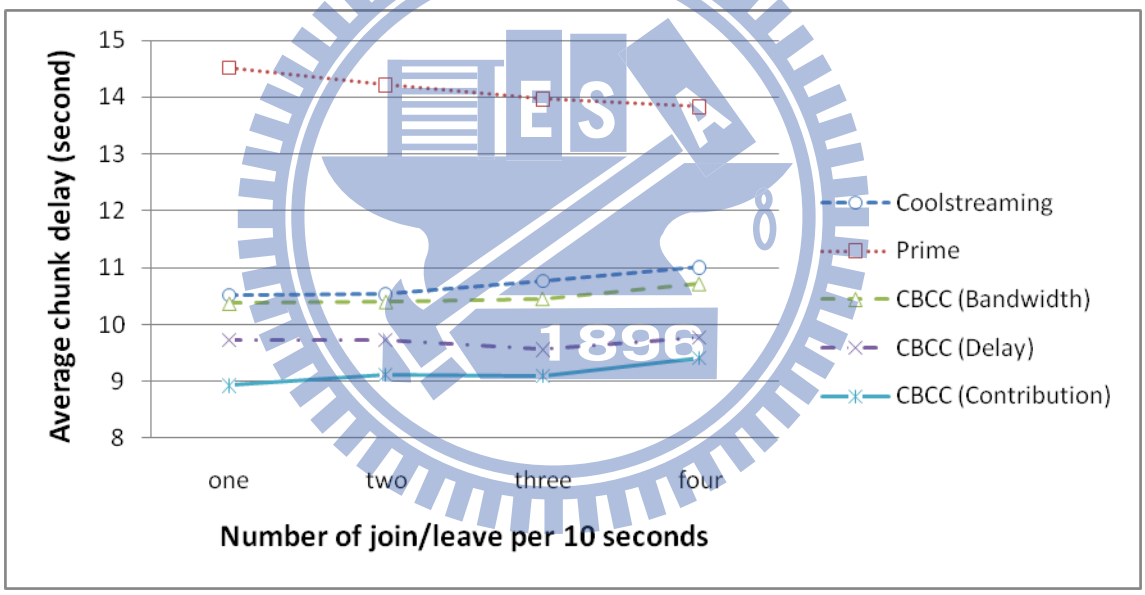


圖 32　上下線頻率不同時，節點收到片段的平均延遲

由以上圖表可看出，上下線頻率增加時，每種方法中，節點的平均接收速率都會降低。但是儘管如此，CBCC (Contribution)依然是表現最好的方法。

接著我們測試來源節點的上傳頻寬對系統的效能造成的影響。在這個部分我們設定來源節點上傳頻寬分別爲 1.6M、2.4M、3.2M、4.0M、4.8M，而一般節點上傳頻寬分布於 300kbps 到 900kbps 之間，每十秒鐘會有一節點上下線。

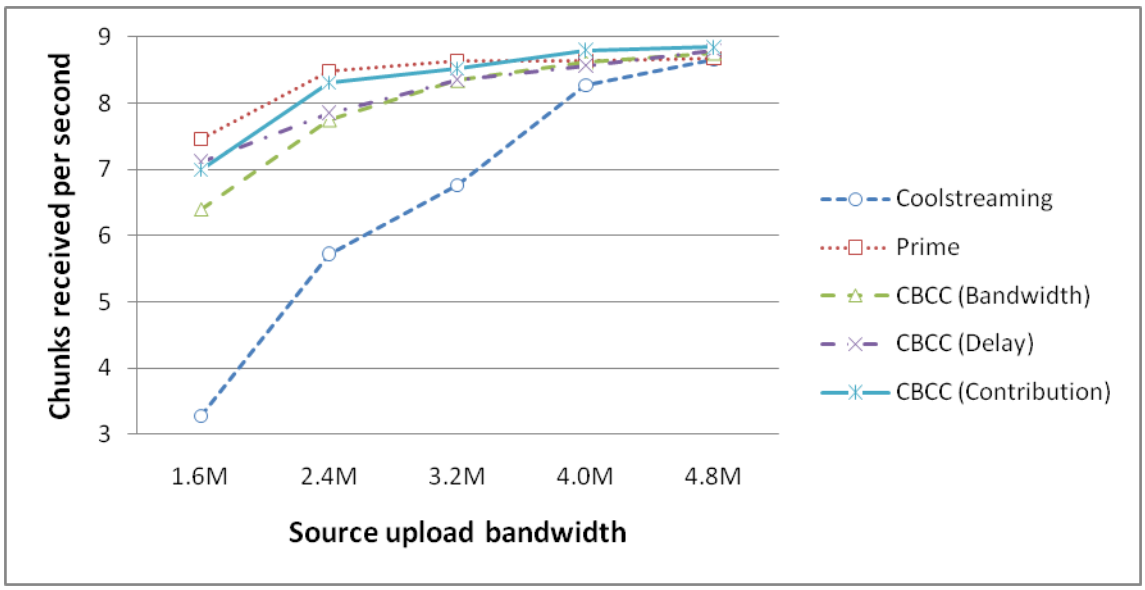


圖 33　來源節點上傳頻寬不同時，節點平均的接收速率

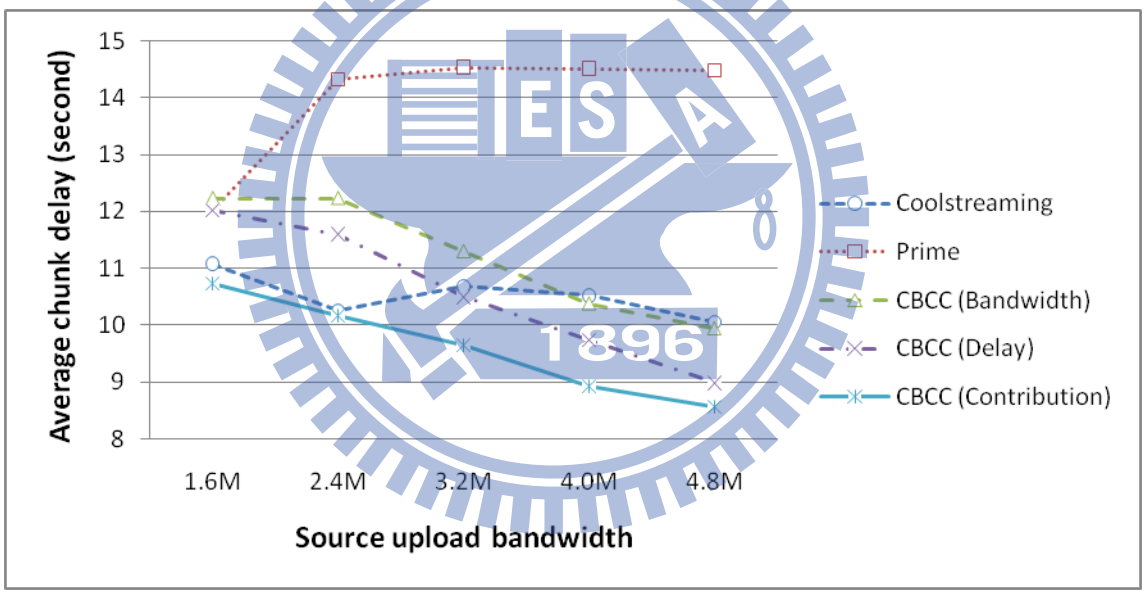


圖 34　來源節點的上傳頻寬不同時，節點收到片段的平均延遲

從上面圖表可看出接收速率方面，Coolstreaming 受到來源節點上傳頻寬的影響最大，我們推測這是因爲 Coolstreaming 使用 TCP 來傳送子串流，因此傳輸速率會平均分配在所有接收者，當來源節點頻寬不足時，造成的結果就是所有的接收者都無法收到完整的子串流。而 CBCC 在頻寬不足時，自然會把傳輸速率分到少數幾個接收者，因此效果會比使用 TCP 還要好。而 Prime 是片段式的架構，不需要提供者提供完整的子串流速率，因此受到來源節點上傳頻寬的影響最小。

再來我們來測試本論文提出的方法是否真的能達到傳輸較多的節點收到較多的資料。這部分我們設定來源節點上傳頻寬爲 4.0Mbps，一般節點上傳頻寬平均分布在 100kbps 到 700kbps，沒有節點上下線。我們會測試 CBCC 只考慮上傳頻寬跟考慮貢獻時，節點的傳輸速率跟接收速率的關係。

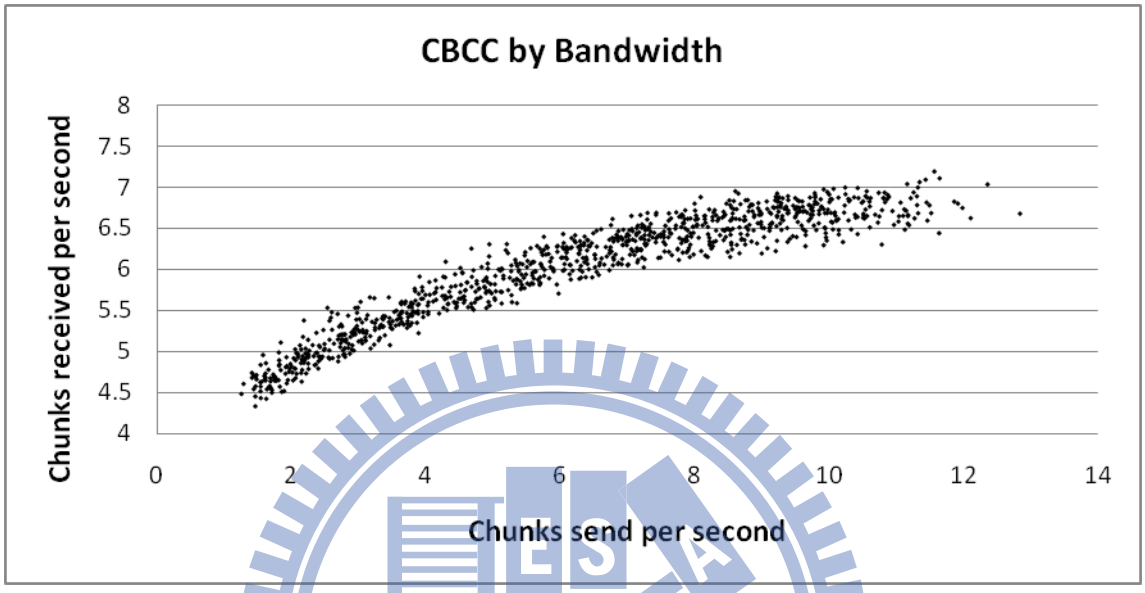


圖 35　考慮上傳頻寬時，節點的傳送速率跟接收速率的關係

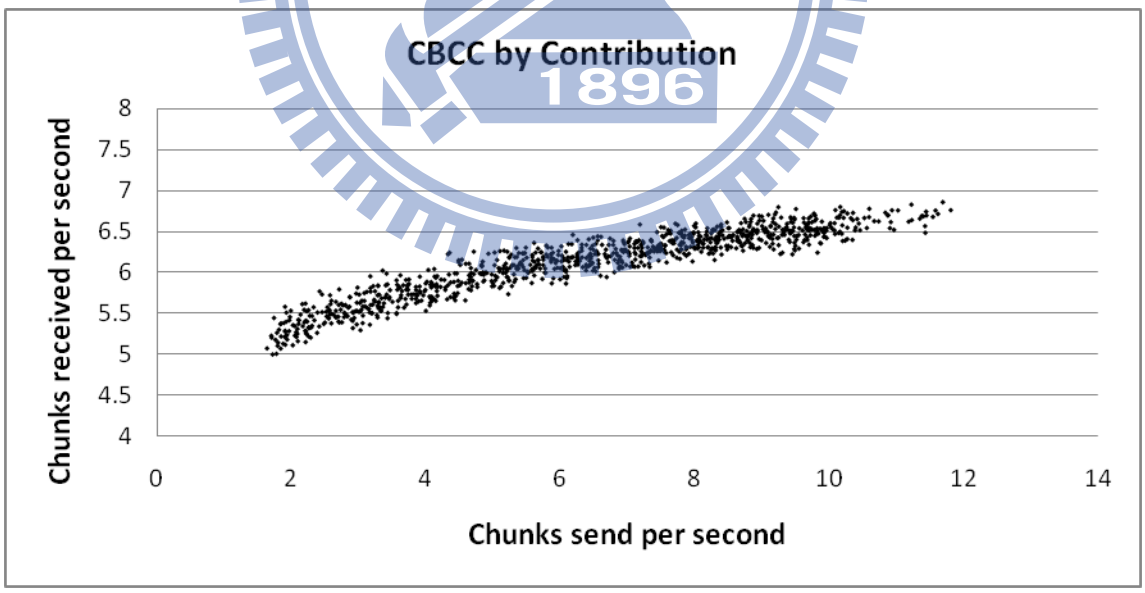


圖 36　考慮貢獻時，節點的傳送速率跟接收速率的關係

由以上圖表可看出，雖然兩種方法都可以達到傳送較多的節點接收較多的效果，但是在只考慮頻寬的狀況下，達到的效果是比較好的。

最後我們要測試 BLACE 所估計出的貢獻是否會跟實際的貢獻符合，在這個實驗中，我們設定來源節點上傳頻寬爲 4.0Mbps，一般節點上傳頻寬平均分布在 100kbps 到 700kbps，沒有節點上下線，並擷取系統運行了 2000 秒後的狀態。首先我們只看單一子串流。

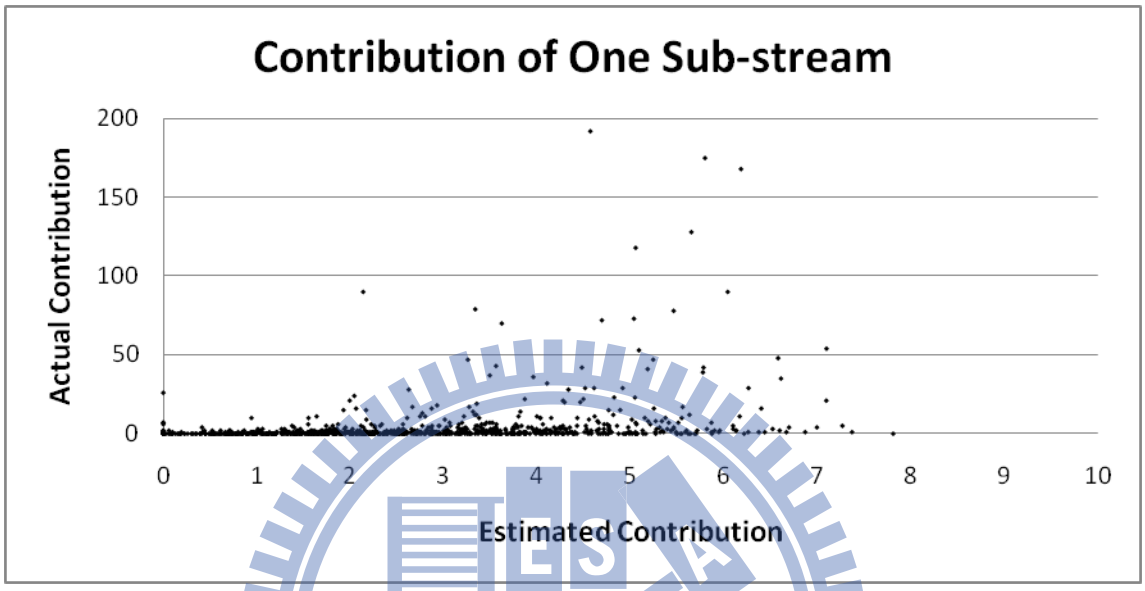


圖 37　節點對一個子串流的實際貢獻和估計的貢獻關係

由於大部分的點 Actual contribution 都分布在 200 以下，所以我們把縱座標軸的上限設爲 200，以便更清楚的看到分布的狀態。我們可以發現，就單一子串流而言，實際的貢獻(即節點把該子串流散布給了多少人)和估計出的貢獻沒什麼明顯的關係。

接著我們合計所有子串流的實際貢獻和估計貢獻，再來看它們的關係。

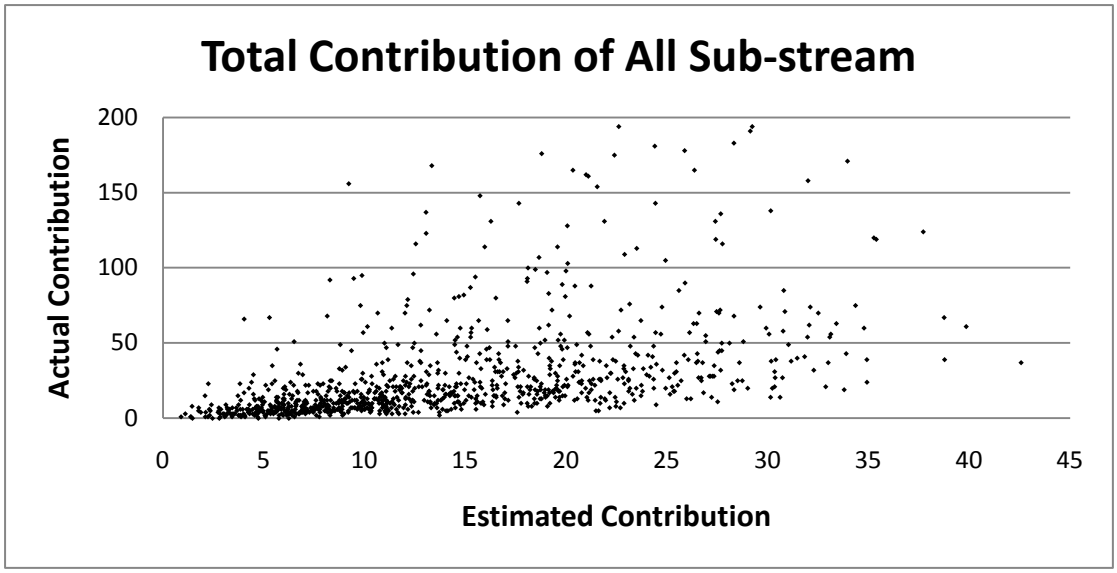


圖 38　節點對所有子串流合計的實際貢獻和合計的估計貢獻的關係

這次我們就可明顯的觀察到估計出的貢獻越大的節點，實際的貢獻也會越大，且底部較密集的部分的點的實際貢獻大略符合估計出的貢獻。但是我們也可以看到，估計出的貢獻和實際的貢獻其實還是沒有完全符合，我們推測的原因如下：第一，本論文提出的方法中，每個節點完全是隨機的選擇提供者，所以頻寬大的節點 A 可能選到頻寬小的節點 B 當作提供者，因此 A 的實際貢獻也會被算進 B 的實際貢獻，導致了頻寬小的 B 實際貢獻比頻寬大的 A 還大。第二，本論文假設每個節點會把頻寬平均的用來傳送每一個子串流，但是根據實驗結果，我們發現節點通常會把頻寬集中用來傳一個子串流。下圖就是我們觀察到的結果。

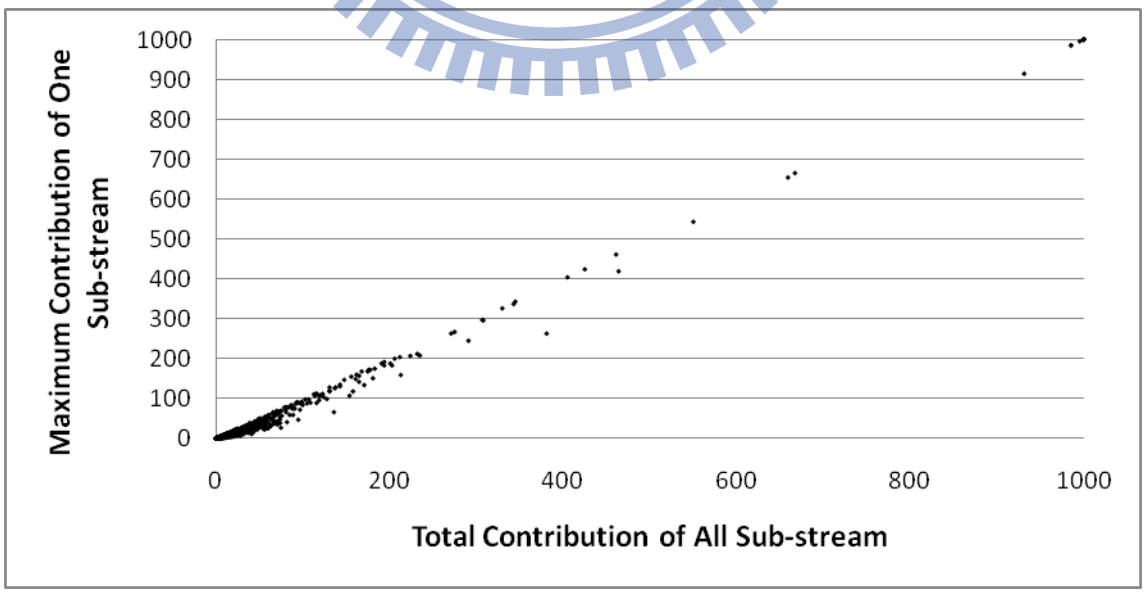


圖 39　節點對所有子串流合計的實際貢獻和對單一子串流最大的實際貢獻的關係

圖 39 的橫軸是每個節點對所有子串流的實際貢獻的合計，縱軸是該節點對貢獻最大的子串流的實際貢獻。我們可以發現節點對單一子串流最大的實際貢獻幾乎等於所有子串流的實際貢獻總和，也就是說，節點的貢獻幾乎都集中在一個子串流上，這也就代表節點會傾向把頻寬集中用來傳送一個子串流，而本論文未考慮到這點，這也是之前只看一個子串流時，實際貢獻和估計的貢獻無法符合的原因，因爲節點的貢獻可能集中在其他的子串流。

第六章　結論與未來工作

6.1　結論

本論文提出兩個機制，BLACE 會根據一個節點的上傳頻寬、連線延遲，以及該節點要求的資料延遲程度來估計該節點的貢獻，而 CBCC 可以根據不同節點的優先權來分配到不同節點的傳輸速率。且根據我們實驗的結果，使用 CBCC 來分配傳輸速率，不論是根據節點頻寬、要求的資料延遲，或是 BLACE 估算出的貢獻，表現都比 Coolstreaming 好。而根據 BLACE 估算出的節點貢獻分配傳輸速率，除了來源節點上傳頻寬小的環境，其他環境表現都比 Prime 好。

由於 Coolstreaming 使用 TCP 來傳輸串流，因此傳輸速率會平均分配在所有接收者，當上傳頻寬不足時，所有的節點都無法以足夠的速率收到子串流，以致於節點會頻繁的更換提供者。而 CBCC 對每個接收者都有不同優先權，當頻寬不足時，傳輸速率會自動分配到優先權高的接收者，所以只有優先權低的接收者會因爲無法接收到足夠速率而更換使用者，因此使用 CBCC 的表現會比使用 TCP 還要好。

另外，隨著系統的平均上傳能力不同，分配優先權時是根據節點的上傳頻寬，或是根據子串流的延遲，也會造成不同的結果。當系統平均上傳能力低時，子串流有可能因爲傳到頻寬不足的節點，以致無法繼續傳播，因此在這種環境下，讓大頻寬的節點有高的優先權，就可以減低子串流傳輸中斷的機率，而提高系統的效能。但是若系統的平均上傳能力高時，子串流傳輸中斷的機率很低，因此不論如何分配優先權，子串流都能夠散布到全部的節點，在這種環境下，先傳輸較新的資料可以更進一步的降低節點收到的資料延遲程度。而本論文提出的 BLACE 兼顧了這兩個方面，當系統平均上傳頻寬低時，節點的貢獻主要取決於節點的上傳頻寬，而當系統平均上傳頻寬高時，節點要求的資料延遲程度就會對貢獻造成比較大的影響，因此，根據 BLACE 算出的貢獻來分配優先權，效果會比只考慮節點上傳頻寬或只考慮資料延遲程度都還要好。

6.2　未來工作

本論文只探討提供者如何分配接收者的傳輸速率的部分，而接收者找尋提供者時，依然是使用隨機尋找的方式，因此有可能會造成大頻寬的節點卻位在底層的情況，而小頻寬的節點會因爲與它競爭頻寬的節點都比較強，因此很難找到可用的提供者。爲了解決這些問題，可以讓每個節點都根據自己的頻寬，挑選與自己相近的節點做爲鄰居(假設來源節點有最大的頻寬)，如此一來每個節點也都只跟自己頻寬相近的節點要求資料，就能更確保大頻寬的節點位在上層，而小頻寬的節點也因爲跟自己競爭頻寬的都是小頻寬的節點，所以就比較有機會找到可用的提供者。此外，本論文估計貢獻的模型中，未考慮節點對不同的子串流也會有不同的傳輸優先權，而只簡單假設節點會平均的把頻寬用來傳輸每個子串流。若將不同子串流的優先權也納入考慮，相信估計出的貢獻會更準確。

參考文獻

[1] X. Hei, Y. Liu, and K. Ross, "IPTV over P2P Streaming Networks: The Mesh-Pull Approach," IEEE Communications Magazine, vol. 46, issue 2, pp. 86-92, FEB 2008.

[2] R. Kumar, Y. Liu and K. Ross, "Stochastic Fluid Theory for P2P Streaming Systems," IEEE Conference on Computer Communications, pp 919-927, MAY 2007.

[3] N. Magharei, R. Rejaie and Y. Guo, "Mesh or Multiple-Tree: A Comparative Study of Live P2P Streaming Approaches," IEEE Conference on Computer Communications, pp. 1424-1432, MAY 2007.

[4] D.-A. Tran, K.-A. Hua and T. Do, "ZIGZAG: An efficient peer-to-peer scheme for media streaming," IEEE Conference on Computer Communications, pp 1283-1292, MAR 2003.

[5] V.-K. Goyal, "Multiple description coding: Compression meets the network," IEEE Signal Processing Magazine, vol. 18, issue. 5, pp 74-93, SEP 2001.

[6] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron and A. Singh, "SplitStream: High-bandwidth content distribution in cooperative environments," International Workshop on Peer-to-Peer Systems, pp 292-303, FEB 2003.

[7] V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy and A.-E. Mohr, "Chainsaw: Eliminating trees from overlay multicast," International Workshop on Peer-to-Peer Systems, pp 127-140, FEB 2005.

[8] V. Venkataraman, K. Yoshida and P. Francis, "Chunkyspread: Heterogeneous unstructured tree-based peer-to-peer multicast," IEEE International Conference on Network Protocols, pp 2-11, NOV 2006.

[9] S. Xie, B. Li, G.-Y. Keting and X. Zhang, "Coolstreaming: Design, theory, and practice," IEEE Transactions on Multimedia, vol. 9, issue 8, pp 1661-1671, DEC 2007.

[10] N. Magharei and R. Rejaie, "PRIME: Peer-to-Peer Receiver-Driven Mesh-Based Streaming," IEEE-ACM Transactions on Networking, vol. 17, issue 4, pp 1052-1065, AUG 2009.

[11] Y.-W.-E. Sung, M.-A. Bishop and S.-G. Rao, "Enabling contribution awareness in an overlay broadcasting system," IEEE Transactions on Multimedia, vol. 9, issue 8, pp. 1605-1620, DEC 2007.

[12] D. Dutta, A. Goel, R. Govindan and H. Zhang, "The design of a distributed rating scheme for peer-to-peer systems," Workshop on Economics of Peer-to-Peer Systems, June 2003.

[13] S. Buchegger and J. Boudec, "A robust reputation system for p2p and mobile ad-hoc networks," Workshop on Economics of Peer-to-Peer Systems, June 2004.

[14] A. Akella, S. Seshan, S. Shenker and I. Stoica, "Exploring Congestion Control," Carnegie Mellon University Computer Science Technical Reports, 2002
http://reports-archive.adm.cs.cmu.edu/

[15] M.-S. Kim, T. Kim, Y.-J. Shin, S.-S. Lam and E.-J. Powers, “A Wavelet-Based Approach to Detect Shared Congestion,” IEEE-ACM Transactions on Networking, vol. 16, issue 4, pp 763-776, AUG 2008.

[16] E. Gavaletz and J. Kaur, “Decomposing RTT-Unfairness in Transport Protocols,” IEEE Workshop on Local and Metropolitan Area Networks, pp 1-6, May 2010.

[17] PlanetLab
http://www.planet-lab.org/

[18] I. Baumgart, B. Heep and S. Krause, “OverSim: A scalable and flexible overlay framework for simulation and real network applications,” IEEE International Conference on Peer-to-Peer Computing, pp 87-88, SEP 2009.

[19] S. Naicken, A. Basu, B. Livingston and S. Rodhetbhai, “A Survey of Peer-to-Peer Network Simulators,” Annual Postgraduate Symposium, JUNE 2006.

[20] The Network Simulator - ns-2
http://www.isi.edu/nsnam/ns/