3 系統架構

在本章中將針對本論文所提出以階層式多重描述編碼法則為基礎之 peer-to-peer 視訊串流傳輸系統的整個架構作一個完整的介紹。首先會說明整個系統的網路拓樸配置狀況，再介紹整個模擬 peer-to-peer 的方式，最後介紹我們建構於 peer-to-peer 網路上的階層式視訊串流系統，由下往上的方式將整個系統作一完整的描述。

3.1 網路拓樸設計

在本論文中，為了模擬目前一般網路上的視訊串流環境，我們主要分成兩個部分來設計網路拓樸：

第一部分為無變動節點，其中又可分為兩種節點，一種為 backbone node，這是整個網路骨幹中最主要的節點，整個 backbone 都是由這些節點所架設起來。另一種為 gateway node，此部分指的是一些 ISP 所架設的路由器，是由 backbone node 所延伸出來的。這些節點屬於網路骨幹部分，在整個網路中屬於較不易變動的節點，圖 3.1 是一個 transit-stub topology 的簡單範例，其中我們可以看到棕色的範圍內為 transit domain，其中的黑色節點即為 backbone node，而每個 backbone node 都會與數個 stub domain 相連，stub domain 即為圖 3.1 中白色圓圈

的部分，而灰色節點代表的即為 gateway node。在本論文中我們使用 GT-ITM(Georgia Tech Internetwork Topology Models)[19]產生含有 100 個節點的 transit-stub 網路作為整個網路的骨幹。

第二部分為變動節點，也可以稱作為 leaf node，此部分指的是網路中最末端的使用者，我們的 peer-to-peer application 均架構在此類型的節點之上。我們在上述的骨幹網路中，將所有的 backbone 及 gateway node 隨機加入數個 leaf node，每一個 leaf node 在網路中所代表的是一個 ADSL 使用者，而在 peer-to-peer 系統中則代表的是一個 peer。

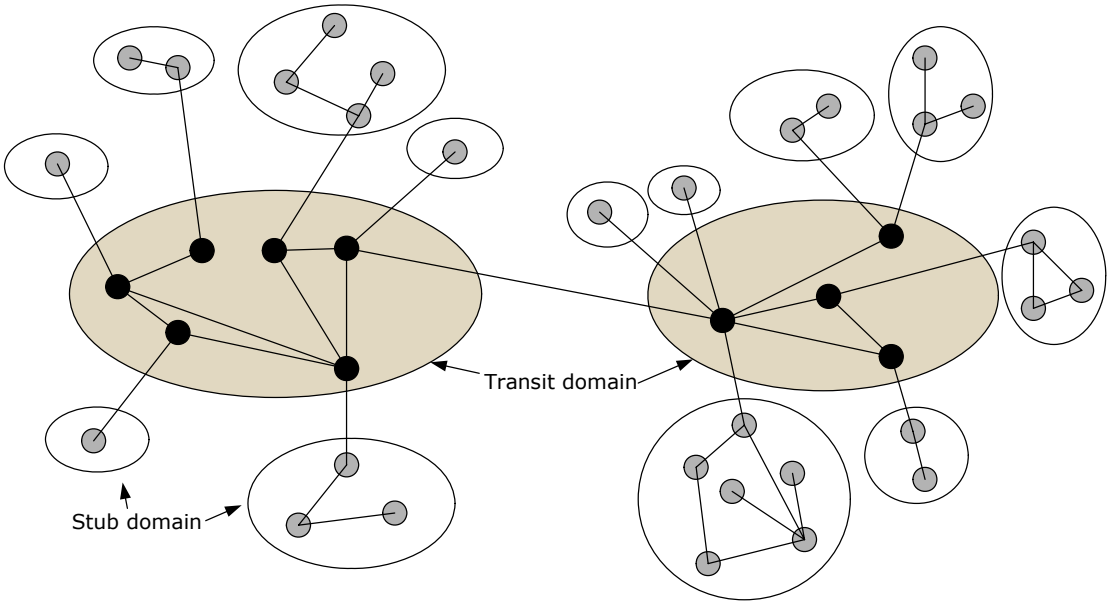

圖 3.1 Example of Transit-Stub topology

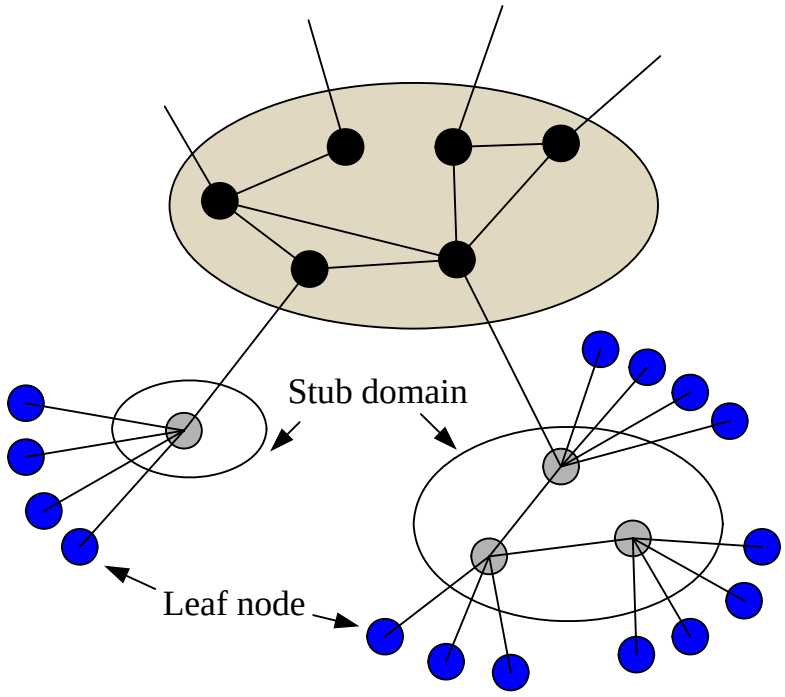

圖 3.2 Leaf Node 在 Transit-Stub Topology 中的連結情形

在圖 3.2 中我們可以看到藍色部分的 node 即為 Leaf node，在我們的架構中，gateway node 代表的為 ISP 機房中的 router，而 leaf node 為 ADSL 使用者。使用這樣的網路拓樸，我們建立了一個簡單的網路模型，用來發展我們的視訊串流系統。

3.2 建構 Peer-to-Peer 網路模擬環境

在本論文中我們採用 NS2 作為我們整個架構的模擬工具。在 NS2 中並沒有可模擬 peer-to-peer 系統的現成元件，為了要模擬出 peer-to-peer 系統，我們使用了 GnutellaSim[15]這個 framework。

GnutellaSim 是為了模擬 peer-to-peer 系統在 packet-level 下，封包實際的傳送

狀況，而產生的一組 framework。它的功能是將 peer-to-peer 系統中每一個 peer 的行為，經由此 framework 將使用者行為轉換為實際上的網路封包。經過這樣的一個轉換過程後，我們可將這些封包放入網路模擬器中進行模擬，即可得到一個 peer-to-peer 系統的模擬環境。

根據 GnutellaSim 的設計理念，凡是屬於 packet-level 的網路模擬器，均可與 GnutellaSim 結合，進而模擬出 peer-to-peer system 環境。GnutellaSim 的原始開發平台即為 NS2，在本論文中我們結合這兩者來作為 peer-to-peer 的網路模擬環境。

3.2.1 GnutellaSim 架構

在 GnutellaSim 的架構中，每個 peer node 之中可以分為三層式的結構，在圖 3.3 中我們可以看到由上而下可分為 PeerApp、PeerAgent 以及 Socket Adaptation Layer 三層。

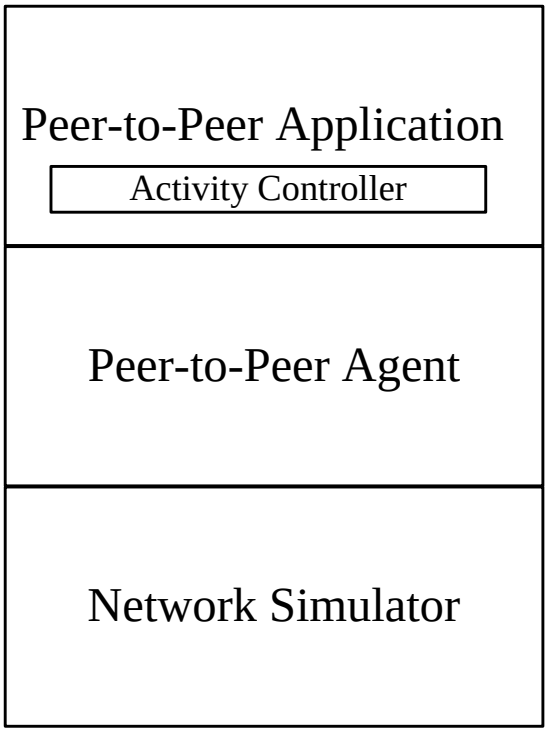

圖 3.3 GnutellaSim 之三層式架構圖

分成三層架構的好處在於各層可分別抽換，而不置於影響其他層的運作。舉例來說，Socket Adaptation Layer 代表的是網路模擬器，將其獨立成為一層，代表凡是支援 Packet-level 的網路模擬器均可套用於此層之中。在實作上，也會出現許多 peer-to-peer application 使用同一種 protocol 的情形，比如說 LimeWire[23]、Gnucleus[24]等，均是基於 Gnutella protocol 之上所發展出來的 peer-to-peer application。某些時候可能我們會比較某兩套軟體的差別，這時候我們只要在 Application 層中作抽換，而無需修改其下的 Protocol 層，這就是分層架構的目的。

3.2.2 Application Layer (PeerApp)

對於 peer-to-peer 系統的使用者來說，PeerApp 是使用者最直接所面對的介面，它所代表的是各種使用者的操作行為。雖然各種不同的 peer-to-peer application 會有不一樣的操作行為，但是大部分的操作行為是具有共同性的。因此，我們可以將 applcation 的各種操作行為作出以下的分類：

- 使用者行為：一些使用者操作軟體的動作，會導致觸發某些 peer-to-peer protocol 訊息，或是改變 peer 的狀態，這類型的行為包括像 Join、leave、idle、search 等等。這類型的行為是屬於由使用者所觸發的。

- 系統行為：為了維持與 peer-to-peer 網路的正常連繫，application 往往會自動與其他 peer 傳遞一些相互溝通訊息，以得知整個網路的節點狀態，確保資料能夠正確的被傳輸無誤。這樣的行為使用者通常是並不會看到的，但卻是讓通訊正常運作的關鍵。

為了能夠在模擬的環境中產生出以上所說的使用者或系統行為，在 GnutellaSim 中，作者實作了一個名為 ActivityController 的模組，該模組的功用便是根據給定的系統參數，以一定的機率來隨機模擬出各種使用者的行為，或是自動的產生系統行為，以維持 peer 與 peer 間的連繫。在實作上，PeerApp 是使用 Timer 來製作，當到達特定的時間時(Time-driven)，ActivityController 會自動

的觸發事件，來產生使用者行為或改變 peer 的狀態。

3.2.3 Protocol Layer (PeerAgent)

PeerAgent 層代表的是各種不同的 peer-to-peer 協定，雖然 GnutellaSim 在此層中是使用 Gnutella protocol 來實作，但由於 protocol 獨立為一層，因此我們也可使用其他具有不同特性的 peer-to-peer protocol 來模擬，比如說結構式的 protocol，如 Chord[17]或是 Pastry[22]等。

3.2.4 Socket Adaptation Layer

這一層的主要功能，是將 PeerAgent 所傳遞的 Protocol Message 轉換為實體上的封包，並交給模擬器去模擬封包的傳送。在 GnutellaSim 的實作中，為了能夠達到 portable，在這一層中設計了 socket-based 的介面，提供了一些必要的 Socket 動作：bind, listen, Connect, send, recv, and poll 等。PeerAgent 在模擬過程中會將 Protocol 的 Message 轉換成各種 socket 的操作，並利用這一層所提供的 Socket 指令進行模擬。

3.3 階層式多重描述視訊串流系統架構

3.3.1 系統概觀

我們的階層式視訊串流系統主要是利用 peer-to-peer 網路來搜尋視訊串流的來源，在找到視訊串流來源後，便向來源端發送視訊串流要求，以達成視訊串流的目的。在本論文整個視訊串流的流程部分可用圖 3.4 來表示：

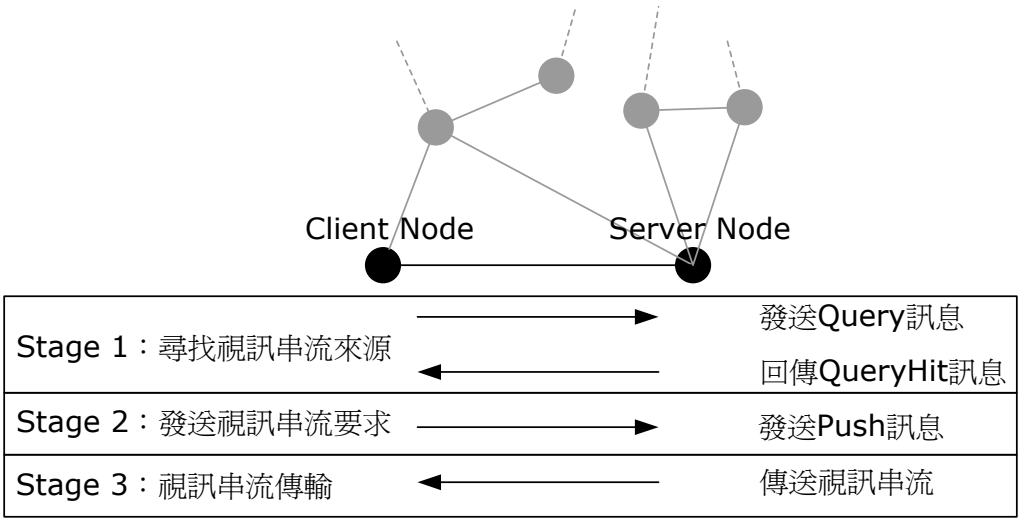

圖 3.4 視訊串流傳輸流程

上圖中我們將整個視訊串流的過程分為三個階段，由 peer-to-peer 系統使用者開始要求視訊串流開始，我們定義該使用者為 Client Node，第一個階段是發送 Query 訊息，向網路上其他 peer 詢問是否有該視訊串流的來源，當被詢問的 peer 擁有該視訊的來源，我們定義該 node 為 Server Node，Server Node 會回傳一個 QueryHit 的訊息給 Client Node，當 Client Node 在一段時間後，會根據所收到

的所有 QueryHit 進行分析，選擇最佳的視訊來源，此時便進入第二階段，Client Node 對最佳的視訊來源發送視訊串流要求，當 Server Node 收到 Client Node 所發送的視訊串流要求後，則進入第三階段，Server Node 開始傳送視訊串流給 Client Node。這樣的一個流程就是本系統進行視訊串流的步驟。

從 Client Node 的角度來看，我們可以將以上視訊串流的步驟以圖 3.5 的有限狀態機(Finite State Machine)來表示：

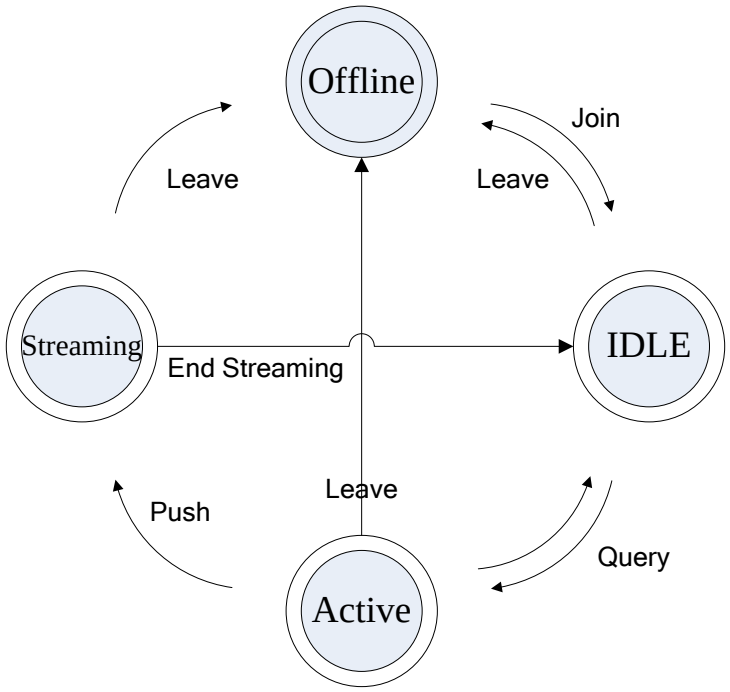

圖 3.5 Client Node 的有限狀態機

圖 3.5 有限狀態機中的各種狀態是根據 Client Node 中使用者的行為而設計，一開始使用者均為 Offline 狀態，當進入 peer-to-peer 網路(Join)後會變為 IDLE 的狀態，此時是 Online 的情形，但並未作任何的動作。一旦使用者作出 Query 的動作後即變成 Active 的狀態，當 Client Node 決定視訊來源後會發送視訊串流

要求，則進入 Streaming 狀態，當視訊串流傳輸完畢後即回到 IDLE 狀態。Client Node 總是在此四種狀態中進行變換。

由於本系統是多重描述編碼，因此在進行視訊串流時會同時有多個來源端。接收端在播放時會根據圖 3.6 的情形來進行視訊串流的播放：

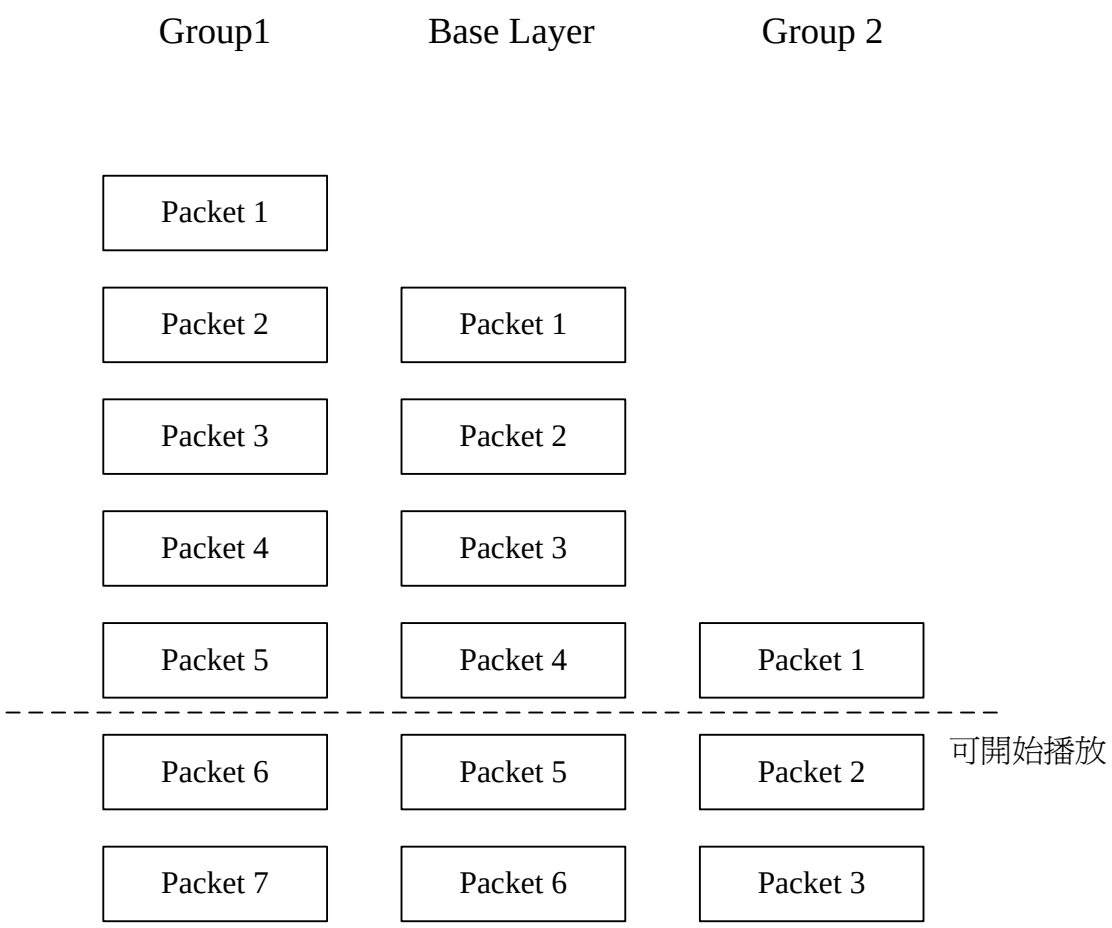

圖 3.6 在多重視訊來源之下的視訊播放情形

在圖 3.6 中我們可以看到系統在接收到 Group1 的第一個視訊串流封包時並不會立刻開始播放，因為其他群組的 Packet 1 仍未抵達。當 Group2 的第一個視訊串流封包到達，此時便可開始播放，在本系統中會視串流長度先擷取一定程度

的緩衝區(視訊長度的 1/3)，以降低網路品質波動所造成的視訊品質影響。Base Layer 指的是視訊在多重描述編碼過程中將各群組重疊之低頻部分將其獨立為 Base Layer，也視為一個群組。

接下來的三個小節我們將分別就視訊串流過程中的三個階段，對本系統作更深入的介紹。

3.3.2 尋找視訊串流來源

在尋找視訊串流的部分，我們使用了 Gnutella Protocol 中的 Query 訊息來對整個 peer-to-peer 網路進行詢問，根據 2.2 節中所提出的多重描述編碼法則，本系統中的視訊串流皆使用多重描述進行編碼，因此同一段視訊可分為多個群組，分別存在於 peer-to-peer 網路中不同的 peer 上。當 peer 接收到來自於其他 peer 的 Query 訊息時，會檢視本身是否擁有該 Query 所需要的視訊的其中一個群組，若發現本身擁有該視訊的某一群組，則會利用 Gnutella Protocol 中的 QueryHit 訊息，回傳一個 QueryHit 訊息，讓詢問者得知我們擁有他所需要的部分視訊，並在 QueryHit 中告知我們所有的是該視訊的哪一個群組。詢問者接收到 QueryHit 訊息之後，會將所得到的 QueryHit 收集為一列表(QueryHit List)，在一定的時間後，根據 QueryHit List，便可找到此 peer-to-peer 網路中擁有目標視訊的列表。

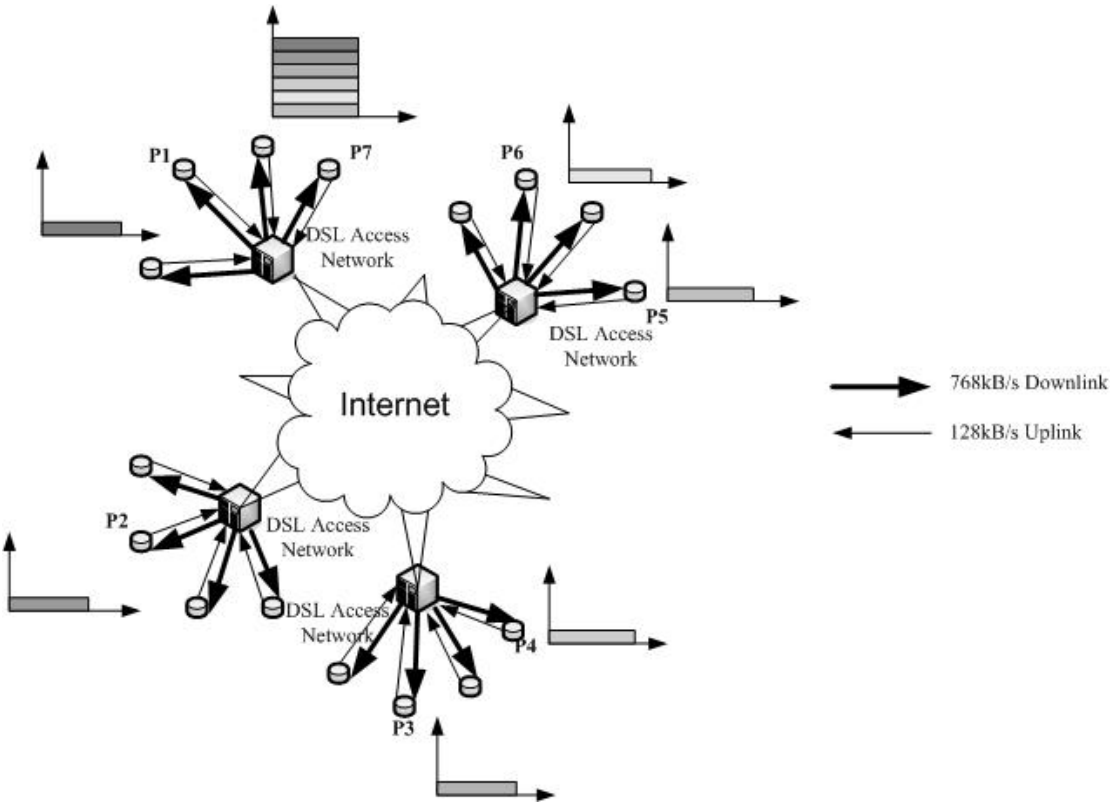

圖 3.7 使用多重描述編碼法則之 peer-to-peer 視訊串流系統

圖 3.7 顯示了一個簡單的視訊串流系統架構圖。假設每一個使用者皆利用 ADSL 連接至網際網路中，且視訊影像利用多重描述編碼法分為 6 個群組，各自分散儲存於 P1 至 P6 的 peer 中。當使用者 P7 需要進行視訊串流時，則對整個 peer-to-peer 網路發送 Query，擁有該視訊中某個群組的 P1 ~ P6 會回傳 QueryHit 給 P7，則 P7 可得到一個 QueryHit List(P1, P2, P3, P4, P5, P6)，P7 就可根據 QueryHit List 來對 P1 ~ P6 發送視訊串流要求

假設由於網路頻寬或傳輸延遲的變化造成 P3 及 P4 的封包無法及時到達 P7，則 P7 仍舊可以利用 P1、P2 及 P5、P6 之封包以較低之品質進行還原，如此的方式可減低頻寬變化或傳輸延遲對於視訊串流接受端的視訊品質影響。同理，

假設 P3 及 P4 擬離開 peer-to-peer 系統，則 P7 也不會因為 P3 及 P4 之離開而造成視訊串流之中斷。同時我們也可以發現即使每個使用者的上傳頻寬有限，無法單獨對一個接受端提供完整的視訊影像，但卻有能力提供部分的視訊內容(指一個群組)，因此透過多重描述編碼法，使得每個使用者都能對於整個 peer-to-peer 系統作出貢獻。

3.3.3 發送視訊串流要求

根據上一節的說明，我們可得到一個 QueryHit List，則我們可在該 List 中尋找最佳的視訊來源，進而發送視訊串流要求。在本系統中我們針對視訊串流之特性，對於 Gnutella Protocol 的 QueryHit 作了一點修改，我們在 QueryHit 訊息中新增了一個叫作 bandwidth 的欄位，此欄位的定義為：

$$Bandwidth = \frac{\text{總上傳頻寬}}{\text{目前服務使用者數目}+1}$$

舉例來說，假設一節點 P 的上傳頻寬大小為 512Kbps，正在提供視訊給 4 個使用者所使用，則若是第五位使用者發送 Query 訊息到 P，P 節點會回傳一個 QueryHit，而其中的 Bandwidth 為 128Kbps(512/4)，第五位使用者可根據此欄位來決定可能的最佳視訊來源。因此使用者在發送視訊串流要求時，可對 QueryHit List 作分析，根據 Bandwidth 找到每個群組的可能最佳視訊來源節點，進而對若干節點發送視訊串流要求。

節點	群組	Bandwidth
P1	Group 1	256Kbps
P2	Group 1	128Kbps
P3	Group 2	512Kbps
P4	Group 2	64Kbps

表 3-1 節點 P 的 QueryHit List

在表 3-1 中我們可以看到一個簡單的 QueryHit List 的範例，當節點 P 搜尋完視訊串流來源之後，得到如表 3-1 的一個 QueryHit List，則節點 P 會對 QueryHit List 進行分析，找出各群組的最佳來源節點。從表中我們可以看出 P1 及 P2 擁有群組 1，P3 及 P4 擁有群組 2。我們先看群組 1 的部分，若是向 P1 發送視訊串流要求，可以得到 256Kbps 的服務，但若是向 P2 要求，則可能只能得到 128Kbps 的服務，因此經過分析後我們會選擇 P1 作為群組 1 的最佳視訊串流來源節點，同理可知 P4 為群組 2 的最佳來源節點。以上的敘述可整理為表 3-2。

節點	群組	Bandwidth	最佳來源
P1	Group 1	256Kbps	Group1 來源
P2	Group 1	128Kbps	
P3	Group 2	64Kbps	
P4	Group 2	512Kbps	Group2 來源

表 3-2 節點 P 的已分析 QueryHit List

在決定了每個群組的最佳視訊來源之後，我們則對來源節點發送視訊串流的要求。此部分我們利用了 Gnutella Protocol 的 Push 訊息，Push 原先的設計是當使用者無法穿透防火牆抓取資料時，會發送 Push 給資料提供者，請資料提供者主動連線至接收端，以類似 FTP 的 Passive 模式，被動的餵送資料給接收端。而

視訊串流無法直接抓取，也是一樣要被動的等待視訊串流資料，所以在我們的架構中，我們使用了 Push 代表向來源節點要求視訊串流。來源節點在接收到 Push 之後，即會開始傳輸視訊串流到接收端。

3.3.4 視訊串流品質動態調整

從視訊提供者的角度來看，在視訊串流的傳輸過程中，可能遇到新的接收端要求視訊串流服務，當使用者所服務的對象數目越來越多，所送出的串流資料超出上傳頻寬，則整個服務的 response time 便開始下降。而由於視訊串流需要穩定的網路品質，一旦 response time 過高，造成封包無法在 playback 之前送到接收端，則視為封包遺失，造成視訊品質的嚴重下降。

為了解決這樣的問題，我們在本論文中應用了階層式多重描述編碼法則(Layered Multiple Description)，將視訊串流進行階層式多重描述編碼，來增加每一個使用者可服務之接受端數目。階層式多重描述法則有二個主要步驟，第一個步驟將輸入影像分解為若干群組，第二個步驟則將每一個群組進行階層式編碼。因此接受端除了僅需要接受部份群組資料便可進行還原之外，對於每一個擬接受的群組，接受端也只需要接受部份階層即可還原。倘若某一個使用者擬服務二個或二個以上的接受端，則該使用者不需要單一群組內所有階層之資料傳送給單一的接受端，而是將半數或是更少的階層資料傳送給每一個接受端。如此雖然上傳

之頻寬有限，但每個 peer-to-peer 使用者皆可服務多個接受端，對於正在接收服務的接收端來說，所得到的視訊串流品質雖略有下降，但 response time 仍舊能夠保持在一定的時間，而不至於超出 playback 時間，使得封包完全丟棄。

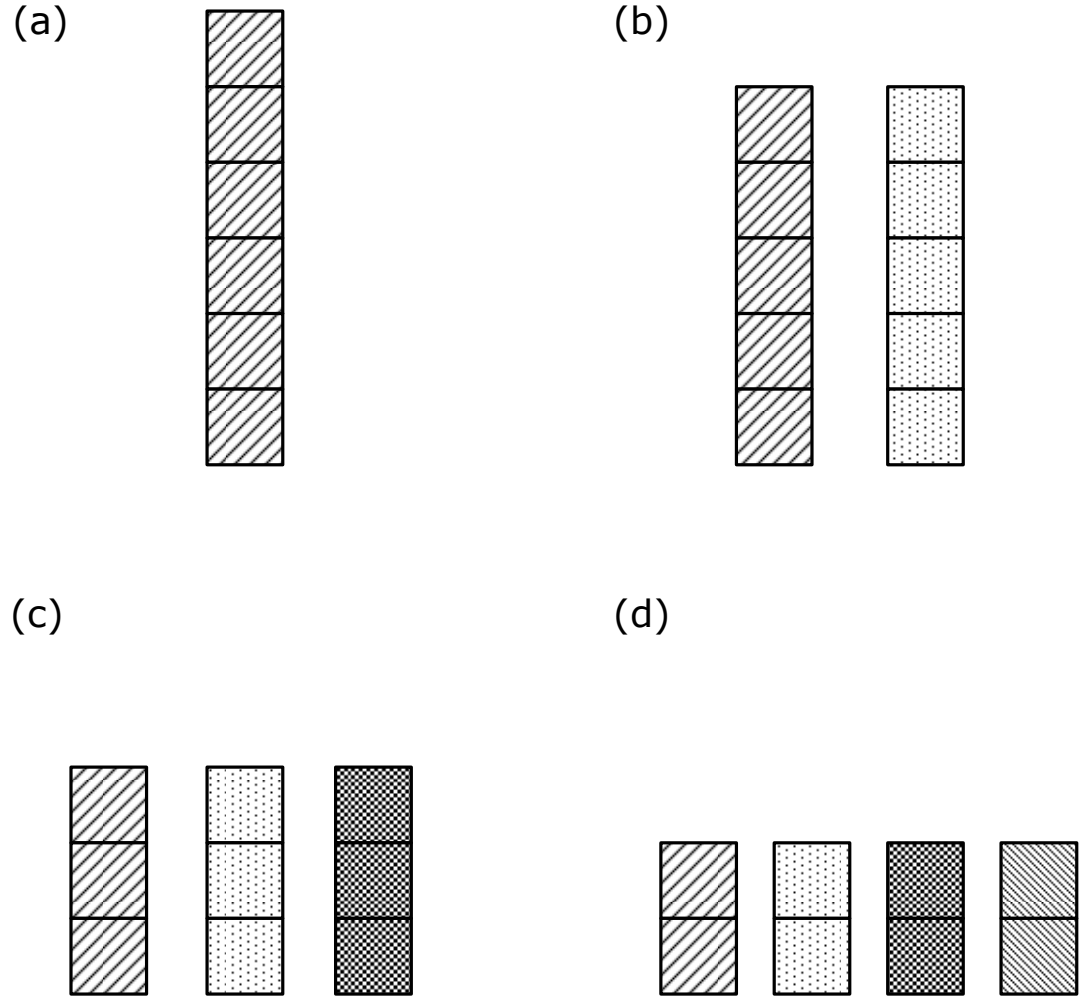

圖 3.8 一個 peer-to-peer 使用者提供串流服務之情形

圖 3.8 顯示依據本節所提出之方法，一個 peer-to-peer 使用者執行傳送服務的情形。假設不同的接受端皆要求不同的視訊資料，每一個群組的資料皆被分為 6 個階層，而每個階層所需之頻寬完全相同。我們定義 b 為傳送每一個階層所需要之頻寬，並假設傳送端上傳之頻寬為 $10b$。如圖 3.8(a)所示，若僅有一個使用

者要求服務，則該傳送端將完整的一個群組(亦即 6 個階層)之資料提供給此接受端。若有二個使用者要求服務，則傳送端會用罄上傳之頻寬。若再加上一個新的服務要求，傳送端需對每一個現有的服務對象削減二個階層，如圖 3.8(c)所示。該傳送端最多可提供資料給 10 個接受端，每個接受端僅接受到一個群組中的最底層資料。