第三章 鏈路層

傳輸訊號之訊框化、通訊連線的管理、流量控制、錯誤偵出、自動重複請求等有關資料的傳輸技術。

3-1鏈路層簡介

在我們探討『**鏈路層**』(Data-Link Layer)(或稱為資料連結層) 之前，我們必須假設有關訊號在網路上傳輸 (每位元訊號) 的各種問題已在實體層 (Physical Layer) 處理完成，我們不需要再去考慮傳輸上的各種問題 (如編碼技巧)。鏈路層主要是考量傳送端如何透過傳輸媒介將『一筆資料』(一連串的二進位資料) 安全地傳送給接收端；當接收端接收到該筆資料時，又如何去判斷資料是否正確。圖 3-1 表示鏈路層應具有之功能，分述如下：

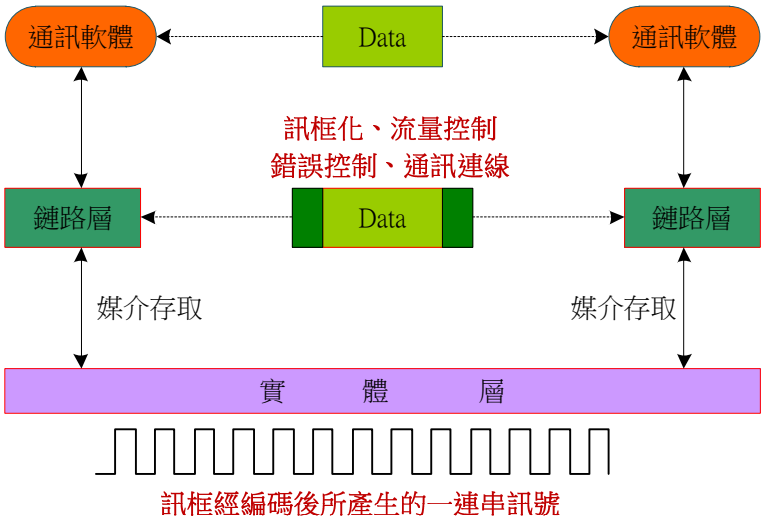


圖 3-1 鏈路層之功能

(A) 訊框化

鏈路層將一個位元串列 (Bit Stream) 資訊傳送給實體層，實體層再依序將每個位元轉換成訊號發送到傳輸媒介，在傳輸媒介上產生一段連續的訊號，因此這個位元串列就稱之為『訊框』(Frame)。也就是說，訊框是工作站在傳輸媒介上傳送的單位，它的長度直接影響傳輸媒介的傳輸效率。而在連續訊號當中，相對於每個訊框的起始與結束也都必須能被辨識出來，這便是訊框化的功能。

(B) 通訊連線管理

鏈路層必須建立工作站之間實體的連線，包括連線的啟動、結束、及連線狀態的管理。同時鏈路層也針對不同的通訊軟體（網路層）提供有：非連接服務、連接服務、非連接附確認服務等三種服務類別。

(C) 流量控制

必須處理傳送與接收之間資料流動的控制，協調傳送設備送出的訊框速度不能高於接收端所能吸收的速度，使通訊連線的效率提到最高。

(D) 錯誤偵出

必須具有「偵測訊框在傳輸媒介中傳送是否發生錯誤」的功能。訊框若要具有可偵測錯誤的功能，傳送之前須做適當的處理（如加入同位元），接收端才能依協議的方法（如檢查同位元）來判斷是否發生錯誤，此協議方法稱為錯誤控制。

(D) 自動重送請求

當接收端發現資料錯誤，如何自動請求傳送端重新傳送。或者，傳送端將資料發送後，如何判斷資料已遺失，而必須重新傳送。

(E) 媒介存取機制

鏈路層必須負責如何取得傳輸媒介的使用權。在點對點或交換式的網路中，每部工作站都有專屬連線；但於多重存取網路上（如 Ethernet 網路），必須制定公平競爭的方法，讓各個工作站能公平取得傳輸媒介的使用權，稱為『媒體存取控制』。針對各工作站的鏈路層也必須有一套定址方法，以分辨不同的來源和目標位址。

鏈路層的架構和實體層之間的關聯較為密切。一般為了適合各種不同的環境需要，而產生不同的網路架構，這些多半以鏈路層來分辨。譬如，區域網路中的 Ethernet 網路、Token-Ring 網路，或較大型網路如 FDDI 網路、ATM 網路，甚至最近風行的寬頻網路。本章以一般鏈路層應具有的基本功能來介紹，針對各種不同網路的特殊功能，也會在相關章節裡介紹到。

其實通訊協定的各個層次所使用的通訊技術，幾乎與鏈路層裡所介紹的技術大同小異，例如流量控制、通訊連線控制、錯誤控制等等。因此我們只要將鏈路層的各種技術研習清楚，以後各個層次就很容易瞭解。

3-2訊框化

到目前為止，我們瞭解『訊框』(或稱資料框，Frame）是鏈路層傳送給實體層的資料封包。它以位元流（Bit Stream）的序列（連續的二進位資料）傳送給實體層，實體層再依序將位元資料轉換成訊號傳送出去，於是在傳輸媒介會上出現一連串的訊號，如圖 3-1 所示。因此，鏈路層必須將資料分割及包裝，使規格大小和內容適於轉換成訊號，也適合於媒介上傳送，這個過程稱為**『訊框化』(Framing)**。

通常工作站欲將資料發送到傳輸媒介之前，會先將原資料(或上層的協定資料單元，PDU）分割成若干個訊框，再依序發送到傳輸媒介上；接收端依序收到訊框後，再將這些訊框組合回原來資料封包（或 PDU），如圖 3-2 所示。分割成較小的訊框，主要有下列兩個原因：

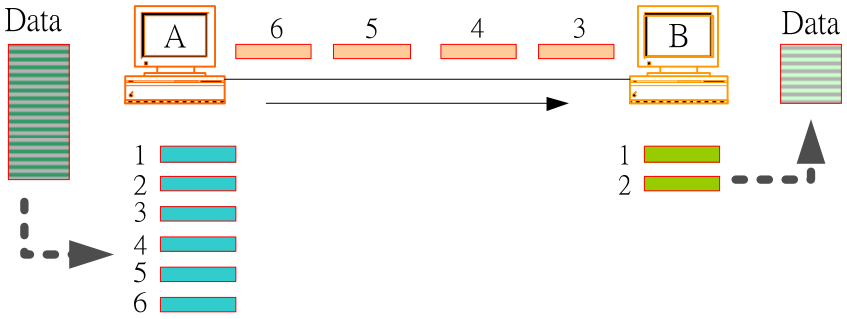


圖 3-2 資料分割傳送

(1) 錯誤控制：資料在傳送過程中，可能會有某些位元發生錯誤，假使一筆未經分割的大資料（假設有 10 Mbits）出了錯，則整個（10 Mbits）的資料都必須重送。如將原始資料分割成若干個小訊框（2 ~ 4 Kbits），即使其中某一訊框發生錯誤，也只需重新傳送該訊框即可。

(2) 共享媒介：一般網路中，傳輸媒介是共用的，如果不將資料分割成若干小訊框，當某一部工作站佔用傳輸媒介並且傳送一筆大資料時，將會佔用過長的時間，而造成其它工作站必須等待很久，也會使網路的整體傳輸效率降低。

當鏈路層以位元流（Bit Stream）的方式將訊框發送到網路上，而在傳輸媒介上產生一連串的訊號。接收端收到該訊號串列時，得去判斷訊號框的開始和結束，因此，訊框必須作特殊的包裝，使接收端可以輕易判別訊框的開始和結束。另外，在訊框裡也必須加入適當的控制訊息，使雙方的通訊協定可充分表達。一般來說，網路上的訊框結構可分為：字元導向（Character-oriented）和位元導向（Bit-oriented）兩大類。一般網路依其功能及環境因素或許稍有差異，但皆以這兩大類為基礎，以下分別述之。

3-2-1 字元導向鏈路控制

『**字元導向鏈路控制**』（Character-Oriented Link Control）是於訊框內插入一個特殊『控制字元』（Control Character）（如 DLE）來表示訊框的開始和結束，因而如是稱之。一般通訊協定都會將訊框分為若干個欄位，每個欄位皆有其特殊功能，以圖 3-3 (a) 訊框為例，一般常用的控制欄位有下列幾種：

- **SYN（Synchronous idle）**：同步空閒訊號。一般都是連續 0 和 1 交替資料之訊號，接收端接收到 SYN 訊號即開始調整本身的時序，使同步於 SYN 訊號，也表示同步於傳送端（請參考 1-8-3 節介紹）。
- **STX（Start of Text）**：利用一個特殊字元，表示後面緊接著傳輸字元（Text）。
- **ETX（End of Text）**：傳輸字元結束後之特殊字元，表示傳輸資料已結束。
- **BCC（Block Check Character）**：訊框的錯誤控制字元
- **DLE（Data Link Escape）**：啟動控制字元。表示後面緊接著為特殊控制字元，如 STX 及 ETX。

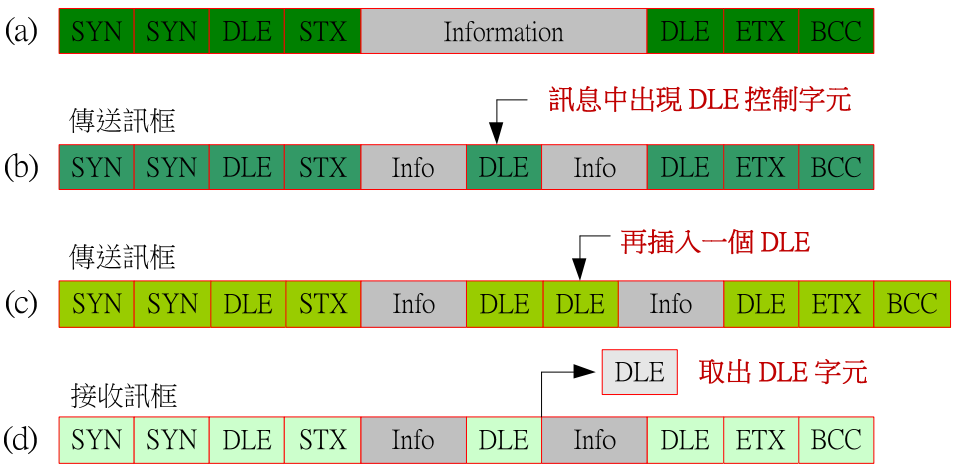


圖 3-3 字元導向訊框結構及字元填塞

圖 3-3 (a) 為正常訊框結構，以 DLE 控制字元表示啟動訊框開始(STX)和結束(ETX)。我們也可以用『DLE STX』來表示訊框的開始，以『DLE ETX』表示結束，這就是『字元導向』(Character-Oriented) 的基本原理。但 DLE 字元也可能出現在訊息欄位內 (即為傳送資料的一部分)，造成接收端判斷的錯誤，以為訊框已結束、開始、或誤解為其他意義，如圖 3-3 (b) 所示。因此，當鏈路層發送訊息時，必須檢查訊息內是否有出現 DLE 字元。如果有出現 DLE 字元，則必須在 DLE 字元後再插入一個 DLE 字元，以聲明該 DLE 是訊息而非控制字元，此動作稱之為『字元填塞』(Character Stuffing)，如圖 3-3 (c) 所示。在訊息之中需要傳遞多少個 DLE，就必須相對插入多少個 DLE 字元；接收端收到 DLE 字元時，如緊接著又收到 DLE，便可判斷前一個 DLE 是訊息而非控制字元，並將第二個 DLE 字元剔除。如圖 3-3 (d) 所示。

3-2-2 位元導向控制鏈路

『**位元導向鏈路控制**』(Bit-Oriented Link Control) 不再以控制字元來標示訊框結構，而是用一個旗號 (Flag) 來表示一個訊框的開始與結束。這個旗號是一組位元字串所組成，一般使用 8 個位元的式樣 (Pattern) 『01111110』，因此稱之為『位元導向』(Bit-Oriented)。圖 3-4 為一般位元導向的訊框結構，其中各欄位功能如下：

Preamble：前置訊號，48 位元。接收端用來調整時序，使同步於傳送端 (請參考 2-8-3 節介紹)。

- **Flag**：旗號欄位，8 位元。以位元式樣『01111110』來表示訊框的開始和結束。
- **Address**：位址欄位，16/32 位元，可變長度。其中包含來源和目的地兩個位址，一般每個位址會用 8 個位元 (8 bits) 或 16 個位元 (16 bits) 來表示。
- **Control**：控制欄位，8/16 位元，可變長度。傳送端將控制訊號放在控制欄位，作為標示本訊框的類型、以及通訊雙方溝通交換控制訊息時使用。
- **Information**：訊息，可變長度。傳送端傳送給接收端之訊息。

- **FCS（Frame Check Sequence）**：訊框的檢查碼，16/32 位元。接收端利用 FCS 來偵測訊框是否發生錯誤。

Preamble	Flag	Address	Control	Information	FCS	Flag

圖 3-4 位元導向之訊框結構圖

由圖 3-4 可知，整個訊框的開始和結束是利用位元式樣『01111110』(7EH) 組成，用這個旗號來表示一個訊框的結束，和下一個訊框的開始。因此所有已啟動的通訊都必須隨時檢視這個旗號，以做為訊框起始的同步。接收端在接收資料時也須隨時檢視這個旗號，以辨識訊框是否結束。但在訊息（或其他欄位）中也非常有可能會出現『01111110』字串，接收端將會誤認為是訊框的控制旗號，而產生錯誤的判斷，所以在整個訊框中不允許有『01111110』的位元串發生。為解決這個問題，資料流 (Data Stream) 中必須有『位元填塞』(Bit Stuffing) 的功能。所謂位元填塞功能就是傳送端在發送資料流（Data Stream）當中，檢查位元串列之中是否有連續 5 個 1。如有，每 5 個連續的 1 之後便自動插入一個 0（旗號欄除外）。接收端在偵測到訊框旗號後，繼續不斷的檢測位元串列 (Bit Stream)，如發現連續 5 個 1，則必須偵測它之後的第 6 個位元是 0 或 1。第 6 位元若是 0 則刪掉（位元填塞），若是 1，則必須偵測它之後的第 7 位元。第 7 位元若是 0 則被認為是旗號。若第 6 及第 7 位元都是 1 時，則表示傳送端要求終止連線的訊號（終止連線旗號）。位元填塞的例子如下：

原來之位元字串：11111111111101111111011110

位元填塞後位元字串：111110111110111011111011011110（0 表示填塞）

並非加入位元填塞的功能就不會出問題，位元填塞有時也會因一個位元在傳送過程發生變化，而產生嚴重的錯誤。如圖 3-5 (a) 所示，在資料流中如果有某個關鍵位元在傳送當中發生變化 (0→1)，就可能導致一個訊框被分割成兩個訊框。另外也有可能在傳輸中旗號串列的某個位元發生錯誤 (1→0)，因而被誤判為一般訊息，導致兩個訊框被合成一個訊框，如圖 3-5 (b) 所示。這些錯誤必須由其他錯誤控制方法偵測出，較常見的方法是在訊框內增加一個計算訊框長度 (Length) 的欄位。接收端計算所收到的訊框長度是否相同於『長度』欄位的值，如果不符合，表示訊框有被合併或分割的可能。

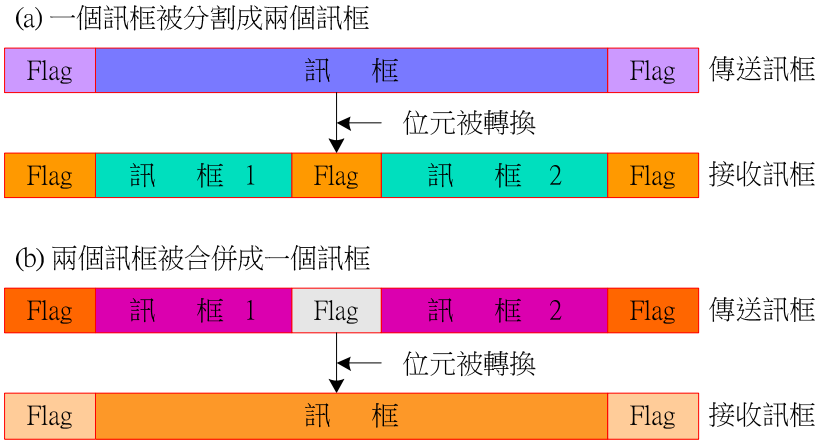


圖 3-5 位元轉換產生之錯誤

3-3 通訊連線管理

鏈路層必須建立兩個工作站之間的通訊連線，雙方再由這條連線互相傳遞訊框。但依照上層通訊軟體的需求，可能需要不同等級的傳輸品質。一般鏈路層都有提供下列三種服務類別：

- **非連接服務（Connectionless Service）**
- **連接導向服務（Connection-oriented Service）**
- **非連接附確認服務（Connectionless with Acknowledge）**

以下將分別介紹這三種服務類別的連線動作，但並非所有通訊協定都依照這些步驟製作。各種通訊協定會依照其特性或環境需求而有所差異，但皆以上列連線類別為基礎。

3-3-1 非連接服務

『**非連接服務**』(ConnectionLess Service, CL)表示通訊雙方未建立連線便開始傳送訊框，由該訊框自己建立連線傳送，傳送完後連線便自動消失。既然未事先建立連線，所以不保證是否可以傳送成功，因此又稱為『不可靠連線』(Unreliable Connection)。如圖 3-6 所示，通訊兩方(DL_A 及 DL_B)傳送資料之前並未建立連線，直接將資料發送給對方。發送端(DL_A)直接將訊框（DL_CL_Data.request）發送到網路上；接收端由網路上收到該訊框（DL_CL_Data.indication)。一般鏈路層做連線管理時所發送的各種訊息（控制訊號），都採

用非連接方式傳送，因為當時並不一定已完成連線的建立。另外，廣播訊息也採用非連接方式：

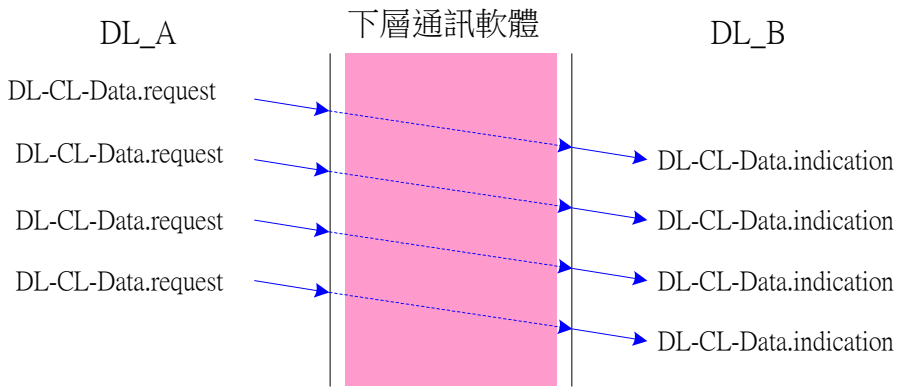


圖 3-6 非連接服務之通訊連線

3-3-2 連接導向服務

『**連接導向服務**』(Connection-oriented Service, CS) 是通訊雙方建立連線後才開始傳送訊框，因能保證訊框可以到達接收端，所以又稱為『可靠性連線』(Reliable Connection)。圖 3-7 為連接導向服務的通訊連線的時序圖，其可區分為三個時相 (Phase) : 連線建立時相、資料傳送時相與連線終止時相，以下分別介紹之。

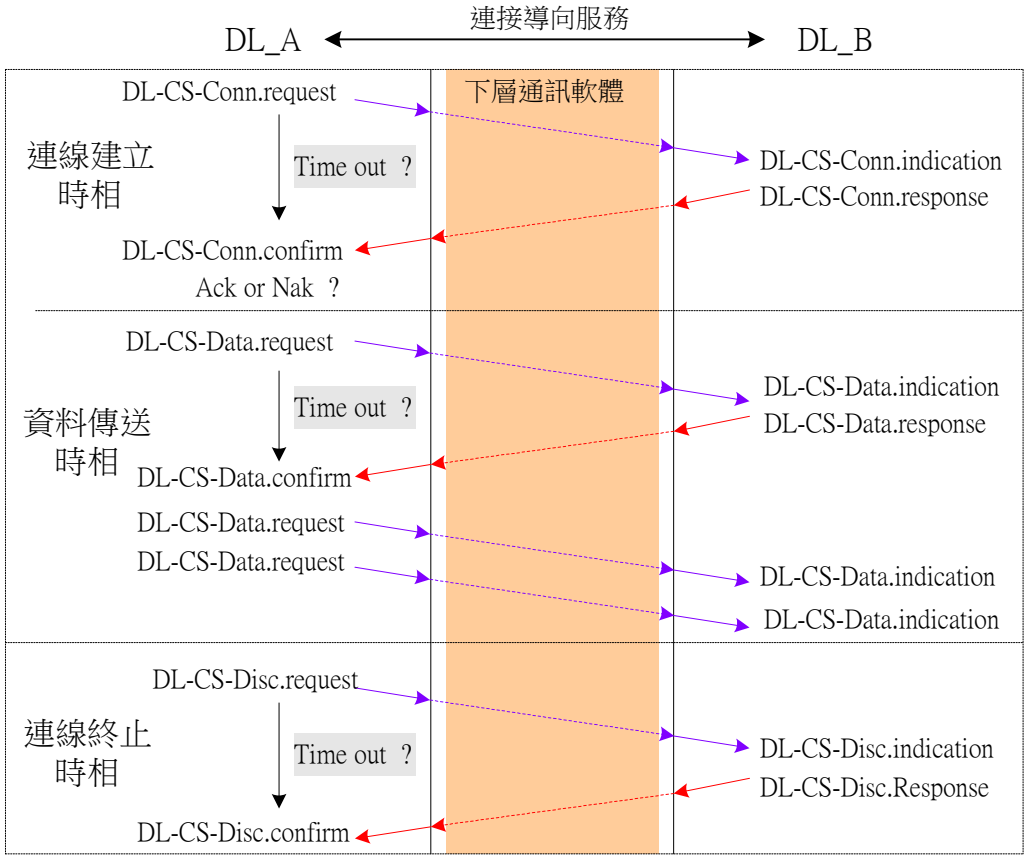


圖 3-7 連接導向服務之通訊連線

(A) 連線建立時相（Connection Establish Phase）

發送端的鏈路層（DL_A）送出要求連線訊息（DL_CS_Conn.request）給接收端（DL_B），對方接收到該訊號（DL_CS_Conn.indication）以後，則回應同意（Acknowledge, Ack）或不同意（Non-acknowledge, Nak）訊息，並將其包裝成回應訊息（DL_CS_Conn.response），回覆給發送端。然後，發送端再由收到的回應訊息（DL_CS_Conn.confirm）判斷對方是否同意連線。如果所收到的回應訊息是確定的（Ack），則表示雙方連線已建立成功。當發送端送出要求連線訊號時，並不能保證該訊息是否可安全傳送到對方。因此，在發送端送出訊息後便啟動計時器（Timer），如在逾時（time out）前未收到回應，便判斷該訊號已經遺失，或對方工作站根本不存在，接著必須決定是否需要重新要求連線。

(B) 資料傳送時相（Data Transfer Phase）

當雙方連線成功後就可以開始傳送資料。首先，發送端將會送出資料訊框（DL_CS_Data.request），並啟動計時器；接收端在收到該訊息（DL_CS_Data.indication）後，會回應該訊號是否正確（DL_CS_Data.response）。發送端由回應訊息（DL_CS_Data.confirm）內判斷該訊框在傳送當中是否安全送達到接收端（或是有發生錯誤？）。

處理連接導向的資料傳送程序比較複雜，因為傳送端可以一次連續發送數筆訊框，但接收端如何來確認是否已全部接收到？又當傳送端將訊框發送後，便啟動自身的計時器。如逾時後未收到任何回應，則表示該訊框在傳輸中遺失，而必須重新傳送；如有收到回應，則重置（Reset）計時器，再繼續傳送訊框。但也有可能接收端的回應訊號傳輸過慢，計時器已逾時還未到達傳送端，造成傳送端重複傳送該訊框，導致接收端所收到的訊框順序不對。因此，訊框在傳送當中可能發生列三個主要問題：

- 訊框遺失（Lost）
- 訊框重複（Duplicate）
- 訊框接收順序不對（Out of order）

關於上述訊框傳送中所可能發生的問題，必須利用一個特殊的機制（流量控制）來處理。一般通訊協定都採用滑動視窗法，我們在下一節將會介紹它。

(C) 連線終止時相（Connection Terminal Phase）

當資料傳送完畢後，任何一方都可以要求終止連線。如圖 3-7 發送端送出要求終止連線訊號（DL_CS_Disc.request），對方收到要求終止連線訊息（DL_CS_ Disc.indication）後，決定是否斷線並回應訊息（DL_CS_Disc.response）給發送端（一般都必須無條件同意斷線）。發送端由對方回應訊息（DL_CS_Disc.confirm）判斷對方是否同意終止連線，如對方確認（Ack）終止連線，或在計時器逾時之前沒有收到回應，便釋放該連線。

3-3-3 非連接附確認服務

『**非連接附確認服務**』（Connectionless with Acknowledge Service, CA）如同非連接服務，在雙方未建立連線的時候，就開始傳送資料（DL_CA_Data.request），並啟動計時器。但要求接收端收到該訊框（DL_CA_Data.indication）後必須即時回應（DL_CA_Data.response）給發送端。發送端由對方回應之訊息（DL_CA_Data.confirm）來判別對方是否有正確收到資料。非連接附確認服務一般應用在需要反應較快的資料訊框傳送，如需要快速傳送、或資料量較少的連線，使用連接導向服務反而會浪費許多連線的時間，此時使用非連接附確認服務較為理想。

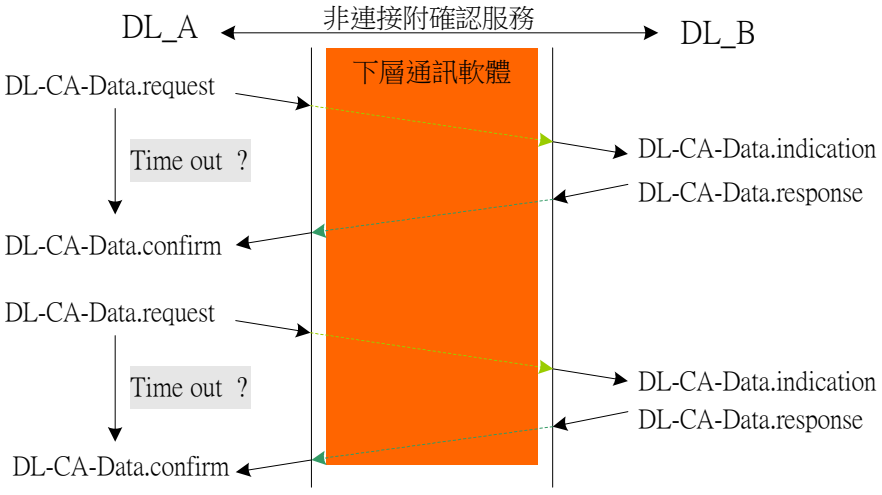


圖 3-8 非連接附確認服務之通訊連線

3-4 流量控制

當兩端的鏈路層建立好通訊連線後，雙方也協議出使用連接導向的通訊服務，下一個問題是，兩端如何利用通訊連線傳送資料？而雙方該如何管理資料訊框在通訊連線上的流動，才會使通訊連線產生最高效率？這些都屬於『**流量控制**』(Flow Control) 的問題。

流量控制除了管理資料在通訊連線上傳輸的順序外，還牽涉到傳送端和接收端緩衝器多寡的管理問題。當電腦欲傳送資料時，將資料依每次傳送封包大小，依序填入『傳送緩衝器』內。當傳送端將資料封包傳送出去後，可能必須經過一段時間後才能到達接收端，接收端是否正常收到資料？該如何回應給傳送端？如果發生問題怎樣重送？傳送端必須確認該訊號已正常傳送完畢，才可將資料封包由傳送緩衝器上刪除，因此在大量連續傳送資料時，傳送緩衝器就必須很大。同樣的，在接收端也必須有一個『接收緩衝器』，接收到資料後判斷資料正常便填入緩衝器內，但因為通訊連線的時間也許很長，每一筆資料封包到達的時間不一定相同，且到達的資料封包不一定按照原來的封包順序，也可能在一串的封包內其中有幾個封包發生錯誤。因此有幾個封包是重新傳送？有幾個封包是重複傳送？都必須等接收端將接收緩衝器內的封包按序號排列完成，才可傳送給上層通訊軟體，所以接收緩衝器有可能要很大。當然由技術層面來講，不論傳送緩衝器或接收緩衝器都不可能無限的擴充，我們希望流量控制的技術能儘量減少緩衝器的需求，而如何重複使用緩衝器以減低緩衝器需求也是流量控制項目之一。一般流量控制有兩種基本方法：

- **停止與等待法（Stop-and-Wait Method）**
- **滑動視窗法（Sliding Window Method）**

多數通訊協定（軟體）都有流量控制機制，也都以上述兩種方法為基礎，至於會採用哪一種，則必須視使用環境而定。以下分別介紹這兩種方法。

3-4-1 停止與等待法

『**停止與等待**』(Stop-and-Wait)的重點在於傳送端將訊息送出後，必須等待接收端回應，再決定是否繼續傳送下一筆資料。如果回應接收正常（Ack），便再傳送下一筆資料；如果逾時未收到回應或收到回應不確認訊息(Nak)，則需重新傳送該筆資料，其動作順序如圖 3-9 所示。

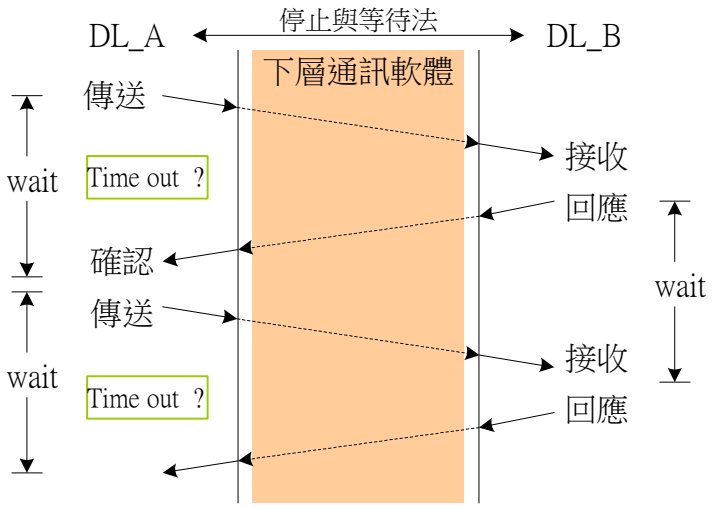


圖 3-9 停止與等待流量控制

由圖 3-9 我們可以發現在停止與等待法中，通訊雙方花費太多的等待時間（大部分時間都在等待對方回應）。但因其處理方法簡單，一般近距離（回應時間較短）或要求快速反應的設備大部分採用此方法。而且傳送和接收雙方只要一個緩衝器便可，一般使用在主機電腦內、或電腦機房內各處理設備之間資料的傳輸，至於長距離傳輸則較不適合。（如 Bluetooth 便採用此方法）

3-4-2 滑動視窗法

當建立連線後，我們希望能充分利用連線來傳輸資料（連接導向服務），如果採用停止與等待方法，通訊連線大部分時間都是空閒著，傳輸效率太差。『滑動視窗法』（Sliding Window）可連續傳送多筆資料，接收端只要回應目前已正確接收到第幾筆資料，一次確認多筆資料，網路上便可連續傳送多筆資料，提高網路的使用率。

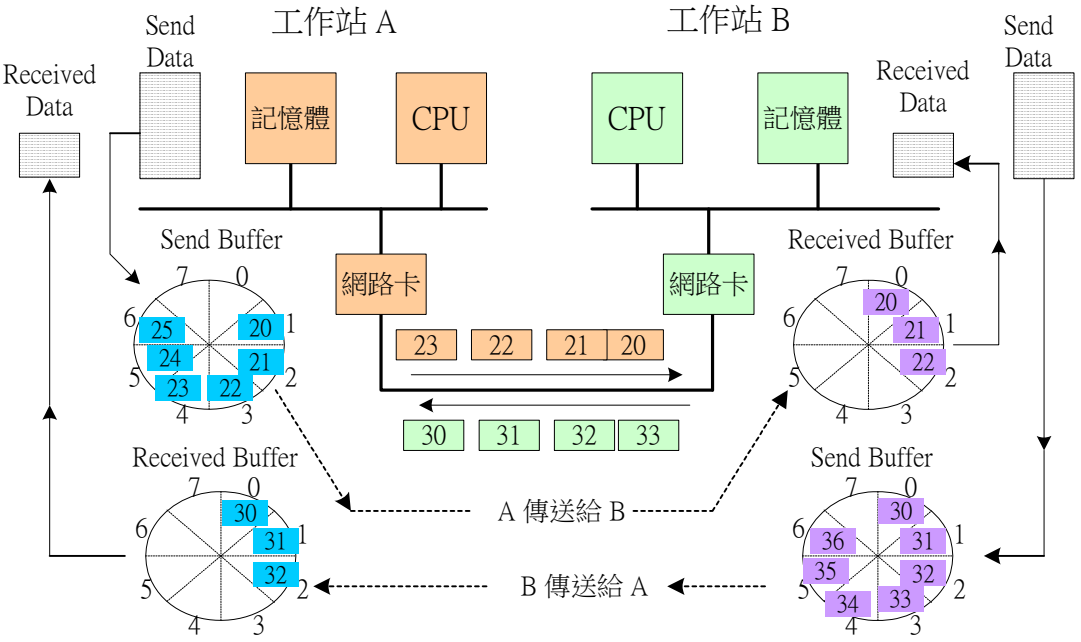


圖 3-10 滑動視窗法

首先我們用圖 3-10 來說明滑動視窗法的大略運作方式，圖 3-10 是以一般區域網路網路卡的動作為例子（其它層次也都類似），運作如下：首先將資料（如 10Mbits）分割成若干個固定大小的訊框（如 4 Kbits），並不全部送給傳送器（一般都是網路卡），因傳送器上記憶體容量可能不夠。又傳送器將本身也將記憶體依照訊框大小分割成若干個容器，稱之為**『傳送緩衝器』(Send Buffers)**。首先主機將欲傳送的訊框依順序填入傳送器的傳送緩衝器，傳送器再就緩衝器資料一一傳送到網路上，傳送成功後清除緩衝器上的資料，再請求主機電腦(中斷 CPU) 再填入待傳的資料，一直到資料傳送完畢為止。

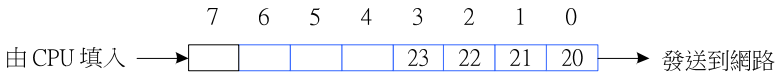
另外，接收端的接收器（網路卡）也將本身的記憶體分割成和訊框大小相同的若干個容器，稱之為**『接收緩衝器』(Received Buffers)**。接收器接收到訊框後便填入接收緩衝器，如果有連續訊框接收正常，便要求（一般採用中斷方式）主機電腦（CPU）來讀取，再清除緩衝器預備接收下一筆訊框。同時還要傳送一個確認訊息給傳送端，表示已正確地接收到幾個訊框，請傳送端繼續發送訊框。

如果通訊雙方都可以同時接收和傳送訊框，那麼雙方都必須具有傳送緩衝器和接收緩衝器。如圖 3-10 所示，工作站 A 的傳送緩衝器對應工作站 B 的接收緩衝器，反之亦然。以下我們分為幾個步驟來介紹滑動視窗法的運作原理。

(A) 傳送視窗與接收視窗

首先我們來介紹什麼是『視窗』(Window) ？其實視窗不過是緩衝記憶體的排列方式。不論傳送緩衝器或接收緩衝器都必須重複使用，如果記憶體以直線序列(Linear Array)排列，如圖 3-11 (a) 所示，資料封包只能由前端填入，由後端取出，記憶體緩衝器就必須不停往後延伸。因此，我們必須將緩衝器之記憶體的前頭和後尾銜接，成為環狀序列 (Ring Array)，以此構成視窗的形狀。以圖 3-11(a) 傳送緩衝器為例，假設有 8 個緩衝器，並由 CPU 填入訊框號碼 20、21、22 及 23 等四個訊框。如果我們將其建構成環狀序列，如圖 3-11 (b) 所示，序列中有兩個指標(Pointer):『前端指標』(Front Pointer) 和『後端指標』(Rear Pointer)。前端指標指向可由 CPU 填入訊框的位置，當 CPU 繼續填入訊框時，指標就一直往前滑動。另外，後端指標則指向將發送到網路上的訊框位置，每發送一個訊框，指標就往前滑動一個緩衝器。CPU 不停的填入；而發送器也不斷的將訊框發送到網路上，兩個指標也就不停的滑動，因此稱之為『滑動視窗法』(Sliding Window)。

(a) Send Buffer (直線序列排列)



(b) Send Window (環狀序列排列)

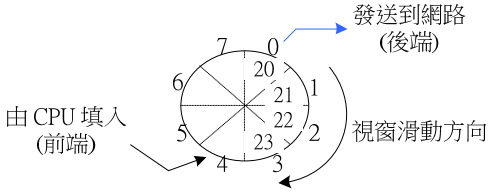
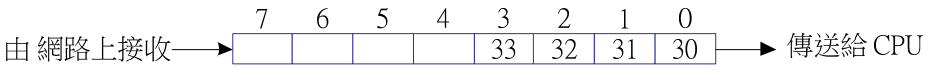


圖 3-11 傳送視窗之緩衝器結構

(a) Received Buffer (直線序列排列)



(b) Received Window (環狀序列排列)

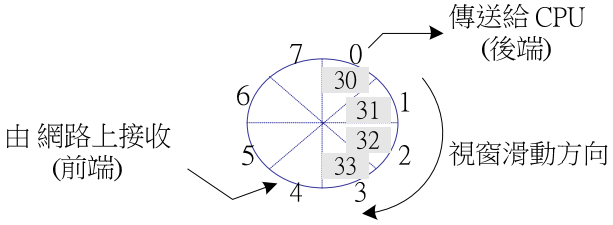


圖 3-12 接收視窗之緩衝器結構

圖 3-12 (a) 為接收緩衝器的直線序列排列架構，我們同樣假設緩衝器大小為 8(0 ~ 7)，如將其排列成環狀序列，便成為圖 3-12 (b) 的接收視窗。接收視窗的前端指標是由網路上接收到訊框之後往前滑動；後端指標是 CPU 讀取訊框後往前滑動。如果接收器已由網路上接收到號碼為 30 ~ 33 等 4 個訊框，它的滑動方向如圖 3-12 (b) 所示。

(B) 訊框順序編號

在鏈路層的訊框格式（其他層次也相同）裡包含兩個序號：

- **N(S)**：表示目前傳送訊框的序號。
- **N(R)**：表示期望對方下次傳送過來的訊框序號，也表示該序號以前的訊框都已正常接收。

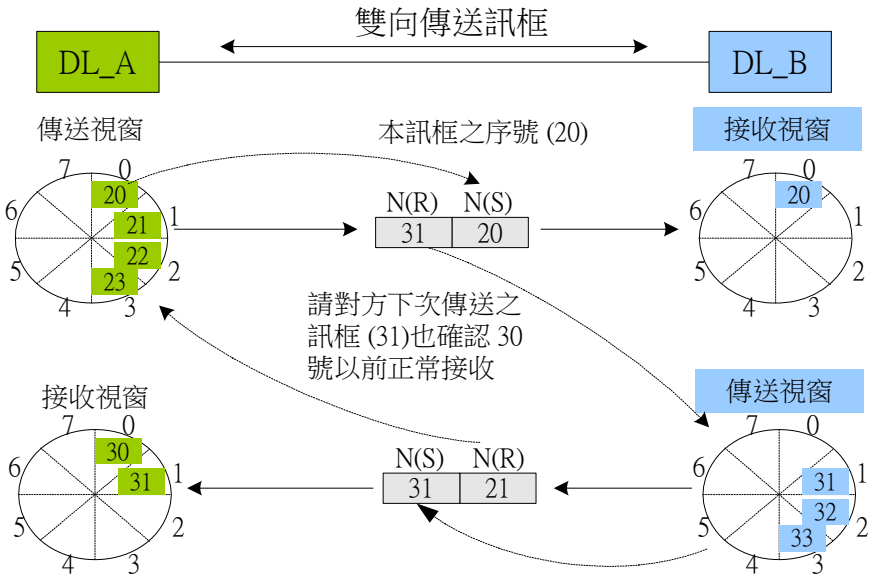


圖 3-13 序號 N(S) 及 N(R) 之功能

我們用圖 3-13 來說明 N(S) 和 N(R) 的運用情形，假設連線雙方（DL_A 和 DL_B）都以全雙工方式相互傳遞訊框。首先 DL_A 發送序號為 20 （N(S) = 20）的訊框給 DL_B，並期望 DL_B 下次傳送 31 號訊框給它（N(R) = 31），這表示 31 號以前的訊框（30、29、28、）都已正常接收。DL_B 收到 DL_A 所傳送訊框後，便瞭解 30 號以前的訊框對方都已正常接收。再傳送第 31 號訊框，並要求對方下次傳送第 21 號訊框(N(S) = 31、N(R) = 21)，即告訴 DL_A 你的 20 號訊框我也正常接收了。由上得知，N(R) 帶有確認的功能。雙方可

連續傳送多筆訊框，再回覆 N(R) 表示某號碼以前的訊框已正常接收。因此在滑動視窗法中，通訊雙方可以連續傳送多筆資料，不像停止與等待法一次只能傳送一個訊框。

滑動視窗的大小是相對應的緩衝器空間。如果以 128 個視窗為例，每個訊框為 4 K Bytes，其緩衝器記憶體空間是 512 KBytes。連同傳送和接收緩衝器就必須具有 1 Mbytes 的空間。視窗號碼和封包序號沒有絕對的關係，因為訊框是動態不停的填入緩衝器；而視窗號碼是固定的緩衝器空間。

一般通訊協定都將訊框序號（N(S)）以亂數號碼（Random Number）來累進計算，並非一定由 0 開始計算。回應序號（N(R)）是依照對方的 N(S) 來計算（如，加一）回應。一般通訊協定都以 16 或 8 位元來代表訊框訊號。

(C) 滑動視窗範例

我們用圖 3-14 來說明滑動視窗運作的範例。首先假設滑動視窗為 0～7，表示有 8 個緩衝器。但最多僅能使用到 7 個緩衝器，才不至於使前端和後端指標相重疊。假設目前 DL_A 預備由 20 號訊框開始發送；而 DL_B 由 30 號開始。以前所傳送之訊框都已確認成功，所有緩衝器（7 個）都空閒。我們用下列 a～f 步驟來描述它的運作情形：

(1) DL_A 發送 20 和 21 號訊框給 DL_B，對方並將其填入接收視窗內。同時 DL_A 要求對方傳遞 30 號訊框，表示以前訊框（29、28、27）都已正常接收。

(2) DL_B 發送 30 和 31 號訊框給 DL_A，並確認 22 號以前訊框(N(R) = 22)。DL_A 收到 N(R) = 22，知道之前所傳 20 和 21 對方已收到，便剔除這兩個訊框，後端指標滑動兩格。而同一時間內 DL_A 的 CPU（或上層通訊軟體）再填入 2 個訊框（25、26）到傳送視窗，也使前端指標向前滑動兩格。

(3) DL_A 傳送三個訊框（22、23、24），其中 23 號遺失，DL_B 發現訊框沒有依照順序到達，缺少 23 號。另 DL_B 之 CPU 又填入 2 個訊框（35、36）至傳送視窗。

(4) DL_B 傳送 32 號訊框，同時要求對方重新傳送 23 號（N(R) = 23）。

(5) DL_A 重送 23 號訊框，並確認 32 號訊框。DL_B 收到 N(R) = 33 便刪除傳送視窗的 31 和 32，並滑動傳送視窗。另一方面，將 N(S) = 23 訊框填入接收視窗，產

生 22、23、24 連續序號，便將這三個訊框傳給 CPU (或上層通訊軟體)，再將接收視窗滑動三格。

(6) DL_B 連續傳送四個訊框(33 ~ 36)，並確認對方 24 號以前訊框(N(R) = 25)。DL_A 收到 N(R) = 25 後便將 22 ~ 23 訊框刪除，並滑動視窗 3 個位置。並將所收的 4 個訊框填入接收視窗內。

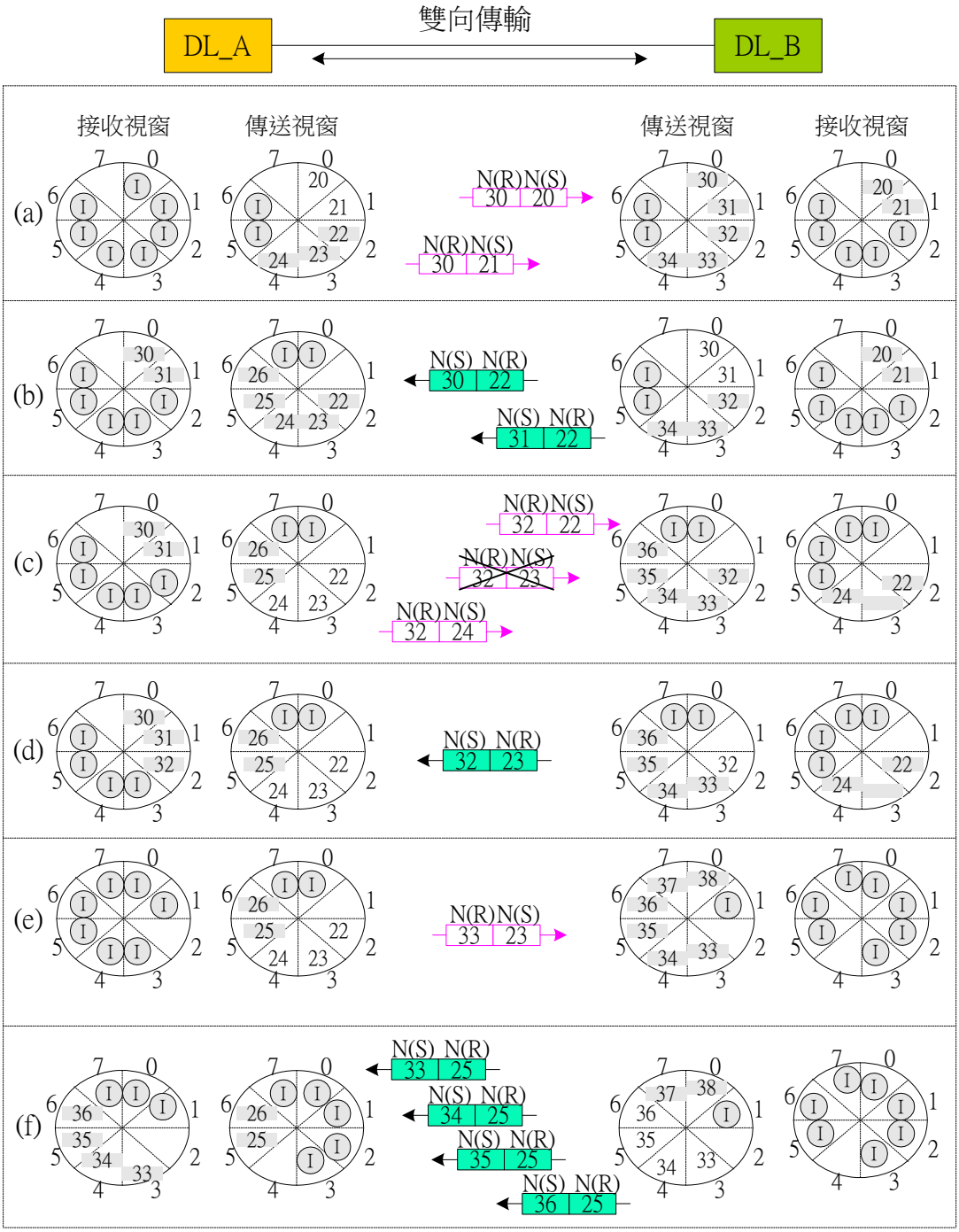


圖 3-14 滑動視窗範例

3-5錯誤偵出

鏈路層將訊框傳送給實體層，實體層再將訊框轉換成連續的訊號後發送到傳輸媒介。訊號在傳輸媒介上容易受環境因素所影響，如電磁波干擾或接續端子接續不良等，而發生訊號串列中某些訊號錯誤或遺失；接收端在接收完訊號後必須有能力去偵測它，到底所收到的資料是否和傳送端所傳送的資料相同。如果要讓接收端有偵測錯誤的能力，則傳送端必須在資料傳送之前，針對傳送資料做一些必要的處理和控制；而接收端在接收完資料之後，則依照雙方的協議來偵測傳輸資料是否有發生錯誤。萬一資料在傳送之中發生錯誤，接收端如何要求傳送端重新傳送？傳送端接到錯誤訊息如何重新傳送？以上所有針對如何保證資料的完整性之措施，稱為『錯誤控制』(Error Control)。錯誤控制包含兩大基本技術：

- **錯誤偵測（Error Detection）：**如何偵測出資料發生錯誤。
- **自動回覆請求（Automatic Repeat reQuest, ARQ）：**傳送資料發生錯誤，如何自動請求重新傳送。(將於 3-6 節介紹)

雖然在通訊協定裡，各層次中的通訊軟體大部分都有提供錯誤控制的管理，但以鏈路層對錯誤控制的管理最為重要，因為它和實體傳輸環境有最直接的關係，錯誤發生的機率最大。所以在鏈路層中所用到的錯誤偵測方法最為嚴謹。但要注意，並非所有錯誤偵測方法都能百分之一百偵測出錯誤，我們只能說較嚴謹的錯誤偵測方法，所能偵測出錯誤的**『或然率』（Probability）**較高。當然愈嚴謹的錯誤偵測方法愈理想，但有時我們為了考慮傳輸效率而不得不放棄它，改採折衷的錯誤偵測方法。資料訊框的長短對於資料錯誤偵出的或然率有絕對的影響，訊框愈長偵測能力愈低，相對的訊框愈短偵測能力愈高。但訊框長度直接影響傳輸效率，愈長傳輸效率愈高；愈短傳輸效率就愈低，因此如何找到雙方的平衡點，也是制定通訊協定的頭痛問題。既然沒有一種錯誤偵測方法可以百分之一百的偵出錯誤，因此接收端在偵測資料後沒有發現錯誤並不能保證沒有錯誤，只能說**『沒有發現錯誤』**(No error detected)，而不能說**『沒有錯誤發生』**(Error Free)。

一般在資料傳輸方面，對於資料錯誤控制的方法，除了錯誤偵出外，還有**『錯誤修正』**(Error Correction)方法。不論錯誤偵出或錯誤修正，都必須產生一個**『多餘碼』**(Redundancy Code)來保護原始資料，它在錯誤偵測方法中稱為**『錯誤偵測碼』**(Error-Detecting Code, EDC)，而在錯誤修正方法中稱為**『錯誤修正碼』**(Error-Correcting Code, ECC)。

在有錯誤控制的情況下，假使原來資料為 n 個位元，所產生的多餘碼為 k 個位元，則要傳輸整筆資料時就必須傳送 t = (n + k) 個位元。但錯誤修正或偵測碼只用來防止資料錯誤，並非真正的資料，因此稱為『多餘碼』。一般而言，多餘碼愈長，則錯誤控制能力（偵出或修正）愈強，但相對的也會減低傳輸效益，因為多餘碼浪費許多傳送的時間。

如採用錯誤修正方法，為了要使資料具有錯誤修正的能力，在資料傳輸中必須加入許多錯誤修正碼。一般來說，傳輸 8 位元資料中必須要有 5 位元的錯誤修正碼，才能偵測 2 位元的錯誤、和修正 1 位元的錯誤（One-bit correction two-bit detection）。但整體傳輸效率只有 8/13（8/(5+8)），顯然非常的低。它主要應用於記憶體和磁碟機之間近距離傳輸的錯誤控制。在範圍較廣的網路上，傳輸效率會影響到整個網路的效益，因此，一般網路只採用錯誤偵出，而不輕易採用錯誤修正（Bluetooth 便採用此法）。以下介紹三種常用錯誤偵測的方法：

- **同位元檢查法（Parity Check）**
- **檢查集檢查法（Check-Sum Check, CS）**
- **循環多餘碼檢查法（Cycle Redundancy Check, CRC）**

通訊協定各個層次都有流量控制及錯誤偵出的機制，所使用的偵錯方法也都大同小異。除了鏈路層因為牽涉到實體傳輸媒介（較容易出錯），而採用循環多餘碼檢查法外，其它層次大多採用檢查集檢查法。我們在這裡介紹各種方法，以後介紹其他層次裡就不再介紹錯誤檢查的基本原理。

3-5-1 同位元檢查

若要進行『**同位元檢查**』（Parity Check），傳送端和接收端在傳送資料之前，雙方必須協議出傳送資料（含檢查用的外加碼）轉化為二進位之後，位元值 "1" 的個數是偶數或是奇數。如協議為偶數個 1，則稱為『偶同位元檢查』（Even Parity Check）；奇數個 1 就稱為『奇同位元檢查』（Odd Parity Check）。為了要使整個資料固定為偶數個或奇數個 1，就必須加入 1 個位元的多餘碼，稱為『同位位元』（Parity Bit）。如果資料位元是 n 個位元，則整個傳輸資料長度 m =（n+1）。如雙方傳送協議採用偶同位元檢查，傳送端必須計算每一筆資料中 1 的數目，再決定同位元是 0 或 1，使 m 位元中有偶數個 1。接收端接收到資料後計算是否有偶

數個 1，如果是偶數個 1，則表示沒有發現錯誤，再將同位位元（Parity bit）拋棄；如果計算出是奇數個 1，那表示資料在傳送中已發生錯誤。同位位元的計算方式如下：

資料	奇同位元檢查	偶同位元檢查
01101100（4 個 1）	1	0
11011011（6 個 1）	1	0
01100100（3 個 1）	0	1

同樣的，如雙方協議使用奇同位元檢查，接收端收到資料後，便計算資料中 1 的數量。如果 1 的數量是奇數，則表示傳送資料沒有發現錯誤，否則（數量是偶數）表示傳送當中已發生錯誤。如果資料長度太長，同位元檢查的偵測錯誤能力就較低，因為位元串列中只要有偶數個位元發生變化（0→1、1→0），同位元檢查就找不出錯誤（位元是 1 的總數是偶數或奇數沒有改變）。因此，每一筆資料不可以太長，每 5～9 個位元產生一個同位位元，其偵測能力較高。同位位元的產生與偵測可以用硬體線路直接運算，不必用軟體額外來計算，也不會浪費電腦處理時間，一般應用在電腦內記憶體偵測或交談式的傳輸線上（RS-232C）。

3-5-2 檢查集檢查

『**檢查集檢查**』（Check-Sum check, CS）是一種以加法為基礎的檢查機制。傳送端欲將資料傳送前，將資料以每個字元（16 位元或 32 位元）為單位，連續累加起來得到一個字元（16 位元、32 位元、或其他長度），稱之為『檢查集』（Check-Sum, CS），也就是多餘檢查碼。如圖 3-15 所示，檢查集（CS）會連同資料（M）一齊（T = M + CS）傳送給接收端。接收端接收到資料後將資料（M'）以同樣的方法（字元長度）連續累加，CS'，如果 CS = CS' 則沒有發現錯誤；否則表示資料已發生錯誤。

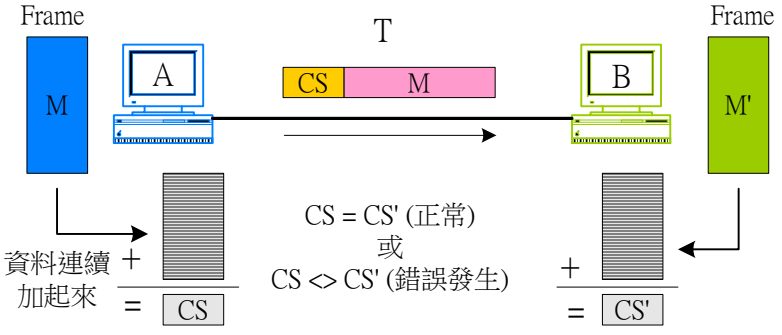


圖 3-15 檢查集檢查運作原理

檢查集不能夠利用硬體線路來建構。通訊軟體必須每次針對要傳送的資料方塊（Block）做運算來計算出檢查集，比較浪費時間。一般應用在較高層通訊協定（如 TCP/IP）或備份（backup）檔案的錯誤偵出使用。

3-5-3 循環多餘碼檢查

『**循環多餘碼檢查**』（Cyclic Redundancy Check，CRC）是利用除法（Modulo 2 運算）來做錯誤的檢出。其原理如圖 3-16 所示，其中通訊兩端必須事先協調好某一個除數（Q，最好為質數），傳送端傳送資料之前先將訊框（M）除以除數（Modulo 2 運算），得到一個餘數（FCS），再將餘數附加在訊框（M）的後端一起（T = M + FCS）傳送給接收端。接收端接收資料（M'）後，也用同樣的除數（Q）除，得到另一個餘數（FCS'）。如果 FCS' 和 T 後面的 FCS 相同（FCS = FCS'）即代表沒有發現錯誤；否則（FCS <> FCS'）已發生錯誤。

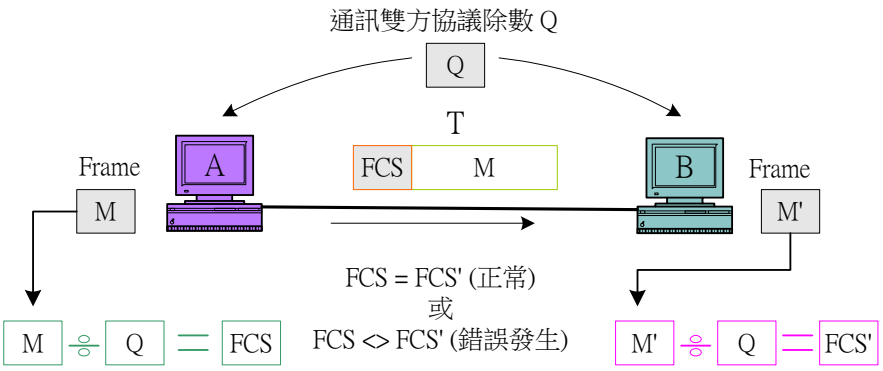


圖 3-16 循環多餘碼檢查運作原理

其實在 CRC 檢查運算裡，並不是利用餘數的比較來判斷是否有錯誤發生。其運算原理如下：一個 n 個位元的傳送資料經過 Modulo 2 的除法運算（除以除數 Q）後，產生的 k 個位元的餘數，稱之為『訊框檢查序列』（Frame Check Sequence，FCS），而將訊框檢查序列附在傳輸資料後面即成為傳送訊框，傳送訊框為 n + k 個位元長的訊息（T），而且正好可以被預先決定好的除數（Q）所整除。然後接收端便利用該除數（Q）來除收到的訊框（T'），若沒有餘數產生，即表示沒有發現錯誤。對於傳送端的 CRC 檢查碼的產生除法器（產生 FCS），以及接收端判斷是否有錯誤發生的除法器（M' 是否可以整除？），兩者都是相同的除法原理。

以下我們用 Modulo 2（模式 2）運算來推演 CRC 除法原理，假設參數如下：【備註：Modulo 2 運算為 XOR 計算，加法與減法都沒有進位和借位的情況】

- M = k 位元長的訊息，放於訊框前面的 k 個位元。
- T = (k + n) 個位元長的傳送訊框。
- F = n 個位元長的 FCS，放置於訊框（T）的後面 n 個位元。
- Q = n + 1 個位元長的除數。

如果推演出：接收端所收到的訊框（T'）和傳送訊框（T）相同，則 T' 可以被 P 整除，也表示沒有錯誤發生。推演如下：

(1) 在二進位算術中，向左移一位相當於乘以 2，向左移 5 個位元相當於乘以 25，因此對整個傳送訊框（T）內容的表示：

$$T = 2^n M + F$$

(2) 傳送訊框 T 是藉由訊息 M 向左移 n 個位元，並且在後面補上 n 個 0。再加上訊框檢查序列 F，便可串接 M 和 F 以產生 T。為了產生 FCS，我們先用 2n M 除以除數 Q：

$$(2^n M) / Q = K + R / Q$$

(3) 除數 Q 的選擇儘可能是二進位數中的質數，產生餘數的機率最大。由上式運算中，我們得到結果是商 K 和餘數 R。餘數 R 會比除數 Q 少一個位元，我們就將這個餘數作為 FCS。則整個傳送訊框為：

$$T = 2^n M + R$$

(4) 如果接收端接收到的訊框沒有錯誤發生，則 T / Q 應該沒有餘數。這條件如何產生？我們來做驗證：

$$T / Q = (2^n M + R) / Q \quad 則$$

$$T / Q = 2^n M / Q + R / Q$$

$$T / Q = K + R / Q + R / Q = K$$

(5) 由於 Modulo 2 運算中，任何相同的數相加是等於零(XOR 運算)，所以上式中 T 可以被 Q 整除，而不會產生非零的餘數。因此對於 FCS 的產生非常容易，只要將 2n

M 除以 Q，所產生的餘數作為 FCS 便可以，再將 2n M 與 FCS 組合成 T。在接收端方面只要用 T 除以 Q，如果沒有餘數，即表示沒有錯誤發生。

(6) 讓我們用一個簡單的例子來說明 FCS 的產生和錯誤檢查：

- **假設：**

  訊息 M = 1011100101 （10 個位元）

  除數 Q = 101011 （6 個位元）

  欲計算出 5 個位元的 FCS。

- **將 M 乘以 25**，得到 101110010100000。

- **將 2^5 M 除以除數 Q**：

```
          1001000011
      ┌──────────────
101011│101110010100000
      │101011
      ├──────
      │  101010
      │  101011
      ├──────
      │    110000
      │    101011
      ├──────
      │     110110
      │     101011
      ├──────
      │      11101
```

圖 3-6-1

- **得到餘數 11101 做為 FCS**。餘數被加到 2^5 M 以產生 101110010111101。

- **若沒有錯誤發生，則接收端應可完整的收到 T，此訊框將可被 Q 整除：**

```
          1001000011
      ┌──────────────
101011│101110010111101
      │101011
      ├──────
      │  101010
      │  101011
      ├──────
      │    111110
      │    101011
      ├──────
      │     101011
      │     101011
      ├──────
      │      00000
```

圖 3-6-2

- 由於沒有餘數產生，因此我們可以**假設沒有錯誤發生**。

CRC 除法原理是利用 Modulo 2 的運算原理推論出來。而 Modulo 2 的運算類似於二進位的「互斥或」閘（XOR Gate）運算（沒有進位的加法、沒有借位的減法）。因此，我們可以利用一個含有 XOR 閘和位移暫存器（Shift Register）來設計 CRC 除法器。

假設:傳送訊框 M(X)是一個可變的二進位函數(多項式表示),而除數 Q(X)為固定函數。因此，我們只要設計出 Q(X) 的 CRC 除法器即可。CRC 除法器的邏輯線路設計方式如下：

(1) 除法器的線路中含有 n 個暫存器，相當於餘數 R(X) 的長度。

(2) 最多含有 n 個互斥或閘。

(3) 在除數 Q(X) 位置是 1 的位置，便接有互斥或閘。

假設：

訊息 M = 1010001110　　M(X) = X9 + X7 + X3 + X2 + X

除數 Q = 110101（長度為 6）　Q(X) = X5 + X4 + X2 + 1

餘數 R(X) 長度為 5

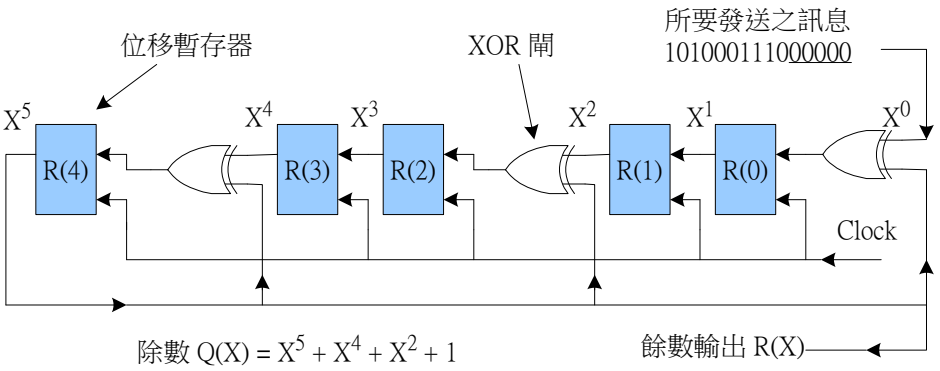


圖 3-17 CRC 除法器範例

圖 3-17 為 Q(X) = X5 + X4 + X2 +1 的 CRC 除法器之邏輯線路設計圖。此運作方式由清除移位暫存器開始，然後訊息由最高位之位元以一次一個位元的方式進入暫存器，以共同脈衝（Clock）來同步移位。由於要等到第一個被除數之位元到達暫存器的最高位元的端點才會有回饋訊號產生,因此前面的五個位元只有移位而已。當一個 1 的位元到達暫存器最左端(R(4))時,從此以後 R(3) 、R(1) 的輸出及輸入位元都會在下一次移位時間和 R(4) 輸出作 XOR 的運算。最前端 R(4)（X5）的輸出會迴旋到輸入端作處理，如果訊息不斷的進入，整個線路的

循環就不停止，因此稱之為『循環多餘碼除法器』(CRC 除法器)。使用該技術的錯誤偵出法也稱為『循環多餘碼檢查』。

在圖中，當訊息 (1010001110) 全部進入除法器後，暫存器上所儲存的值 (11010) 便是餘數 R(X)，亦是 FCS 的值。又當 0 進入除法器時，相當於所有暫存器的內容往左移一位。因此，我們在訊息後面補上 5 個 0，當 5 個時脈後，FCS 的值就從 R(4) 暫存器上輸出，也將其放入訊框中的 FCS 欄位內。

圖 3-18 是傳送端的 CRC 除法器。當在傳送資料時，多工器在 0 的位置，表示資料一方面傳送到網路上，一方面進入除法器以產生 FCS。當資料傳送完後，繼續送入補 0 的位元時，將多工器位置切換到 1 的位置，繼續傳送 FCS 的位元。因此，FCS 欄位就緊接著訊息 M 之後。不論接收端或傳送端的除法器都可架設硬體線路來運算，而且可以依照資料的輸入或輸出，一個位元接一個位元的進入除法器。也可看出 CRC 除法器完全不會佔用傳送時間。而且傳送端的 FCS 產生和接收端錯誤檢出之 CRC 除法器都是相同，因此，我們只要製作一個 CRC 除法器即可，一般都將其製造在網路卡上。

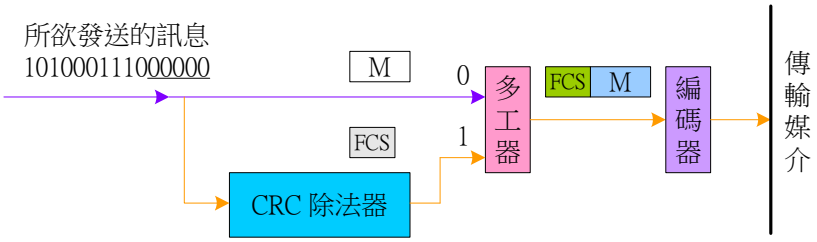


圖 3-18 CRC 檢查碼的硬體架構

由以上的推演我們可以瞭解，所使用的除數 Q(X) 對錯誤的檢出能力有很大關係。尤其，所欲傳送的訊框是可變長度 (限制在某長度範圍以內)，訊框愈長錯誤檢出能力就愈低。唯一的方法就是增加除數的長度，但相對所產生的多餘碼 (FCS) 也就愈長，傳輸效率也會降低。另外，在某固定長度的除數應該是什麼？當然所選的除數必須儘可能會產生餘數。因此，除數最好是選擇質數 (二進位的質數)，最有可能產生餘數。經過許多數學家的推演，有下列幾種通訊協定上較常使用的 CRC 碼。一般區域網路所使用的是能產生 32 位元長餘數的 CRC-32。

- CRC-12 = X12 + X11 + X3 + X2 + X + 1
- CRC-16 = X16 + X15 + X2 + 1

- CRC-CCITT = X16 + X12 + X5 + 1
- CRC-32 = X32 + X26 + X23 + X22 + X16 + X12 + X11 +
- X10 + X8 + X7 + X5 + X4 + X2 + X + 1

3-6 自動重複請求

『**自動重複請求**』（Automatic Repeat reQuest, ARQ）表示兩個通訊實體之間有資料流動時，接收端發現訊框已發生錯誤，它要如何告訴傳送端？傳送端應採何種方法重新傳送(re-transmit)？我們上一節已介紹過錯誤偵出的方法，這一節我們將介紹發生錯誤之後，如何重送的方法。一般通訊協定自動重複請求的機制有下列三種：

- **停止與等待 ARQ（Stop-and-Wait Automatic Repeat reQuest）**
- **退後 N ARQ（Go-Back-N Automatic Repeat reQuest）**
- **選擇性重複 ARQ（Select-Repeat Automatic Repeat reQuest）**

3-6-1 停止與等待自動重複請求

『**停止與等待 ARQ**』（Stop-and-Wait ARQ）是建構在停止與等待的流量控制技術上。如圖 3-9 所示，傳送端送出一個訊框後，便需等待接收端回應確認訊號（Acknowledge），在接收端未回答訊號前，傳送端不會再傳送訊框。

停止與等待之 ARQ 有兩種錯誤情況：

(1) 傳送之訊框受損。

接收端利用錯誤偵測技術偵出錯誤發生，放棄該訊框並回應一個負確認訊號（Non-acknowledge, NAK）給傳送端，這時候傳送端必須有能力再重送同樣的訊框。為了克服這個問題，傳送端必須有一個緩衝器保留訊框複本，訊框傳送出去後一直到收到確認訊號才可將複本拋棄，否則複本將會被重新傳送。

(2) 回應之確認訊號（ACK）受損。

接收端收到訊框並判斷訊框正確後，回應確認訊號。但確認訊號卻在傳送中受損。傳送端本身在訊框傳送出去後便開始計時，如計時器的時間已逾時（Time out）未收到確認或不確

認訊號（NAK），則會認為該訊框已遺失，而重送該訊框。又可能回應訊號在途中受損，同樣的訊框也會被再度傳送。在這種情況下，接收端必須有能力去判斷是否重複接收訊框。為了解決這個問題，我們將訊框編號以 0 及 1 交替使用，第一次為訊框 0、第二次是訊框 1、再下一個是訊框 0。接收端回應確認訊號也以 ACK0、ACK1 交替使用。和滑動視窗控制相同，ACK0 作為編號為 1 之訊框的接收確認訊息，表示接收端已準備接收編號為 0 的訊框。

由圖 3-9 可了解停止與等待之 ARQ，通訊中的雙方大部分時間都在等待對方回應或傳送資料。對於比較遠的通訊便不適合，因為傳輸延遲時間愈久，等待的時間就愈久。然而的操作方法簡單、且只需一個緩衝器的特性，卻是其他方法所不能及，因此使用於近距離傳輸尤佳。

3-6-2 退後-N 自動重複請求

『**退後-N ARQ**』（Go-back-N ARQ）是針對流量控制中的滑動視窗法所設計。在滑動視窗法中，傳送端可以連續傳送多筆訊框，每一筆訊框上都有編號。接收端回應要求希望接收第幾號訊框，表示該號數以前的訊框都已接收正常並確認之。退後-N 的設計是傳送端連續傳送多筆訊框後，接收端只回應正確收到第幾筆訊框，以後的訊框表示負確認。負確認以後的訊框都必須重新傳送，表示退後到首個不正確傳送的訊框，將該處以後的訊框全部重送，不管其中訊框是否傳送正常。接收端在偵出錯誤訊框後，便將以後的訊框丟棄，等待對方再重送。

如圖 3-19 所示，**傳送端連續發送 6 個訊框（編號 0、1、2、3、4、5）之後，接收到回應訊框 2 的負確認(NAK 2)，這時候接收端就將編號 2 以後的訊框全部丟棄(含 3、4、5)。**傳送端也退後到自編號 2 以後的訊框都全部重新傳送。因此，退後-N ARQ 較不需要大量的緩衝器來儲存未按順序到達的訊框，只要儲存連續的正確訊框即可。

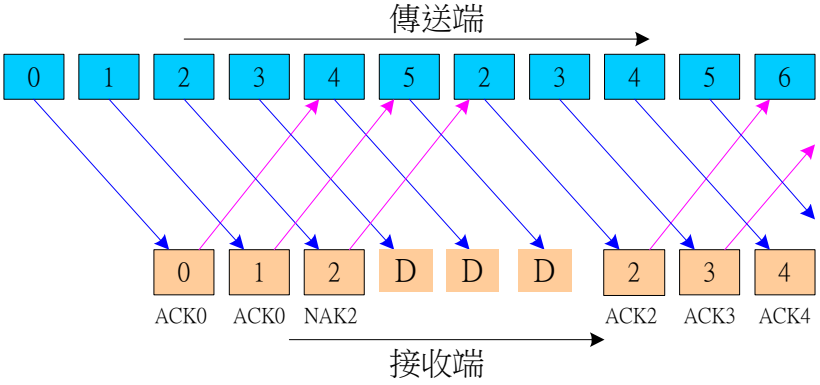


圖 3-19 後退-N ARQ 之運作程序

3-6-3 選擇重送自動重複請求

雖然退後-N ARQ 可以應用在滑動視窗法的錯誤控制、也可節省大量的緩衝器空間，但目前記憶體價格便宜，一般網路卡上都裝設有大量的緩衝器，記憶體空間已不是大問題。加上目前網路傳輸速率較快，也許會一次傳送大量的連續訊框。如果其中某一訊框發生錯誤，就拋棄該訊框以後的連續訊框，這對傳輸效率來講太不划算。**『選擇重送 ARQ』(Select-Repeat ARQ)** 就是針對這個缺點設計，在連續訊框中有某些訊框發生錯誤，只要針對錯誤的訊框重送就好，不要全部重送，這對整個傳輸效率來講會提昇許多。如圖 3-20 所示，**訊框編號 2 發生錯誤，只要重送該訊框即可。**但選擇重送方法，在傳送端必須保留更多的訊框複本，一直到對方有回應連續訊框都正常接收，才可拋棄訊框複本。又在接收端方面處理運作也較複雜，接收到的訊框也許會不按照訊框編號順序，必須將訊框依照順序排列後，再傳送給上層通訊軟體。接收端方面也需要有大量緩衝器，而且緩衝器溢流(overflow)的情況也非常容易發生。因此採用選擇重送的滑動視窗法時，必須有能力處理這個問題。

一般各個層次的通訊軟體都有流量控制，不管採用何種錯誤檢出方法，大部分都還是採用選擇重送 ARQ 機制。為了預防緩衝器溢流的問題，在傳輸封包的控制區塊裡有一個視窗（window）欄位（如，TCP 封包），接收端利用這個欄位告訴傳送端還剩下幾個緩衝器可以傳送封包。

如果接收端回應訊號裡 window = 0，表示緩衝器已滿，請暫停傳送封包。在鏈路層方面，在訊框的控制欄位也有一個 P/F 位元（如 LLC 封包），接收端回應傳送端時，如果 P/F = P 表示還有空閒緩衝器可以存放，可再傳送訊框；如果 P/F = F 時，表示緩衝器已滿，請暫停傳送訊框。

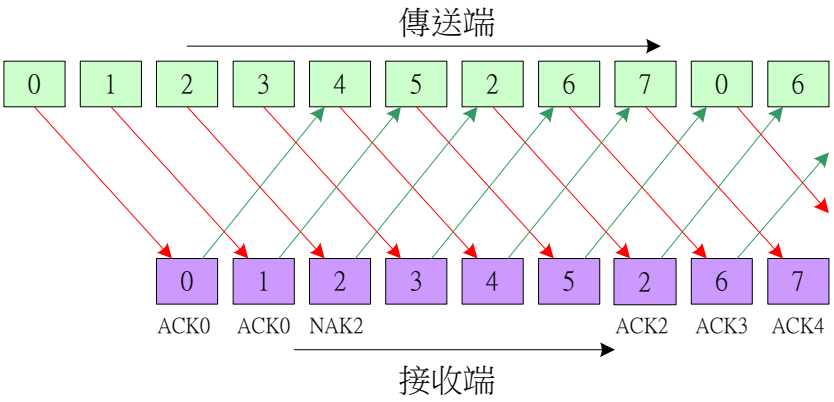


圖 3-20 選擇重送 ARQ 之運作程序

3-7傳輸媒介存取機制

網路上的傳輸媒介大多是共享的。也就是說，工作站之間互相通訊是利用共享傳輸媒介，宛如汽車欲前往任何地方必須透過共享的公路一樣。但對於工作站之間如何去取得媒介的使用權，一般網路有下列三種基本的存取控制機制：

- 點對點存取機制（Point to Point Policy）
- 多重存取存取機制（Multiple Access Policy）
- 交換存取機制（Switching Policy）

各種存取方法延伸出不同的網路類型，以下分別介紹其特性。

3-7-1 點對點存取機制

『點對點的存取機制』（Point to Point Policy）表示通訊的工作站之間有一條專屬連線，工作站所欲通訊的對象已被硬體連線固定。也就是說，工作站只能依照連線方向傳給下一個工作站。它如欲傳送給其它工作站，也必須透過它的下一個工作站轉送。最典型的點對點存取機制是環狀網路，如圖 3-21 所示。工作站 A 只能將訊息傳遞給工作站 B（硬體連線限制）、工作站 B 只能傳送給工作站 C 等等。工作站 A 如欲傳送訊息給工作站 E，它的訊息必須經過工作站 B、C、D 轉送才可到達。

點對點方式下，通訊雙方有固定的專線連結，不需要媒介存取的特殊控制，只要訊號按照順序傳遞給下一個工作站即可。因此，點對點存取機制大多是單向的。例如，Token-Ring 網路、FDDI 網路都屬於這種機制。

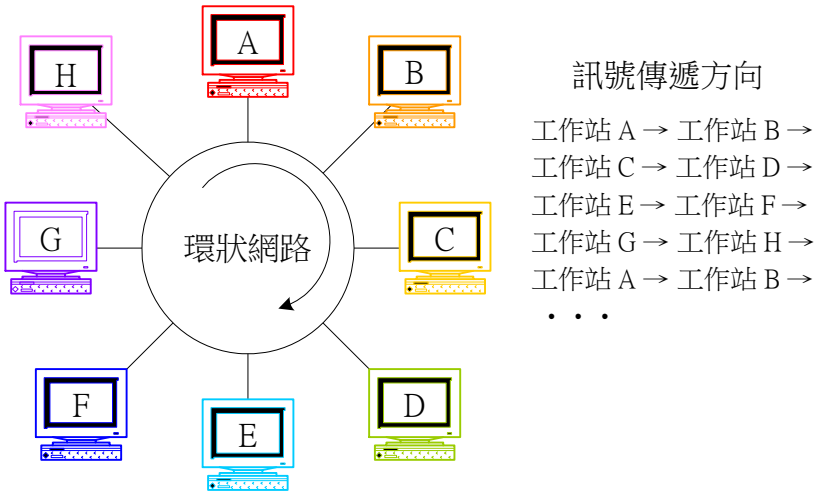


圖 3-21 點對點存取機制範例

3-7-2 多重存取機制

『多重存取機制』(Multiple Access Policy)表示一條傳輸媒介可讓多個工作站共同存取。網路上所有工作站都是直接連上傳輸媒介，但工作站欲傳送資料，必先取得媒介的使用權，才可將訊息發送到網路上。基本上，傳輸媒介上同一時間內只允許一部工作站發送訊號。最典型的多重存取機制是匯流排架構，如圖 3-22 所示。工作站 A 欲傳送給工作站 B，必須先取得媒介使用權 (依照不同協定制定)，然後將訊息廣播到媒介上；工作站 B 再由媒介上取得資料。

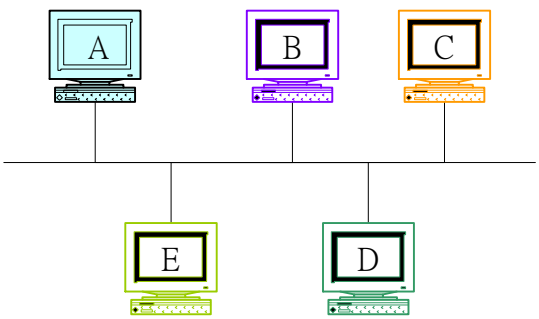


圖 3-22 多重存取機制範例

在多重存取架構之下，網路連線非常簡單，只要一條傳輸媒介就可以將所有工作站串接在一起，不像點對點方式，每兩個工作站之間都必須有一條專屬連線。但它對於工作站之間

如何取得傳輸媒介的控制較為複雜。而且當傳送端將訊息發送出去後，並無任何方法可以保證該訊息可安全到達目的地。目前最廣泛的 Ethernet 網路就是使用多重存取機制(CSMA/CD 協定)。

3-7-3 交換機制

在『**交換機制**』(Switching Policy) 的媒介存取控制之下，所有傳輸媒介都連接到『交換機』(Switch)，由交換機來轉送訊息。如同點對點方式，網路上每一部工作站都有專屬連線；但交換機制是連接到交換機，不似點對點方式是連接到下一個工作站。而且，交換機制的專屬連線是雙向性 (半雙工或全雙工)；而點對點方式是單方向性的。典型的交換機制如圖 3-23 所示。如果工作站 A 欲將資料傳送給工作站 B，首先得透過專屬連線將訊框傳送給交換機，交換機再依照訊框所指定的目的位址轉送到工作站 B 所連接的埠口 (Port) 上，再由該埠口連接的傳輸媒介傳送到工作站 B。

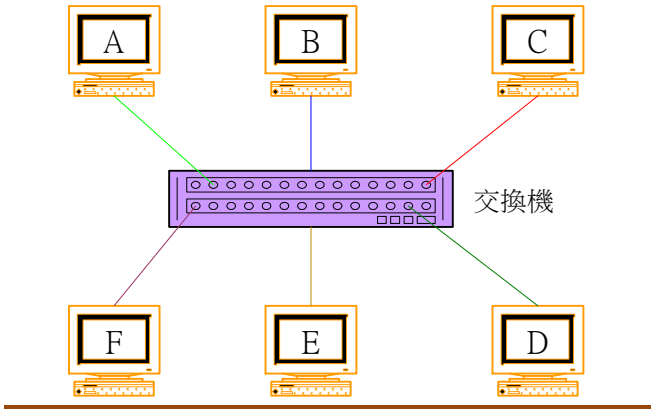


圖 3-23 交換機制範例

交換機制的交換機上必須擁有若干個連接埠口(或稱通訊埠)，且交換機本身必須記錄(或學習得到) 每一埠口所連接的工作站位址 (或名稱)。工作站只要銜接到交換機埠口上就可以通訊，因此，它和交換機之間並沒有特殊的通訊協定來處理，而只純粹是傳送、接收和交換的動作。但在傳輸速率較高的網路上，交換機的處理速度必須非常快速，也必須擁有大量緩衝器，以調解各通訊埠口之間的不同速率。交換機制目前在網路上使用非常普遍，而且一般都使用於傳輸骨幹，例如 Ethernet Switch、ATM 網路等等。

不同的媒介存取機制會延伸出各種網路架構，本書第二、三部份將針對區域網路和電信網路的媒介存取控制作完整介紹。

習 題

1. 請敘述鏈路層（Data-Link Layer）應具有之功能。
2. 為何資料在傳送之前必須分為若干個固定大小的訊框？
3. 何謂訊框化（Framing）？有哪兩種基本結構？請分別說明其特性。
4. 何謂字元填塞（Character Stuffing）？請敘述其功能。
5. 何謂位元填塞（Bit Stuffing）？請敘述其功能。
6. 一般鏈路層提供哪三種通訊連線服務？請分別敘述其特性。
7. 何謂停止與等待流量控制法？請說明其運作程序。
8. 何謂滑動視窗流量控制法？請說明其運作程序。
9. 如圖 3-14 (e) 滑動視窗範例中，如果工作站 A（DL_A）欲連續傳送 4 個訊框給 LL_B，請問他所傳送的訊框順序如何？為什麼？
10. 請利用虛擬碼(Pseudo-code)寫出滑動視窗法中的傳送視窗和接收視窗的運作程序。假設緩衝器空間為 16。
11. 為何一般網路通訊軟體都需要錯誤控制？尤其鏈路層最為重要，請說明其原因？
12. 何謂同位元檢查？請說明其原理。
13. 何謂檢查集檢查？請說明其原理。
14. 何謂循環多餘碼檢查？請說明其工作原理。
15. 如果資料長度為 8 位元，請計算出同位元錯誤檢出的或然率？
16. 何謂 Modulo 2 運算？請利用 10101100 及 10110011 驗證 Modulo 2 的加法。又利用 0110 及 1101 驗證 Modulo 2 的乘法。
17. 依照圖 3-17 Q(X) = X5 + X4 + X2 + 1 的除法器，如果輸入訊息為 M = 1010001110，請依照同步脈衝（Clock Pulse），繪出每輸入一個位元時，移位暫存器上的輸出值。

當所有位元都進入時，暫存器所儲存的值是否為餘數（11010）？再輸入 5 個 0，是否可將餘數全部輸出？

18. 請設計出 Q(X) = X9 + X7 + X5 + X4 + X + 1 的 CRC 除法器。

19. 請繪出 CRC-32 除法器的線路圖。

20. 請找出 5 個 16 位元的二進位質數，並說明其原理。

21. 何謂停止與等待法的自動重複請求？請說明其運作情形。

22. 何謂退後 N 自動重複請求（Go-Back-N ARQ）？請說明其運作情形。

23. 何謂選擇性重複自動重複請求（Select-Repeat ARQ）？請說明其運作情形。

24. 何謂傳輸媒介存取機制？基本上有哪幾種機制？請說明其運作情形。