國立臺灣師範大學應用電子科技學系

碩士論文

指導教授：黃政吉博士

IEEE 802.11e HCCA 中改善傳輸效能之動態排程演算法

A Dynamic Scheduling Algorithm for Performance Improvement in IEEE 802.11e HCCA

研究生：張哲瑜　撰

中 華 民 國 97 年 7 月

國立臺灣師範大學應用電子科技學系

碩士論文

指導教授：黃政吉博士

IEEE 802.11e HCCA 中改善傳輸效能之動態排程演算法

A Dynamic Scheduling Algorithm for Performance Improvement in IEEE 802.11e HCCA

研究生：張哲瑜　撰

中 華 民 國 97 年 7 月

IEEE 802.11e HCCA 中改善傳輸效能之動態排程演算法

學生：張哲瑜　　指導教授：黃政吉博士

國立臺灣師範大學應用電子科技學系碩士班

摘　　要

在 IEEE802.11e 的修正草案中，提出了根據平均的封包大小和資料產生率來計算 TD (TXOP Duration)及 SI (Service Interval)的演算法，但卻無法適用於 VBR (variable bit rate)資料流。因此，在相關研究裡提出一個 TXOP timer 的機制，利用可變的輪詢週期以及可變的 TD 來適應 VBR 的資料型態。此外在 TXOP timer 的基礎上，更有研究提出了利用佇列的資料量資訊 QS (Queue Size)來分配準確的 TXOP。雖然在 TXOP 配置方面可因此接近於完美，但在資料流量控管方面仍然有很大的改善空間。這是因為 TXOP timer 的增長速率為一平均值，因此在服務 VBR 封包時會導致若干封包無法在延遲範圍內接受服務。本論文將提出一個有效的排程演算法來改善 TXOP timer 機制，而作法主要是將 QSTA (QoS Station)的服務分成輪詢及傳送兩部分。在輪詢部分將使用 VBR 最短的服務區間以準確的掌握 VBR 的封包產生時間；在傳送部分針對 TXOP timer 的傳輸限制提出無 timer 傳輸條件以及根據 QS 調整之變動 timer 傳輸條件來改善傳輸效能。經由模擬發現本論文作法可以有效的改善封包的延遲及遺失，並且能達到較高的吞吐量。

關鍵字：wireless local area network (WLAN), IEEE 802.11e, scheduling

A Dynamic Scheduling Algorithm for Performance Improvement in IEEE 802.11e HCCA

student：Che-Yu Chang **Advisors：Dr. Jeng-Ji Huang,**

Institute of Applied Electronics Technology

National Taiwan Noraml University

ABSTRACT

In IEEE 802.11e, the provision of parameterized quality of service (QoS) is enabled by a polling-based scheduling. The scheduling deals with assignments of transmission opportunities (TXOPs) to QoS stations (QSTAs) at proper polling time instants, and it is inefficient for variable bit rate (VBR) traffic streams if both of the TXOP durations and the polling instants are estimated based on mean values declared in the respective traffic specifications (TSPECs). Although the efficiency can be improved by adapting TXOP durations according to the backlogged traffic reports issued by QSTAs, a problem still remains if polling instants are determined by mean-value timers. In this paper, we first point out the performance impairment that may be caused by mean-value timers, then a traffic scheduling algorithm using an adaptive timer is proposed to remedy this problem.

Keywords: wireless local area network (WLAN), IEEE 802.11e, scheduling.

誌　　謝

首先誠摯的感謝指導教授黃政吉博士的細心教導使我能了解無線網路這門技術。在研究過程中不時的與我討論並指點我正確的方向，使我在這兩年獲益匪淺。而老師對研究的嚴謹更是我學習的典範。

感謝張森境、汪威霆、陳宜玄學長們不厭其煩的指出我研究中的缺失，且總能在我迷惘時為我解惑，也感謝台師大應用電子科技研究所全體同學的幫忙，恭喜我們順利走過這兩年。實驗室的吳宗哲、劉益興學弟當然也不能忘記，你們的幫忙及搞笑我銘感在心。

最後感謝在我背後的默默支持我的家人，沒有你們的體諒、包容，我便不會有現在的成就。

目　　錄

中文摘要……i
英文摘要……ii
誌　　謝……iii
目　　錄……iv
圖目錄……vi
表目錄……vii
第一章　緒論……1
1.1　前言……1
1.2　研究目的……4
1.3　其他相關研究……6
1.4　論文架構……7
第二章　相關知識及作法介紹……7
2.1 HCCA 機制介紹……7
2.2 HCCA 運作……8
2.2.1 HCCA 運作原理……8
2.2.2 ns-2 簡介……9
2.2.3 HCCA 的程式實作……10
2.2.4 NS2HCCA……12
2.3 HCCA 參考排程演算法……13
2.3.1 排程原理……13
2.3.2 程式實作……15
2.4 SETT-EDD 排程演算法……18
2.4.1 排程原理……18
2.4.2 程式實作……21
2.5 ARROW 排程演算法……24
2.5.1 排程原理……24

2.5.2 程式實作……26
2.6 適應性排程演算法……29
2.6.1 排程原理……29
2.6.2 程式實作……30
第三章 論文作法介紹……33
3.1 ARROW 與適應性排程比較……33
3.2 論文作法……36
3.3 程式實作……38
第四章 模擬結果……45
4.1 模擬情境與參數設定……45
4.2 模擬相關程式……45
4.2.1 Otcl 程式……45
4.2.2 awk 程式……47
4.3 模擬結果……49
4.3.1 系統吞吐量(throughput)分析……49
4.3.2 封包平均延遲(mean delay)分析……50
4.3.3 封包遺失率(loss rate)分析……50
第五章 結論……52
參考文獻……53

圖 目 錄

圖 1-1：封包分佈與產生區間分佈……2
圖 1-2：token bucket 示意圖……3
圖 2-1：TSPEC 資訊格式……7
圖 2-2：IEEE802.11e HCF 範例……9
圖 2-3：HCCA 模擬結構……10
圖 2-4：MAC 封包交換程序……11
圖 2-5：Simple 示意圖……15
圖 2-6：SETT-EDD 範例……20
圖 2-7：SETT-EDD 模擬結構示意圖……21
圖 2-8：Add_Timer()結構示意圖……22
圖 2-9：ARROW 傳輸範例……24
圖 2-10：服務時間 t_{due} 與 Start Service Time 的關係示意圖……29
圖 3-1：ARROW 服務固定週期 VBR(如 H.261 與 MPEG4)的情況……33
圖 3-2：ARROW 服務非固定週期 VBR(如 H.263)的情況……34
圖 3-3：適應性排程服務固定產生週期和非固定產生週期的情況……35
圖 3-4：適應性排成滿載之情況……35
圖 3-5：論文作法服務非固定週期 VBR 之情況……36
圖 3-6：論文作法封包遺失之情況……37
圖 3-7：無傳輸限制在服務非固定封包產生週期之情況……38
圖 3-8：[9]所提出之事件程序……39
圖 3-9：論文所提出之事件流程……40
圖 4-1：系統吞吐量成效圖……49
圖 4-2：平均封包延遲時間分佈圖……50
圖 4-3：平均封包遺失率分佈圖……50

表 目 錄

表 1-1：VBR 特性統整表……...……3
表 3-1：ARROW 及適應性排程之特性統整表……..……35
表 4-1：系統參數……45

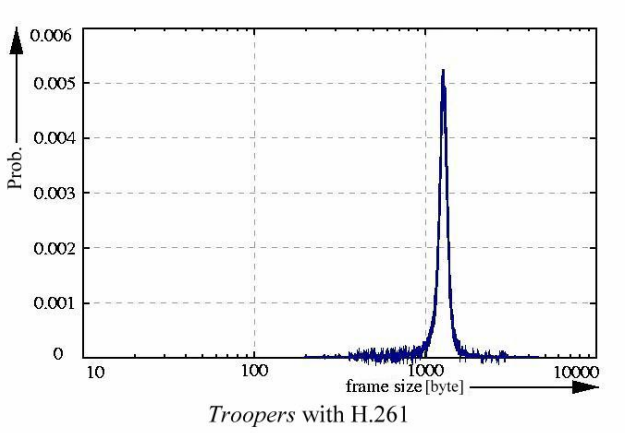
第一章　緒論

1.1 前言

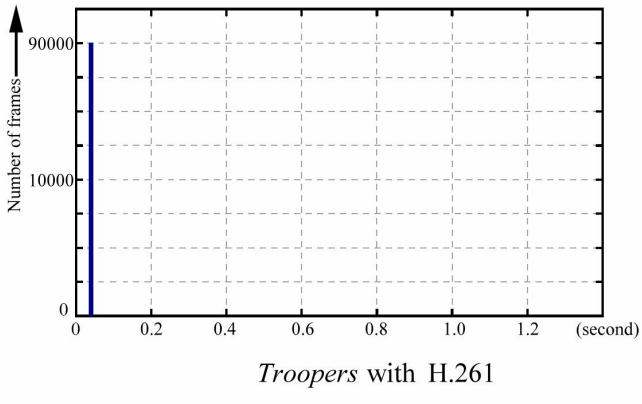
隨著無線區域網路的日益普及，也帶動了影音多媒體在無線網路的相關應用。在無線網路上傳遞多媒體資料，最主要的困難在於維持媒體資料的時效性，較有效率的作法是利用所謂的多媒體影音串流技術(streaming)。多媒體影音串流技術的原理，是把每個要播出的資料單元先分割成許多大小適當的封包，再傳送到使用者端。由於類比影音經轉換成數位形式後，資料量是非常驚人的；並且數位影音對時間的敏感性很強，即時性要求很高。因此，數位影音的傳輸一般先經過壓縮使檔案變小後，經由網際網路傳輸到用戶端後，解壓縮以顯示或播放影音內容。

目前廣泛使用的壓縮技術主要有 MPEG4、H.261、H.263 等，不同的壓縮技術之間有不同的特點，以適應不同的應用，在[1]中提供了許多原始影像經過不同壓縮技術壓縮後的 trace 檔。以電影 *Jurassic park* 及 *Troopers* 為例[1]，經過 MPEG4、H261 及 H.263 壓縮技術壓縮後的封包大小與產生區間分佈如圖 1-1 所示。我們可發現三種不同壓縮技術的封包大小分佈有著很大的差異性；並且在封包的產生間隔方面，H.263 有著高變動性的封包產生間隔，不同於 H.261 及 MPEG4 固定的間隔。不同壓縮技術所產生的串流特性統整如表 1-1 所示。

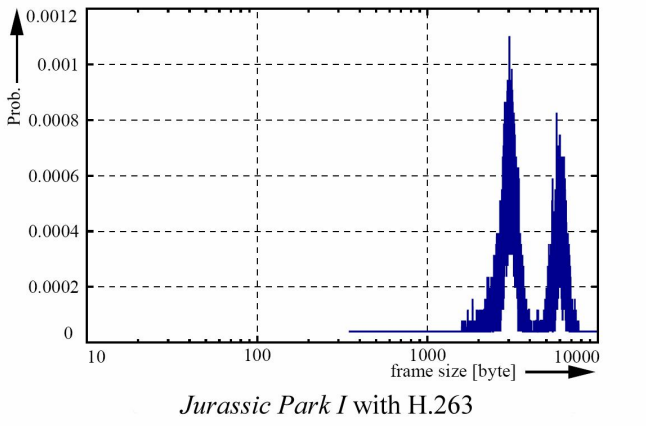
經由上述壓縮技術產生的影像串流便是產生變動性資料率 VBR (variable bit rate)的資料串流。如何確保 VBR 串流的服務品質(Quality of Service，QoS)是現今許多研究的主要課題。



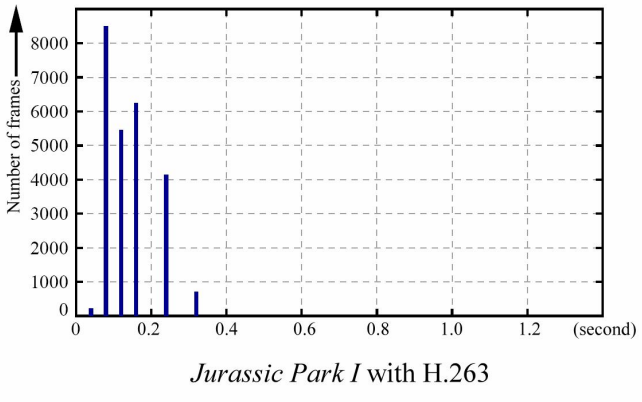
(a)



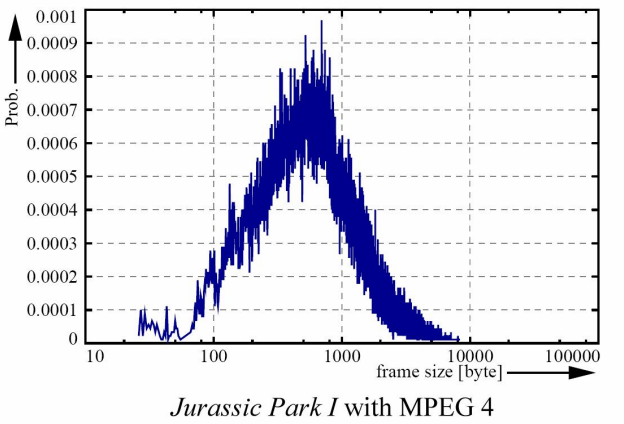
(b)



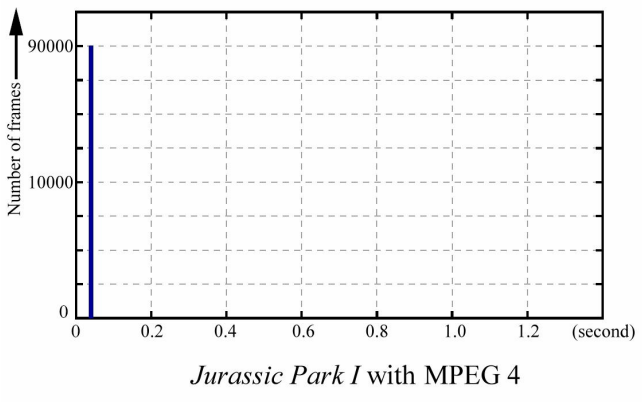
(c)



(d)



(e)



(f)

圖 1-1：封包分佈與產生區間分佈：(a)、(b) H.261；(c)、(d) H.263；(e)、(f)MPEG4。(資料來源：[1])

表 1-1：VBR 特性統整表

	H.261	MPEG4	H.263
Packet Size	*variable*	*variable*	*variable*
Packet interval	*constant*	*constant*	*variable*
Mean Data Rate	*constant*	*variable*	*constant*

在無線區域網路(wireless local area network，WLAN)中，以碰撞機制為基礎之 IEEE 802.11 DCF (distributed coordination function)，無法完整地確保即時性 VBR 串流之服務品質。因此，802.11e 標準[2]改善傳統 802.11 的無線網路媒介存取(medium access control，MAC)協定，以提供更多即時性資料的服務品質保證。

雖然 802.11e 提供了增強的服務品質保證，但對於 HCCA 排程的資料流量監督(policing)以及傳輸時間(TXOP Duration)分配仍然缺乏有效的管理。因此，在[3]裡提出了一個 TXOP timer 的機制，利用 QSTA (QoS Station)等待的時間及 timer 增長率(TD/mSI)，以配置變動的傳輸時間。TXOP timer 可視為類似有線網路中之 token bucket 的效果，如圖 1-2 所示。

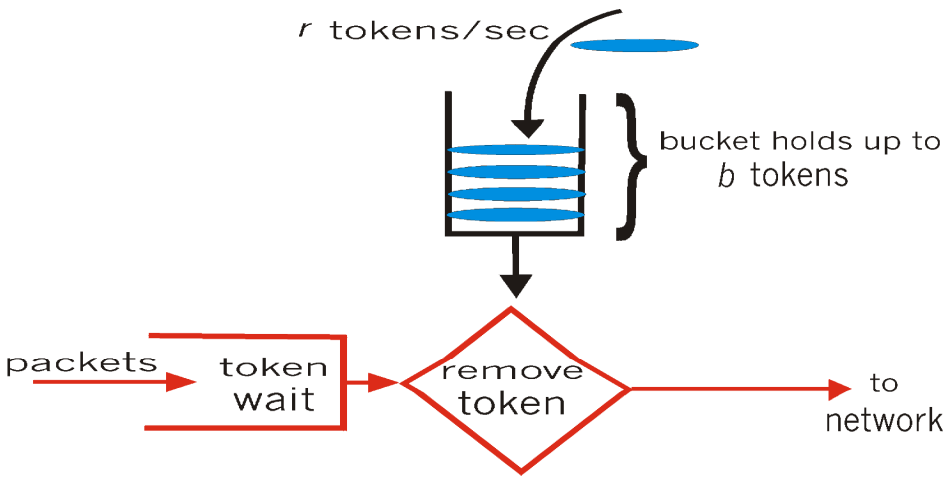


圖 1-2：token bucket 示意圖。

假設一個 token bucket 最大可以存放 b 個 token，並且以速率 r (token/sec)累增。倘若傳送一個封包需要一個 token，則所能傳送的封包數便取決於累積的 token 量。如果 token bucket 沒有任何 token 又需要傳送封包，便必須等待 token 產生。如此，可利用 token bucket 的 token 增長率來控制資料流量。而[3]所提出的 TXOP timer 機制，timer 會以 TD/mSI 的速率累增，其中 mSI (minimal Service interval)為資料流的最小輪詢週期，其計算方法為平均封包大小除以平均資料產生率(L/ρ)；所以 timer 會每隔 mSI 增長一個 TD。除此之外，為了保證過大封包可以有效傳送，TXOP timer 機制另設定了一個傳輸門檻值，只有 timer 在大於門檻值的情況下才能傳送資料，而此門檻值通常設定為資料流裡最大封包所需的傳送時間。所以如果一個資料流有不尋常的產生速率，便可以利用 TXOP timer 所設定的傳輸條件來限制資料的傳送量。

1.2 研究目的

本論文主要是在 802.11e HCCA TXOP 配置的基礎上製作一個動態調整 timer 傳輸條件的排程演算法。亦即考慮流量控管的前提下針對各個 QSTA 所產生資料流的個別特性調整傳輸條件，以提高無線網路通道的整體傳輸效能(throughput)、改善封包延遲(delay)及遺失率(loss rate)。

802.11e 的參考排程演算法(reference scheduler)中，僅考慮資料流(Traffic Stream，TS)的平均資料產生速率(mean data rate)、最大服務間隔時間(maximum service interval，MSI)、實體層速率(physical rate，PHY)等參數。因此，每一個 QSTA 所配置的傳送時間 TD 將會是一個平均值。然而，如果資料流的資料特性為 VBR，則選用平均的 TD 會是非常不符合效率的。

針對這個問題，[3]提出 SETT-EDD 演算法，改用 TXOP timer 的方式取代原本固定的 SI 以及平均的 TD，使得通道所能服務的 QSTA 個數能夠獲得改善，並利用 TXOP timer 的傳輸條件達到流量控管的效果。但 802.11e 參考排程演算法與 SETT-EDD 中 TD 之配置，皆是以估測(estimate)的方式獲得，因此不免造成通道的浪費。於是[4]提出了利用 Queue Size (位於 802.11e 封包內 header 的 QoS 欄位)來計算出正確的 TXOP 需求，以達到無線通道的完整利用。然而，[4]為了計算準確的 TD，封包的傳送必須分成『上傳 QS』、『分配 TD』兩階段來完成，需花費兩次的輪詢時間。另一方面，因為[3]、[4]的輪詢頻率取決於 QSTA 的 TXOP timer 增長速率(TD/mSI)；並且只有在 TXOP timer 大於傳輸門檻值才能接受輪詢。若服務的 VBR 為固定封包產生區間(如 H.261、MPEG4)，其資料產生較密集且規律，因此所計算出來的最小輪詢週期 mSI 會相近於資料流的封包產生週期，也就是 timer 的增長速率會相近於資料的產生速率(資料流每隔 mSI 產生一個封包，而 timer 每 mSI 增長一個 TD)。以圖 1-1 中的 H.261 為例，封包的產生週期固定為 0.04s，而由平均封包 1279byte 除以平均資料率 256kbit/s 而得的 mSI 為 0.0399s。然而，若是服務變動封包產生區間的 VBR (如 H.263)，其資料產生較分散且不規律，這使得資料流的封包比其他壓縮技術來的龐大，計算出來的最小輪詢週期 mSI 也就比多數的封包產生週期來的長；也就是說 timer 的增長速率會小於資料流內多數封包的產生速率，因此使得封包無法在的延遲範圍內輪詢而造成封包遺失。以圖 1-1 中的 H.263 為例，封包的產生週期為 0.04、0.08、0.12、0.16…等，而由平均封包 4533byte 除以平均資料率 256kbit/s 而得的 mSI 為 0.1416s。

另一方面，在[5]裡提出了一個適應性排程演算法以改善 SETT-EDD，使之能服務更多種類型的 VBR。主要的作法是不使用 TXOP timer 的機制(也就是不設定傳輸條件)，而將 QSTA 的服務開始時間(ServiceStartTime)參數列入排程的考慮因素，並根據不同類型的 VBR 提供適當的輪詢週期。而[5]又額外考慮 QSTA 的佇列情況以確保封包能在延遲範圍內接受輪詢。但由於[5]的 TD 仍是用平均值的方

式獲得，將造成可接受服務的 QSTA 個數僅維持一定數目。

為了改善封包的延遲及遺失而仍然能維持高吞吐量，本論文將設計一個結合[4][5]優點的 HCCA 排程器。首先，在考慮 QSTA 的佇列情況下，將 QSTA 的服務分成輪詢及傳送兩部分。在輪詢部分，我們以 VBR 最短的封包產生區間作為 QSTA 的輪詢週期，以準確的掌握變動封包產生區間的 VBR，並根據 QSTA 的 Queue Size 傳輸需求計算準確的傳輸時間；而在傳送部分我們主要針對 TXOP timer 的傳輸限制分別提出『無傳輸限制條件』以及『可根據 QS 傳送時間調整的變動傳輸條件』來改善傳輸延遲時間。

1.3 其他相關研究

除了上述以802.11e參考排程演算法為基礎之SETT-TDD [3]、ARROW [4]及[5]之外，在[6]中針對HCCA排程演算法的TXOP分配問題，HC會根據各個QSTA前一次輪詢及當次輪詢的佇列差量，利用PID控制器(proportional-integral-derivative controller)的誤差修正原理，以調整TD的分配。在[7]中，針對802.11e的HCCA提出考慮MAC層以及跨層級(cross-layer)的適應方法，並使用類神經網路(neural networks)的學習系統，修改MAC層的參數並進而改善802.11e的服務品質。另外，[8]針對HCCA排程演算法的TXOP分配問題，各個QSTA的佇列量將根據資料流的延遲時間會以權重(weight)表示，延遲越久的資料將會有越大的權重值，並且以線性規劃(linear programming)的方式計算最佳化的TD分配。然而，[6] 與[8]之計算並未將各個封包之延遲限制(delay bound)加入考慮，所得之模擬結果僅為封包之平均時間延遲。

1.4 論文架構

在本論文後續的章節中， 第二章主要是針對IEEE 802.11e HCCA、以及相關作法的詳細介紹(包含ns-2程式碼)，第三章將介紹我們所提出的通道排程演算法(包含ns-2程式碼)，第四章為實驗結果，第五章為結論。

第二章 相關知識及作法介紹

2.1 HCCA 機制介紹

當IEEE802.11e執行HCCA時，QSTA會先將資料流(Traffic Stream，TS)的相關參數(Traffic SPECification，TSPEC)上傳至混合控制器(hybrid controller，HC)，TSPEC的資訊格式如圖2-1所示：

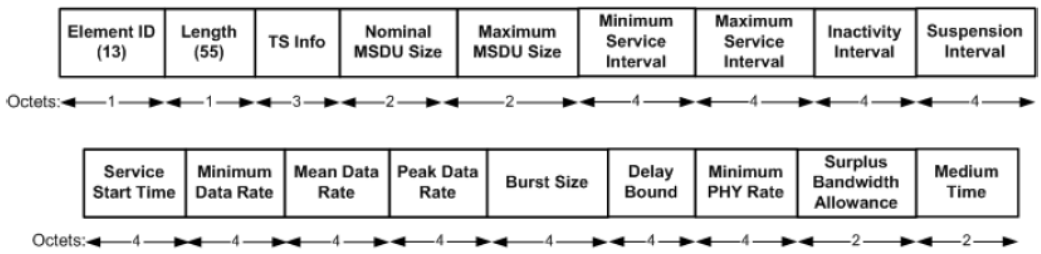


圖2-1：TSPEC資訊格式。(資料來源：[2])

由於802.11e使用封包叢集之作法，也就是，QSTA取得了通道使用權時，可以在給予的時間內連續傳送多個訊框，而不用重新競爭通道的使用權。因此，若 $QSTA_i$ 有 n_i 個資料流，可利用其TSPEC參數計算叢集服務(aggregate service)之排程參數：

- 最小 TXOP 長度(minimum TXOP Duration，mTD)：為 $QSTA_i$ 所有資料流中之最大封包產生時，所需花費的傳送時間。mTD 主要是為了

避免最大封包產生時，因 TD 太小而無法順利將其傳送出去，形成死結(deadlock)。M_{ij} 、R_{ij} 分別為第 j 個資料流最大封包大小和實體速率。

$$mTD_i = \max_{j\in[1,n_i]}\left(\frac{M_{ij}}{R_{ij}}\right) \quad (1)$$

- 最大 TXOP 長度(Maximum TXOP Duration，MTD)：當所有資料流均產生最大叢集時，所需花費的傳送時間。

$$MTD_i = \frac{\sum_{j=1}^{n_i} MBS_{ij}}{R_i} \quad (2)$$

其中，MBS_{ij} 為第 j 個資料流之最大叢集長度(maximum burst size)。

- 最小服務週期(minimum Service Interval，mSI)：為兩次 TXOP 配置之間的最小時間間隔。

$$mSI_i = \min_{j\in n_i}(mSI_{ij}) \quad (3)$$

其中，mSI_{ij} 為第 j 個資料流之最小服務週期。

- 最大服務週期(Maximum Service Interval，MSI)：為兩次 TXOP 配置之間的最大時間間隔。

$$MSI_i \le D_i - MTD_i \quad (4)$$

其中，$D_i = \min_{j\in n_i}(D_{ij})$ 。

2.2 HCCA 的運作

2.2.1 HCCA 運作原理

在HCF的機制中，HC (Hybrid Coordinator)於任何時刻都可以在等待一段PIFS (PCF interframe space)的時間後，立即傳送MSDU。由於PIFS的長度小於DIFS

(DCF interframe space)，因此HC擁有比較高的優先權等級。圖2-2為IEEE802.11e HCF的範例：在免競爭週期(contention-free period，CFP)中，HC所輪詢之QSTA，將得到一段Polled TXOP的時段以進行傳輸；而其他QSTA必須等待，直到被輪詢之後才能進行傳輸。在競爭週期(contention period，CP)中，除了各個QSTA會去競爭通道的使用權之外，HC可以藉由傳送一個QoS CF-Polled的訊框，配置QSTA一段Polled TXOP的時段。HC在無線傳輸媒介閒置了一個PIFS的時間後，不需要執行後退(backoff)程序就可以傳送QoS CF-Polled訊框。所以，HC可以利用其主控無線傳輸媒介存取的優勢，給予QSTA傳送即時性資料封包的機會。HC在競爭週期內，給予一個或多個QSTA存取的時期，稱之為CAP (Controlled Access Period)。

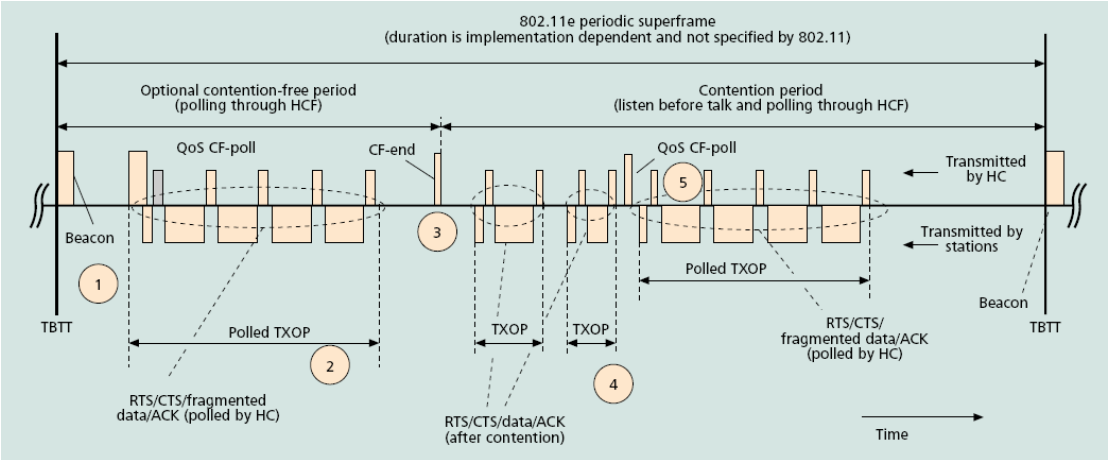


圖 2-2：IEEE802.11e HCF 範例，其中包含 CFP 與 CP 期間之 QoS CF-Poll。

(資料來源：[2])

2.2.2 ns-2 簡介

ns-2 是一個針對網路技術開發的免費模擬軟體，研究人員使用它可以很容易的進行網路技術的開發，而且發展到今天，它所包含的模組已經非常豐富，幾乎涉及到了網路技術的所有方面。所以，ns-2 成了目前學術界廣泛使用的一種網路模擬軟體。在每年國內外發表的有關網路技術的學術論文中，利用 ns-2 給出模擬結果的文章最多，通過這種方法得出的研究結果也是被學術界所普遍認可的。

ns-2 使用 C++和 Otcl 作為開發語言。ns-2 模擬分兩個層次；一個是僅編寫 OTcl 語言的層次；另一個是編寫 C++和 OTcl 語言的層次。前者利用 ns-2 已有的網路元素來實現模擬，不需修改 ns-2 本身，只需編寫 OTcl 腳本；而當 ns-2 沒有所需的網路元素時，則需要利用後者來增加所需網路元素，也就是增加新的 C++和 OTcl 類別，編寫新的 OTcl 劇本。

2.2.3 HCCA 的程式實作

在 HCCA 的模擬方面，將採用[9]所提出的模擬結構，如圖 2-3 所示：

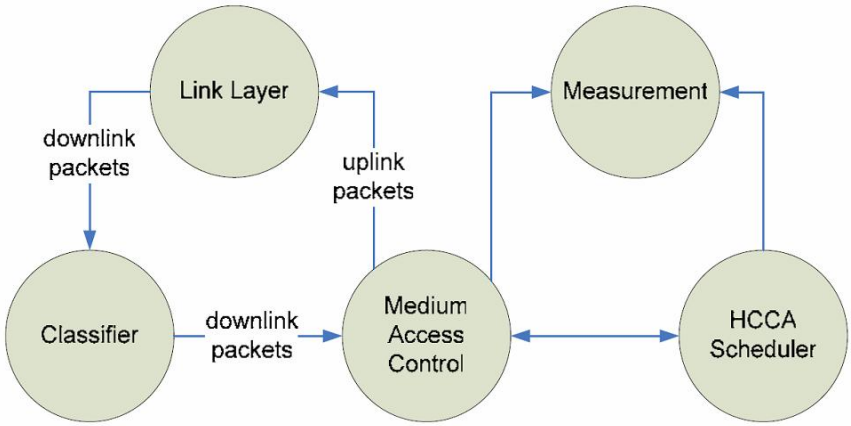


圖 2-3：HCCA 模擬結構。(資料來源：[9])

在此結構中最主要部份為 Classifier、HCCA Scheduler、MAC 模組。其中 Classifier 的主要工作是對可服務資料流所產生的封包做分類的動作。每一個被分類的封包將擁有一個 Traffic Identifier (TID)，決定此封包將於 HCCA 或 EDCA 週期處理，並且只有從 Link Layer 到 MAC Layer (downlink)的封包才需要做分類的動作，因為 uplink 的封包只是要傳送至更高的層級，不需要任何的排程或是優先權差別待遇。在 MAC 模組中主要的工作為處理資料傳遞時所需要的封包交換動作，並定義所有在封包交換的過程中會產生的事件，分別為：

- HCCA_HAS_CONTROL：掌握通道的使用權。在 QSTA 方面，只有當 QAP 發送輪詢的封包給 QSTA 才能擁有使用權，而在 QAP 方面，當通道閒置超

過 PIFS 或是 QSTA 傳完資料後便能取得使用權。

- HCCA_LOST_CONTROL：失去通道使用權。在 QSTA 方面，當資料傳完時便會失去使用權，在 QAP 方面，當發送輪詢的資訊給 QSTA，並且 QSTA 有做回應時便會輸去使用權。
- HCCA_DATA_RECV：當 QAP 或 QSTA 接收到的封包為資料封包時便會產生此事件。
- HCCA_RECV：當 QAP 或 QSTA 接收到任何封包時便會產生此事件。
- HCCA_SUCCESS：當 QAP 或 QSTA 成功傳送資料並且接收到 ACK 封包時便會產生此事件。
- HCCA_TRANSMIT：當 QAP 或 QSTA 須開始傳送封包時便會產生此事件。
- HCCA_TX_END：當 QSTA 使用完 TXOP 時，在 QAP 端便會產生此事件。
- HCCA_CAP_HAND ：當 CAP 週期開始執行時便會產生此事件。

圖 2-4 為 MAC 模組執行封包交換時，各個事件的發生點：

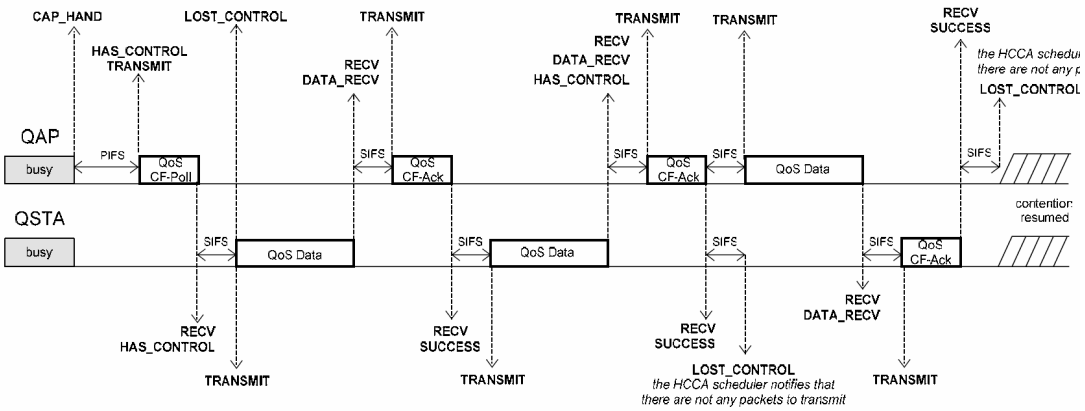


圖 2-4：MAC 封包交換程序。(資料來源：[9])

在開始執行 CAP 週期時，會產生一個 CAP_HAND 的事件。接著在經過 PIFS 的時間後 QAP 掌握通道的使用權，並且傳送一個 CF-Poll 的封包給 QSTA，因此會產生 HAS_CONTROL 以及 TRANSMIT 的事件。而當 QSTA 接收到 CF-Poll 封包的時候，便能獲得通道的使用權並且產生 RECV 以及 HAS_CONTROL 事件。

然後經過 SIFS 的時間後 QSTA 便會開始傳送資料封包，並產生 TRANSMIT 的事件。而當 QAP 接收到資料封包時便會產生 RECV 以及 DATA_RECV 的事件，並且經過 SIFS 時間傳送一個 ACK 封包以及產生 TRANSMIT 事件。接著 QSTA 接收到 ACK 封包時便會產生 RECV 以及 SUCCESS 事件代表一個資訊封包的完整傳遞。當 QSTA 使用完 TXOP 後，QAP 端便會產生 HAS_CONTROL 事件重新獲得通道使用權。

在執行 MAC 封包交換動作的時機最主要是由 HCCA Scheduler 所掌控的。HCCA Scheduler 中存在了一個 CAP_Timer 以通知 MAC 模組何時該進行 CAP 週期，並且 HCCA Scheduler 也會擁有一套排程演算法，負責產生輪詢表以通知 MAC 當 QAP 掌握通道使用權時該向哪一個 QSTA 傳送 CF-Poll 封包。其中本論文在以 ns-2 實現的部份最主要的工作，也就是將 HCCA Scheduler 演算法修改成本論文所提出的演算法。

2.2.4 NS2HCCA

NS2HCCA 為[9]提供的一個 802.11e 排程規約，在實作裡分別有 mac-802_11e.cc、mac-802_11e.h、mac-hccasched.cc、mac-hccasched.h、mac-hccasched_oneflow.cc、mac-hccasched_oneflow.h、mac-hccasched_periodic.cc、mac-hccasched_periodic.h、mac-hccasched_map.cc、mac-hccasched_map.h、mac-hccasched_map_ref.cc、mac-hccasched_map_ref.h。

以上檔案，mac-802_11e.cc、mac-802_11e.h 主要是負責封包的交換作業；mac-hccasched.cc、mac-hccasched.h 描述 QSTA 和 AP 皆須具備的功能函數；mac-hccasched_oneflow.cc、mac-hccasched_oneflow.h 以及 mac-hccasched_periodic.cc、mac-hccasched_periodic.h 分別描述 QSTA 和 AP 個別需擁有的功能函數；mac-hccasched_map.cc、mac-hccasched_map.h、

mac-hccasched_map_ref.cc、mac-hccasched_map_ref.h 定義了 AP 在做通道排程時所用的方法。

在 mac-802_11e 檔案裡定義了三個類別，分別為 Status、SchedulerTxData、Mac802_11e。Status 主要表示一個 QAP 或 QSTA 的狀態；SchedulerTxData 則表示 QAP 或 QSTA 所要傳送的資料型態；例如：poll frame、data frame、ack frame 等；而 Mac802_11e 用來描述 QAP 或 QSTA 對於通道以及資料的控制函數。

在 mac-hccasched、mac-hccasched_oneflow、mac-hccasched_periodic 中分別定義了 MacHccaSched、MacHccaSchedQSTA_oneflow、MacHccaSchedQAP_per 三個類別，其中 MacHccaSchedQSTA_oneflow、MacHccaSchedQAP_per 皆為 MacHccaSched 的子類別。此三個類別分主要定義 QSTA 或 QAP 封包產生所需執行的 addTSPEC、enque、deque 等。

mac-hccasched_map、mac-hccasched_map_ref 則分別定義了 MacHccaSchedMap、MacHccaSchedMap_ref 兩個類別，其中 MacHccaSchedMap_ref 為 MacHccaSchedMap 的子類別。主要是用來計算有關通道排程的各個參數值，如各個 QSTA 所擁有的 mSI 及 MSI，Simple scheduler 所需決定的 SI 和 TD，以及決定出排程的 polling List。

2.3 HCCA 參考排程演算法(或稱為 Simple Scheduler)

2.3.1 排程原理

在 802.11e 的參考排程演算法中，以輪詢方式進行資料傳輸的免競爭週期或 CAP 裡，必須根據下列步驟計算服務週期(Service Interval，SI)、TXOP 大小、以

及允入控制單元(Admission Control Unit，ACU)。

Step 1：HC 根據所有 STAs 提出 TSPEC 參數中之最大服務週期(Maximum Service Interval，MSI)，挑選出其中最小的 minMax SI。接著再由導引信號週期(beacon interval)的因數中，選出小於 minMax SI 的最大者，作為共同的 SI。例如：

- Beacon Interval = 500 ms。
- 三個 STAs 的 MSI 要求分別為 180 ms、150 ms 和 200 ms。
- 則選出 125 ms 作為共同的 SI (因 125 為 500 的因數中，小於 180、150 和 200 之最大者)。

Step 2：HC 根據 STA 所提出之其他 TSPEC 參數，計算應分配多少 polled TXOP 給該 STA。

- 首先，利用 Mean Data Rate (r_i)及 Nominal MSDU Size (L_i)，計算 $QSTA_i$ 在 SI 內可以傳送的 MSDUs 數目 N_i ：

$$N_i = \left\lceil \frac{SI \times r_i}{L_i} \right\rceil \qquad (5)$$

- 然後，利用 PHY (R_i)、Maximum MSDU Size (M)、Overhead (O)計算分配給 $QSTA_i$ 的 $TXOP_i$ (應至少可傳輸一個 Maximal MSDU)：

$$TXOP_i = \max\left(\frac{N_i \times L_i}{R_i} + O, \frac{M}{R_i} + O \right) \qquad (6)$$

- 最後，HC (如圖 2-5 所示，以 i、j、k 為例)在每個 SI 內，依序對各個 STA 提供 Polled TXOP：

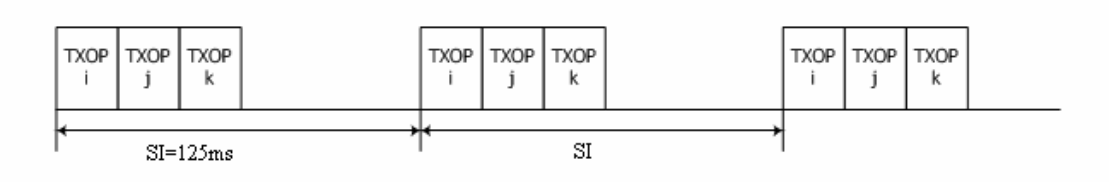


圖 2-5：HC 在共同的服務區間(SI)內，依序對各個 STA 提供 Polled TXOP。

(資料來源：[2])

Step 3 (ACU)：若系統已服務 *k* 個資料流，當第 *k+1* 個資料流提出申請時：

■ 若

$$\frac{TXOP_{k+1}}{SI} + \sum_{i=1}^{k} \frac{TXOP_k}{SI} \le \frac{T - T_{CP}}{T} \quad , \qquad (7)$$

其中 T 為 Beacon 週期，而 T_{CP} 為 EDCA 之週期長度。

■ 則表示 SI 尚有空間，ACU 將允許第 *k+1* 條資料流進入系統。

2.3.2 程式實作

Simple Scheduler 是在 MacHccaSchedMap_ref 裡實現的排程演算法，主要程式碼如下：

```
txop_desc*
MacHccaSchedMap_ref::createTXOPDesc(TSPEC& tspec)
{
    //每建立一個資料流，便會建立一個新的 txop_desc 來存放新加入的 tspec 參
    //數
    txop_desc* newdesc          = new txop_desc;
    newdesc->next               = 0;
    newdesc->tid                = tspec.tid;
    newdesc->qsta               = tspec.qsta;
    newdesc->direction          = tspec.direction;
    newdesc->periodic           = tspec.periodic;
    newdesc->mean_rate          = tspec.mean_data_rate;
    newdesc->max_SI             = tspec.maximum_SI;
    newdesc->mean_sdu_size      = tspec.nominal_SDU_size;
    newdesc->max_sdu_size       = tspec.maximum_SDU_size;
    newdesc->db                 = tspec.delay_bound;
    //計算出一個平均封包所需的傳送時間
    newdesc->mean_time          = phymib_.getSIFS() +
    mac_->txtime( tspec.nominal_SDU_size + phymib_.getHdrLen11e(),
    mac_->getDataRate()) + phymib_.getSIFS() +
    mac_->txtime(phymib_.getHdrLen11e(), mac_->getBasicRate());
    //計算出一個最大封包所需的傳送時間
    newdesc->max_time           = phymib_.getSIFS() +
    mac_->txtime( tspec.maximum_SDU_size + phymib_.getHdrLen11e(),
```

```
    mac_->getDataRate()) + phymib_.getSIFS() +
    mac_->txtime(phymib_.getHdrLen11e(), mac_->getBasicRate());
    //如果新加入的 tspec 是 DOWNLINK 也就是由 QAP 傳至 QSTA 則不用
    //另外計算 polling 時間，反之則需計算
    if ( tspec.direction == TSPEC::DOWNLINK )
    {
        //pi 即為 polling 時間
        newdesc->pi = 0.0;
    }
    else
    {
        //polling 時間設定為傳送一個 polling 封包的時間加上 SIFS
        newdesc->pi = mac_->txtime(phymib_.getHdrLen11e(), mac_->getBasicRate()) +
        phymib_.getSIFS();
    }
    //傳回 newdesc
    return newdesc;
}


int
MacHccaSchedMap_ref::createMap (TSPEC& tspec)
{
    txop_desc* p;
    //建立一個新的 txop_desc 並存入由前一個函式所產生的*txop_desc
    txop_desc* newdesc = createTXOPDesc(tspec);
    //並將 newdesc 加入 txop_desc 串列的最後面
    //head 為 MacHccaSchedMap_ref 的一個成員函數，存放 txop_desc 串列第一
    個位址
    if ( head != 0 )
    {
        p = head;
        while ( p->next != 0 ) p = p->next;
        p->next = newdesc;
    }
    else
    {
        head = newdesc;
    }
    //當新增加一個 newdesc，n_elem 便會累增，n_elem 存放 txop_desc 串列的
    個數
    ++n_elem;
    //計算出符合 txop_desc 串列裡每一個 txop_desc 的 SI
    SI = -1.0;
    p   = head;
    while ( p != 0 )
    {
        if ( SI < 0 || SI > p->max_SI )
            SI = p->max_SI;
        p = p->next;
    }
    //計算出各個 txop_desc 的 txop duration
    p = head;
```

```
while ( p != 0 )
{
    //如果 txop_desc 為週期性且為固定封包大小則將 txop 設定為下
    if(p->periodic || !ref_vbr_MaxMSDU)
    {
        //計算出一個 SI 平均產生的封包個數 N
        double N = ceil(SI * p->mean_rate / (8.0 * p->mean_sdu_size));
        //取 N*NTD 及 mTD 的兩者最大值
        p->txop =mac_->getPhyMib().getMaxPropagationDelay()
        + max( N * p->mean_time + p->pi, p->max_time + p->pi);
    }
    else
    {
        //將 SI 及封包大小皆換為最大值，再算出產生最大封包的個數
        //此為程式作者提出的方式，非為標準定義
        double num_pkt =
        (int)floor(p->max_SI * p->mean_rate /(p->max_sdu_size * 8));
        p->txop = num_pkt * p->max_time;
        double rem = (int)ceil(p->max_SI * p->mean_rate / 8)
        - num_pkt * p->max_sdu_size;

        if(rem > 1)
            p->txop += mac_->getFrameSeqTime(rem);
        if(p->txop < p->max_time)
        {
            p->txop = p->max_time;
        }
        p->txop += mac_->getPhyMib().getMaxPropagationDelay() +
        p->pi + phymib_.getPIFS();
    }
    //如果計算出來的 txop 超過 TXOP_LIMIT 則設為 TXOP_LIMIT
    if ( p->txop > MAC_MAX_TXOP_LIMIT )
    {
        p->txop = MAC_MAX_TXOP_LIMIT;
    }
    p = p->next;
}
//加總串列裡所有的 txop 長度，並確認小於 SI
double txop_sum = 0.0;
p = head;
while ( p != 0 )
{
    txop_sum += p->txop;
    p = p->next;
}
if ( txop_sum > SI )
{
    //當回傳 1 代表 txop 製作失敗
    return 1;
}
curr_desc = head;
first_call = 1;
```

```
    // 傳回 0 代表 list 完成
    return 0;
}

//當 QAP 選擇到第一個 txop_desc 作服務後，便將指標指向下一個 txop_desc
void
MacHccaSchedMap_ref::next_txop(void)
{
    if(curr_desc->next)
    {
        TXOP_start_time = TXOP_start_time + curr_desc->txop –
        phymib_.getSIFS();
        curr_desc = curr_desc->next;
    }
    else
    {
        SI_start_time += SI;
        TXOP_start_time = SI_start_time - phymib_.getSIFS();
        curr_desc = head;
   }
}
```

2.4 SETT-EDD 排程演算法

2.4.1 排程原理

雖然 Simple 排程演算法提供一個所有 QSTA 皆能接受之共同的 SI，就固定資料量及固定產生時間之 CBR 有不錯的成效；但是，對於變動資料量及變動產生時間之 VBR，則無法分配最適當的 polled TXOP。因此，當資料量突然增加，便會造成傳送時間不足，導致資料必須延遲更多時間甚至造成資料遺失；而當資料量較小時，不免形成通道的浪費。

針對 VBR，許多研究學者提出比 Simple 排程演算法更具成效的排程演算法。[3]首先提出 SETT-EDD (Scheduling based on Estimated Transmission Times – Earliest Due Date)演算法，此演算法針對 Simple 排程演算法有下列改進作法：

- 以不同的 SI，輪詢不同的 QSTA (而非以相同的 SI，輪詢所有的 QSTA)

■ 允許分配不同長度的 TXOP (而非固定的 TXOP)

SETT-EDD 演算法中，每個 QSTA 都設置一個 TXOP 計時器(Timer)，並以 QSTA 個別的 TD/mSI 速率增長(直到 Timer 的最大值 MTD)，QSTA 所分配的 TXOP 便由當時 Timer 量所決定。除此之外，SETT-EDD 演算法也計算出 QSTA 下次最早輪詢的時間 $t+mSI$ 、以及最晚輪詢的時間 $t+mSI$ (t 為 QSTA 當次輪詢的時間)，來決定被輪詢之優先順序。其中，最晚輪詢的時間又稱為到期時間(Due Date)。所以當 QAP 於 t'，在決定下一個輪詢的 QSTA 時，會執行以下步驟：

Step 1：首先，每個 QSTA 會根據 TD/mSI 累積 Timer；並且 QAP 會在排程的當時(t')，先從所有 QSTAs 中，挑選出符合 $Timer \geq mTD$ 的 QSTA。

Step 2：接著，QAP 由符合 **Step 1** 條件的 QSTAs 中，挑選出到期時間距離現在時間(t')最接近的 QSTA (也就是最早到期者)進行輪詢；而所分配的 TXOP 長度，等於 Timer 的量。

Step 3：最後，當 QSTA 傳輸結束，將通道使用權還給 AP 時，QSTA 會將所使用之 TXOP 時間從 Timer 中扣除；接著，該 QSTA 之 Timer 會繼續增長。而 QAP 則回到 **Step 1**，進行下一個 QSTA 的輪詢。

舉例說明，假設系統中有 i、j、k 三個QSTA，其Timer量、mTD、Timer增長速率(TD/mSI)、及Due Date，分別如圖2-6所示：

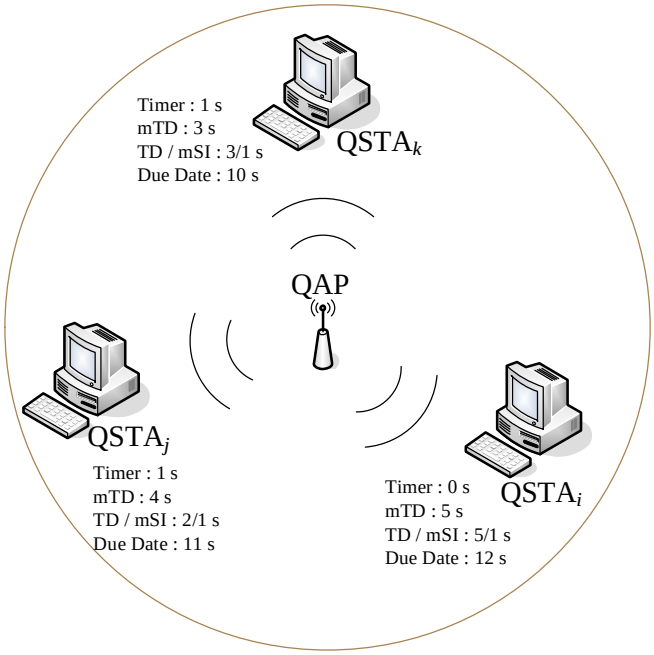


圖2-6：SETT-EDD範例：系統中有i、j、k三個QSTAs，QAP根據其Timer量、mTD、Timer增長速率(TD/mSI)、及Due Date決定輪詢對象。

經過1 s後，三個QSTA之Timer將分別為5 s、3 s、及4 s；由於mTD分別為5 s、4 s、及3 s，因此可知$QSTA_i$與$QSTA_k$滿足**Step 1**條件；但由於$QSTA_k$擁有較接近的到期時間，便由$QSTA_k$取得輪詢資格，並由QAP分配得4 s的TXOP長度。

如果$QSTA_i$被輪詢的時間為 t，並且在 t 時間消耗了TD_i，由TD/mSI可知再經過mSI的時間後，$Timer_i$的量就有可能會大於等於mTD_i；所以，$QSTA_i$下次能夠被輪詢的最早時間至少為$t+mSI_i$。倘若$QSTA_i$已經滿足$Timer_i \geq mTD_i$的條件，但其他QSTA卻擁有更接近的Due Date，則會由其他QSTA優先取得服務；直到$QSTA_i$的Due Date相較於其他QSTA為最小的時後。因此，倘若下次服務的時間為t'則應滿足：

$$t + mSI_i \leq t' \leq t + MSI_i \qquad (8)$$

2.4.2 程式實作

關於我們所撰寫的 SETT-EDD Scheduler 是由在[9]所提出的程式碼的基礎上加以增設及修改函式。而關於 SETT-EDD 程式的實作方法的示意圖如圖 2-7 所示，在 simple 的程式碼裡我們在每個 txop_desc 裡多加了一個 timer，增長的速率為 TD/mSI ，並且增加 Add_Timer( )函式做 timer 的增長。另一方面，因為本論文所設定的封包延遲時間並不會也並不允許在同一時間內同時傳送兩個封包以上的狀態，換句話說，最大的叢集封包大小(Maximum burst size)僅等於單一個最大封包大小(Maximum MSDU size)，所以本論文將 timer 的最大量設定為單次傳送最大封包所需要的時間(mTD)。因此在當次 txop_desc 的 TXOP 結束後，我們會在結束的時間點利用 Add_Timer( )計算出 timer 下次累積到 mTD 的時間，也就是下次的服務時間(service_time)，並將服務時間加上 max_SI 作為下次到期時間(EDD_time)。接著挑選出最早到期的 QSTA(EDD_STA)作為下次服務的 txop_desc，並將 EDD_STA 的 txop 設為 txop_timer 值，txop_timer 值清空。

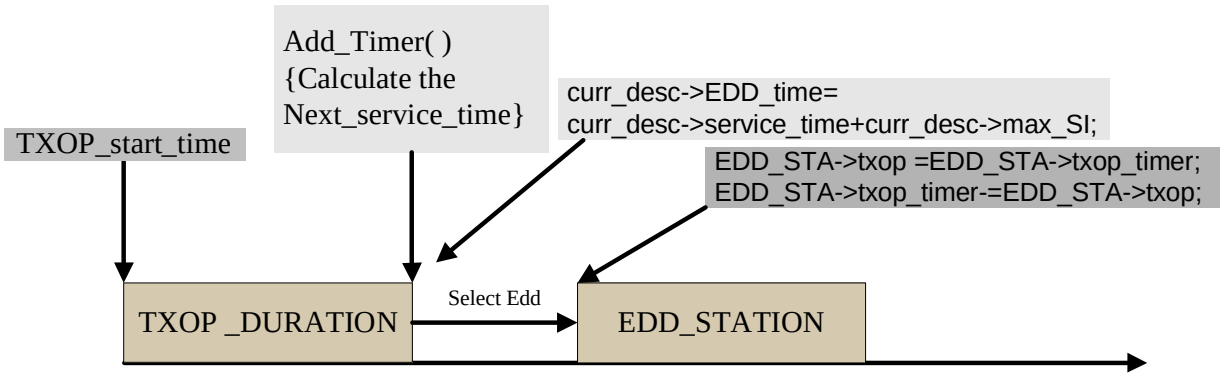


圖 2-7：SETT-EDD 模擬結構示意圖

針對 Add_Timer( )的詳細說明如圖 2-8 所示。Add_Timer( )會從 QSTA 當次服務的時間以及 QSTA 當次服務完的 timer 量，利用 timer 的增長率 TD/mSI 來計算增長到 mTD 所需要的時間，接著再與 min_SI 取最大者作為下次可以服務的時間。

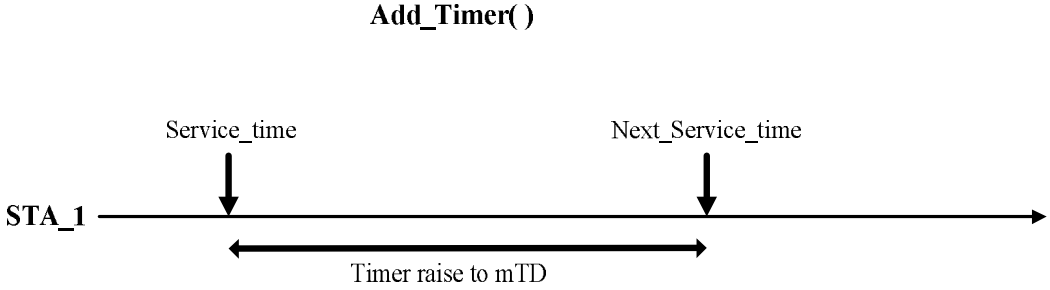


圖 2-8：Add_Timer()結構示意圖

simple scheduler 程式碼的修改部分及增加的部分如下：

```
int
MacHccaSchedMap_ref::createMap (TSPEC& tspec)
{
	txop_desc* p;
.....
	//增加部分
	p = head;
	while ( p != 0 )
	{
		p->min_SI=p->mean_sdu_size*8/p->mean_rate;
		p->txop_timer=p->txop;
		p->timer_max=p->txop;
		p->timer_rate=p->txop/p->min_SI;
		p->service_time=0;
		p->EDD_time=p->max_SI
		p=p->next;
	}
.....
}

void
MacHccaSchedMap_ref::next_txop(void)
{
	//修改部分***********
	txop_desc* p;
	txop_desc* EDD;
	//利用 add_timer()決定出下次的服務時間(service_time)，並將服務時間加上
	//max_SI 作為下次到期時間(EDD_time)
	add_timer();
	curr_desc->EDD_time= curr_desc->service_time+curr_desc->max_SI;
	//將時間點移至 txop 結束的時間
	TXOP_start_time = TXOP_start_time + curr_desc->txop – phymib_.getSIFS();
	EDD = new txop_desc;
	p = head;
	EDD->EDD_time=-1.0;
	//選出最早到期的 txop_desc
	while (p != 0)
```

```
        {
            if ( EDD->EDD_time < 0 || p->EDD_time < EDD->EDD_time)
            {
                EDD=p;
            }
            p=p->next;
        }
        curr_desc=EDD;
        free EDD;
        //如果所選到 txop_desc 的 service_time 大於 TXOP_start_time 則將排程時間
        //移至 service_time
        if(TXOP_start_time <= curr_desc->service_time)
        {
            TXOP_start_time=curr_desc->service_time;
        }
        //將所選到 txop_desc 的 txop 設定為 timer 值，並將 timer 清空
        curr_desc->txop =curr_desc->txop_timer;
        curr_desc->txop_timer=0;
        //*******************
        //將以下程式碼刪除
        if(curr_desc->next)
        {
            TXOP_start_time = TXOP_start_time + curr_desc->txop
            phymib_.getSIFS();
           curr_desc = curr_desc->next;
        }
        else
        {
            SI_start_time += SI;
            TXOP_start_time = SI_start_time - phymib_.getSIFS();
            curr_desc = head;
        }
}
void
MacHccaSchedMap_ref::add_timer(void)
{
        txop_desc* p;
        p = head;
        double delta_TD;
        double delta_time;
        while (p != 0)
        {
            //只針對當次服務的 QSTA 計算出 timer 大於 mTD 所需要的時間，也就
            //是下次的服務時間
            if(p->qsta==curr_desc->qsta)
            {
                delta_TD= p->timer_max - p->txop_timer;
                delta_time= delta_TD/p->timer_rate;
                if(delta_TD>=0)
                {
                    //比較 timer 滿足 mTD 所需要的時間(delta_time)以及 min_SI
                    if(TXOP_start_time +delta_time > TXOP_start_time +min_SI)
                    {
```

```
                    //如果 delta_time 大於 min_SI 則設定下次服務時間為當次
                    //服務時間加上 delta_time
                    p->service_time=TXOP_start_time +delta_time;
                }
                else
                {
                    //如果小於則設定下次服務時間為當次服務時間加上//min_SI
                    p->service_time=TXOP_start_time +min_SI;
                }
                p->txop_timer=p->timer_max;
            }
        }
        p=p->next;
    }
}
```

2.5 ARROW 排程演算法

2.5.1 排程原理

SETT-EDD 排程演算法雖然能夠改善 HCF 的吞吐量,但是在 TD 方面仍是利用估測的方式計算，不但在通道的使用方面容易造成浪費，當資料為 VBR 時，也有可能導致分配不足的現象。因此，[4]提出了 ARROW (Adaptive Resource Reservation Over WLANs)演算法,依循 SETT-EDD 的理論基礎,再搭配 QueueSize (位於 802.11e 的 QoS field)的資訊，以達到 TXOP 的正確配置。

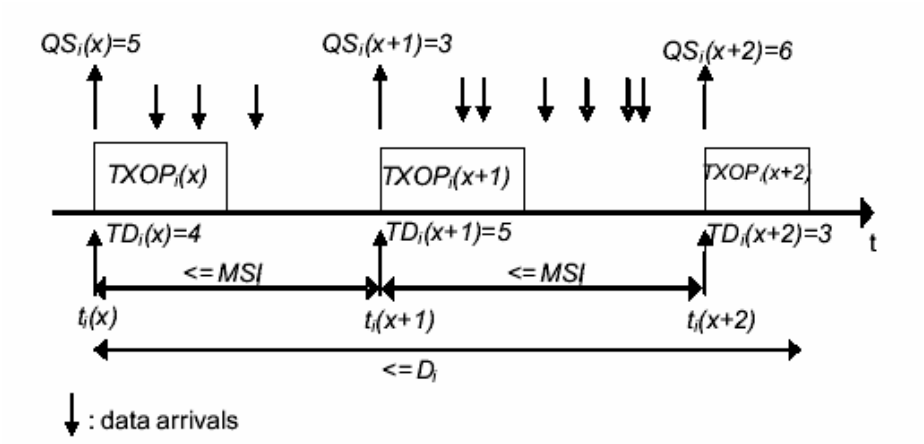


圖 2-9：ARROW 傳輸範例：$QSTA_i$ 分別於第 *x, x+1,*與 *x+2* 次輪詢時，配置之 TD 與前一次輪詢上傳 QS 之間的關係示意圖。(資料來源：[4])

以圖 2-9 進行說明。在$t_i(x)$時，QAP 會根據前一個 TXOP 所上傳的 QoS field 以及 TSPEC 參數，分配給$QSTA_i$一段$TXOP_i(x)$ (或$TD_i(x)$)的傳輸時間；而當$QSTA_i$於$TXOP_i(x)$期間上傳資料時，也會利用 QoS field 將 QueueSize ($QS_i(x)$)資訊上傳至 QAP。因此，在$t_i(x+1)$時，QAP 會根據$QSTA_i$所提出的$QS_i(x)$需求，分配$TXOP_i(x+1)$，長度為$TD_i(x+1)(=QS_i(x))$。而在$[t_i(x),\ t_i(x+1)]$間，若$QSTA_i$有新資料產生，$QSTA_i$便會利用所分配的$TXOP_i(x+1)$時間，上傳新的 QueueSize ($QS_i(x+1)$)；同樣地，在$t_i(x+2)$，QAP 根據$QS_i(x+1)$需求，分配$TD_i(x+2)$ $(=QS_i(x+1))$的時間給$QSTA_i$，而可再上傳$QS_i(x+2)$。從以上的敘述可以發現，經由 QoS field，QAP 將可以獲得 QSTA 最正確的即時需求，進而分配最正確的 TD。因此，當資料型態為 VBR 時，可以隨時掌握用戶端正確的封包資訊。

然而，在$[t_i(x),\ t_i(x+1)]$所產生的封包，必須要等到$t_i(x+2)$才能被傳送。因此，為了不超過封包的 Delay Bound，假設最壞的情況下兩次的服務週期(service intervals)皆為MSI_i，如圖 2-9，且$TD_i(x+2)$恰等於MTD_i時，下式必須滿足：

$$D_i \geq 2MSI_i + MTD_i \Leftrightarrow MSI_i \leq \frac{D_i - MTD_i}{2} \qquad (9)$$

若加入考慮重傳的可能，則上式可表示成：

$$MSI_i \leq \frac{D_i - MTD_i}{2+m} \qquad (10)$$

其中，m為最大的重傳次數。

而在實際排程將會遵循以下流程：

Step 1：排程器等待通道變成閒置(idle)狀態。

Step 2：

(a)接著從所有加入排程的 *QSTA* 挑選出滿足 $t_i + mSI_i$ 。

(b)並且 *TXOP timer* 大於最大封包的傳輸時間(*mTD*)。

Step 3：如果沒有任何 *QSTA* 滿足 **Step2** 的條件則繼續等待至至少有一個 *QSTA* 滿足。

Step 4：如果至少有一個 *QSTA* 滿足，則選擇出最早到期的 *QSTA*，也就是選擇擁有最小 $t_i + MSI_i$ 的 $QSTA_i$。

Step 5：

(a)接著根據各個 $QSTA_i$ 所屬資料流的 QS_{ij} 計算 TD_{ij}

$$TD_{ij} = \max\left(\frac{QS_{ij}}{R_{ij}} + O, mTD_{ij}\right) \tag{11}$$

而若 QS 為零，則 *TD* 配置為傳送 *Null-data* 的時間。

(b)而 $QSTA_i$ 總共的所需要的 TD_i 為

$$TD_i = \sum_{j=1}^{n_i} TD_{ij} \tag{12}$$

(c)再將 TD_i 與 $QSTA_i$ 的 *TXOP timer* (T_i)取最小值

$$TD_i = \min(TD_i, T_i) \tag{13}$$

Step 6：接著將 T_i 減去 TD_i 後回到 **Step1**。

2.5.2 程式實作

在 ARROW 實作方面，主要是在我們所實作的 SETT-EDD 模擬上做進一步的修改。為了達到準確分配 TXOP 的效果，我們在 802.11e 的每個資料封包的標頭加入了一個 QueueSize 的欄位，以便 QSTA 上傳資料時能將 QS 資訊一併上傳。而在服務週期方面，也根據 ARROW 文獻將 MSI 縮短為一半。其新增的欄位以及修改程式如下：

```
struct hdr_mac802_11e {
```

```
    struct frame_control dh_fc;
    u_int16_t                dh_duration;
    int our_queue_size;//新增加的欄位，存放 QS 資訊
    u_char                   dh_ra[ETHER_ADDR_LEN];
    u_char                   dh_ta[ETHER_ADDR_LEN];
    u_char                   dh_3a[ETHER_ADDR_LEN];
    u_int16_t                dh_scontrol;
    u_char                   dh_body[0];
    struct qos_control     dh_qc;
};
```

以上結構位於 802.11.h 中

```
int
MacHccaSchedMap_ref::createMap (TSPEC& tspec)
{
      txop_desc* p;
.....
      //修改 SETT-EDD 部分
      p = head;
      while ( p != 0 )
      {
            //將 max_SI 縮短為一半並設定為 txop_desc 第一次上傳的到期時間
            p->EDD_time=p->max_SI/2;
            p->min_SI=p->mean_sdu_size*8/p->mean_rate;
            p->txop_timer=p->txop;
            p->timer_max=p->txop;
            //將 timer_rate 改為每 mSI 增長 p->mean_time+p->pi
            p->timer_rate= (p->mean_time+p->pi)/p->min_SI;
            p->service_time=0
            p=p->next;
      }
.....
}


void
MacHccaSchedMap_ref::next_txop(void)
{
      //修改 SETT-EDD 部分**********
      txop_desc* p;
      txop_desc* EDD;
      add_timer();
      //將下次的 service_time 加上 max_SI/2 作為下次的 EDD_time
      curr_desc->EDD_time= curr_desc->service_time+curr_desc->max_SI/2;
......
}
```

// deque 為 MacHccaSchedQAP_per 的成員函式，其中一部份功能為分配 txop //duration 給 QSTA，而我們在分配 txop duration 的部分作了修改。

```
void
MacHccaSchedQAP_per::deque(SchedulerTxData& schedTx_)
{
.....
	//執行 update_qs()更新 AP 所擁有的 QSTA 佇列資訊
	update_qs();
	//接著執行 update_txop(desc->qsta)更新 txop duration
	map_builder_->update_txop(desc->qsta);
.....
}


void
MacHccaSchedQAP_per::update_qs(void)
{
	//更新每一個 Queue_Size
	for(int i=0;i<21;i++)
	{
		map_builder_->qsta_qs[i]=queue_packet_size[i];
	}
}


void
MacHccaSchedMap_ref::update_txop(int qsta)
{
	txop_desc* p = head;
	double new_txop = 0;
	while ( p != 0 )
	{
		if(p->qsta==qsta)
		{
			//根據 Queue_Size 計算出新的 txop duration
			new_txop = phymib_.getSIFS() + mac_->txtime( qsta_qs[p->qsta] +
					phymib_.getHdrLen11e(), mac_->getDataRate())+
					phymib_.getSIFS() +
					mac_->txtime(phymib_.getHdrLen11e(), mac_->getBasicRate());
			if(qsta_qs[p->qsta]!=0)
			{
				p->txop=new_txop + p->pi;
				//將 timer 減去 txop
				if(p->txop_timer<p->txop)
				{
					p->txop=p->txop_timer;
				}
				p->txop_timer-= p->txop;
			}
			else
			{
				p->txop=p->pi;
			}
		}
		p=p->next;
```

```
    }
}
```

2.6 適應性排程演算法

2.6.1 排程原理

由於ARROW的輪詢週期是由*TXOP timer*的增長率所決定，所以輪詢週期為一平均值，倘若服務的VBR為變動性區間的類型，則將會造成封包超過延遲範圍而遺失。因此[5]提出了適應性排程演算法(Adaptive Multimedia QoS Scheduler)，根據所服務的VBR類型提供不同的輪詢週期。首先在QAP端必須獲得所有加入排程資料流的服務開始時間(Service Start Time)，以及資料流的最小服務區間，接著在排程的過程中，若資料流上次的服務時間為t'則下次的服務時間t_{due_j}及到期時間t_{dead_j}將由下式決定：

$$\begin{aligned} t_{due_j} &= SST_j + k \cdot mSI_j \\ k &= \min_{l}\{t' \le (SST_j + l \cdot mSI_j)\} \\ t_{dead_j} &= t_{due_j} + MSI_j \end{aligned} \qquad (14)$$

如此經由 SST 及 mSI 可推算出最適當的輪詢時間。以圖 2-10 為例，倘若當次開始服務時間為圖中 Polling time 位置所示，則經由開始服務時間 Start Service Time 即可推算出下次服務時間 t_{due} 為兩倍的 mSI。

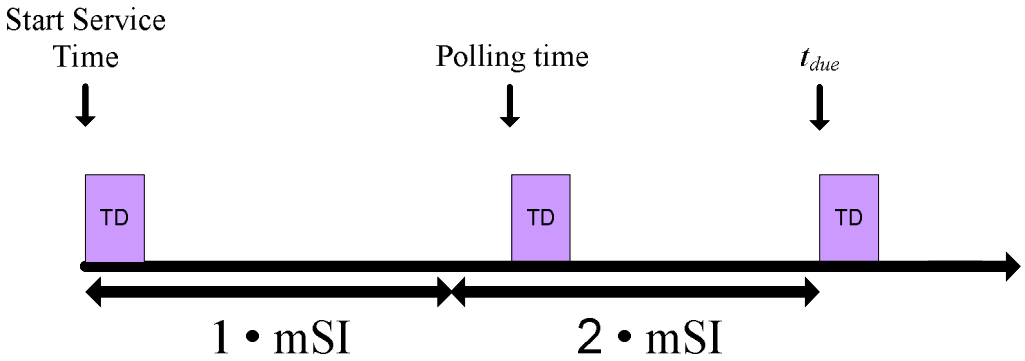


圖 2-10：服務時間 t_{due} 與 Start Service Time 的關係示意圖。

此外，[5]又額外考慮 *QSTA* 的佇列情況以確保封包能在延遲範圍內接受輪詢，若 t' 及 t'' 分別為 *QSTA* 當次 *TXOP* 的開始及結束時間，t_{duej} 及 t_{deadj} 分別為 *QSTA* 所屬資料流 j 的下一次的服務及到期時間，t_{reqj} 為資料流 j 的傳送需求，則 *QSTA* 下次服務的 *mQSI* 及 *MQSI* 計算如 15 式所示：

$$mQSI = \begin{cases} \min\limits_{1\le j\le n}\{t_{duej} - t''\}, if \forall j : t_{reqj} = 0 \\ 0, if \exists j : t_{reqj} \neq 0, if \exists j : t_{reqj} \neq 0 \end{cases}$$
$$MQSI = \begin{cases} \min\limits_{1\le j\le n}\{t_{deadj} - t''\}, if \forall j : t_{reqj} = 0 \\ \min\limits_{j\in\{t_{reqj}\neq 0\}}\{t' + MSI_j\}, if \exists j : t_{reqj} \neq 0 \end{cases} \quad (15)$$

式中如果 *QSTA* 的所有資料流 t_{reqj} 皆為零，則代表 *QSTA* 的佇列未存放任何資料，所以將 *QSTA* 下次服務的 *mQSI* 及 *MQSI* 設定為所屬資料流內最小的服務及到期時間；倘若至少有一個 t_{reqj} 不為零，則代表 *QSTA* 有資料產生並存放於佇列中，所以便將服務時間設為零，到期時間設為 $t' + MSI_j$，代表可即刻被服務。

2.6.2 程式實作

在適應性排程的程式實作方面，主要是捨棄了原 SETT-EDD 的 timer 作法而改用考慮 Start service time 並利用 14 式來決定出下一次的服務時間，但在 txop 分配方面還是利用 simple scheduler 的分配方式，所以新增及修改的程式內容如下：

```
int
MacHccaSchedMap_ref::createMap (TSPEC& tspec)
{
    txop_desc* p;
.....
    //修改 SETT-EDD 部分
    p = head;
    while ( p != 0 )
    {
        p->min_SI=p->mean_sdu_size*8/p->mean_rate;
```

```
            //將資料流的 sst(start service time)列入考慮
            p->service_time=p->sst;
            p->EDD_time=p->service_time+p->max_SI;
            p=p->next;
        }
.....
}


void
MacHccaSchedMap_ref::next_txop(void)
{
    //修改 SETT-EDD 部分**********
    txop_desc* p;
    txop_desc* EDD;
    //利用以下迴圈計算出最小大於當次服務時間的 k 值
    while((curr_desc->sst+k*curr_desc->min_SI) <= TXOP_start_time)
    {
        k++;
    }
    //將下次的服務時間 service_time 設定為 k 倍的 mSI，並且將下次的到期時間
    // EDD_time 設定為服務時間加上 max_SI
    curr_desc->service_time= k * curr_desc->min_SI+(curr_desc->sst+0.04);
    curr_desc->EDD_time=curr_desc->service_time+curr_desc->max_SI;
    //******************
    //將時間點移至 txop 結束的時間
    TXOP_start_time = TXOP_start_time + curr_desc->txop – phymib_.getSIFS();
    EDD = new txop_desc;
    //選出最早到期的 txop_desc
    .....
    //刪除下列兩行使得每次分配的 txop 皆為原 simple scheduler 設定值
    curr_desc->txop =curr_desc->txop_timer;
    curr_desc->txop_timer=0;
}
```

為了在編寫 otcl 時能夠使資料流亂數分佈，所以作了以下修改

```
proc create_connections {} {
    .....
    .....
    //設定 n 為 0 到 0.04 間的均勻分佈
    set n [$rng uniform 0 0.04]
    if { $opt(hcca) == 1 } {
        [$qsta($i) getMac 0] tclas $i 1
        [$qsta($i) getMac 0] tspec 1 [expr $i + 1] \
        //為了使otcl能傳回開始時間給Mac802_11e類別所以在下列程式後面多加一變
        //數以傳回開始時間
        uplink 0 0 0 0 0 0 0 0 .040 0 1 $n
        [$qap getMac 0] tspec 1 [expr $i + 1] \
```

```
            uplink 1279 6171 0 0 .040 0 256000 0 0 0 0 $n
        }
}
```

如此在 Mac802_11e 類別便可接收開始服務時間的資訊

```
int
Mac802_11e::command (int argc, const char*const* argv)
{
        .....
        }
        else if ( argc == 17 && strcmp(argv[1], "tspec") == 0    )
        {
                .....
                t.peak_data_rate                = atof(argv[12]);
                t.delay_bound                   = atof(argv[13]);
                t.minimum_PHY_rate              = atof(argv[14]);
                t.periodic                      = ( atoi(argv[15]) == 1 ) ? true : false;
                //資料流開始服務的時間
                t.start_service_time            = atof(argv[16]);
                .....
                return TCL_OK;
        }
}

txop_desc*
MacHccaSchedMap_ref::createTXOPDesc(TSPEC& tspec)
{
        .....
        .....
        newdesc->next                   = 0;
        newdesc->tid                    = tspec.tid;
        newdesc->qsta                   = tspec.qsta;
        newdesc->direction              = tspec.direction;
        newdesc->periodic               = tspec.periodic;
        newdesc->mean_rate              = tspec.mean_data_rate;
        .....
        //把原本從 otcl 取得的 sst 列入 txop_desc 的資訊
        newdesc->sst                    = tspec.start_service_time;
        return newdesc;
}
```

第三章 論文作法介紹

3.1 ARROW 與適應性排程比較

如前章節所述，ARROW 利用了 Queue Size 的資訊來達到準確的 TD 配置，並且提高通道的使用量。在流量控管方面，ARROW 使用平均值所計算的 timer，並且由 timer 傳輸條件來決定輪詢的時間。當服務的 VBR 類型為固定週期的 VBR 時(如 H.261 與 MPEG4)，將如圖 3-1 所示：

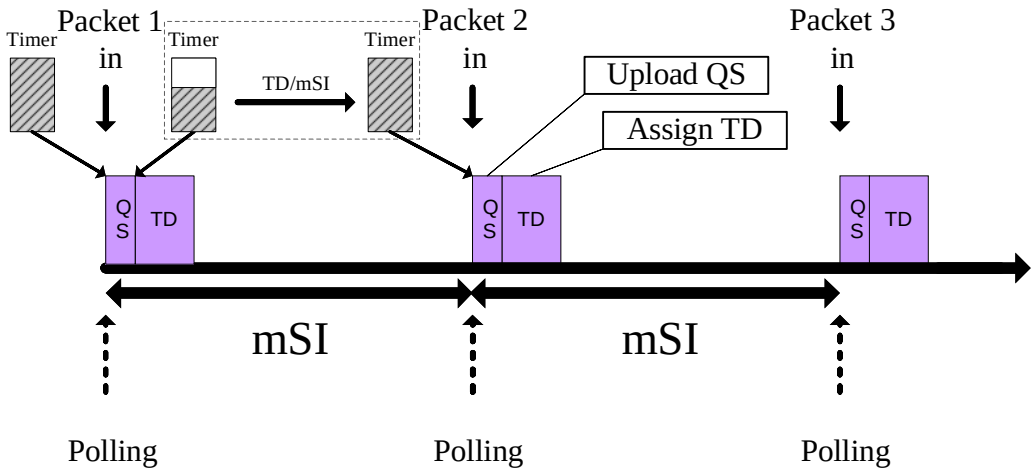


圖 3-1：ARROW 服務固定週期 VBR(如 H.261 與 MPEG4)的情況

在圖中我們可發現封包的傳送必須經過上傳 QS 及分配 TD 兩部分，並且封包的產生週期是固定的。以圖中 Packet 1 為例，當 Packet 1 產生時會先經由輪詢把封包的大小(也就是 QS)上傳至 QAP，接著 QAP 會分配經由 QS 所計算的 TD 給 QSTA 傳送 Packet 1，並且在分配 TD 的同時會將 QSTA 的 timer 減去 TD 量(與 SETT-EDD 將 timer 清空不同)，再以 TD/mSI 的增長率增長至 timer 的最大值進行下一次的輪詢。由於平均封包除以平均資料率而得的 mSI 會相近於封包產生的間隔，所以 timer 在每個封包產生的時間皆會增長一傳送平均封包的 TD (TD/mSI)。倘若 timer 的傳輸條件設定為 mTD，且每個封包在傳送時所需要的

TD 皆相近於平均封包所需傳送的時間，則 timer 在每個封包產生點都能滿足傳送條件而接受輪詢。因此，在服務固定週期的 VBR 會有不錯的成效。

而當服務的 VBR 類型為非固定週期的 VBR 時(如 H.263)，將如圖 3-2 所示。

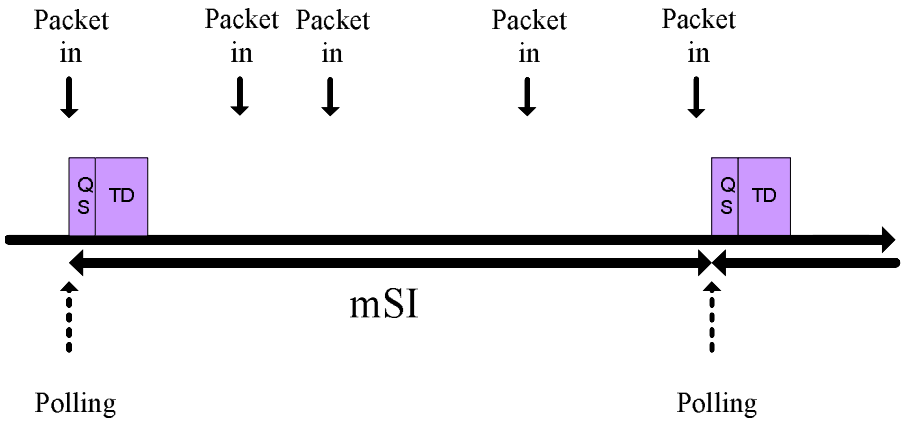


圖 3-2：ARROW 服務非固定週期 VBR(如 H.263)的情況

在圖 3-2 中我們可發現封包的產生週期相對於圖 3-1 是不固定的。並且經由平均封包除以平均資料率而得的 mSI 會與封包產生間隔有著極大的差異，這使得輪詢的週期將大於若干個封包產生週期，造成封包不能在延遲範圍接受輪詢而遺失。在另一方面，由於非固定週期 VBR 的封包變異量極大於固定產生週期的 VBR，這使得 QSTA 在經過一次的輪詢後，timer 需要比 mSI 更多的時間增長至 mTD，原本遺失量已經夠大，又因為此情況使得成效更加不彰。如此，我們可發現 ARROW 的作法將非常不適用於服務非固定週期的 VBR。

在適應性排程方面，同前章節所述，儘管能適應各種不同 VBR 類型的資料流[5]，如圖 3-3 所示。

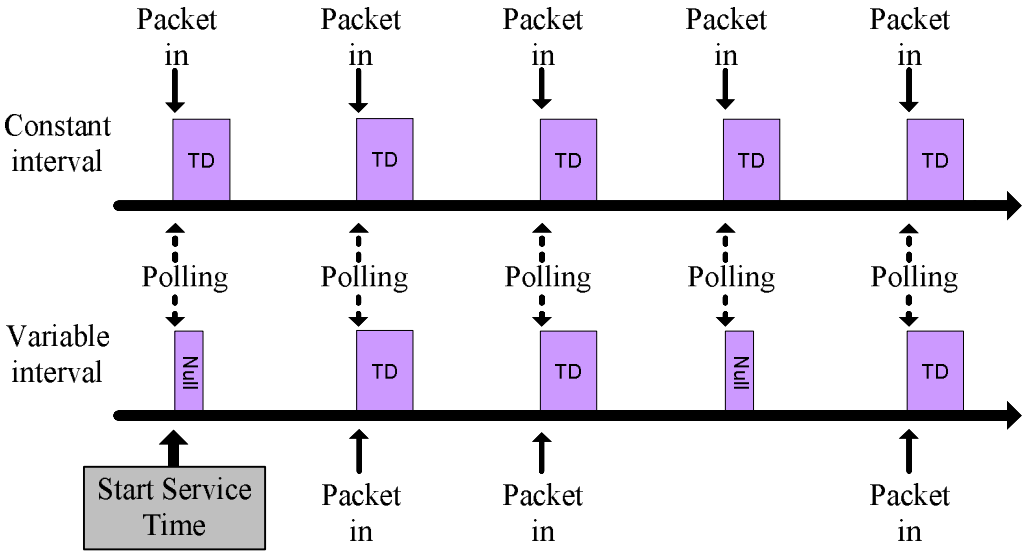


圖 3-3：適應性排程服務固定產生週期和非固定產生週期的情況

在圖 3-3 中，在服務固定及非固定產生週期的資料流時，會以最小資料的產生區間的倍數作為輪詢的週期，所以可以準確的取得資料產生時間並且傳送。但在其他方面，如圖 3-4 所示，由於適應性排程並沒有 QS 的觀念，所以 QAP 在分配 TD 是採用估測的方式計算，因此使得通道使用者僅維持少量數目，不免造成通道的浪費。

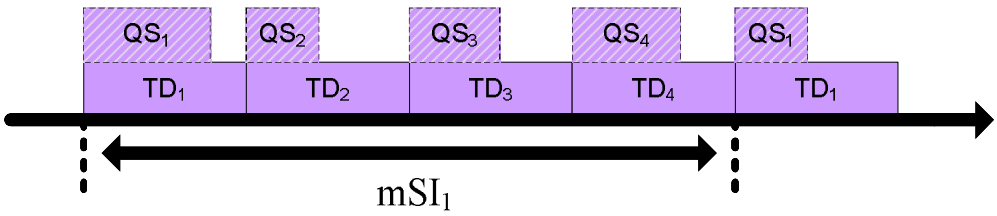


圖 3-4：適應性排成滿載之情況

因此針對 ARROW 及適應性排程可以統整出下列表格：

表 3-1：ARROW 及適應性排程之特性統整表

	輪詢週期	TD 分配
ARROW	由 Timer 決定	準確
適應性排程	最短封包產生週期	估測

在本論文作法裡，將結合 ARROW 及適應性排程的優點(表格紅色部分)，設計出一個最高成效的通道排程器。

3.2 論文作法

經由前一章節我們可發現，使用 TXOP timer 的技術會造成封包遺失的原因有輪詢及 timer 傳輸條件兩方面，因此本論文作法將著重於改善這兩方面的成效。在輪詢部分為了準確獲得封包產生時間，所以使用最短的封包產生區間作為輪詢的週期，取代原本由 TXOP timer 所決定的輪詢週期；而在 timer 傳輸條件部分，本論文使用以上傳的 QS 所需要之傳送時間來動態調整 TXOP timer 的傳輸條件，取代原本固定的 mTD(最大封包傳送時間)傳送條件，如圖 3-5 所示。

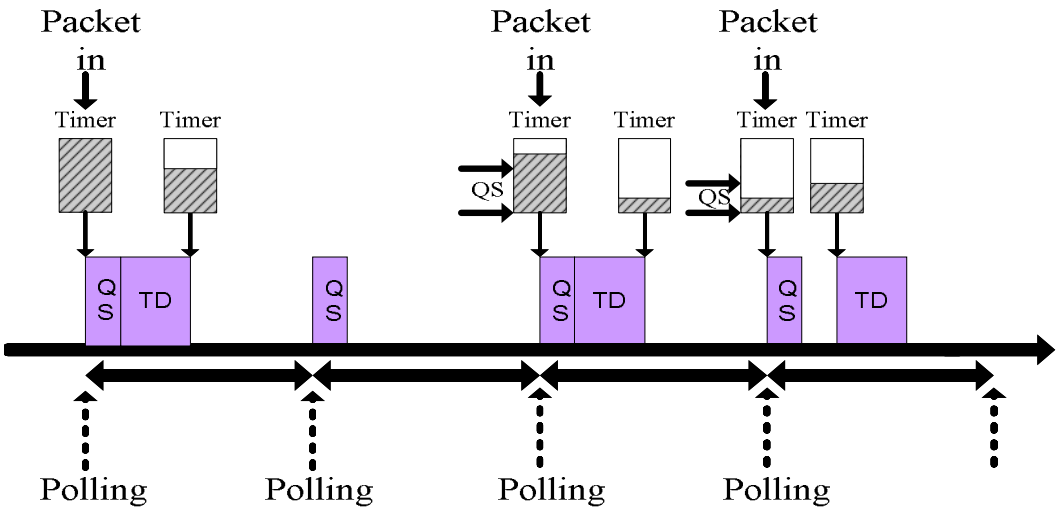


圖 3-5：論文作法服務非固定週期 VBR 之情況

在圖 3-5 中可發現，本論文使用最小封包產生區間作為輪詢週期，可有效的掌握封包產生時間；而在資料流接受輪詢時，由於本論文使用可動態調整的 timer 傳輸條件，所以如果 QAP 在接收輪詢的 QSTA 所上傳之 QS 後，如果發現 QSTA 的 timer 大於 QS 所需傳送的時間，QAP 便可馬上分配 TD 讓 QSTA 傳送資料，而不用等待 timer 增長至 mTD 的時間。

雖然已經改善了輪詢以及 timer 傳輸條件，但在 timer 的增長率方面，由於本論文仍使用 TD/mSI 的平均速率，如果 QAP 在接收輪詢的 QSTA 所上傳之 QS 後，發現 QSTA 的 timer 小於 QS 所需傳送的時間，必須等待 timer 增長至 QS 的傳送時間才能傳送資料，因此還是會有封包超過延遲時間而導致遺失的可能。而此情況的封包遺失與否以圖 3-6 及 13 式作說明：

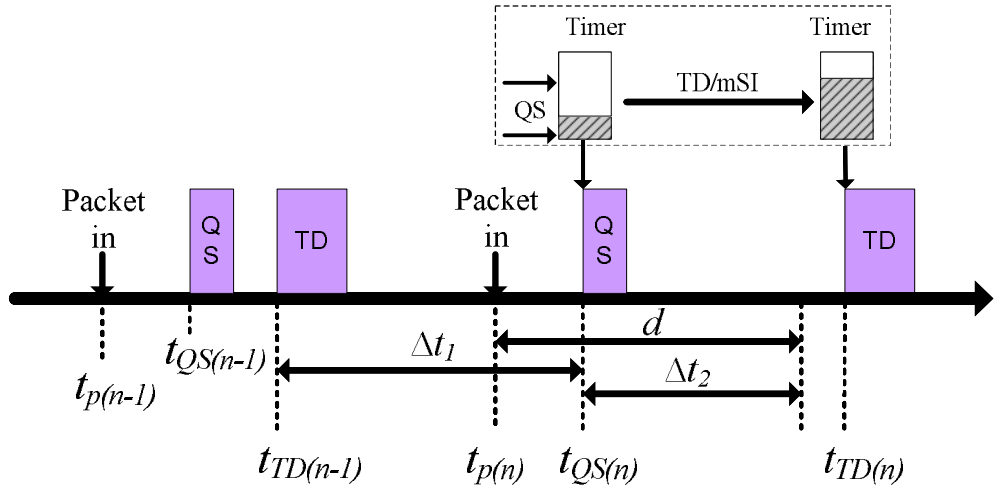


圖 3-6：論文作法封包遺失之情況

$$Loss = \begin{cases} 0 & , T_{t_{QS(n)}} + r \cdot \Delta t_2 \geq \dfrac{QS_{(n)}}{R} \\ 1 & , others \end{cases}$$

$$T_{t_{QS(n)}} = T_{t_{QS(n-1)}} + \Delta t_1 \cdot r \qquad (1)$$
$$\Delta t_1 = t_{QS(n)} - t_{TD(n-1)}$$
$$\Delta t_2 = t_{p(n)} + d - t_{QS(n)}$$

圖中 $t_{p(n)}$、$t_{QS(n)}$、$t_{TD(n)}$分別為第 n 次的封包產生、QS 上傳、TD 分配時間，d 為允許的延遲時間。一開始在 $t_{QS(n)}$的時間點上傳 QS 時，timer 會根據此刻時間與前一次分配 TD 的時間差(Δt_1)以及增長率 r 來增長 timer 量。接著將增長後的 timer 再加上 $t_{QS(n)}$到封包延遲範圍之間(Δt_2)能增長的 timer 量，若大於 QS 所需的 TD 量($\frac{QS_{(n)}}{R}$)，則代表封包能夠被傳送；反之則遺失。一言以蔽之，也就是第 n 次上傳 QS 時的 timer 量加上 timer 在 QS 延遲範圍內的增長量，必須大於 QS 的傳輸時間，才可能在 QS 的延遲範圍內傳送資料。

接著，為了求得系統的最大服務個數，我們考慮了一個無限大的 TXOP timer，使得 timer 隨時能滿足傳輸條件。換句話說，就是不設定任何的 timer 傳輸條件，只要 QAP 在接收上傳的 QS 之後，發現 QSTA 有資料存放於佇列中，就立刻讓 QSTA 傳送佇列的資料。如此，再搭配最短輪詢週期，則通道的使用與否，將取決於 QSTA 在輪詢週期是否有封包產生。如果服務的資料流類型為固定封包產生週期，則系統所能服務的最大 QSTA 個數便是在輪詢區間裡能服務的最大 QSTA 數目。舉例說明，假設輪詢的週期為 40ms，且每個 QSTA 傳送封包的時間為 10ms，則服務固定封包產生週期的 VBR 最多只能容納 4 個 QSTA。而如果服務的資料流類型為非固定封包產生週期時，由於資料流在每一段輪詢區間並不一定會產生封包。如圖 3-7 所示：

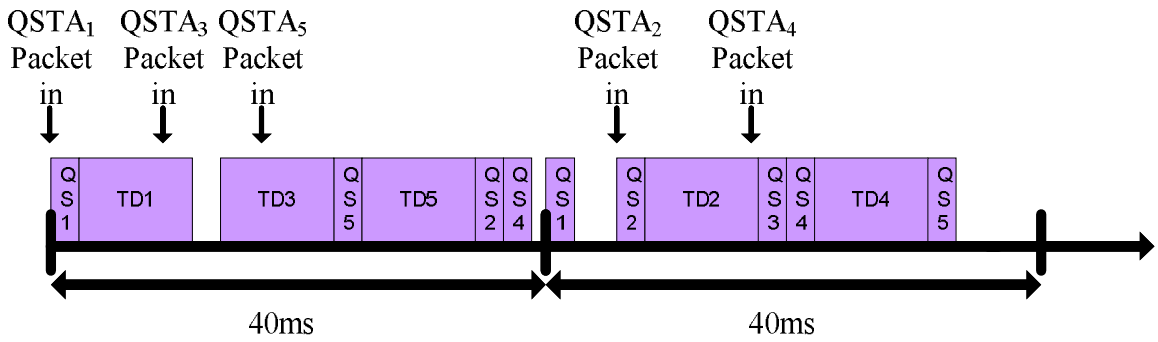


圖 3-7：無傳輸限制在服務非固定封包產生週期之情況

在圖中的第一段區間只有 QSTA1、3、5 有產生資料，第二段區間只有 QSTA2、4 有資料產生。如此，使用前面敘述的假設，則系統將可達到比服務固定產生週期的 VBR 更多的數目。

3.3 程式實作

在本論文作法的程式實作方面，由於本論文使用[9]所提出的模擬事件程序，如圖 3-8 所示：

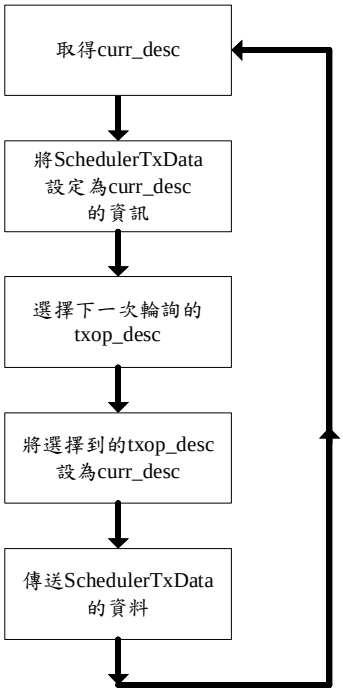


圖 3-8：[9]所提出之事件程序

由於排程器在取得當次輪詢的 QSTA 並且尚未傳送資料時，就已經選擇了下次要輪詢的 QSTA。而本論文作法需要利用當次被輪詢所上傳的 QS 資訊來決定下次被服務的時間，如果上傳的 QS 小於當次 QSTA 的 timer，則當次輪詢完的 QSTA 需要在下一次選擇 QSTA 時被選擇到，但以原本的事件程序作法並無法知道當次 QS 的資訊。因此本論文使用了一個小技巧，如圖 3-9 所示：

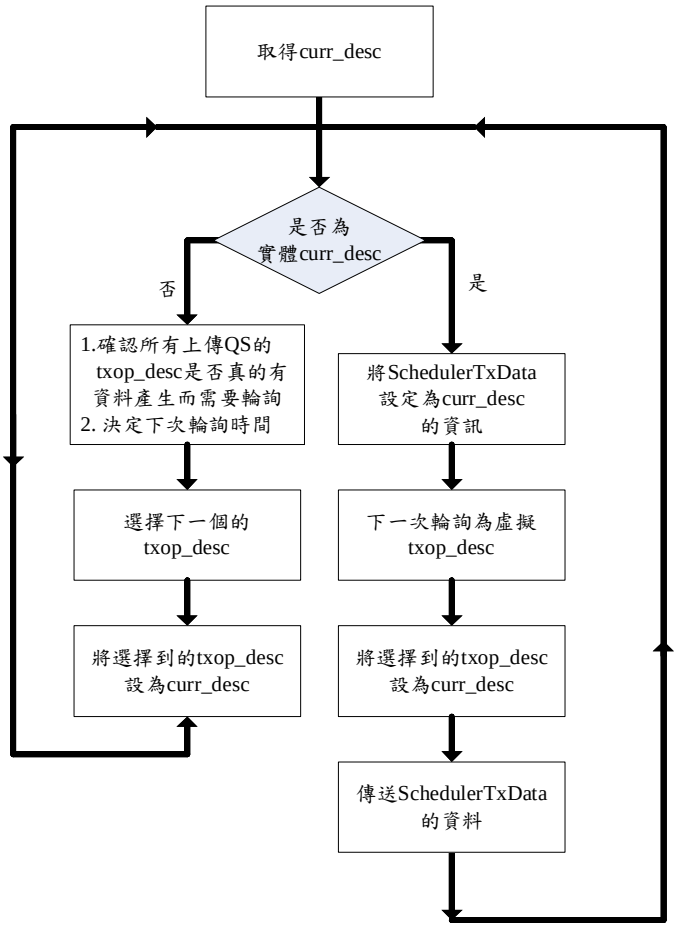


圖 3-9：論文所提出之事件流程

本論文加入了一個虛擬的 txop_desc，並且在取得當次輪詢的 QSTA 之後，如果當次輪詢的 QSTA 是實體的 txop_desc，則在選擇下次輪詢的 QSTA 時，便會選擇到虛擬的 txop_desc。而當傳送完當次 QSTA 的資料後，在取得新的輪詢 QSTA 時會判別是否為虛擬 txop_desc，如果為是，則 QAP 會確認所有已經上傳 QS 的 QSTA 是否真的有資料產生而需要輪詢，如果有資料產生則會計算出 QSTA 下一次需要被輪詢的時間，接著再選擇下一個實體 txop_desc，如此便能將原事件程序修改成本論文的事件程序。

所以程式碼的修改部分及增加的部分如下：

```
void
MacHccaSchedQAP_per::deque(SchedulerTxData& schedTx_)
{
    while(1)
    {
```

```
            desc            = map_builder_->getTXOPDesc();
            schedTx_.tid_ = desc->tid;
            schedTx_.dst_ = desc->qsta;
            //修改成如果選擇到虛擬 txop_desc(desc->qsta==0)時則執行下列函式並選擇下
            //一個 txop_desc
            if(desc->qsta==0)
            {
                //更新 QS 的資訊
                update_qs();
                //執行 check_QS()，如果上一次輪詢的 QSTA 真的有資料產生則將
                //qs_upload 設定為 true
                map_builder_->check_QS();
                //選擇下一個 desc_txop
                map_builder_->next_txop();
                //重新執行取得 desc_txop
                continue;
            }
            if(desc->direction == TSPEC::DOWNLINK)
            {
                ….
            }
            //如果選擇到的 txop_desc 為實體 desc 則執行下列函式
            else
            {
                ….
                //執行 update_txop()來取得 TD 並且由 timer 扣除
                map_builder_->update_txop(desc->qsta);
                //將 QAP 端記錄的 QS 清為零
                queue_packet_size[desc->qsta]=0;
                //選擇下一個 desc_txop，此時會選擇到虛擬 txop_desc
                map_builder_->next_txop();
                return;
            }
        }
}

void
MacHccaSchedMap_ref::next_txop(void)
{
        txop_desc* p;
        txop_desc* EDD;

        //如果當次選擇到的 txop_desc 為虛擬的 txop_desc 則，針對有上傳 QS 的 txop_desc
        //計算出下次的輪詢時間
        if(curr_desc->qsta==0)
        {
            p = head;
            while (p != 0)
            {
                //如果 QSTA 真的有資料產生，便計算出下次輪詢時間
                if(p->qs_upload==true)
                {
                    //如果已經計算出下次輪詢時間 txop_desc 則不再計算，TD_set 為新
```

```
                    //增的旗標變數，僅為計算判別用
                    if(p->TD_set==false)
                    {
                        //將 timer 累增
                        p->txop_timer+=
                        (Scheduler::instance().clock()-p->service_time)*p->timer_rate;
                        if(p->txop_timer>p->timer_max)
                        {
                            p->txop_timer=p->timer_max;
                        }
                        //計算出 QS 所需要的傳送時間
                        double demand_txop =
                        phymib_.getSIFS() + mac_->txtime( qsta_qs[p->qsta] +
                        phymib_.getHdrLen11e(), mac_->getDataRate())+
                        phymib_.getSIFS() + mac_->txtime(phymib_.getHdrLen11e(),
                        mac_->getBasicRate()) + p->pi;
                        //如果 timer 小於 QS 所需要的傳送時間則計算出 timer 增長到//
                        能傳送 QS 的時間
                        if(demand_txop>p->txop_timer)
                        {
                            Double delta_time=
                            (demand_txop-p->txop_timer)/p->timer_rate;
                            p->service_time=Scheduler::instance().clock()+delta_time;
                            p->EDD_time=p->service_time+p->max_SI/2;
                            p->txop_timer=demand_txop;
                        }
                        //如果 timer 大於 QS 所需要的傳送時間則設定為馬上可以輪詢
                        else
                        {
                            p->service_time=Scheduler::instance().clock();
                            p->EDD_time=p->service_time+p->max_SI/2;
                        }
                        //將 TD_set 設定為 true 代表已經計算
                        p->TD_set=true;
                    }
                }
                p=p->next;
            }
        }
        //如果為實體 txop_desc 則下次輪詢時間設為下列
        else
        {
            curr_desc->service_time=curr_desc->service_time+0.04;
            curr_desc->EDD_time=curr_desc->service_time+curr_desc->max_SI/2;
        }

        TXOP_start_time = TXOP_start_time + curr_desc->txop - phymib_.getSIFS();

        //找出下一個最早到期的 qsta**********
        p = head;
        EDD = new txop_desc;
        EDD->EDD_time=-1.0;
        EDD->service_time=0;
```

```
        EDD->qsta=0;
        EDD->txop=0;

        //如果當次選到的 txop_desc 為虛擬 txop_desc 則選擇出下一個最早到期 txop_desc
        if(curr_desc->qsta==0)
        {
            while (p != 0)
            {
                if ( EDD->EDD_time < 0 || p->EDD_time < EDD->EDD_time)
                {
                    EDD=p;
                }
                p=p->next;
            }
        }
        curr_desc=EDD;
        //*************************************

        if(TXOP_start_time <= curr_desc->service_time)
        {
            TXOP_start_time=curr_desc->service_time;
        }
}


void
MacHccaSchedMap_ref::check_QS(void)
{
        txop_desc* p = head;
        //針對所有 QSTA，如果真的有上傳資料則設定 qs_upload=true
        while ( p != 0 )
        {
            if(qsta_qs[p->qsta]!=0)
            {
                p->qs_upload=true;
            }
            else
            {
                p->qs_upload=false;
            }
            p=p->next;
        }
}

void
MacHccaSchedMap_ref::update_txop(int qsta)
{
        txop_desc* p = head;
        double new_txop = 0;
        int k=0;
        while ( p != 0 )
        {
            //針對當次選擇到的 txop_desc 計算出分配的 TD
```

```
            if(p->qsta==qsta)
            {
                new_txop =
                phymib_.getSIFS() + mac_->txtime( qsta_qs[p->qsta] +
                phymib_.getHdrLen11e(), mac_->getDataRate())+
                phymib_.getSIFS() +
                mac_->txtime(phymib_.getHdrLen11e(), mac_->getBasicRate());
                //如果 QAP 端記錄的 QS 不為零則設定為 new_txop + p->pi，並從 timer
                //扣除
                if(qsta_qs[p->qsta]!=0)
                {
                    p->txop=new_txop + p->pi;
                    if(p->txop_timer>=p->txop)
                    {
                        p->txop_timer-=p->txop;
                    }
                    else
                    {
                        p->txop=p->txop_timer;
                        p->txop_timer=0;
                    }
                    //利用適應性排程的方式選擇出下一次輪詢的時間
                    while(((p->sst+0.04)+k*0.04) <= TXOP_start_time)
                    {
                        k++;
                    }
                    p->service_time= (k-1) * 0.04+p->sst;
                    //將 QAP 端記錄的 QS 清為零
                    qsta_qs[p->qsta]=0;
                    //將 TD_set 設為 false 代表需重新計算
                    p->TD_set=false;
                }
                else
                {
                    p->txop=p->pi;
                }
            }
            p=p->next;
        }
    }
```

第四章 模擬結果

4.1 模擬情境與參數設定

在模擬方面，本論文考慮一個 QSTA 累增的情境，並且針對 ARROW、適應性排程以及本論文的作法(QS 傳輸條件、無傳輸條件)分別在 H.261、H.263、MPEG4 的成效比較。使用的系統參數如表 4-1 所示：

表 4-1：系統參數

TSPEC Parameter	H.261 ***Starship Troopers***	H.263 ***Jurassic Park I***	MPEG4 ***Jurassic Park I***
Mean data rate(Kbps)	256	256	268
Delay bound (ms)	40	40	40
Nominal MSDU size(bytes)	1280	4534	1340
Maximum MSDU size(bytes)	4927	11817	8511
Maximum burst size(bytes)	4927	11817	8511
Peak data rate(Kbps)	986	1418	1702
Minimum service interval (ms)	40	40	40
Maximum service interval (ms)	40	40	40

4.2 模擬相關程式

4.2.1 Otcl 程式

```
;#讀取標頭檔
source our_header.tcl

# default argument options
set opt(start)          0.0                     ;# traffic start time
set opt(n)              1                       ;# number of real-time stations
set opt(hcca)           1                       ;# HCCA is on
;#由鍵盤讀取輸入資訊
getopt $argc $argv
;#將val(n)設為鍵盤輸入的QSTA個數
set val(n)              $opt(n)

#
# create connections
#

proc create_connections {} {
    global ns opt val qsta qap defaultRNG       ;# input

      ;#設定VBRtrace檔**********
      set original_file_name Verbose_Troopers_256.dat
      set trace_file_name video.dat
      set original_file_id [open $original_file_name r]
      set trace_file_id [open $trace_file_name w]
      set last_time 0

      while {[eof $original_file_id] == 0} {
            gets $original_file_id current_line
            if {[string length $current_line] == 0 ||
            [string compare [string index $current_line 0] "#"] == 0} {
                  continue
            }
            scan $current_line "%d%s%d" next_time type length
            set time [expr 1000*($next_time-$last_time)]
            set last_time $next_time
            puts -nonewline $trace_file_id [binary format "II" $time $length]
      }
      close $original_file_id
      close $trace_file_id
      # set the simulation end time:
      set end_sim_time [expr 1.0*$last_time/1000+0.001]
      # read the video trace file:
      set trace_file [new Tracefile]
      $trace_file filename $trace_file_name
      ;#**************************
      ;#設定rng為亂數產生物件並指定亂數種子為鍵盤輸入值$opt(our_seed)
      set rng [new RNG]
      $rng seed $opt(our_seed)
      ;#設定網路拓樸
      for { set i 0 } { $i < $opt(n) } { incr i } {
            set agt_dst($i)         [new Agent/Null]
            set agt_src($i)         [new Agent/UDP]
            $agt_src($i) set packetSize_ 5000
```

```
            $agt_src($i) set fid_               $i
            set app($i)                 [new Application/Traffic/Trace]
            $app($i)    attach-tracefile $trace_file
            $ns attach-agent $qsta($i) $agt_src($i)
            $ns attach-agent $qap    $agt_dst($i)
            if { $i == 0} {
                  if { $opt(hcca) == 1 } {
                        [$qsta($i) getMac 0] tclas $i 1
                        [$qsta($i) getMac 0] tspec 1 [expr $i + 1] \
                        uplink 0 0 0 0 0 0 0 0 .040 0 1 0
                        ;#設定tspec
                        [$qap getMac 0] tspec 1 [expr $i + 1] \
                        uplink 1280 4927 0 0 .040 0 256000 0 0 0 0 0
                  }
                  $ns connect $agt_src($i) $agt_dst($i)
                  $app($i) attach-agent $agt_src($i)
                  $ns at [expr $opt(start)+ 0 ] "$app($i) start"
            } else {
                  ;#設定n為均勻分佈的變數值
                  set n [$rng uniform 0 0.04]
                  if { $opt(hcca) == 1 } {
                        [$qsta($i) getMac 0] tclas $i 1
                        [$qsta($i) getMac 0] tspec 1 [expr $i + 1] \
                        uplink 0 0 0 0 0 0 0 0 .040 0 1 $n
                        ;#設定tspec
                        [$qap getMac 0] tspec 1 [expr $i + 1] \
                        uplink 1280 4927 0 0 .040 0 256000 0 0 0 0 $n
                  }

                  $ns connect $agt_src($i) $agt_dst($i)
                  $app($i) attach-agent $agt_src($i)
                  $ns at [expr $opt(start)+$n] "$app($i) start"
            }
      }
}
;#設定初始參數
init
if { $opt(hcca) } {
      $ns at 0.0 status
      ;#設定hcca開始時間
      $ns at $opt(start) "hcca-start"
}
;#開始執行ns
$ns run
```

4.2.2 awk 程式

```
BEGIN
{
      #設定使用變數
      highest_packet_id = 0;
      highest_time = 0;
```

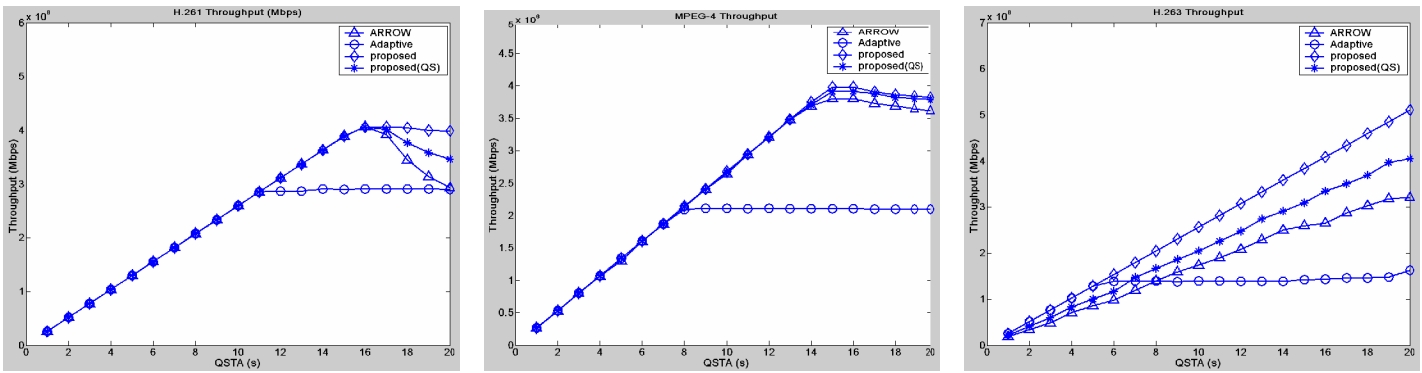
```
        packet_num = 0;
        packet_total_delay = 0;
        mean_delay = 0;
        total_packet_size = 0;
        throughput = 0;
        loss_num = 0
        loss_rate = 0;
}
{
        #將 trace 檔裡面的資料放入下列變數
        event = $1;
        time = $2;
        packet_id = $6;
        packet_class = $7;
        packet_size = $8;
        #取得最高的封包編號。
        if ( packet_id > highest_packet_id )
        {
                highest_packet_id = packet_id;
        }
        #取得最高的時間。
        if ( time > highest_time )
        {
                highest_time = time;
        }
        #如果事件為 s 則紀錄封包的產生時間。
        if ( event == "s" )
        {
                start_time[packet_id] = time;
        }
        #如果事件為 r 則記錄封包的接收時間，並且累加總封包量。
        if ( event == "r" )
        {
                end_time[packet_id] = time;
                total_packet_size+=packet_size;
        }
        #如果事件為 d 則將接收時間設為-1 以表示封包遺失。
        if ( event == "d" )
        {
                end_time[packet_id] = -1;
        }
}
END
{
        for ( packet_id = 0; packet_id <= highest_packet_id; packet_id++ )
        {
                start        = start_time[packet_id];
                end          = end_time[packet_id];
                #將 end 減掉 start 則為封包的延遲時間
                packet_duration = end - start;

                #如果封包延遲時間大於零則將接收的封包個數及總封包延遲時間累加。
                if ( start < end )
```

```
            {
                packet_num++;
                packet_total_delay+=packet_duration;
            }
        }
        #將最高的封包編號減去總接收的封包個數則為遺失封包的個數。
        loss_num=highest_packet_id-packet_num;
        #將遺失個數除以最高封包編號則為遺失率。
        loss_rate=loss_num/highest_packet_id;
        #將總延遲時間除以總接收封包個數則為平均延遲時間。
        mean_delay=packet_total_delay/packet_num;
        #將總封包大小除以最高時間則為系統吞吐量。
        throughput=(total_packet_size*8)/highest_time;
}
```

4.3 模擬結果

4.3.1 系統吞吐量(throughput)分析

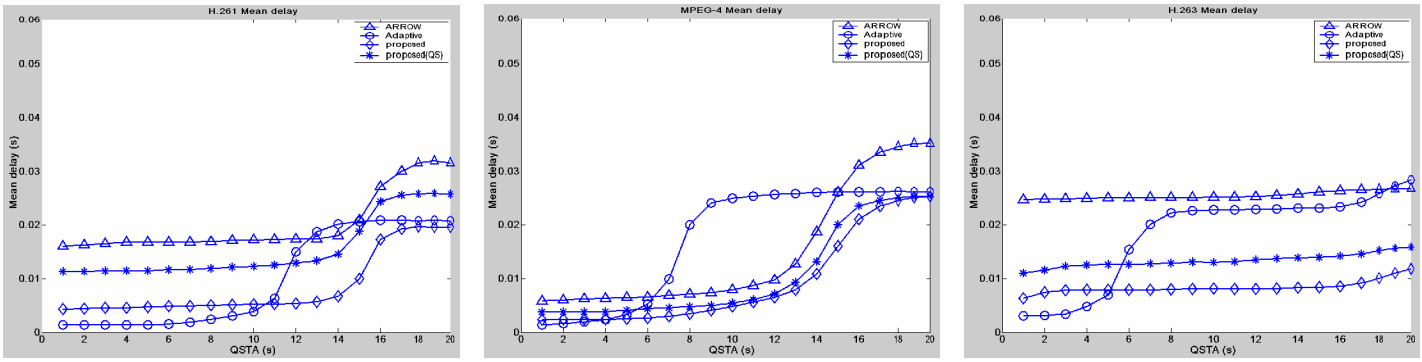


(a) (b) (c)

圖 4-1：系統吞吐量成效圖：(a) H.261；(b) MPEG4；(c) H.263。

在圖中可以發現適應性排程演算法的系統容量遠小於本論文所提出的兩種作法以及 ARROW。這是因為適應性排程演算法使用估測的方式來計算 TD。而本論文及 ARROW 皆是使用 QS 的資訊來分配最準確的 TD，因此在較多使用者的情況下依然能使得系統的吞吐量隨著使用者個數的上升而增加。在另一方面，本論文針對 ARROW 作了傳輸條件以及輪詢週期的修改，所以在 H.263 的吞吐量成效圖中可發現本論文的兩種作法均有較高的的封包增長率(資料量/QSTA)。

4.3.2 封包平均延遲(mean delay)分析

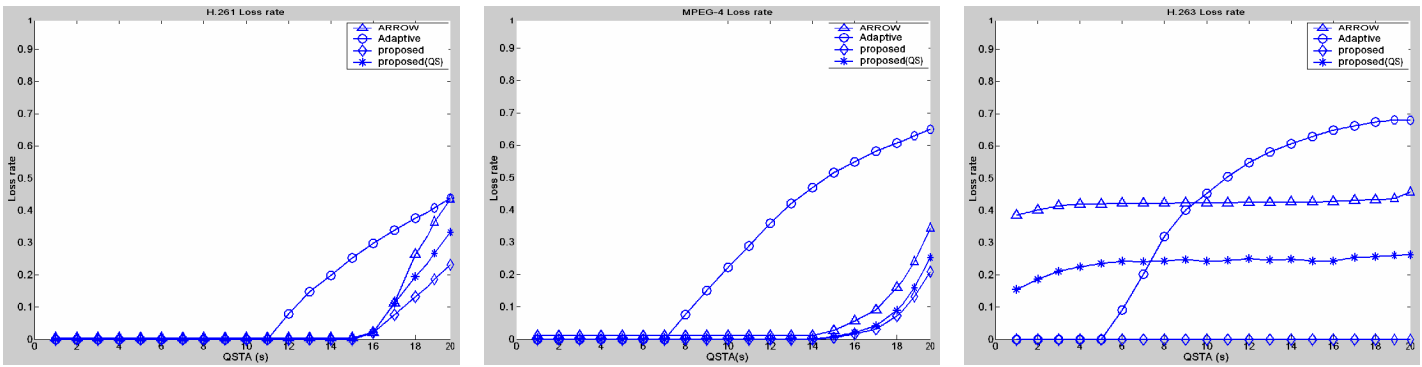


(a) (b) (c)

圖 4-2：平均封包延遲時間分佈圖：(a) H.261；(b) MPEG4；(c) H.263。

由圖中可發現，本論文兩種作法的平均封包延遲均小於 ARROW，其主要原因為本論文使用動態調整的傳輸條件,所以當 QSTA 有資料需要傳送時便只需等待 timer 量增長至傳送 QS 所需的 TD 量;相對於 ARROW 必須使得 timer 增長至最大封包傳送時間(mTD)。另一方面，ARROW 使用 timer 來決定輪詢週期，並不能準確掌握封包的產生週期。而本論文使用最小的封包產生週期作為輪詢週期，所以在封包產生時能讓封包在 timer 允許的範圍能立刻被輪詢。

4.3.3 封包遺失率(loss rate)分析



(a) (b) (c)

圖 4-3：平均封包遺失率分佈圖：(a) H.261；(b) MPEG4；(c) H.263。

在遺失率分佈圖中很明顯的可以看出適應性排程的遺失的封包數目非常的龐大，主要的問題還是在於使用估測值的 TD 而使得無法服務容量以上的 QSTA。在 H.263 封包遺失率方面，ARROW 造成封包遺失的主要原因為利用一平均值增長的 timer 來決定輪詢的時間，使得若干封包因為超過延遲範圍而遺失。而本論文所提出的動態調整傳輸條件演算法雖然仍是使用平均值增長的 timer，但由於本論文使用了最小封包產生區間的輪詢週期並搭配變動的傳輸條件，所以在輪詢時能準確的掌握封包的產生時間；在傳送時讓封包在 timer 允許的範圍內盡快被傳送，所以能有效的降低封包遺失率。

另一方面，雖然 H.261 及 MPEG4 的封包產生間隔均為 40ms，但 ARROW 在允許的系統容量內仍然會有封包的遺失，其主要問題在於平均封包與最大封包的差異度太大，舉例說明，當 QSTA 在傳送完最大封包之後，便必須等待 timer 增長至 mTD 才能進行下一次的輪詢。假設平均的 TD (Nominal TD，NTD)為 5s，TD/mSI 為 5s/40ms，而如果 mTD 為 NTD 的兩倍也就是 10s，則 timer 必須花兩倍 mSI 才能夠進行下一次的輪詢，因此便會超過封包的延遲範圍而造成遺失。而本論文所提出的變動傳輸條件演算法是利用上傳時的 QS 大小來決定下次輪詢(分配 TD)的時間，所以並不會有 ARROW 的缺點。

第五章 結論

如何在無線網路傳送 VBR 封包而能維持高傳輸效能，是現今無線網路研究的主要重點。本論文在此方面提出了在 802.11e HCCA TXOP 配置的基礎上，製作一個動態調整傳輸條件的排程演算法，將 QSTA 的服務分成輪詢及傳送兩部分。在輪詢部分我們以 VBR 最短的服務區間作為 QSTA 的輪詢週期，以準確的掌握變動區間 VBR 的封包產生時間；而在傳送部分我們主要針對 TXOP timer 的傳輸限制分別提出無 timer 傳輸限制條件，以及可根據 QS 傳送時間調整的變動 timer 傳輸條件來改善傳輸效能。

經由模擬發現本論文所提出的作法能有效的改善封包的延遲及遺失，並且能達到較高的吞吐量。而在未來本論文將延續此研究基礎，設計一個具公平性且能因應 VBR 類型而動態調整的 TXOP timer。

參 考 文 獻

[1] [Online] Video Traces for Network Performance Evaluation.

Available: http://trace.eas.asu.edu/

[2] IEEE Std. 802.11e, "Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications– Medium Access Control (MAC) Quality of Service Enhancements," Nov. 2005.

[3] A. Grilo, M. Macedo, and M. Nunes, "A Scheduling Algorithm for QoS Support in IEEE802.11e Networks," *IEEE Wirel. Commun. Mag.*, vol. 10, no. 3, June 2003, pp. 36–43.

[4] D. Skyrianoglou, N. Passas, and A. K. Salkintzis, "ARROW: An Efficient Traffic Scheduling Algorithm for IEEE 802.11e HCCA," *IEEE Trans. Wirel. Commun.*, vol. 5, no. 12, Dec. 2006, pp. 3558–3567.

[5] I. Inan, F. Keceli, and E. Ayanoglu, "An Adaptive Multimedia QoS Scheduler for 802.11e Wireless LANs", *IEEE ICC*, vol. 11, June 2006, pp. 5263–5270.

[6] G. Boggia, P. Camarda, L.A. Grieco, and S. Mascolo, "Feedback-Based Control for Providing Real-Time Services with the 802.11e MAC," *IEEE/ACM Trans. Net.*, vol. 15, no. 2, Apr. 2007, pp. 323–333.

[7] Chiapin Wang, Po-Chiang Lin, T. Lin," A Cross-Layer Adaptation Scheme for Improving IEEE 802.11e QoS by Learning", *IEEE Trans. Neural Networks*, Vol. 17, Issue 6, Nov. 2006, pp. 1661 – 1665.

[8] B. Makarevitch, "Delay reduction for 802.11e hybrid coordinator," *Elect. Lett.*, vol. 40, May 2004, pp. 708–709.

[9] [Online] IEEE 802.11e HCCA simulation using the Network Simulator 2

Available: http://info.iet.unipi.it/~cng/ns2hcca/