基於 DiffServ Edge Routers 之網路處理器分類處理效能之研究
Performance Evaluation on Classifiers in the Network Processor Based on DiffServ Edge Routers

二、研究計畫摘要—請對研究計畫之問題依下列各項述明：

(1) 研究動機與研究問題

近年來，由於網路發展之迅速，大量資料在散佈全球之網際網路中往來流通。這當中，擔任指揮封包交通腳色的重要功程－Router (路由器)便背負起集結各地封包並正確地將各封包傳送出去的重責大任。然而，封包之路由速度，取決於路由器之實作方式及相關演算法。也因此，路由之速度成爲網路大環境速度之主要瓶頸。

在這當中，有三種方法可用來實作路由之功能：1. 軟體實作, 2. 硬體實作, 3. 介於兩者之間的網路處理器實作。以下便一一概述三者的優缺點：

軟體實作方面，使用一般用途處理器，完全靠程式處理資料之路由這個方法使用一般大家都熟悉的軟硬體架構，容易且方便實作。然而就在同時，將導致許多簡單的功能需要經過複雜的運算或是大量的 clock cycles，大幅降低原本應有的效能。

此時，最佳化硬體實作 Application Specific Integrated Circuit – ASIC (特殊用途積體電路)便是不錯的解決之道。對於路由之功能做硬體最佳化，可以極短時間完成許多軟體實作需要耗費大量時間甚至是無法完成的工作。傳統之 ASIC 雖能大幅提升資料處理速度，然而因爲針對特殊用途所作之特殊通訊協定已最佳化燒死在晶片中，缺乏演算法和功能之改進提昇的能力，而導致升級十分困難。此外，ASIC 之開發過程也相當複雜，需要較長之開發週期，且更新或加強網路協定困難，所需付出的成本也相對較高。綜合以上觀點看來，ASIC 硬體實作雖有速度上的優勢，然而仍有其美中不足的地方。

也因此，而衍生出第三種折中的解決方案－Network Processor or Network Processing Unit – NPU (網路處理器)。此種處理器爲可程式化之半導體元件，有專門處理網路資料所設計的最佳化功能，包括用來支援高速的封包分類和修改封包的硬體及指令集，提供足夠的彈性以供功能之改進與提升，又不失其硬體最佳化而相對快速之優點。此外在

NPU 設計當中，通常也考慮到 I/O 頻寬的提升。基於以上理由，不難發現網路處理器架構的同時涵括了前兩者之優點，並將前兩者之缺點減至最低，後繼發展指日可待，也因此成爲此次專題計畫所要研究的對象。

於此領域相關之研究，有 IEEE Network Journal July/August 2003 之「 DiffServ Edge Routers over Network Processors: Implementation and Evaluation 」。此篇論文使用 Range Matching 演算法來實作 DiffServ (Differentiated Services)，並記錄相關資料封包傳送速度，加以分析比較。DiffServ 機制爲提供使用者接收不同服務等級之封包，根據某種方法將封包分類成多重場域(Multiple Fields - MFs)並標示上 per-hop behavior – PHB 用以定義此封包之特殊型爲以及該如何被處理。每一個經定義過的 PHB 都會對應到一個 DiffServ Code Point – DSCP 值用在分類排程上。想當然耳，具有較高等級之封包會比較低等級之封包還要早被排程，使得不同種類的封包依據其實際需求而分配到不同等級的網路資源，對於重要的封包能給予其較快的傳輸，而不需要太快傳輸之封包則給予其較慢之排程，各司其速。在這當中也不難發現，處理器除了運送封包之外的時間，幾乎都在執行分類(Classification)與排程(Scheduling)的工作，分類與排程也就理所當然對於整體路由速度有著關鍵的影響力，也因此成爲 DiffServ 實作的主要瓶頸。

目前已有許多演算法被提出來，試圖縮短分類排程的時間。然而實際上基於平台的限制，例如記憶體容量太小等，某些演算法並不完全行的通。此次專題計畫之研究題目定爲「基於 DiffServ Edge Routers 之網路處理器分類處理效能之研究」 (Performance Evaluation on Classifiers in the Network Processor Based on DiffServ Edge Routers)，便想利用不同的演算法來實作檢討與比較，試圖在有限的資源上得到更高的效能結果，進而改善路由之速度，而使之前提到之分類與排程這大瓶頸對於整體速度的影響能減到最小。

此外，也由於之前對於網路概念的生疏，想藉此機會多學習網路概念及架構，藉此提升自我實力。或許此次研究計畫僅涵蓋網路的某一層面，尚沒辦法通盤了解整個網路的運作方式，然而知識的累積，需要經過細心及耐心的薰陶，方能有所進展。希望經過此次研究計畫能使自己於知識方面更上一層樓，同時也提升自己的研究方面的經驗，方法，思考與態度，以利於未來從事於不管是研究上，或是就業職場上的相關工作。

(2) 研究方法及步驟

本研究計畫之首要工作便是閱讀 NPU 之相關軟硬體技術資料。熟悉硬體架構，學習軟體修改開發，才有辦法實作。另外也須了解 DiffServ 之相關技術以及相關演算法。關於此次研究計畫之設備及預計使用之路

由分類演算法，將在接下來的篇幅加以說明。

很幸運的，張老師的實驗室已有以 IXP1200 網路處理器爲基礎的路由器之研究與開發的初步成果，也因此能夠得到一些相關經驗。其內容大致上是將 L3Forward 中 microengines 執行的程式，由原來名爲 MicroCode 的組合語言，改用一般人熟悉的 C 語言實作出來。在這當中，首當其衝的就是必須經過重重的安裝設定及操作上的學習，嘗試及錯誤，十分辛苦。從師長那邊得知學長在這方面下了很多工夫，也花了不少時間，令小弟十分佩服。也由於有學長的開拓，所以節省了許多不必要的試誤，同時也留下了許多寶貴的經驗。

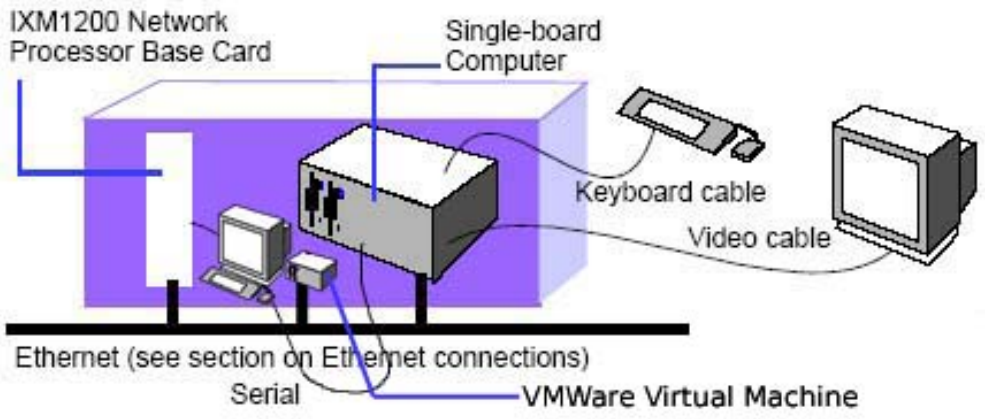
本次專題計畫所採用的 NPU 系統，是由 Radisys 所生產的 ENP-2505，以 IXP1200 搭配著四個 10/100 Mb ports 的開發板。其中，IXP1200 爲 Intel 公司所發展 Internet eXchange Architecture – IXA (網路交換架構)之 NPU 的其中一個型號。由 1 個高速的核心 StrongARM 處理器及 6 個可程式化之 microengines 組成，每個 microengine 支援 4 個硬體執行緒(thread)。

在這當中，路由的工作可分爲兩類：control plane 及 data plane。一般來說，StrongARM 主要負責的是 control plane，而 microengines 則是負責 data plane。此兩工作之差別在於，data plane 必須以網路速度(line speed)來處理封包，儘可能的接收封包及做相關分類排程工作，使較重要的封包不至於在不重要的封包流中延遲甚至遺失，並立即將封包送至正確目的地；至於 control plane 則是預期接收很少量的封包，唯有當路由器改變或建立新連結時才需要重新建立最短路徑的 routing table。此外，另一項重要的差別在於 data plane 對資料的處理應儘可能的少，如 IP header 的驗證，TTL 的減少，重新計算 checksum，從 routing table 中尋找適當的輸出 port 等，都是需要極少運算的簡單工作；相反地，control plane 則需要執行高運算量的工作，如爲建立 routing table 而執行最短路徑演算法等。也因此兩者之間的工作分配將使得運算資源有效地被運用。

至於資料的進出流向，大致上是由 MAC 層接收進入 RFIFO (Receive FIFO)，接收之 Microengine 從封包的目的 IP 及 SRAM 中的 Routing table 查得要送出之下一個路由 IP 位址，同時將封包送入 SDRAM 暫存。接下來交由傳送之 Microengine 把封包從 SDRAM 中讀出放入 TFIFO (Transmit FIFO)後再由 MAC 層送出。而我們將會在接收到封包後對封包做處理，並依優先等級來做分類，以達成增加效能之目標。

詳細的 IXP1200 架構簡述如下：

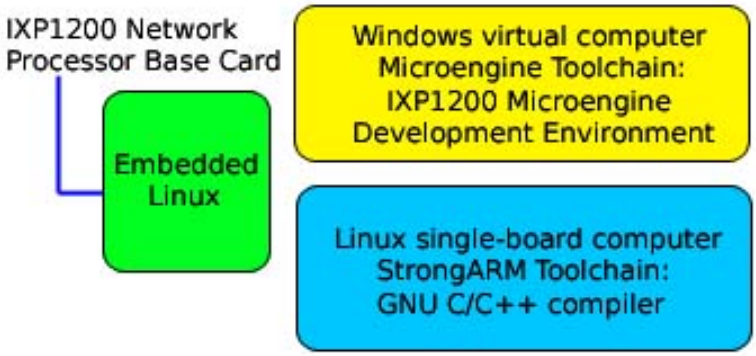
1) 環境介紹：



如上圖所示，IXP1200 安插於一台 Linux 的 Single-board Computer 之 PCI 插槽上，其中一條 RS232 Serial port 與 Linux 主機相連。而位於主幾中間那台小小的迷你電腦則是使用 Microsoft Windows 爲作業平台，以網路與 Linux 主機及 IXP1200 相連。上面裝有 Intel 發展的 Microengine compiler – Workbench，方便模擬測試及除錯之用。

爲了節省硬體設備資源，我們使用 VMWare 軟體於 Linux 主機中虛擬出一台 Windows 的虛擬主機。也就是說圖中位於主機裡的那台迷你小電腦，使用虛擬網路以虛擬主機的形式位於 Linux 主機中。

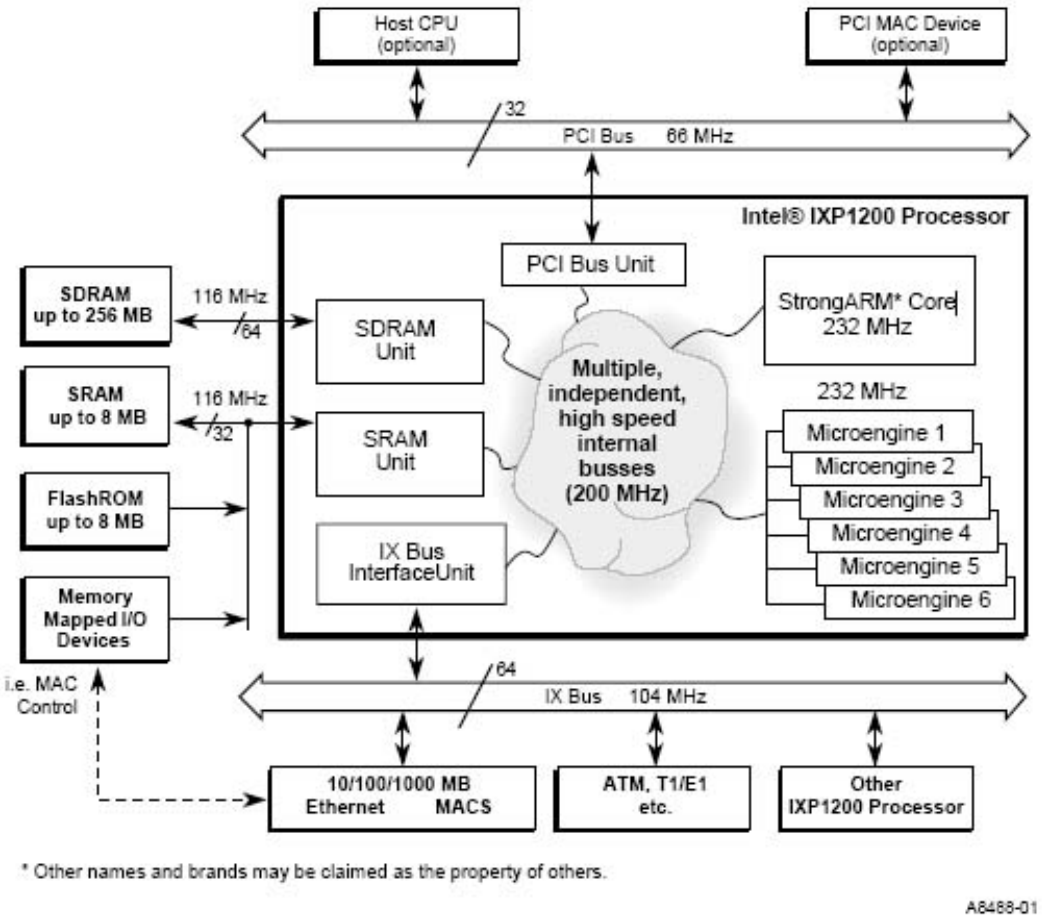
總而言之，整個架構大致上可用下圖的三大平台來表示：



虛擬 Windows 平台和 Linux 主機上的 IXP1200 透過虛擬網路互相連結溝通。這樣一來，便可在 Windows 平台上使用方便的開發軟體 Workbench 直接將 microcodes 實際套用在 IXP1200 上，作相關除錯或測試結果之用。

2) 硬體介紹：

IXP1200 大致上可分爲六大單位(Unit)，如下圖所示：



1 232MHz StrongARM,
6 232MHz Microengines,
256MB 116MHz SDRAM,
8MB 116MHz SRAM,
104MHz IX Bus Interface (包含 RFIFO, TFIFO 等),
及 66 Hz PCI Bus 等。
此六單位由 200MHz 之 Internal Buses 擔任資料互流的工作。

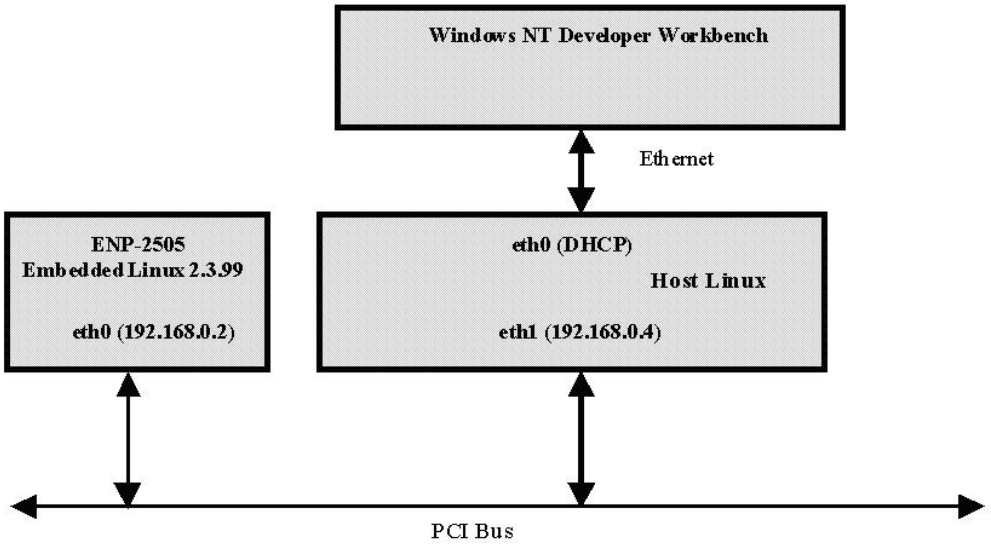
其中 SRAM 主要用來儲存 routing table，以方便 Microengines 來查詢，而 SDRAM 則主要用來儲存整個封包。正如之前所述，封包經由 MAC 到 RFIFO，由 SRAM 查詢 routing table 後暫存 SDRAM，最後經 TFIFO 再由 MAC 層送出。而我們所需要做的工作便是在接收封包的同時將他們快速的分類，以供後續之使用。

3) 安裝設定介紹：
大致上都是參考其文件 SDK_Install.pdf，於 Linux 主機上裝好 Linux 後一步步照著指示，更新 kernel，安裝套件，做相關設定。都完成之後可以/opt/ixasdk/enp-2505/bootixp/bootixp 來開啓 ENP-2505。而 ENP-2505 與 Linux 主機之間則由 NAT 進行交流，ENP-2505 的 NFS 檔案系統會被 mount 在/nfs 下。

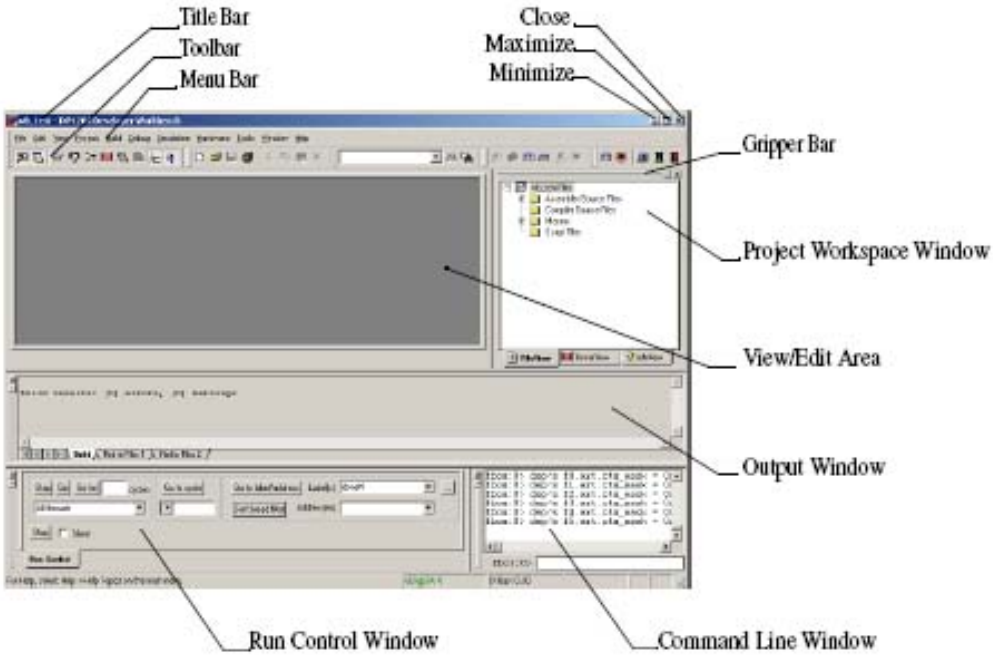
至於上述的虛擬主機則是裝好 VMWare 套件後，以其安裝一台虛擬的 Windows 主機，設定好區網後再裝上開發工具 Workbench 及模擬 Linux 環境的 Cygwin。

其中比較需要注意的是，由於我們使用 VMWare 來虛擬 Windows 主機，因此必須設定 iptables 使虛擬主機與 IXP1200 板子互相資料傳送沒有問題。若沒設定 iptables，則使用 Workbench 來實際執行 microcode 將無法接收到 IXP1200 之中斷回應。設定方式爲在 Linux 主機上執行 iptables -t nat -A POSTROUTING -o vmnet8 -s 192.168.0.2 -j MASQUERADE，其中 vmnet8 表示虛擬主機之虛擬網卡，192.168.0.2 爲 ENP-2505 之區網 ip。

整體網路架構將如下圖所示：



4) 發展工具介紹：
此次專題計畫預計使用之開發工具爲 Intel 公司發展的 IXP1200 Developer Workbench，使用者畫面如下：

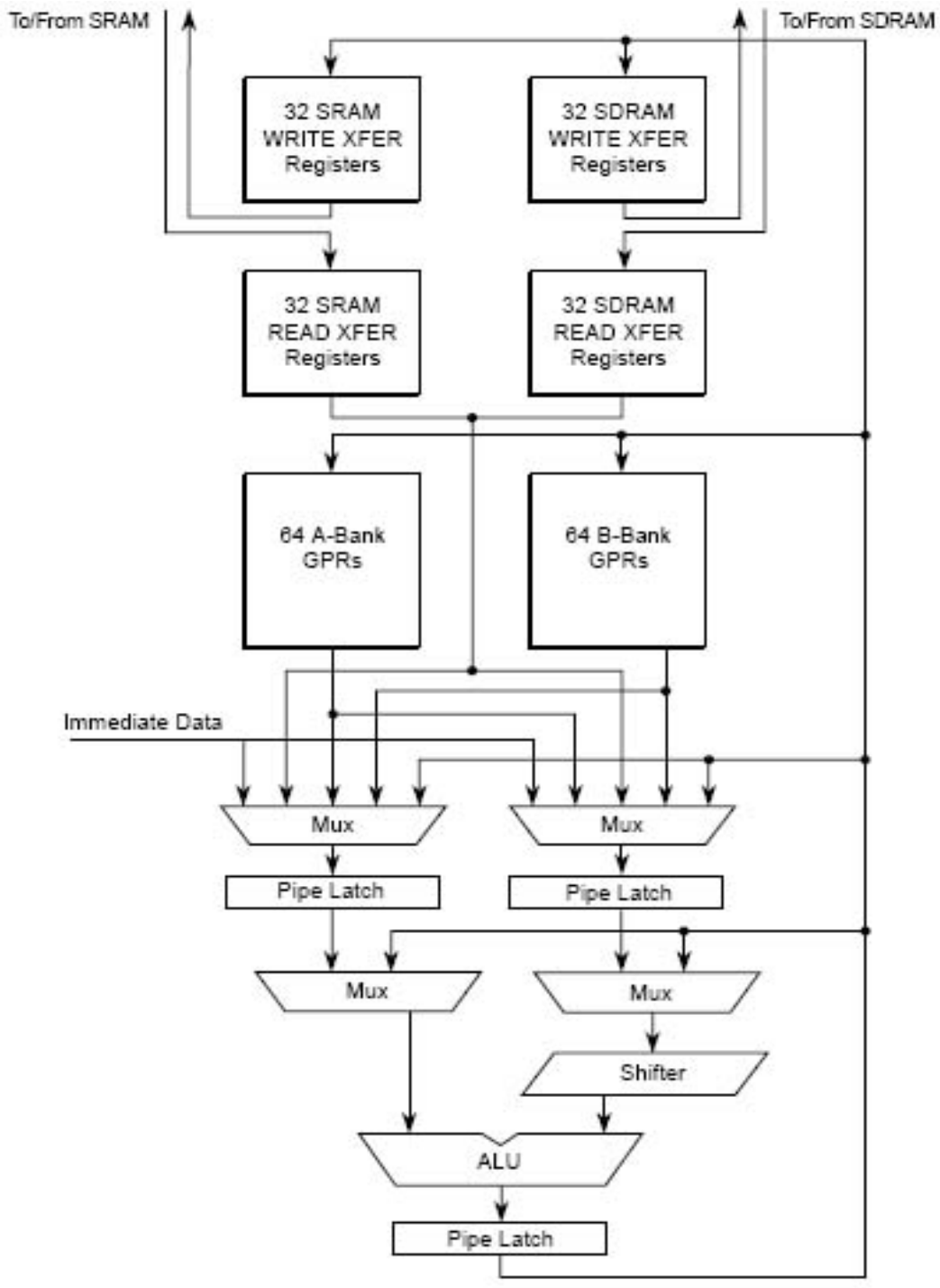


此開發工具提供我們 coding，偵錯除錯及實作(虛擬實作或實體實作)之相關功能，並可予以中斷而實際檢視記憶體或暫存器等資料流的相關數據，相當方便實用。詳細操作內容於其 Tools Guide 文件中有詳細說明，將不在此多加闡述。

5) 基本 Microcode 程式設計：
在做相關介紹之前，先提到一些資料詞彙：

Term	*Bits*
Byte	8
Word	16
Longword	32
Quadword	64

接著看看 Microengines 的 data path 運算流程以利我們理解其運作方式：



以下我們舉一個開發工具 Workbench 中所附之範例程式例子 (L3_8_1Gig.dwp: Line 47 ~ 86)，將之轉換成我們容易理解的 C 語法，來大致瀏覽其 microcode 的形式。其中「;」爲 microcode 之註解，而「//」則是自己加上對應後面 code 的 C 語法。

```
; Run the dispatch loop
.while(1)
dl_start#:
      // DL_SASource();
DL_SASource[ ]
```

```
        // if (dl_buffer_handle == IX_BUFFER_NULL) goto Main_Dispatch#;
alu[--, dl_buffer_handle, -, IX_BUFFER_NULL]
br=0[Main_Dispatch#], guess_branch
        // if (dl_next_block's 8 bits == IPV4_TAG) goto IPv4Fwd#;
br=byte[dl_next_block, 0, IPV4_TAG, IPv4Fwd#]
        // else goto DL_Drop#;
br[DL_Drop#]

Main_Dispatch#:
; ME (main) dispatch loop
        // EthernetIngress();
EthernetIngress[ ]
ingress_done#:
        // if (dl_buffer_handle == IX_BUFFER_NULL) goto dl_start#;
alu[--, dl_buffer_handle, -, IX_BUFFER_NULL]
br=0[dl_start#]
        // if (dl_next_block's 8 bits != 1) goto DL_Drop#;
br!=byte[dl_next_block, 0, 1, DL_Drop#]
IPv4Fwd#:
        // IPv4UnicastFwd();
IPv4UnicastFwd[]
ipv4_done#:
        // if (dl_next_block's 8 bits == 1) goto DL_MESink#;
br=byte[dl_next_block, 0, 1, DL_MESink#]
        // if (dl_next_block's 8 bits != IX_EXCEPTION) goto DL_Drop#;
br!=byte[dl_next_block, 0, IX_EXCEPTION, DL_Drop#]
        // { local ace_tag;
.local ace_tag
        // ace_tag = IPV4_TAG;
immed[ace_tag, IPV4_TAG]
        // DL_SetAceTag(ace_tag);
DL_SetAceTag[ace_tag]
        // }
.endlocal

DL_SASink#:
        // DL_SASink();
DL_SASink[ ]
.continue

DL_MESink#:
        // DL_MESink();
DL_MESink[ ]
ingress_eop#:                   ; for simulation perf measurements
```

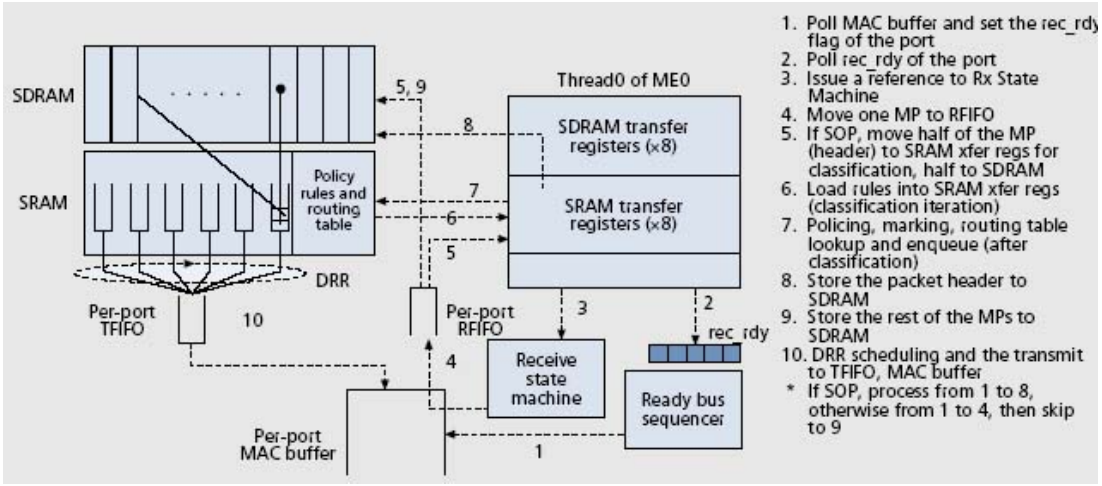
```
nop
.continue

DL_Drop#:
    // DL_Drop();
DL_Drop[ ]

.endw
```

在此以 alu 爲例子稍微做個解釋。alu 之格式爲 alu[a, b, op, c]，表示 a = b op c;的意思。也就是說如果我們用 alu[a, b, +, c]，則代表執行 a = b + c;。至於其他部分詳細的 instructions 格式及用法，包括剛才提到的 alu，都可參考在 Workbench 中的 IXP1200 Prog Ref Manual.pdf，它有十分詳盡且有系統的說明。

了解硬體及程式設計方向之後，接下來的動作便是著手實作 DiffServ 了。下圖爲在 IXP1200 上實作 DiffServ 之封包流示意圖：



至於之前提到之分類演算法，目前預計將使用 1. Range Matching, 2. Fast Binary Search 來做比較。其中 Range Matching 爲上面所提到 IEEE 論文中所使用之演算法。此兩演算法分別簡述如下：

1) Range Matching:

把 rule table 建成一個 Binary Trie，根據 prefix 的 length，最多這個 Binary Trie 可到達 32 層。在第 16 層建一個相對應的 array，所以這個 array 總共有 2^{16} 個 node，每個 node 底下可能還有 subtrie。對於每個 node 的 subtrie，建立一個一對一的 array，每個 prefix 有相對包含的最小值和最大值。所以在這個 array 裡，我們把 subtrie 所包含的每個 prefix 所包含的最大和最小值儲存下來，根據這些直下去排序，便可得到一個排序好的 array。

在建 trie 的過程中，每建一個 prefix 就有一個相對應的 Rule ID。當我們把建立起來的 trie 做 complete 的過程中，這些 Rule ID 的資訊就可以傳遞下來。當我們要找 address A，就去找到 Narrowest enclosing range。

當一個 IP Packet 近來後，我們可以把前 16 bits 擷取出來當作 index，在事先建好的 16-bit prefix table 中查詢。如果發現個 index 裡所含的資訊直出它還有 subtrie，再對後 16 bits 去事先建立好的 range array 查詢，即可得到一個 IP Packet 符合 rule table 裡的 rule ID Chain。

在二維的 Packet Classification 裡，對 Source IP 做上述方法的查詢，可以得到一個 Rule ID Chain 的 Source IP Bitmap。對 Destination IP，使用同樣的方法，也可以得到一個 Destination IP Bitmap。針對這兩個 Bitmap 做 AND 的動作，就可以得到這一個 IP Pair 所 Matching Rule Tables 的 Rule ID。

2) Fast Binary Search:

把 rule table 建成一個 Binary Trie，根據 prefix 的 length，最多這個 Binary Trie 可到達 32 層。在建 trie 的過程中，每建一個 prefix 就有一個相對應的 Rule ID。跟上述方法不同的是，在對建立起來的 trie 做 complete 的過程中，我們把 enclose 在一個 prefix 下的 node 當成是 auxiliary prefix，這些 Rule ID 的資訊就可以傳遞下來。

在第 16 層建立一個相對應的 array，因此這個 array 總共有 2^{16} 個 node，每個 node 底下可能還有 subtrie。對於每個 node 的 subtrie，建立一個一對一的 array，利用「中序」方法走訪這個 subtrie，把所走訪到的 original prefix 和 auxiliary prefix 都記錄在這個 array 中。由於利用中序方法走訪，所以得到的是一個按照我們所定義大小關係排序的 array($0 < * < 1$)。

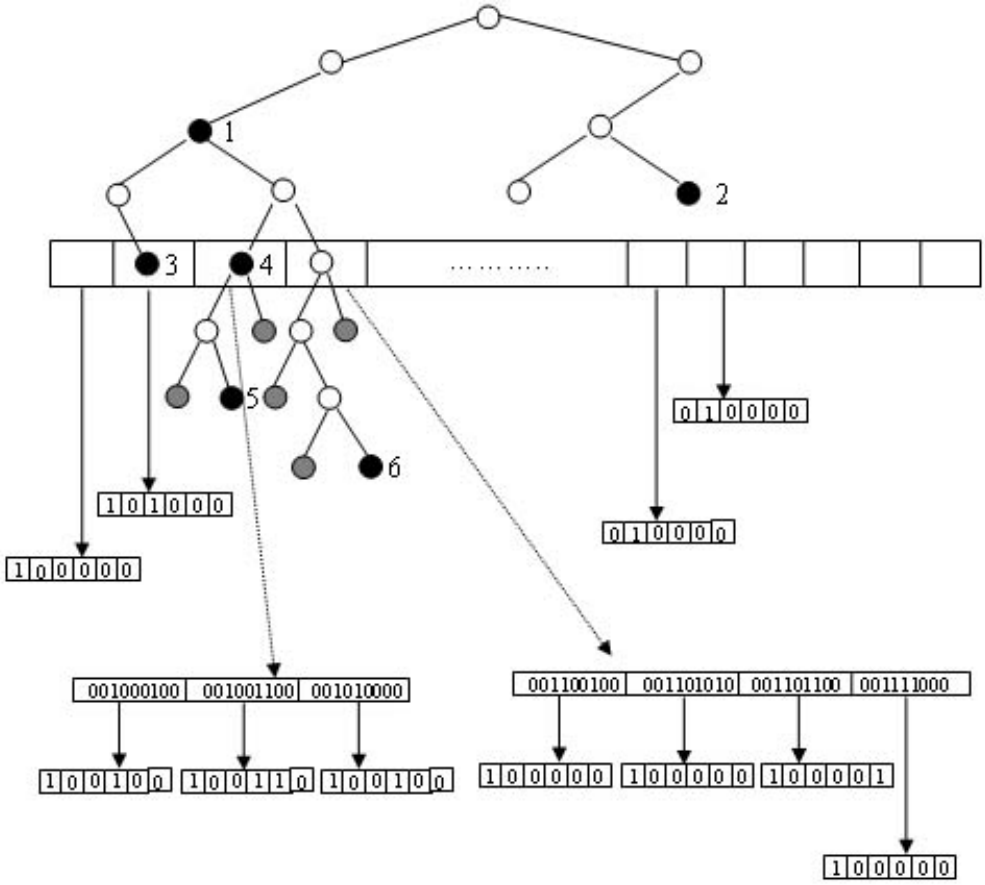
在這個 array 中，不難發現，當相鄰兩個 node 都是 auxiliary prefix 並且記錄著相同資訊時，我們可以把他們 merge 起來。而記錄下來的 prefix 是它們兩個所共有的 Longest Common Prefix。經由這個方法，可以很清楚地得到 merge 完後的 node 個數必定小於 2N (N 爲這個 subtrie 的 original prefix 個數)。在上個演算法中，因爲我們記錄每個 Original Prefix 的最大最小值，所以可以輕易地找到我們所記錄的 array node 個數爲 2N。這也是這個演算法在 storage 的議題上比之前演算法還要好的地方。

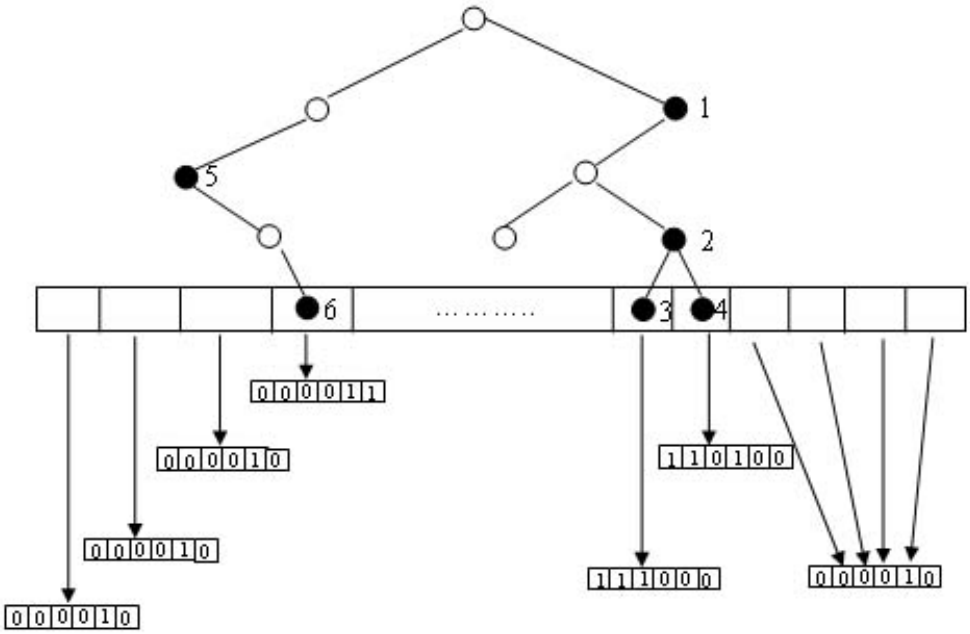
當一個 IP Packet 近來後，我們可以把前 16 bits 擷取出來當作

index，在事先建好的 16-bit prefix table 中查詢。如果發現個 index 裡所含的資訊直出它還有 subtrie，再對後 16 bits 去事先建立好的 range array 查詢，即可得到一個 IP Packet 符合 rule table 裡的 rule ID Chain。

在二維的 Packet Classification 裡，對 Source IP 做上述方法的查詢，可以得到一個 Rule ID Chain 的 Source IP Bitmap。對 Destination IP，使用同樣的方法，也可以得到一個 Destination IP Bitmap。針對這兩個 Bitmap 做 AND 的動作，就可以得到這一個 IP Pair 所 Matching Rule Tables 的 Rule ID。

以下舉個例子來看看：





在此以 8-bit 爲例：

Rule #	Source IP	Destination IP
1	3.0/2	8.0/1
2	6.0/3	11.3/3
3	1.1/4	10.3/4
4	2.0/4	11.8/4
5	2.4/6	3.0/2
6	3.6/7	3.0/4

IP Pair (3.7, 3.10)，從第一個圖我們可以得到 bit vector: 100001，第二個圖可以得到 bit vector: 000011。所以從這二個 bit vectors 去做"AND"我們可以得到 000001，即可知這個 IP Pair match "Rule 6"。

完成實作後，實際在 IXP1200 上收送封包，取得封包收送狀態得到相關數據，以此繪表並比較不同條件不同情況之差異，同時也比較不同演算法之差異，並討論未來改善空間。

(3) 預期結果

正如之前所述，這次專題題目定爲「基於 DiffServ Edge Routers 之網路處理器分類處理效能之研究」。因此，在完成並得到最後結果之前，預期將會學到許多之前所提到的已知及未知的相關知識。包括 router 的運作方式，NPU 之架構概念及實作方式，IXP1200 之相關技術及應用，封包格式及 DiffServ 分類處理之概念等。

於實作方面，兩種 DiffServ 演算法將實作出來並加以比較，而預期

Fast Binary Search 將會比 Range Matching 的分類效率還要佳。

此外，於實作經驗上，也預期自己會有更多的體驗。想必在此次專題計畫中，必定會遭受許多挫折，會灰心，會失望。然而經過這一連串的薰陶，將會使自己得到許多寶貴的經驗，對日後的後續研究奠定不可磨滅的基礎。

(4) 參考文獻

[1] IXP1200 Programming, Erik Johnson & Aaron Kunze, 2002 Intel Press

[2] Intel IXP1200 Network Processor Family – IXA SDK 2.01 Getting Started, Intel doc. No. A68374-002, December 2001.

[3] Intel IXP1200 Network Processor Family – Hardware Reference Manual, Intel doc. No. 278303-008, August 2001.

[4] Intel IXP1200 Network Processor Family – Microcode Programmer's Reference Manual, Intel doc. No. 278304-009, August 2001.

[5] Intel IXP1200 Network Processor Family – Development Tools User's Guide, Intel doc. No. 278302-009, December, 2001

[6] Ying-Dar Lin, Yi-Neng Lin, Shun-Chin Yang and Yu-Sheng Lin, " DiffServ Edge Routers over Network Processors : Implementation and Evaluation", IEEE Network July/August 2003

[7] Intel IXA 官方網站 http://www.intel.com/design/network/ixa.htm

[8] Intel IXP1200 官方網站 http://www.intel.com/design/network/products/npfamily/ixp1200.htm

[9] Intel IXA University Program http://www.ixaedu.com/home/

(5)需要指導教授指導內容

1. 相關網路基本常識
2. 有效率之研究方法及認知

表 C802