在嵌入式 Linux 系統上的新一代網路協定轉換軟體開發

The Development of Network Address Translation and Protocol Translation on Embedded Linux

廖永申

大葉大學資訊工程研究所

r8906015@mail.dyu.edu.tw

王欣平

大葉大學資訊工程研究所

swang@mail.dyu.edu.tw

摘要

近年來網路以倍數成長，使得現有的 IPv4 網路位址即將不敷使用。為解決此問題，新一代的網際網路通訊協定 IPv6已被提出。從 IPv4 向 IPv6 的過渡仍需相當長的過程，在此期間，必須讓現有的 IPv4 應用軟體仍可繼續使用。因此 IETF 訂定了一系列的標準，提供了多種轉移的過渡策略。其中網路協定轉換機制 (NAT-PT) 正在 Windows、FreeBSD、Linux 等系統上研發。

本文提出在居家用閘道器(Residential Gateway) 建置網路協定轉換機置 (NAT-PT)，來解決家用網路與新一代網際網路的連接問題，同時分析模組效能。基於嵌入式系統對電源消耗量的考量，分析重點放在低功率消耗的 NAT-PT 模組設計上，針對 NAT-PT 部分模組進行演算法最佳化的研究，並分析記憶體的存取指令數來減少記憶體的電源消耗。

關鍵詞：嵌入式系統、LINUX、IPv6、閘道器、NAT-PT

ABSTRACT

Rapid growth of Internet participants and development of mobile computing have escalated the depletion of limited 32bits IPv4 addresses. In responding, newer IPv6 that equipped with 128 bits addresses is developed. The transition of IPv4 to IPv6 takes time, hence IETF develops a series of schemes for the Internet to sustain both IPv4 and IPv6 during the transition. Among these schemes, NAT-PT is well adapted for its simplicity.

This paper concerns implementation of NAT-PT on an embedded based Residential Gateway and assesses the power dissipation of the implementation using dynamic profiling and cross-compiling techniques.

Key Words：Embedded system, Linux,IPv6, Gateway, NAT-PT

一、前言

隨著網際網路的普及，短短十數年間明顯的成長至現今的規模。網路上每一台主機都需要一個唯一的 IP 位址，因此目前 32 bits 的 IPv4 位址也因需求級數成長而即將使用殆盡。特別是隨著無線網路普及，隨身攜帶以嵌入式系統為基底的手機、個人數位助理 (PDA)，和家中的各項電器設備都將與網路連接，也因此 IP 位址不足的問題顯得越來越嚴重。為解決此問題，網際網路工程師工作小組 (Internet Engineering Task Force-IETF)提出了新一代的網際網路協定 IPv6〔4〕。IPv6 改良了 IPv4 的許多問題，特別是 128 bits 的定址長度解決了目前網路 IP 位址不足的問題。

在不久的將來，會有相當數目的網路系統將由 IPv4 升級為 IPv6，屆時網際網路上將同時存在著 IPv4 和 IPv6 的網路設備，這將使網路傳輸發生問題。針對此一問題 IETF(Internet Engineer Task Force)提出了幾種方法，讓在這個過渡期受到的影響降到最少〔5 - 6〕。在 IETF 提出的幾種方法中，本文選擇了簡便的 NAT-PT 作為轉換機制。目前 NAT-PT 在 PC 上已有部份實作，如 Microsoft、Cisco、BT、KAME 等公司相繼在 Windows、FreeBSD、Linux 等平台上開發了 NAT-PT 的實作程式。

本文主旨在探討 NAT-PT 在嵌入式的居家用閘道器(Residential Gateway)上的開發測試與效能分析。同時基於嵌入式系統對電源消耗量的特殊考量，本文所探討的部份重點在 NAT-PT 在嵌入式系統上記憶體資料存取部份的功率消耗分析評估，並針對部分 NAT-PT 模組改用較佳的搜尋演算法提昇執行效能。

我們使用一般 PC 建置居家用閘道器 (Residential Gateway)，搭配 Linux 系統，透過 NAT-PT 達到連接 IPv4 及 IPv6 網路位址的目標。使用動態側描 (Profiling) 方式，統計 NAT-PT 模組的函數呼叫情形來分析 NAT-PT

模組的執行效能。進一步統計 ARM 指令集的記憶體存取指令數，來分析記憶體存取的功率消耗。

二、相關研究

(一)IPv6 標頭

IPv6 的標頭很類似 IPv4，由基本 IPv6 標頭和選擇性的延伸標頭所組成。IPv6 標頭共使用 40 個位元組的長度，包括 8 個欄位與兩個位址。圖 1 為 IPv6 標頭格式。

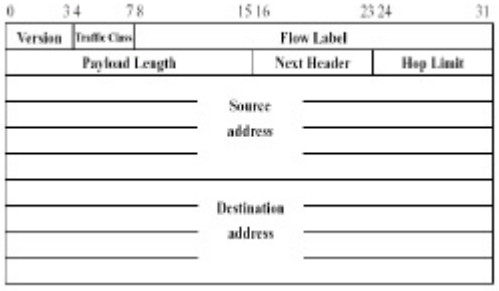

圖 1 IPv6 標頭格式

表 1 IPv4 與 IPv6 的欄位差異

IPv4 欄位	IPv6 欄位
Header Length	欄位取消 (固定長度 40Bytes)
TOS	Traffic Class + Flow Label
Length	Payload Length
TTL	Hop Limit
Header Checksum	欄位取消
32 bits Address	128 bits Address

表 1 簡列出 IPv4 與 IPv6 的欄位差異。由於 IPv6 採固定表頭長度，因此取消 IPv4 表頭長度(Header Length)。服務型式(TOS)欄位由其它機制取代。由於 IPv6 只支援端點對端點(end-to-end)分割，故不再需要識別(Identification)、旗號(Flags)和區段移補(Fragment offset)欄位。靠媒介存取(media access)控制程序中的檢查和，因此取消表頭檢查和(Header Checksum)，減少表頭處理的負擔。有些欄位重新命名並重新定義。IPv4 的整體長度(Length)由 IPv6 的封包承載長度(Payload Length)取代。協定型式(Protocol Type)欄位重新命名成下一表頭(Next Header)。還可增加延伸表頭(Extension Header)。存活時間(Time to live)欄位變更成跳躍點限制(hop limit)以符合實際情形。增加了新的優先順序(Priority)和訊流標記(Flow Label)欄位，來支援即時服務。

(二) 網路協定轉換機制

建置新的 IPv6 網路應該在對既有 IPv4 網路、系統業者、及終端使用者干擾最小的狀況下完成。IETF 目前提供了三種機制可協助 IPv4 順利轉移至 IPv6，它們分別是雙堆疊機制(Dual stack)、通道機制(Tunneling)，以及轉換機制(Translator)。

使用雙堆疊機制的前提是每一主機的作業系統本身必須同時支援 IPv4 及 IPv6，而通道機制祇適用於兩個 IPv6 網域的連線。相對的，轉換機制則無上述限制，且它的架構簡單，因此本文選定了屬於轉換機制中適合在閘道器上建置的 NAT-PT〔11〕〔14-15〕作為居家用閘道器上的轉換機制。

居家用閘道器上的封包轉換機制簡述於下，當閘道器上的 NAT-PT 模組收到來自 eth0 由居家側 IPv6 電腦傳來的封包後，NAT-PT 模組會先搜尋 IP 位址對映表中是否已有相符的 v6-v4 對映位址，如果沒有就在可用位址列中取得一個新的 IPv4 的位址，再將封包轉成 IPv4 的位址後，經 eth1 傳入網際網路。同樣的，當閘道器收到來自 IPv4 端的封包後，經由 IP 位址對映表轉換成 IPv6 封包後，透過 eth0 傳進 IPv6 家用網路中的對映主機。

NAT-PT 主要為位址的對映轉換，屬於資料為主(Data Dominate)的應用，當在搜尋對映表時，必須對記憶體中的資料做大量的比對，因此記憶體存取是影響效能的因素之一。同時其它研究指出〔9〕，在嵌入式系統上記憶體存取為主要的功率消耗，因此如何有效規劃 NAT-PT 記憶體存取，對系統整體效能及減少功率耗損是很重要的課題。

(三) 嵌入式系統

嵌入式系統和 PC 主要的不同在於，嵌入式系統大多針對特定用途設計，因此可用資源相較一般多用途的 PC 要少，同時嵌入式系統的可攜式特性和體積輕小的需求使它對電源消耗量非常敏感。在日常生活中常見的嵌入式系統產品有手機、全球衛星定位系統(GPS)、個人數位助理(PDA)、Set-Top-Box 等。

嵌入式系統功率消耗多寡不僅影響機器的待機時間，同時相應產生的熱效應也影響了系統的穩定性、效能和價格。而省電功能的嵌入式系統研究近年來相當盛行〔7〕〔12-13〕。同時其它研究也指出記憶體資料存取佔系統

電源消耗量大部份〔3〕〔10〕，因此本文所探討的部份重點將在的 NAT-PT 在嵌入式系統上記憶體資料存取部份的功率消耗分析評估。

(四) 搜尋演算法

當我們要讀取、更新或刪除一筆資料，就必須對資料進行搜尋。演算法的好壞對效能的影響很大，在此我們探討循序搜尋、二分搜尋與雜湊搜尋演算法。並對各種演算法在 ARM 上的記憶體存取指令進行研究，尋找出哪種搜尋演算法的電源消耗比較低。

三、系統架構

(一) 測試與分析系統架構說明

我們使用 PC 建置居家用閘道器，透過對 NAT-PT 的動態側描，統計 NAT-PT 模組的函數呼叫情形，佐以統計 ARM 指令集的記憶體存取指令數，來分析功率消耗。圖 2 中電腦 A、B 構成一 IPv6 子網路，用來模擬家用網路中的家用主機，這些家用主機執行 IPv6 協定。以上居家子網路通訊閘設定為電腦 C，執行 NAT-PT 轉換機制用來模擬設計中的居家用閘道器。在閘道器電腦 C 上設置兩個乙太網路介面卡，分別是負責下傳(downstream) 用來連接 IPv6 的家用網路的 eth0，及上傳(upstream) 用來連接 IPv4 的網際網路的 eth1。

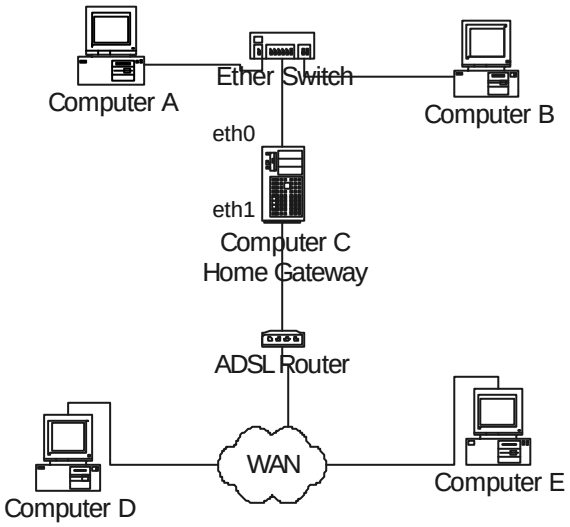

圖 2 系統架構圖

架構中所需軟體及相關設定簡述如下。在居家用閘道器上的軟體需求主要包括 Linux 作業系統及 NAT-PT 模組。此處我們安裝的是 RedHat 7.3 系統核心為 2.4.18 的版本，另外我們自 Electronic Telecommunications Research Institute 〔20〕取得 NAT-PT 模組的原始碼。將取得的 NAT-PT 模組的原始碼自行編譯。

使用 modprobe ipv6 指令將 IPv6 模組載入 Linux 核心。我們使用 route 程式來設定路由表。在居家 IPv6 網域的電腦 A、B 上我們使用 Windows XP 作業系統。相關網路設定我們使用 Windows XP 的 IPv6 設定程式 ipv6，來設定 IPv6 位址和路由表。

(二) 測試與分析步驟說明

首先本文利用 GNU(Free Software Foundation) 開發的程式開發工具 gprof 針對 NAT-PT〔20〕在 PC 上的實作做動態側描(Profiling)以取得 NAT-PT 程式執行中的各函數呼叫次數及函數呼叫圖(Call Graph)，這部份的工作主要是找出 NAT-PT 中最常執行的程序供後續最佳化處理。

其次是利用 GNU Cross-compiler 產生 NAT-PT 的 ARM 組合語言程式碼，最後利用先前動態側描及編譯得到的 NAT-PT 組合語言程式碼來計算執行 NAT-PT 中各模組所需存取記憶體的次數。詳細測試與分析內容如下。

(三) Profiling 及測試封包

進行 Profiling 前，首先我們使用 gcc 編譯器版本 2.96 搭配 glibc 版本 2.2，並設定 -pg 編譯選項重新編譯及連結產生新的 NAT-PT 執行檔〔2〕。經由編譯器 -pg 選項，編譯時程式中會加入系統指令來追蹤呼叫函數的次數及時間。而在程式連結時，在程式中會加入統計的程序，藉以產生所需的資訊。因為 NAT-PT 主要的功能為提供 IPv4/IPv6 協定的轉換，因此針對 NAT-PT 測試時只要是能夠產生 IPv6 的封包就可以用來測試，而本文此處使用的是系統提供的 ping6 來產生測試封包。

整個測試過程中我們反覆的在電腦 A、B 上產生數千個 ping6 封包，對網路上事先選定的十台電腦發出，封包透過居家用閘道器中的 NAT-PT 轉換成 IPv4 封包送出，然後封包穿過 ADSL 路由器傳送到 Internet 上，接著被 ping 的主機分別回傳 ICMP 的封包回居家用閘道器，閘道器收到回應的訊息後再將 IPv4 轉成 IPv6 封包，分別傳給對映的電腦 A、B。

經由編譯程式時開啟側描的選項的設定，此時程式執行後會產生一個統計資料檔，這個統計資料檔可以使用 gprof 來觀看數據。另外我們使用 AT&T 的 DOT〔1〕程式根據上述統計資料檔產生函數呼叫圖。側描 NAT-PT 結果如圖 3 所示。我們統計了 NAT-PT 各模組的呼叫次數，發現了幾個模組花費較多的執行時間。因此要提昇效能，就必須對這些模組進行最佳化。

(四) Cross-Compiler 安裝及 ARM 程式碼產生

Profiling 結果提供 NAT-PT 函式層

(Functional Level)各函式執行的次數及彼此呼叫關係的資訊，這些資訊有助於 NAT-PT 最佳資料結構及演算法的開發。但是如果我們要做系統架構最佳化設計的話，我們還需要指令層(Instruction Level)的統計資料。特別是需要計憶體存取指令數目及存取計憶體位址等資訊。這部份的工作我們透過 GNU Cross-compiler 達成，相關內容及方法分述於下。

相關在 PC 上安裝 Cross-Compiler 部份，我們可以從網路上下載適合自己平台的 ARM-Linux Cross-Compiler〔16〕，也可以自己編譯產生。如果要自己編譯原始檔，必須下載下列幾個檔案並安裝: (1)Bin Utilities (Binutils) 版本 2.x ，(2)GCC Compiler 版本 2.9x，(3)GLIBC Library 版本 2.x〔17 - 18〕。詳細編譯 Cross-compiler 的步驟可參考〔19〕。

為了得到 ARM 的組合語言程式碼，我們必須在使用 Cross-compiler 編譯的過程中使用 -S 選項，同時為了將迴圈的展開讓我們可以得到較詳盡的 ARM 程式碼，我們也在編譯過程中加入了 –funroll-loops 選項。一個參考編譯指令如下:

arm-linux-gcc –S –funroll-loops nat-pt.c

編譯結果會產生以.s 的延伸檔名的 ARM 的組合語言程式碼檔案。此處礙於篇幅限制，本文僅列出各模組相關統計資料如表 2 ARM 程式碼統計表所示。

我們統計了記憶體存取數，發現了幾個模組花費較多的記憶體存取指令在資料的存取或搬移上。因此要降低功率的消耗，就必須針對這些模組進行最佳化。我們知道高階程式語言的語法會影響到編譯後的組合語言程式碼，而搜尋的快或慢影響很大，因此我們針對資料搜尋比對這部分的程式語法來最佳化。

(五) 搜尋方式研究

目前 NAT-PT 模組所使用的資料搜尋演算法為循序搜尋法(Sequential Search)，最差情形下比較次數為 N，平均比較次數為 $\frac{1}{N}(1+2+3+...+N)=\frac{N+1}{2}$ 。時間複雜度為 $O(n)$ 。如果使用二分搜尋法(Binary Search)，則最差情形下比較次數為 $\lfloor \log_2 n \rfloor +1$，平均比較次數為 $\lfloor \log_2 n \rfloor + \frac{1}{2}$，時間複雜度為 $O(\log_2 n)$。

假設 IP 位址數為 20，使用循序搜尋法最差情況下須比較 20 次，而使用二分搜尋法最差情況下也只需比較 5 次。另外，如果使用雜湊(Hash)搜尋法，時間複雜度只需 $O(1)$ ，但需要較多的記憶體空間，因當雜湊表太滿時，碰撞率會因空間擁擠而急遽升高，因此使用雜湊搜尋法會比較浪費記憶體空間。

針對記憶體的電源消耗我們實際分析了循序搜尋、二分搜尋和雜湊法。因為速度快的演算法不一定記憶體的存取指令數就一定會比較低，所以我們使用 Cross-compiler 編譯各搜尋法程式碼，得到 ARM 的組合語言程式碼，來比較哪種搜尋演算法的記憶體存取數比較少，電源消耗也會比較少。

四、結果分析

(一) 結果分析

實際測試 Pin6 結果，封包傳遞的預估值和實際測試結果會有些許差異。這些差異發生的原因是在實際測試中，有外來的 Internet 封包進入閘道器，也有些 Ping6封包會因各種網路原因而產生遺失，例如頻寬限制、封包碰撞或路由器忙碌，導致傳送和接收封包數有些許不同，但就本文的主旨而言，這些細微差異並不影響資料的參考性。

表 2 中 ARM 指令總數代表 ARM 的記憶體存取 LDR/ STR 指令總數。記憶體存取數為模組呼叫次數乘以 ARM 指令總數然後均值後得到的比率。雖然我們編譯時已展開所有迴圈，但動態的迴圈數並無法預先知道，因此此處我們得到記憶體存取數近似值，詳細的記憶體存取數須靠指令層的模擬取得。

我們統計了模組的呼叫次數和記憶體存取數，發現以下幾個模組花費較多的時間在資料的存取或搬移上，write_packet 、PT_Translate_IPv6_Packet、PT_Translate_IPv4_Packet 、translate_ICMPv4_dgram 、translate_ICMPv6_dgram、Get_IPv6_hardware_address、get_IPv4_hardware_address 、NAT_Translate_IPv6_To_IPv4、NAT_Translate_IPv4_ To_IPv6。

要降低功率的消耗，我們就必須針對這些函數模組進行最佳化。其中 translate_ICMPv6_dgram 及 translate_ICMPv4_dgram 部分是 Application Dependent，也就是說它們是執行 ping6 的結果。get_IPv6_hardware_address、get_IPv4_hardware_ address，因為涉及傳送 ARP 封包獲得 MAC 位址動作，所以執行時間較長。write_ packet 主要功能為送出封包，將指標變數傳給網路卡驅動程式。以上這幾個函式因為和NAT-PT 轉換機制並無絕對關聯，所以我們將不去討論它們，我們討論的重點放在 PT_Translate_IPv6_Packet 、PT_Translate_IPv4_Packet 、 NAT_Translate_

IPv6_To_IPv4 和 NAT_Translate_IPv4_To_IPv6 模組。

另外我們觀察 read_packet_IPv4 與 read_packet_IPv6 的執行時間，很明顯的 read_packet_IPv6 的時間是 read_packet_IPv4 的兩倍以上，因為 IPv6 的標頭佔用 40 個位元組，比 IPv4 的 20 位元組多出一倍，所以延長了封包處理的時間。

圖 3 顯示由 Functional Level 側描得到的函數呼叫圖 (Call graph)。圖中顯示了模組名稱、模組執行的關係。從圖 3 函數呼叫圖可觀察到，PT_Translate_IPv6_Packet 呼叫 NAT_Translate_IPv6_To_IPv4 來轉換位址，而 NAT_Translate_IPv6_To_IPv4 主要轉換 IPv6 的位址為 IPv4 位址。同樣的 NAT_Translate_IPv4_To_IPv6 主要轉換 IPv4 的位址為 IPv6 位址。

值得注意的是 NAT_Translate_IPv6_To_IPv4 和 NAT_Translate_IPv4_To_IPv6 存取相同一位址對映表，位址對映表的資料結構如下所示。

```
struct Mapping_List {
struct in_addr        IPv4_Address;
unsigned short        IPv4_Port;
struct in6_addr       IPv6_Address;
unsigned short        IPv6_Port;
unsigned int          Number_of_Sessions;
unsigned int          TCP_Timer_Value;
unsigned int          UDP_Timer_Value;
struct Mapping_List   *pNext;
};
```

IPv6 位址每次必須比對 16 位元組，長度是 IPv4 的 4 倍，如果同時有 20 個位址對映關係表，在最壞情況下必須比對 20 x 16=320 位元組的資料，因此採用最佳化的搜尋方式有助於減少資料比對的次數及時間。所以我們探討的重點在位址對映表的搜尋演算法上，希望減少資料比對的次數及時間。

(二) 演算法分析

如表 3，我們統計了幾種搜尋演算法的 ARM 記憶體存取指令數。知道二分搜尋和雜湊法(Linear Probing)的記憶體存取是最少的，而循序搜尋演算法則是存取次數最多的。另外對雜湊法我們分別測試了兩種碰撞(hash collision)的處理方式，結果顯示分別鏈結(separate chaining)比線性預測(Linear Probing)記憶體存取數多。因為分別鏈結在處理雜湊函數的碰撞問題需要產生新的鏈結串列，所以記憶體存取數比線性預測多。相關於各不同搜尋演算法的詳細討論請參考〔1〕〔8〕。

另外我們特別統計了資料 Insert 和 Search 副程式的記憶體存取數，因為在 Insert 和 Search 過程中，需要大量的存取到記憶體，且會包含大量的迴圈，迴圈的次數影響整體演算法的效率，更與電源的消耗有密切的關係。

在 NAT-PT 模組中，位址資料數預設值為 20 筆。在區域網路中因為主機數量遠少於封包數量，因此討論重點將在資料的搜尋部分。如果搜尋次數可以降低，效能就會提昇，記憶體存取數也會降低，電源消耗也會降低，因此好的搜尋演算法是很重要的。在實際的測試當中，發現到不同的程式語法也會影響到記憶體存取的指令數量，因此使用好的程式語法也有助於減少記憶體存取數並增加程式執行的效能。由表 3 我們得知，在雜湊法中又以 Linear Probing 的記憶體存取指令數較少。

我們知道閘道器在一秒鐘內收到的封包數量可能極為龐大，因此我們必須選擇好的演算法來縮短處理封包的時間，縮短封包延遲。

五、結論

以上測試結果顯示實作的 Linux 居家用閘道器能透過 NAT-PT 能夠進行 IPv6 和 IPv4 間的封包轉換使得 IPv6 和 IPv4 的主機可以互相連線。經由動態側描的結果，和 ARM 指令集的分析，我們知道該針對那些模組最佳化，並減少記憶體存取數，來減少電源的消耗。經由實際測試，我們知道雜湊法(Linear Probing)的時間複雜度最低，效能最好。可縮短資料搜尋及比對的時間和次數來提昇系統效能。以效能和記憶體存取數來看，雜湊法(Linear Probing)也是好的選擇，因此選用雜湊法(Linear Probing)在電源消耗和執行效能上都是最好的選擇。

未來發展部分，在嵌入式系統方面，我們可根據上述測試方法、結果，來設計、規劃嵌入式系統，減少記憶體存取次數來降低電源消耗〔9〕，並針對記憶體有限的嵌入式系統，規劃記憶體的資料結構和管理記憶體的配置。在 NAT-PT 模組方面，希望在未來可支援讓一 IPv4 Address 利用 Port Number 對應到多個 IPv6 主機。

六、參考文獻

[1] 張邵勳, 蔡志敏, “演算法入門與進階:使用 C 語言”, 台北市:松崗,民 80

[2] David B.Stewart, “Measuring Execution Time and Read-Time Performance,” *Embedded Systems Conference* San Francisco, CA, April 2001.

[3] F. Catthoor, S. Wuytack, L. Nachtergaele, A. Vandecappelle, F. Balasa, and E.D. Greef, “Custom Memory Management Methodology-Exploration of Memory Organization for Embedded Multimedia System Design,” Kluwer Academic Publishers, 1998.

[4] S. Deering and R. Hinden, “Internet Protocol, Version 6 (IPv6) Specification,” RFC 2460, 1998.

[5] R. Gilligan and E. Nordmark, “Transition Mechanisms for IPv6 Hosts and Routers,” RFC 1933, 1996.

[6] R. Gilligan and E. Nordmark, “Transition Mechanisms for IPv6 Hosts and Routers,” RFC 2893, 2000.

[7] Patrick Hicks, Matthew Walnock, and Robert Michael Owens, “Analysis of power consumption in memory hierarchies,” Low Power Electronics and Design, 1997. Proceedings., 1997 International Symposium on , pp. 239-242, 1997.

[8] Ellis Horowitz, Sanguthevar Rajasekaran, and Sartaj Sahni, “Computer Algorithms,” New York: Computer Science Press, 1998.

[9] J.C.B. Mattos, M. Kreutz, and L. Carro, “Low-power control architecture for embedded processors,” Integrated Circuits and Systems Design, 2002. Proceedings. 15th Symposium on, pp. 221- 226, 2002.

[10] Wen-Tsong Shiue and Chaitali Chakrabarti, “Memory Exploration for Low Power Embedded Systems,” Journal of VLSI Signal Processing, pp.167-178, Nov 2001.

[11] E. Nordmark., “Stateless IP/ICMP Translation Algorithm (SIIT) ,” RFC 2765, 2000.

[12] Wen-Tsong Shiue and Chaitali Chakrabarti , “Memory exploration for low power embedded systems,” ISCAS '99, Volume: 1 ,pp. 250-253, 1999.

[13] Hiroyuki Tomiyama, Tohru Ishihara, Akihiko Inoue, and Hiroto Yasuura, “Instruction scheduling for power reduction in processor-based system design,” Design, Automation and Test in Europe, 1998., Proceedings , pp. 855- 860, 1998.

[14] Xiaoyu Zhao and Yan Ma, “Linux Based NAT-PT Gateway Implementation,”

[15] G. Tsirtsis and P. Srisuresh, “Network Address Translation- Protocol Translation (NAT-PT) ,” RFC 2766 , 2000.SDF

[16] http://www.lart.tudelft.nl/lartware/compile-tools

[17] http://gcc.gnu.org/onlinedocs/gcc-3.0/gcc_4.html#SEC54

[18] http://gcc.gnu.org/onlinedocs/gcc/ARM-Options.html

[19] http://www.ailis.de/~k/knowledge/crosscompiling

[20] http://www.ipv6.or.kr

[21] http://www.research.att.com/sw/tools/graphviz

表 2 ARM 程式碼統計表

模組名稱	ARM 指令總數	模組呼叫次數	記憶體存取	模組執行時間
write_packet	70	23924	12.05%	1.80%
PT_Translate_IPv6_Packet	205	8030	11.85%	10.74%
PT_Translate_IPv4_Packet	203	7989	11.67%	3.60%
translate_ICMPv4_dgram	196	7890	11.13%	18.22%
Get_Ipv6_hardware_address	180	7989	10.35%	12.60%
Get_Ipv4_hardware_address	169	7973	9.70%	12.62%
translate_ICMPv6_dgram	153	8000	8.81%	3.59%
NAT_Translate_IPv6_To_IPv4	92	8030	5.32%	5.37%
NAT_Translate_IPv4_To_IPv6	66	7974	3.79%	16.23%
create_IPv4_header	48	8083	2.79%	3.56%
read_packet_IPv6	46	8030	2.66%	5.37%
Util_compute_checksum_pseudo	23	16008	2.65%	0.90%
read_packet_IPv4	46	7989	2.64%	1.80%
Create_IPv6_header	46	7989	2.64%	1.80%

表 3 搜尋演算法記憶體存取指令數

演算法 / ARM 函數	循序搜尋	二分搜尋	雜湊法 (Linear Probing)	雜湊法 (separate chaining)
整體總指令數	81	49	51	73
Insert **指令**	28	25	16	25
Search **指令**	6	10	17	18

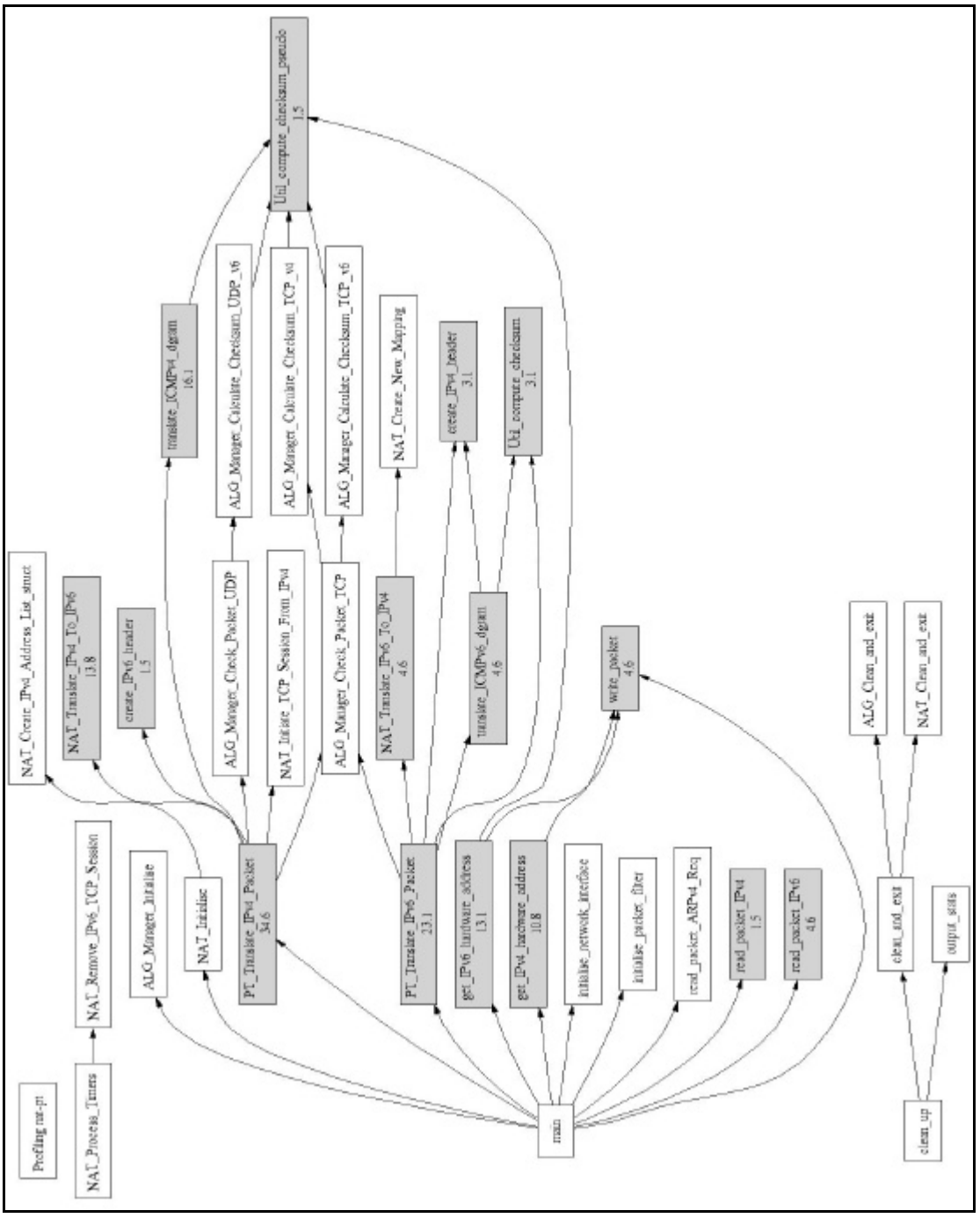

圖 3 函數呼叫圖