# 國立交通大學

## 資訊學院　資訊學程

## 碩 士 論 文

設計與實作 Linux 作業系統的

網路行為分析之工具

**Design and Implementation of a Networking Behavior Analysis Tool for Linux Operating System**

研 究 生：徐勝威

指導教授：曾建超　教授

中 華 民 國 九 十 八 年 九 月

# 設計與實作 Linux 作業系統的
# 網路行為分析之工具

Design and Implementation of a Networking Behavior
Analysis Tool for Linux Operating System


研 究 生：徐勝威　　Student：Sheng-Wei Hsu

指導教授：曾建超　　Advisor：Chien-Chao Tseng


國 立 交 通 大 學

資訊學院　資訊學程

碩 士 論 文

A Thesis

Submitted to College of Computer Science
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of Master of Science
in
Computer Science
Sep 2009

Hsinchu, Taiwan, Republic of China

中華民國九十八年九月

# 設計與實作 Linux 作業系統的

# 網路行為分析之工具

研究生： 徐勝威　　指導教授： 曾建超 博士

國立交通大學資訊學院 資訊學程 碩士班

## 摘　要

隨著資訊與網路科技的快速發展，造就了數位多媒體裝置的興起，使得資訊家電與網路的整合成為一個新的課題。同時，也因為市場的需求的增加，傳統的資訊產品開發方式已經不能滿足市場上迫切的需求，網通市場因此導向了快速整合一途。

隨著資訊產品的整合加速，產品的開發週期也因此受到了壓縮，相對的測試時間也跟著縮短，使得市場上充斥著各種性能良誘不齊的多媒體網路裝置。因此，如何開發出一個網路評比工具，用來快速的分析網路裝置的行為模式，進而分析出其整體效能，甚至是找出系統效能上的瓶頸，也是一個很重要的課題。

目前為止，很少看到有這樣的工具存在於市面上，也因此開發出能夠監控網路核心行為的工具程式便成了一件刻不容緩的事情。

首先，這樣的工具程式一定要能具備如下幾項功能，才能滿足分析網路行為的基本需求。(1)必須具備分析網路堆疊核心函數的功能，藉此將封包進出網路堆疊的路徑透明化(2)必須具備有監控網路核心事件發生的功能，網路行為會受到諸多因素而改變如：網路斷線、IP 位址改變等(3)必須具備有封包截取器，才能解析封包內容，以了解網路協議的行為(4)必須具備有能夠快速分析網路行為與系統效能之能力。

本論文主要目標在於開發一個網路效能評比工具 NBA(Networking Behavior Analysys Tool)以整合上述所有功能，以量測在不同網路層所花費的時間。藉此工具找出封包在傳送及接收時所發生的延遲時間以及分析網路通訊堆疊的整體效能。

論文的最後，將會實測出本軟體的處理能力，以確保對核心原始碼的改動不會造成太多的負面效應，以影響測試結果的公正性。

# Design and Implementation of a Networking Behavior Analysis Tool for Linux Operating System

Student： Sheng-Wei Hsu　　Advisor： Dr. Chien-Chao Tseng

Degree Program of Computer Science
National Chiao Tung University

## ABSTRACT

With the swift advance in information technology and internet, digital multimdedia devices have quickly emerged. The integration of informational home appliance and networking has becoming a new topic. In the meanwhile, due to the stronger market demand, the conventional development model of information products failed to fulfill the market requirements. Thus internet and commnunication product market has lead to the trend of rapid integration.

The rapid integration of information products reduces their development circle; meanwhile, time to test the products is also cut down. Therefor, there are an evalutaion tool, which can analyse the behavior model of the network equipments, conclude an integral efficiency, even find out the troubles of the of system, also is an important issue.

So far it's hard to find such a tool, and that is why developing a software which can monitor the network core behavior is going to be of great urgent.

First, this kind of tool program should provide functions as below to reach the basic needs of network behavior analysis: (1) it must support of analysis the key function of networking protocol stack to fint out the time delay of packet transmitting and receiving(2)It must support of monitor kernel event function because the networking behavior will be change for following factors: Lost link, IP address

changed …etc (3)It must support packet sniffer function. So that people can decode ethernet packet to know the behavior of networking protocol stack.(4) It must support the capability for rapid analysis system behavior and system performance.

The subject of the thesis is to design a networking analysis tool,which called NBA (Networking Behavior Analysys Tool) to integrated above functions in order to measure the delay time of different network layers. By using this tool, we can find out the time delay of packet transmitting and receiving and the overall performance of network protocol stack.

At last, the thesis intends to test the process capibility of the software to make sure the kernel code changes won't result in negative efficiency, and influence the reliability and validity of the test result.

# 誌　謝

本論文得以順利完成，首先要感謝我的指導教授—曾建超 博士在這二年來所給予我的指導及幫忙。一步一步引領我進入到了這一個領域來參與研究與學習。同時也要向我的口試委員：嚴力行博士、張弘鑫 博士、曹孝櫟 博士致上謝意，感謝他們能在百忙中撥空審查我的論文並給予寶貴的意見。另外，也非常感謝蘇俊彰學長能夠抽空幫我校閱論文。

最後，感謝我的太太及家人的犧牲與奉獻，讓我能夠在工作之餘，還能放心的專注於論文研究，也才可以順利的完成此篇論文。

# 目　錄

# 圖目錄

# 表目錄

# 第一章 緒論

## 1.1 研究動機

隨著網際網路的快速掘起，使得多媒體行動式數位通訊裝置成為市場主流。綜觀目前市面上能夠找的到用來評比網路通訊裝置的工具並不多，因此本論文主要的目標，在於設計一套網路通訊裝置的評比工具，以用來分析與衡量通訊裝置的處理效能。更可用於協助系統開發者快速地尋找通訊裝置的系統效能瓶頸。

本論文主要包含有四大部份：(1)核心網路追蹤工具之設計與系統架構， 藉由在 Linux Kernel 安插探針(probe)的動作，快速地顯示出網路通訊協定堆疊內的核心函式。(2)網路事件通知機制，透過 Netlink 來獲取網路通訊協定堆疊內的重要資訊，如:網卡的 Register、Unregister、IP address change、routing table change、MAC address change…等等(3)封包擷取工具，對網路卡上的 IEEE 802.3/802.11 封包作擷取，能夠直接解譯出所感興趣的網路通訊協定。(4)網路效能分析工具，具備從系統應用層乃至於網路層的效能分析。

其中部分(1)由許凱程學長於[2]的論文中已實作完畢，部分(2)由曹敏峰學長於[3]的論文中實作完畢。本論文除了針對(1)、(2)兩部分加以改良之外，並完成了(3)、(4)兩部分。目標便是能夠整合此四大方向，將現有的工具及文獻做有系統的整合，以發展出一套完整的評比工具。

## 1.2 研究目標

本論文的研究目標在於發展一個基於 Linux 作業系統上的網路通訊協定堆疊的效能評比工具，讓嵌入式裝置發展者可以以此工具來分析裝置本身的核心網路行為，包括了：

封包在核心內的傳輸/接收流程

封包傳送時的延遲

封包接收時的延遲

IP 位址的取得

Routing Table 的動態改變

Protocol 在傳送與接收時的延遲

## 1.3 論文內容

關於本篇論文的章節編排與內容簡介如下：

第一章：緒論，介紹本篇論文的研究動機以及希望達成的目標。

第二章：背景知識介紹，介紹本篇論文所要探討主題以及所應用到的先備知識，包括作業系統、核心模組以及檔案系統。

第三章：相關研究，簡介與本篇論文主題相關的文獻及工具，包括了封包過濾工具以及 Linux 網路事件擷取工具。

第四章：Linux 核心網路追蹤工具之設計與系統架構，說明本論文的系統架構與各個軟體元件。

第五章：Linux 核心網路通訊協定分析工具之設計與系統架構，說明本論文通訊協定分析工具的實作細節，以及針對 Linux 內部所做的修改部分。

第六章：Linux 核心網路追蹤與分析工具之實作，說明本論文所開發的系統平台之實作細節。

第七章：實驗結果與貢獻，介紹本工具效能評估的實驗結果分析。

第八章：結論與未來工作，總結本篇論文的結果，以及未來可繼續研究的方向。

# 第二章 背景知識介紹

## 2.1 核心程式與使用者程式間的行程通訊機制

Linux 作業系統透過權限做管理，將作業系統化分為 kernel-space 與 user space，這兩者間最大的差異即是 user-space 程式無法直接存取到 kernel-space 的記憶體[1]。主要的原因就在於作業系統對運行於 kernel-space 的程式做保護。簡而言之，kernel-space 的程式具有較高的權限，而 user-space 的程式則具有較低的權限。但是，兩者之間還是存在有適當的通訊機制可以用來做溝通與管理。比較常見的作法有：

IOCTL，是用來實現應用程式用和驅動程式溝通的函數，讓應用程式可以對某個裝置下命令，一般而言在 Linux 環境下，對裝置驅動程式的存取與對檔案做存取的方法是一樣的，相關的應用程式範例如下：

```
static int myioctl (struct inode *inode, struct file *file, unsigned int cmd,
                    ulong arg)
{
    switch (cmd)
    {
       case 1:
            printk ("Triger command 1.");    /* Command value = 1 */
    }
}

static struct file_operations my_fops =
{
    owner: THIS_MODULE, ioctl : myioctl,
};

static devfs_handle_t devfs_handle;

static int __init my_init(void)
{
    int ret;

    ret = register_chrdev(“my_ioctl”, &my_fops);
    return 0;
}
```

Table 2-1 核心驅動程式使用 IOCTL 範例

首先，必須在驅動程式當中對檔案操作元的 ioctl 指標做註冊的動作，之後才

能在使用者程式當中，調用驅動程式中的 ioctl 函數。

```
int main (int argc, char *argv[])
{
        int fd;

        fd = open("/dev/debug", O_RDONLY);
        ioctl(fd, 1, num);
        close(fd);
        return 0;
}
```

Table 2-2 使用者程式調用 IOCTL 範例

/PROC 檔案系統，是用來實現應用程式和驅動程式溝通的另外一種方法，讓應用程式可以對某個裝置下命令，相關的應用程式範例如下：

```
#include <linux/proc_fs.h>

static struct proc_dir_entry *proc_ func = NULL;

static int read_func_state (char* page, char **start, off_t offset, int
                            count,int *eof, void *data)
{
     int len = 0;

     printk (“This is linux kernel.\n”);
     return len;
}

static int __init my_init(void)
{
     proc_ func = create_proc_entry("mydebug", 0644,NULL);
     if    (proc_func)
     {
        proc_func->read_proc = my_read;
     }
}
```

Table 2-3 核心驅動程式使用 /PROC 範例

```
# cat /proc/mydebug
# This is linux kernel.
#
#
```

Table 2-4 使用者程式呼叫 /PROC 範例

IOCTL 或 /PROC 檔案系統，都可以用來取得核心記憶體內的資料。但此兩種行程通訊方式都同時存在著效能不佳的問題。主要的原因即是資料從核心記憶體拷貝到使用者程式中的虛擬記憶體分頁時，必須透過另外一個共享記憶體

做緩存，這個多餘的拷貝動作會浪廢許多 CPU 資源。

MMAP (Memory MAP)記憶體映射機制，是 Linux 上一種非常有趣的功能，主要是透過「shared file」來做訊息的資料傳遞，在程式當中將檔案 mapping 到自己的記憶體空間（process address space）時，mmap system call 便會建立相對應的 VMA[4] 區段；觀念上來說，只要透過 mmap system call 將檔案 mapping 到記憶體，便會產生對應的 VMA 區段。VMA (Virtal Memory Area) 主要是為了讓 Linux 能夠有效率地管理記憶體，而從 paging 系統發展而來。VMA 可以讓 Linux 以 process 的角度來管理虛擬位址空間，有可能執行記憶體需求超過可用實體記憶體的應用程式。而將 shared file mapping 到 process address space 的 system call 為 'mmap()'。

此種方法也可被應用來提供使用者程式直接存取裝置記憶體(device memory)的能力，讓行程可以共用一個程式庫或記憶體區塊。以下是其中一個簡單的範例：

```
void main (void)
{
    int RxCount = 0, TxCount = 0, mfd = -1;
    size_t size = BUFFERSIZE;    /* 2MB */
    char * mptr;

    mfd = open("/dev/bridge", O_RDWR | O_SYNC);
    mptr = mmap(0, size, PROT_READ | PROT_WRITE, MAP_FILE |
                    MAP_SHARED, mfd, 0);
    memcpy (&RxCount,mptr,2);
    memcpy (&TxCount,mptr+2,2);
}
```

Table 2-5 使用者程式使用 MMAP 範例

```
struct file_operations my_fops = {
.mmap = my_mmap,
};
static int my_mmap(struct file * filp, struct vm_area_struct * vma)
{
   int ret = remap_pfn_range(vma, vma->vm_start,
            virt_to_phys((void*)((ulong) buffer_area)) >> PAGE_SHIFT,
            vma->vm  end-vma->vm  start, PAGE  SHARED);
```

Table 2-6 註冊 MMAP 函數至檔案操作元

使用 MMAP，除了在使用者程式中必須呼叫記憶體映射的 API 之外，在裝置驅動程式上也必須對檔案操作結構中的 MMAP 指標做註冊的動作。

```
#include <sys/mman.h>
void * mmap (void *addr, size_t len, int prot, int flags, int fd, off_t offset);
int munmap(void *start, size_t length);
```

Table 2-7 MMAP 函數原型

addr 參數提供給 Linux Kernel 最佳的檔案對應位址，用以告訴 Linux Kernel 從那個起始位址開始映射檔案。一般情況都是填 '0'，此函數將回傳指向映射區域的真實起始位址。

len 參數表示欲映射多少 Byte 的記憶體長度。

port 參數描述用何種方式來對映射區域做記憶體保護，MMAP 提供如下幾種記憶體保護機制：

PROT_EXEC – 分頁記憶體可被執行

PROT_READ - 分頁記憶體可被讀取。

PROT_WRITE - 分頁記憶體可被寫入。

PROT_NONE - 分頁記憶體不可被存取。

flag 參數用來描述檔案對應的類型，MMAP 提供如下幾種記憶體映射方式：

MAP_FIXED - 強制 mmap( ) 要遵守 addr 的設定，如果無法對應至該位

址，則呼叫失敗。

MAP_SHARED – 當指定為以 MAP_SHARED 作映射時，則使用者程式對映射記憶體的改動將會使的被映射到的實體記憶體內容，一併被修改。

MAP_PRIVATE -當指定為以 MAP_ PRIVATE 作映射時，程式間的對應狀態無法分享，該檔案會對應至一個副本，使用者程式對映射記憶體的改動將不會更動到實體記憶體內容，

fd 參數指到欲映射的檔案。

offset 參數一般設為 0，表示從檔頭開始映射。

當使用者程式將實體記憶體 mapping 到虛擬記憶體空間（Process address space）時，mmap system call 便會建立起相對應的 VMA 區段。

```
[root@localhost ~]# ps -aux|grep prog
Warning: bad syntax, perhaps a bogus '-'? See /usr/share/doc/procps-3.2.7/FAQ
root      3956  0.0  0.0  18872   840 pts/0    Sl+  17:51   0:00 ./prog
root      4014  0.0  0.0   5208   664 pts/3    S+   17:52   0:00 grep prog
[root@localhost ~]# cat /proc/3956/maps
08048000-08058000 r-xp 00000000 08:02 7097997    /home/rober/prog/prog
08058000-08059000 rw-p 0000f000 08:02 7097997    /home/rober/prog/prog
08059000-080b9000 rw-p 00000000 00:00 0
09444000-09465000 rw-p 00000000 00:00 0          [heap]
461a7000-461aa000 rw-p 00000000 00:00 0
461d7000-461da000 r-xp 00000000 08:02 7458937    /lib/libdl-2.7.so
461da000-461db000 r--p 00002000 08:02 7458937    /lib/libdl-2.7.so
461db000-461dc000 rw-p 00003000 08:02 7458937    /lib/libdl-2.7.so
461de000-461f3000 r-xp 00000000 08:02 7458939    /lib/libpthread-2.7.so
461f3000-461f4000 r--p 00014000 08:02 7458939    /lib/libpthread-2.7.so
461f4000-461f5000 rw-p 00015000 08:02 7458939    /lib/libpthread-2.7.so
461f5000-461f7000 rw-p 00000000 00:00 0
b6d61000-b6d62000 ---p 00000000 00:00 0
b6d62000-b7562000 rw-p 00000000 00:00 0
b7562000-b7763000 rw-s 11800000 00:0f 15607      /dev/bridge
b7763000-b7764000 ---p 00000000 00:00 0
```

Figure 2-1 觀察使用者程式內的虛擬記憶體區段配置

採用 MMAP 做行程通訊的一個顯而易見的好處是效率高，因為使用者程式可以直接讀寫核心程式所配置的記憶體，而不需要做任何資料的拷貝。對於像 IOCTL 或 /PROC 檔案系統等方式在做行程通訊時，皆須將資料寫入共享記憶體，然後才能將資料從共享記憶體讀出或寫入目的記憶體，這樣的拷貝動作會影響到系統效能，所以 MMAP 會比較滿足本論文的需求。

Netlink Socket 訊息交換機制，Netlink 是 Linux 作業系統上一種特有的使用者

空間與核心空間的訊息交換機制。類似於 BSD 中的 AF_RUTE，目前的 Linux Kernel 使用 netlink 的範圍很廣，包括 Routing，Firewall，Netfilter，IPSec …等等，都有使用 netlink 來和使用者程式做互動。Netlink 是一種在 kernel space 與 user space 之間進行雙向資料傳輸非常好的方式，使用者只要使用標準的 socket API 就可以使用 netlink 提供的強大功能，Kernel 則需要使用專門的內核 API 來使用 netlink。

相較於 IOCTL 以及 /PROC 而言，netlink 具備的優點如下：

(1)可以發送 Broadcast packet 給不同的使用者行程。

(2)Kernel 可以使用 netlink 首先發起對話，但 IOCTL 以及/PROC 只能由使用者程式發起調用。

(3)使用方便，用戶僅需要在 include/linux/netlink.h 中增加一個新的 netlink 協定定義即可，核心程式和用戶程式就可以立即通過 socket API 使用該 netlink 協定類型進行資料交換。

## 2.2 Libpcap 封包截取函數庫

libpcap 是 unix/linux/Solaris/BSD/Mac OS X/HP-UX 平臺下的封包截取函式庫，libpcap 是由 Lawrence Berkeley National Laboratory (LBNL) 所開發而成的封包截取函數庫。目前市面上大多數網路監控軟體都是以它為基礎。如：TCPDUMP、Ethereal、Wireshark[5]、Sniffer[6] …等等。主要是因為 Libpcap 可以在絕大多數的 Linux/Unix 平臺下工作，簡單、容易使用也是它之所以被廣泛採用的原因。

本論文即是使用此函數庫，來對進出 Linux 作業系統的 Ethernet 封包作攔截與過濾，再搭配上客制化的 Protocol Analyzer，即可對 IEEE 802.3/802.11 的封包內容做解析，以提供使用者從 Layer 2 至 Layer 7 重要的參考資訊。

Libpcap 0.9.8 目前版本提供了 20 多個 API 函數，利用這些 API 函數即可完成本論文中封包截取工具所需要的網路封包監聽功能。

## 2.3 PTHREAD 執行緒

在 Linux 作業系統上，執行緒被用來解決多工與獨立運作的問題。在目前的 Linux 版本當中，最常被使用的執行緒函數庫為符合 POSIX 1003.1c 執行緒標準的 Pthreads 執行緒程式庫。POSIX 1003.1c 執行緒標準是由 IEEE 作業系統技術委員會於 1995 年所訂定而成的，在這個程式庫中的函數都是以 pthread 字串開頭。如：pthread_create 用來創建新的執行緒，pthread_exit 用來退出執行緒。其函數原型與說明如下：

```
#include <pthread.h>

int pthread_create (pthread_t *restrict thread, const pthread_attr_t *restrict
                    attr, void *(*start_routine)(void*), void *restrict arg);
```

Table 2-8 pthread_create 函數原型

在 Linux 系統中，在建立新的執行緒時必須呼叫 pthread_create API。pthread_create 總共有四個參數。第一個參數為 thread，thread 參數為 pthread_t 型別物件的參照。當成功完成 pthread_create 呼叫之後，*thread 將會參照一個唯一的整數執行緒 ID 數值。如果此值被設定為 NULL，當函數返回時並不會回傳執行緒 ID。第二個參數為*attr，它參照到 pthread 動態屬性結構。pthread 結構屬性如下：

```
Typedef struct __pthread_attr_s {
        int __detastate;
        int __schepolicy;
        struct __sched_param __schedparam;
        int __inheritsched;
        int __scope;
        size_t __quardsize;
        int __stackaddr_set;
        void *__stackaddr;
        size_t *__stacksize;
}pthread_attr_t;
```

Table 2-9 執行緒的屬性

一般而言，第二個參數都是填入 NULL，表示使用 pthread 的執行緒屬性初始值，在某些情況下，如果希望改變執行緒屬性的設定，這時後可以使用

```
#include <pthread.h>

int pthread_attr_init (pthread_attr_t * attr);
```

pthread_attr_init 函數來作屬性變更，其函數的原型如下：

Table 2-10 pthread_attr_init 函數

在此須注意的是，新建立的執行緒與原有的行程是共享行程記憶體的。具體的範例程式如下：

```
#include <pthread.h>

void * PacketProcess(void)
{
    /* Put your packet process handler in here */
}

int main (void)
{
    int ret;
    pthread_t    thread_sniffer;
    ret = pthread_create(&thread_sniffer, NULL, (void *)&PacketProcess,
                         NULL);
}
```

Table 2-11 pthread 使用範例

最後，程式碼在做編譯時也必須連結到 pthread 函數庫。

# 第三章　相關研究

本章將會介紹本論文的相關研究，包括了(1)網路通訊協定堆疊之分析 - 藉由在 Linux Kernel 安插探針(probe)的動作，快速地顯示出網路通訊協定堆疊內的核心函式。(2)網路事件通知機制 - 透過 Netlink 來獲取網路通訊協定堆疊內的重要資訊，如:網卡的 Register、Unregister、IP address changed、routing table changed、MAC address changed…等等(3)封包擷取工具。

## 3.1 網路通訊協定堆疊之分析

此部分首先將介紹本實驗室之前的一些相關研究，包括了曹敏峰之「Linux 網路通訊協定堆疊之高效率動態的指令嵌入平台之設計與實作」

該論文的主要目標便是將 Linux 網路通訊協定堆疊中每個重要的函式都加入探針，以用來 Trace 封包收送時會經過那些核心函數。對於「探針」的處理，主要是在網路通訊協定堆疊的核心函數中，加入一個函式指標呼叫，讓 Linux Kernel 在對 Packet 做處理時也能呼叫客制化的函數，已用來記錄當 Packet 經過核心函數時的系統時間。相關的範例如下：

```
/* linux\net\core\dev.c */
int (*dev_queue_xmit_Func_Start)(struct sk_buff *skb)=0;
EXPORT_SYMBOL(dev_queue_xmit_Func_Start);
int dev_queue_xmit(struct sk_buff *skb)
{
        struct net_device *dev = skb->dev;
        struct Qdisc *q;
        int rc = -ENOMEM;

        if    (dev_queue_xmit_Func_Start)
           dev_queue_xmit_Func_Start (skb);
```

Table 3-1 網路通訊協定堆疊的探針處理

對於探針的處理部份，該論文提出了對 sk_buff 結構做修改的方法，以對進出核心函式的封包加以處理，透過增加額外 TAG 的方式到 sk_buff 結構中，來指出封包目前所在的核心函式為何？以及替封包標上一個 Sequence number，以方便與 Ethereal 資料做對照。

所有的日誌資料都是儲存在驅動程式裡面事先配置好的陣列當中，陣列的容量也會受到 Linux Kernel 的基本限制 (4MB)，使用上並不能儲存太多的日誌。本論文因此提出了不同的日誌處理方式，並對日誌檔的備份方法加以改良，以達到讓使用者可以長時間監控核心函式的目的。

## 3.2 網路事件通知機制

此部分首先將介紹本實驗室之前的一些相關研究，包括了許凱程之「支援具網路感知應用程式的中介軟體之設計與實作」。

該論文的主要目標在於設計和實作出一個軟體發展架構，讓程式開發人員在此架構下可以方便的開發具網路感知的應用程式。其中對於網路事件的通知機制，主要的設計方法，即是在對 Linux 的核心函數做修改，並將重要的網路介面訊息做分類，以增加 Linux Kernel 對網路介面訊息更好的支援。以下列出了 Kernel Patch 前與 Kernel Patch 後的源始碼來做參考。

```
static int rtnetlink_event(struct notifier_block
      *this, ulong event, void *ptr)
{
       struct net_device *dev = ptr;
       switch (event) {
       case NETDEV_UNREGISTER:
              rtmsg_ifinfo(RTM_DELLINK, dev, ~0U);
              break;
       case NETDEV_REGISTER:
              rtmsg_ifinfo(RTM_NEWLINK, dev, ~0U);
              break;
       case NETDEV_UP:
       case NETDEV_DOWN:
              rtmsg_ifinfo(RTM_NEWLINK, dev, IFF_UP|
                            IFF_RUNNING);
              break;
       case NETDEV_CHANGE:
       case NETDEV_GOING_DOWN:
              break;
       default:
              rtmsg_ifinfo(RTM_NEWLINK, dev, 0);
              break;
       }
       return NOTIFY_DONE;
}
```

Table 3-2 Linux Kernel Patch 前

```
static int rtnetlink_event(struct notifier_block *this, ulong event, void *ptr)
{
        struct net_device *dev = ptr;
        switch (event) {
        case NETDEV_UNREGISTER:
                rtmsg_ifinfo(RTM_DELLINK, dev, ~0U);
                win_rtmsg_ifinfo(event, RTM_SYSTEM_EVENT, dev, ~0U);
                break;
        case NETDEV_REGISTER:
                rtmsg_ifinfo(RTM_NEWLINK, dev, ~0U);
                win_rtmsg_ifinfo(event, RTM_SYSTEM_EVENT, dev, ~0U);
                break;
        case NETDEV_UP:
        case NETDEV_DOWN:
                rtmsg_ifinfo(RTM_NEWLINK, dev, IFF_UP|IFF_RUNNING);
                win_rtmsg_ifinfo(event, RTM_DRIVER_EVENT, dev, ~0U);
                break;
        case NETDEV_CHANGENAME:
                rtmsg_ifinfo(RTM_NEWLINK, dev, 0);
                win_rtmsg_ifinfo(event, RTM_SYSTEM_EVENT, dev, ~0U);
                break;
```

Table 3-3 Linux Kernel Patch 後

此部份對 Linux 源始碼的改動，主要在於提供底層的網路卡事件更好的支援，讓軟體開發人員不用去了解系統內部的底層運作機制，透過事件的觸發，來得到所需的核心訊息以減少寫程式出錯的機會。

本論文則是使用不同的觀點來處理核心的網路事件，因為目前的 Linux Kernel 雖然沒有針對單一的事件各別的廣播，但是在事件狀態改變時，還是會產生 RTM_NEWLINK 事件來通知使用者程式。所以是可以在不修改 Linux Kernel 源碼的情況底下了解到到真正事件的發生原因。

## 3.3 封包擷取工具

Libpcap 是 unix/linux/Solaris/BSD/Mac OS X/HP-UX 平臺下的封包截取函式庫，目前市面上大多數網路監控軟體都是以它為基礎來做開發。Libpcap 在效能上原先存在有短封包效能不好的問題，因為大量的短封包傳輸會造就更頻繁的資料搬移發生，使得效能受到極大的影響。近年來由於運用了新的處理技術，

原有的問題已獲得了實質上的改善。主要的改進方法在於使用 MMAP 將網卡上的 ring buffer 映射到使用者程式中做處理，而不需要經過額外的搬移與拷貝動作，來達到將 Ethernet Packet Buffer 從 kernel space 到 user space 來到「零拷貝」的目標了。

其動作原理詳述如下。一般而言，網路卡介面皆具有 Descriptor 暫存器可以用來指向存放網路封包的網路封包緩衝區記憶體位址，驅動程式即是事先先將記憶體指標存放在此結構當中。每當網卡接收中斷發生時，驅動程式即可在中斷控制程式內去驅動 DMA 將網卡上的封包搬進 Descriptor 所指向的記憶體位址。簡而言之，如果可以事先配置好固定位址的記憶體供網卡使用，使用者程式當然也可以透過 MMAP 將此塊記憶體映射到使用者程式去做處理。

# 第四章 Linux 核心網路追蹤工具之設計與系統架構

本章將介紹論文中 Linux 核心網路通訊協定堆疊追蹤工具(Linux kernel network protocol stack tracing tool)的設計架構。首先會先對此工具的設計概念及目標做介紹，之後在對整個系統架構詳加說明。

## 4.1 設計概念及目標

隨著經濟的快速起飛，網路產品供應商在產品開發上講求快速整合，以因應市場上的迫切需求，但產品往往就在急就章的情況底下，往往不能兼顧軟體的品質，因此本論文的主要目標在於開發出一套高效率的網路核心評比工具。

因此，如何方便的取得網路核心內部的重要資訊來做分析，是本論文的主要目標。本工具不僅要具備可以分析有線網路裝置的行為能力，還要能具備分析無線網路裝置的行為能力。從 Linux 網路協議與網路行為來做追縱，並將此一程序透明化，以幫助程式開發者驗證產品的基本功能與探查通訊裝置的系統效能瓶頸。

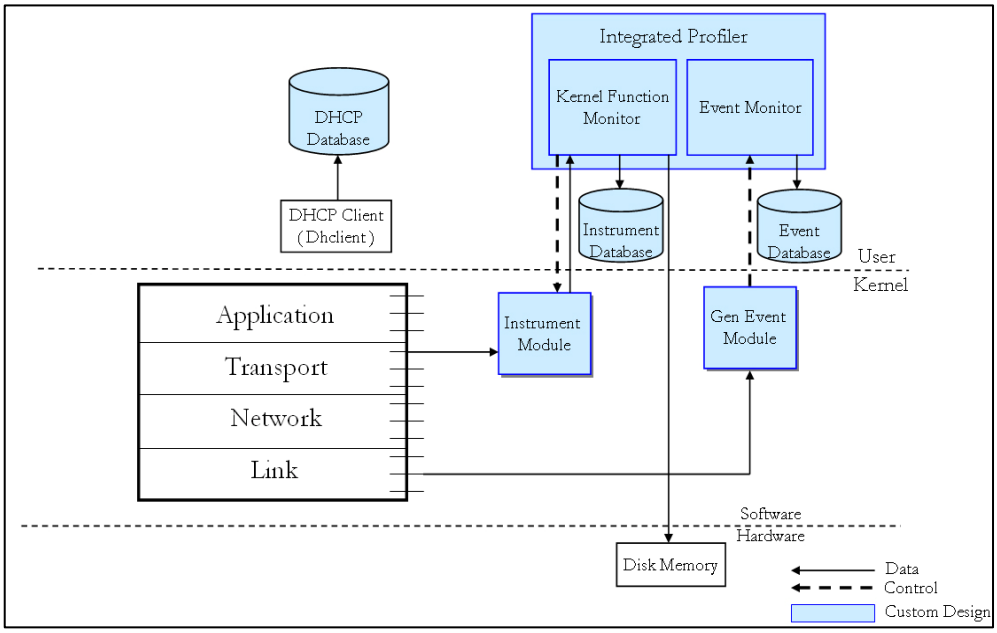



Figure 4-1 系統架構圖

## 4.2 系統架構

圖中主要有三大部分：(1)藍色部分為本論文新增的主要元件，包含有 Integrated Analyzer、Log Database、Integrated Profiler、Instrument Module (2) 紅色部份為 Open Source code，包括有 DHCP Protocol，Libpcap (3) 黑色空心部分為原始作業系統所擁有的部分，包括了 TCP/IP Stack、Device Driver。簡單說明此三大部份：

Integrated Profiler

此單元包含有(1)Packet Sniffer，使用 libpcap 函數庫來截取進出 Linux 系統的網路封包，並將 Sniffer 資料存入 RAW.DAT 檔案裡。(2)Event Monitor，透過 Netlink 訊息交換機制來監控 Kernel 中的 Router Event 與網路介面 Event。(3)Kernel Function Monitor，持續的 Polling Instrument Module 內的 Log Buffer，並將資訊備份到磁碟檔案 KLOG.DAT 中。

DHCP Client

Fedora Linux[7]使用 dhclient 來處理網路介面卡上 IP 的獲得。一般來說，只要有開啟 NetworkManager 與 NetworkManagerDispatcher 這兩個服務，Linux 就會自動的進行IP取得的動作。但前提是使用者必須把網路介面設定成使用 DHCP 來取得 IP Address。

本論文會對此程式做修改，當 DHCP state machine 狀態改變時，隨即將當時的狀態資訊與系統時間寫入系統日誌檔案中/var/dhclient.log。

Kernel Patch

透過對 Linux 核心中的重要網路函數做 Patch，來 Trace IP 封包進出 Linux 核心所經過的系統函數。透過插入 Probe 函數到 Linux Kernel 中，來達到呼叫客制化函數的目的。

Libpcap

Integrated Profiler 中的 Packet Sniffer 即是用此函數庫來監聽網卡上的封包。

## 4.3 系統說明

### 4.3.1 Instrument Module

在曹敏峰[3]的論文當中，提出了一個很基礎的概念，稱作 Naive approach，裡面包含有三個部分：

核心補釘(Kernel patching)-透過對 Linux 核心中的重要函數做 Patch，來 Trace 封包進出 Linux 核心的傳送路徑與與接收路徑。

記錄(Logging)，將 Trace 過程中所產生的重要資訊存放在系統記憶體內。

分析(Analysis)，從系統記憶體內，讀出 Log 資訊來做統計分析。

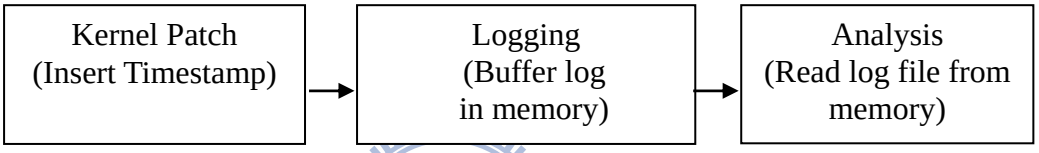



Figure 4-2 Naive approach

本論文對此三部份有新的詮釋與應用，相關差異如下：

在曹敏峰於[3]的論文當中，透過對 sock、sk_buff 資料結構新增兩個額外的欄位 probe_flag 與 probe_seq 來對進入 Linux 的網路封包做加 TAG 的動作。probe_flag 用來指出封包是來自於傳送路徑或者是接收路徑，probe_seq 用來對封包加上一個序號，所有的日誌都是儲存在預先配置的字元陣列當中。其大小為 5000000 Byte。本論文對日誌結構做了些許修正，不僅可以達到相同的要求，對於後續將日誌檔與 Packet Sniffer 的資料整合會有更大的幫助。原始日誌格式如下所示：

```
[1214127586][593195] netif_rx_start: seq[1]
[1214127586][593207] ip_rcv_start: seq[1] id[0]
[1214127586][593215] icmp_rcv_start: seq[1] id[0]
[1214127586][593249] netif_rx_start: seq[2]
[1214127586][593256] ip_rcv_start: seq[2] id[3975]
[1214127586][593264] icmp_rcv_start: seq[2] id[3975]
[1214127587][592198] netif  rx  start: seq[3]
```

Table 4-1 日誌檔案範例

每一筆日誌紀錄的格式為： [秒] [微秒] [函數名稱] [序號] [Identification]

在曹敏峰於[3]的論文當中，使用本文格式將資料寫入到記憶體中做暫存，也因為記憶體大小受到限制，所以資料寫滿後會從陣列起始位址繼續覆蓋下去。

本論文將日誌的格式做了如下的修正，除了時間部份外(timeval)，其餘部份皆為新增欄位。

```
struct _RX_DB_
{
    unsigned char signature;       /* 0xAA for RX*/
    unsigned char header[14];
    unsigned char sip[4];
    unsigned char dip[4];
    int len;
    int ident;
    struct timeval netif_rx_Func_Start;
    struct timeval net_rx_action_Func_Start;
    …
```

Table 4-2 新版日誌格式

header 欄位用來存放 Ethernet 封包的表頭部份，sip 與 dip 欄位則用來存放 IP 表頭中的來源 IP 位址與目的 IP 位址。Len 表示封包的長度，ident 欄位則是用來存放 IP 表頭中的 Identification 值，其餘欄位則用來儲存封包所經過各個重要核心函數時的系統時間。透過增加 Ethernet 表頭到日誌格式中的主要目的在於補足廣播封包或 Multicast 封包，其 Identification 欄位值為零，會導致該筆記錄在封包截取器資料庫中找不到對相映資料的問題。舉例來說，一般的 ARP 封包，其 Identification 欄位永遠為零，此時如果有 Layer 2 封包的表頭部份做參考，就可以輕易的在封包截取器資料庫中找到相對映的該筆資料了。Multicast 封包也會遇到同樣的問題，透過 Layer 2 封包的表頭部份及 sip 與 dip，即可於封包截取器資料庫中找到相對應的該筆封包記錄。

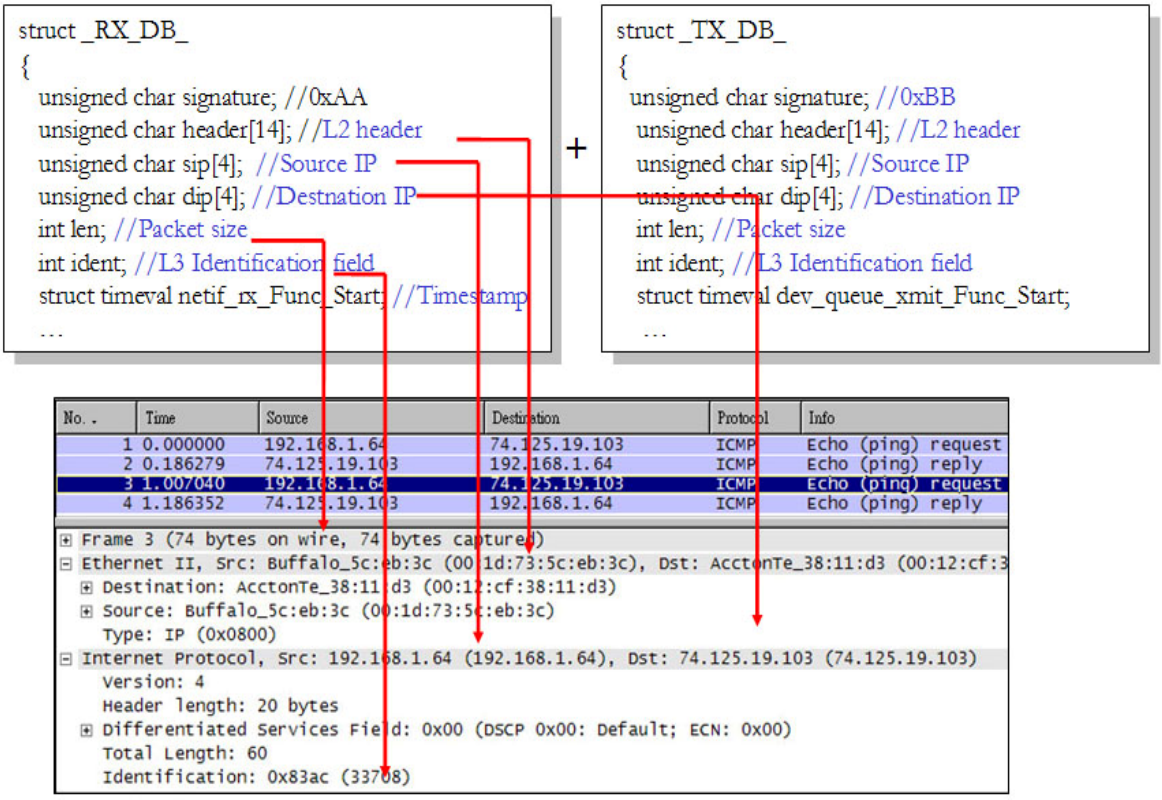



Figure 4-3 傳送與接收日誌格式與 IP 封包欄位對照表

### 4.3.2 Kernel function monitor

本論文運用 MMAP 記憶體映射機制[8] [9]，將 Instrument Module 內的日誌緩衝區映射到使用者程式中的虛擬記憶體分頁。讓使用者程式能夠直接存取到此塊記憶體，以加速日誌檔案備份的速度。在還沒有使用此種方法時，雖然還是可以透過 IOCTL 或/PROC 檔案系統來取得核心記憶體內的資料，但是這兩種行程通訊方式都同時存在著效能不彰的問題。最主要原因即是資料從核心記憶體拷貝到使用者程式中的記憶體時，必須透過另一個共享記憶體做緩存，這個多餘的拷貝動作會浪廢許多 CPU 資源，不僅僅會造成封包的漏失，同時也會影響到系統的處理效能。

使用 MMAP 的做法，雖然可以加速資料備份時的速度，但相對的也不是沒缺點的。缺點就在於資料量太大，存取效能會受到磁碟的存取速度與儲存演算法等因素的影響，而影響到系統的整體效能。當然，這個問題是可以透過對演算法的改進與昇級硬體設備等方法來獲的改善。

### 4.3.3 Event monitor

Event Monitor 使用 Linux 上的 Netlink 來監控核心網路事件的發生，此程式能夠監控的核心訊息如下所示：

NETDEV_REGISTER - 網路裝置界面被新增至 Linux 系統中。

NETDEV_UNREGISTER - 網路裝置界面從 Linux 系統中被移除。

NETDEV_UP - 某個網路裝置界面被啟動了。

NETDEV_DOWN - 某個網路裝置界面被關閉了。

NETDEV_CHANGEMTU - 某個網路裝置界面的 MTU 大小被改變了。

NETDEV_CHANGEADR - 某個網路裝置界面的 MAC Address 被改變了。

IP CHANGE - 某個網路裝置界面的 IP 位址被改變了。

DEFAULT GATEWAY CHANGE - 預設閘道被改變了。

ROUTING TABLE CHANGE - 路由表資料被改變了。

以 Linux Kernel 2.6.30 來說，本身就已經支援下列幾種網路介面事件：NETDEV_UNREGISTER 、 NETDEV_REGISTER 、 NETDEV_UP 、 NETDEV_DOWN、NETDEV_CHANGE、NETDEV_GOING_DOWN、DEFAULT GATEWAY CHANGE、ROUTING TABLE CHANGE、IP CHANGE 等。本論文增加了偵測 MTU 與 MAC 位址改變的功能。

這兩項新增加的功能，在實作上並不困難，主要就在於 Linux 本身已經能夠主動地送出 RTM_NEWLINK 訊息到使用者程式。因此只要在使用者程式中收到此訊息時將網路介面上的 MTU 與 MAC address 讀出來即可判斷出這兩個屬性值是否已經改變。

### 4.3.4 Packet Sniffer

本單元最主要的功能在於截取所有進出 Linux 作業系統的網路封包，這部份可以透過 Libpcap 函數庫來完成。以系統觀點的角度來看，最主要的目的在於做到跟 Instrument Module 的 Kernel 日誌做整合，用以提供從 Layer 2 至 Layer 7，

再到 Kernel 層級的完整資訊。

因此為了達到高處理效能的設計目的，本單元會運行在單一的執行緒上，並透過讀入外部設定檔，來調整欲監控的網路介面以及最大的封包截取長度。此單元執行時的處理效能，同時將會受到硬體性能好壞所影響。

### 4.3.5 DHCP Protocol

Dynamic Host Configuration Protocol (DHCP)是一個網路應用程式，被網路裝置用來做 IP 位址的取得以達到連上網際網路的目的。DHCP 于 1993 年 10 月成為標準，目前的定義可以在 RFC2131 中找到。

一般來說，Linux 作業系統會依據網路卡的設定檔，來決定以何種方式來取得 IP 位址，此設定檔可以在檔案系統上的/etc/sysconfig/networking/devices 目錄底下被找到。其檔案的內容如下所示：

```
# Intel Corporation 82566MC Gigabit Network Connection
DEVICE=eth0
BOOTPROTO=dhcp
HWADDR=00:1C:25:93:35:39
ONBOOT=yes
TYPE=Ethernet
```

Table 4-3 網路介面設定檔範例

如上所示，此設定檔說明了 eth0 網路介面的 IP 取得方式為使用 DHCP Protocol。因此，Linux 系統會在開機階段開啟這檔案，並檢查 BOOTPROTO 欄位值來決定 IP 取得的方式，是要用 DHCP Protocol 還是設定成固定 IP。這是 Linux 系統在啟動時的行為。之後，如果系統有開啟 NetworkManager 或是 NetworkManagerDispatcher 這兩個服務，這兩個服務程式會持續監控允許被其所管理的網路介面。當網路卡斷線時，此程式會嘗試進行 IP 取得的動作。舉例來說，當 Wireless 界面與 AP 失聯時，此程式會找出符合連線需求的下一個網路節點，並呼叫 DHCP Protocol 來進行 IP 位址的取得動作。

從 DHCP Protocol 的定義之中，我們知道從 DHCP 伺服器獲得 IP 資訊的 4 個階段中，DHCP 用戶端會出現有 4 種封包（DHCP DISCOVERY，DHCP OFFER，DHCP REQUEST，DHCP ACK）。

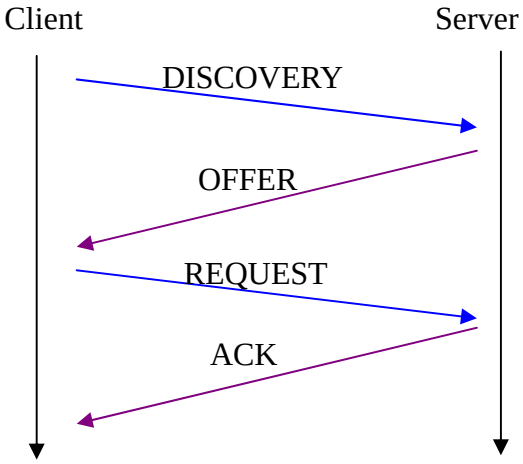



Figure 4-4 DHCP Client 運作流程圖

這四個階段所經過的時間會影響到 IP 取得的速度快慢，對系統效能來說非常重要，本論文即透過修改此 DHCP Client 源始碼來監控此操作流程。並會在 DHCP 原始碼狀態改變時，將事件狀態與系統時間寫入 DHCP 日誌檔中。

# 第五章 Linux 核心網路通訊協定分析工具之設計與系統架構

## 5.1 開發環境需求

本章將介紹本論文中 Linux 核心網路通訊協定分析工具(Linux kernel network protocol stack analyzer)的設計架構。首先會先對此工具的設計概念及目標做介紹，之後再對整個系統架構詳加說明。

## 5.1 設計概念及目標

目前，對於網路通訊裝置系統核心的網路通訊協定處理程序(processes)延遲以及核心事件(kernel event)，並沒有一套完整的分析工具能夠協助開發者了解封包在進出嵌入式系統時所造成的傳輸延遲、反應時間以及封包遺漏的發生原因，這些因素都會對系統運作效能造成重大影響。

有鍵於此，的確有必要建立一個工具程式來檢視網路通訊裝置的網路核心行為，進而對網路通訊裝置做評量。不僅可以用來找出系統效能的瓶頸，相信透過分析核心網路堆疊內的行為，也可以對不好的地方來加以改善，對系統效能的提昇將會有很大的助益。

## 5.2 系統架構

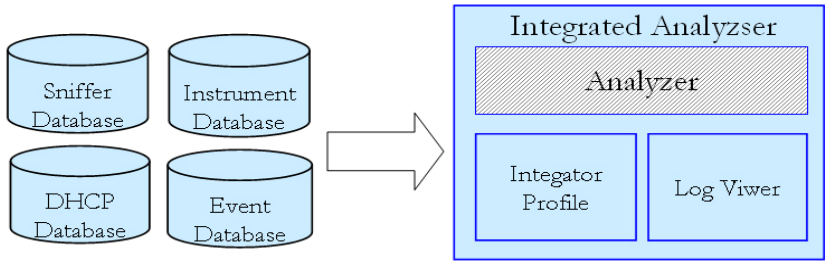



Figure 5-1 系統架構圖

此單元是由 Java 語言撰寫而成，包含有(1)Packet Viewer，這是一個類似

Ethereal 的封包檢視界面，此工具可以從 Packet Sniffer 資料庫中解譯出封包格式，目前已支援 UDP/TCP/ICMP/IGMP/ARP/DHCP/IEEE 802.11 等 Protocol。(2)Kernel Viewer，此工具可以用來顯示接收或者傳送封包所經過的核心函數名稱與時間。(3)Analysis Module，此工具用來提供網路分析結果的重要資訊。簡單說明此三大部份。

## 5.3 系統說明

### 5.3.1 Packet Viewer

Packet Viewer 是個類似 Ethereal 的軟體,其主要目的在於瀏覽所有進出 Linux 作業系統的網路封包，包括有線網路介面、無線網路介面都要能攔截下來。最主要必須呈現的資訊有(1)封包進出網路層的系統時間(2)封包的來源與目的位址(3)封包的格式，包含有 TCP/UDP/ARP/IGMP/ICMP/IEEE 802.11 等格式。(4)封包格式解析，對於常見的網路協議必須提供支援，如 ARP Protocol、DHCP Protocol、IEEE 802.11 等等。

Packet Viewer 所顯示的資訊，可以了解到：

(1)網路裝置是否有掉包的情形發生。

(2)從封包與封包間的時間差可以概括了解到網路裝置的處理效能。對於延遲時間過長或不合理的延遲時間發生都可以清楚的看到。

(3)當解析出封包內容後，有助於進一步的觀察 IP Protocol 的網路行為，如：可以計算出從送出 DHCP Discovery 封包到收到 DHCP ACK 封包之間所花費的時間有多少。還可以配合啟動 Wireless 板卡上的監聽模式來觀察 Wireless Station 與 Station 之間的網路行為。

### 5.3.2 Kernel Viewer

Kernel Viewer 可以用來呈現 Linux Kernel 的網路核心事件，網路核心事件包含有網路介面啟動與關閉、IP 位址改變、網卡 MAC 位址的改變、網路介面卡的註冊與解除、Routing Table 的改變，Default Gateway 的改變等等。

透過這個界面還可以將 Kernel 內所紀錄下來的日誌顯示出來，用以分析網路協議堆疊的行為。此單元最主要的目的在於將 Linux Kernel 的 IP Stack 透明化，讓使用者可以清楚地看到封包經過網路堆疊所留下的足跡。進而從網路核心事件中去分析出網路協議層的處理效能。

### 5.3.3 Analysis Module

此模組可用來統計 Handover 行為中最重要的 6 種延遲時間，詳情可參考[10]。

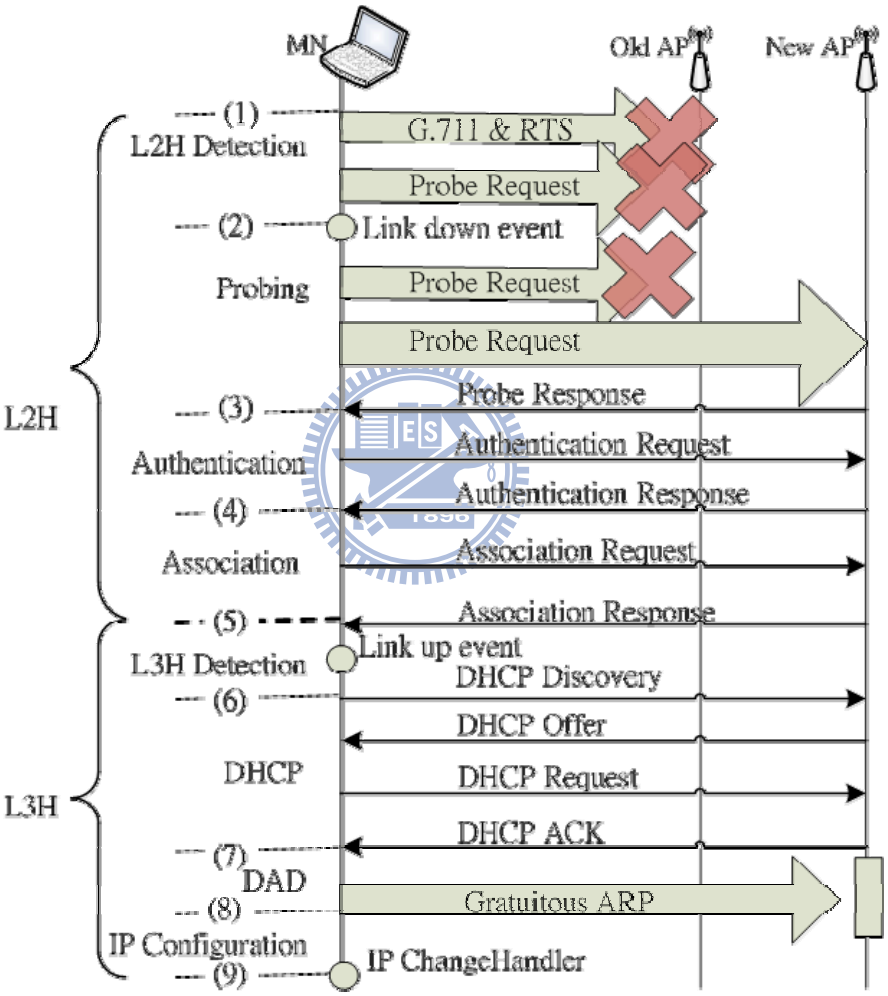



Figure 5-2 Handover 流程圖

(1) L2H Detection Method and Delay

L2H Detection Delay = Link Down - First G.711 Retries

(2) Probing and Probe Delay

Probing Delay = Last Probe Response – Link Down

(3) Authentication Delay

Authentication = Authentication Response – last Probe Response

(4) Association Delay

Association = Association Response – Authentication Response

(5) L3H Detection Method and Delay

L3H = DHCP Discovery – Association Response

(6) DHCP Delay

DHCP = DHCP ACK - DHCP Discovery

# 第六章 Linux 核心網路追蹤與分析工具之實作

## 6.1 開發環境需求

以下為本論文工具開發實作所使用的環境：

系統核心：Linux kernel 2.6.30.1

發行套件：Fedora Core 8

編譯工具：GNU GCC 4.1.2、Java JDK 1.6.0.13

## 6.2 Linux Kernel Patch

本論文使用曹敏峰之「Linux 網路通訊協定堆疊之高效率動態的指令嵌入平台之設計與實作」的論文[3]中的方法來對 Linux Kernel 2.6.30.1 做 Patch。6.2 Instrument Module 設定方式

Instrument Module 提供文字界面模式來對 Probe 函數做設定。當程式進入主選單時，可以選擇(1)或(2)來重新設定所欲 Probe 的函數。

```
t) Config TX Probe Function
r) Config RX Probe Function
s) Start
p) Stop
g) Generate log file
q) Quit

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 a) [X]inet_sendmsg_Func_Init        n) [X]tcp_send_ack_Func_Start         +
 b) [X]icmp_reply_Func_Init          o) [X]tcp_send_synack_Func_Start      +
 c) [X]icmp_send_Func_Init           p) [X]tcp_retransmit_skb_Func_Start   +
 d) [X]tcp_write_xmit_Func_Init      q) [X]tcp_transmit_skb_Func_Start     +
 e) [X]tcp_connect_Func_Init         r) [X]icmp_send_Func_Start            +
 f) [X]tcp_send_ack_Func_Init        s) [X]icmp_reply_Func_Start           +
 g) [X]tcp_send_synack_Func_Init     t) [X]ip_queue_xmit_Func_Start        +
 h) [X]tcp_retransmit_skb_Func_Init  u) [X]ip_append_page_Func_Start       +
 i) [X]inet_sendmsg_Func_Start       v) [X]ip_append_data_Func_Start       +
 j) [X]tcp_sendmsg_Func_Start        w) [X]ip_output_Func_Start            +
 k) [X]tcp_push_one_Func_Start       x) [X]ip_finish_output_Func_Start     +
 l) [X]tcp_write_xmit_Func_Start     y) [X]ip_finish_output2_Func_Start    +
 m) [X]tcp_connect_Func_Start        z) [X]dev_queue_xmit_Func_Start       +
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Figure 6-1 函數設定畫面

## 6.3 TCP 封包於 Linux Kernel 中之路徑分析

本節以 TCP 封包於網路核心中所經過的核心函數來做說明[11]。

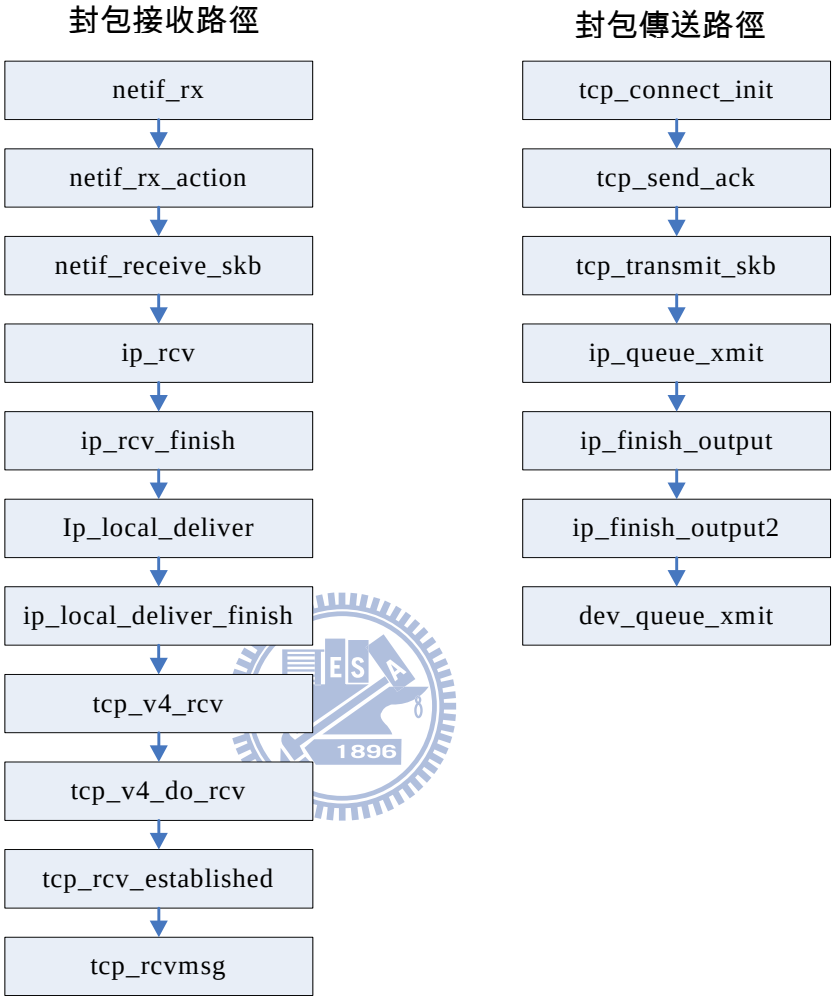



Figure 6-2 TCP 封包於 Linux 上的傳輸路徑

當新的封包進到網卡後，裝置驅動程式通常會呼叫 netif_rx 函式。當封包進入 netif_rx 函式後，netif_rx 函式會對 sk_buff 的相關結構做初始設定，在更新完有關擁塞等級的統計資料後，接著 CPU 這會把封包存放到 CPU 私有的輸入佇列之中並觸發 softirq(NET_RX_SOFTIRQ)以通知核心來處理此封包。

接著 netif_rx_action 會被呼叫，以用來處理進入的封包。netif_rx_action 函數內則是會呼叫 process_backlog 來對輸入佇列中的封包做處理，而 process_backlog 會用 netif_receive_skb 來處理入境的封包。

核心接著會呼叫 ip_rcv 函數來處理 IP 封包，ip_rcv 主要的工作在於對 IP 包

做基本的健全度檢查，當 ip_rcv_finish 被呼叫時，即表示這個檢查以經完成了。接著 ip_rcv_finish 會再調用 ip_route_input 來做路由處理，以判斷封包是要進行轉發(ip_forward)到其他主機還是傳到協議層去做檢查，如果封包是給本機的話，則會進入到 ip_loca_deliver 函數去做處理。

封包在 ip_loca_deliver 函數內的主要工作在於檢查是否對封包做重組(ip_defrag)的動作，IP 封包的表頭會存放所需的資訊。之後會進入 ip_local_deliver_finish，在 ip_local_deliver_finish 函數裡面透過呼叫 ip_prot->handler 函數指標來調用 tcp_v4_rcv 做處理。

TCP 封包接收函數 tcp_v4_rcv 會對 TCP 封包做處理，之後便會呼叫 tcp_v4_do_rcv，當送出 SYN 包後，socket 對應的 sock 的狀態變成 TCPF_SYN_SENT，網卡會在收到 ack 傳到 TCP 層時，會根據 TCPF_SYN_SENT 的狀態，做相關判斷後再發送 ack 包。至此，將 socket 的狀態改為連接建立，即 TCP_ESTABLISHED。從協議層接收數據的動作，由上而下的觸發動作一直到 tcp_recvmsg 函數為止。

傳送方向則比較單純，由上而下的觸發動作為 tcp_connect_init、tcp_send_ack、tcp_transmit_skb、ip_queue_xmit、ip_finish_output、ip_finish_output2、dev_queue_xmit。如需詳情可參考[10]。

## 6.4 Kernel function monitor 的實作

本論文運用 MMAP 記憶體映射機制，將 Instrument Module 內的日誌緩衝區映射到使用者程式中的虛擬記憶體分頁。讓使用者程式能夠直接存取到此塊記憶體，以加速日誌檔案備份的速度。首先，必須於核心程式內將緩衝區做映射的動作。範例如下。

```
buffer = (unsigned char *)(((ulong)buffer_ptr+PAGE_SIZE-1) & PAGE_MASK);

for   (virt_addr=(ulong) buffer; virt_addr < (ulong) buffer+BUFFERSIZE;
                    virt_addr+=PAGE_SIZE)
      SetPageReserved(virt_to_page(virt_addr));
```

Table 6-1 MMAP 核心範例程式-1

除此之外，還必需於檔案操作單元中對 MMAP 指標做註冊的動作。使用者程式才能夠調用到 mmap 函數

```
struct file_operations my_fops =
{.mmap     =  my_mmap, };
static int my_mmap(struct file * filp, struct vm_area_struct * vma)
{
   int ret = remap_pfn_range(vma, vma->vm_start,
            virt_to_phys((void*)((ulong) buffer)) >> PAGE_SHIFT,
            vma->vm_end-vma->vm_start, PAGE_SHARED);
```

Table 6-2 MMAP 核心範例程式-2

最後才能於使用者程式中使用 MMAP API 呼叫來存取此共享區域。請參閱 Table 2-5 使用者程式使用 MMAP 範例。

本論文中，將使用一個獨立的執行緒來對 Linux Kernel 內的日誌資料進行備份，在此執行緒內，使用 Polling 的方式檢查核心內的接收與傳送計數器，再根據計數器值計算出相對映的接收緩衝區與傳送緩衝區的起始位址，並對資料進行備份作業。核心內預先配置了 2MB 大小的緩衝區，其中 1MB 做為接收緩衝區，另外 1MB 當做傳送緩衝區。以現在一筆接收記錄 178 Byte 來說，1MB 的記憶體空間可以儲存 5890 筆記錄於其中。一筆傳送記錄大小為 250 Byte，1MB 的記憶體空間也可以儲存 4194 筆記錄。簡而言之，每一個接收或傳送的封包會佔用陣列內的一個元素，並依序存放，直到用完最後一個元素，再從頭開始覆蓋第一個元素。

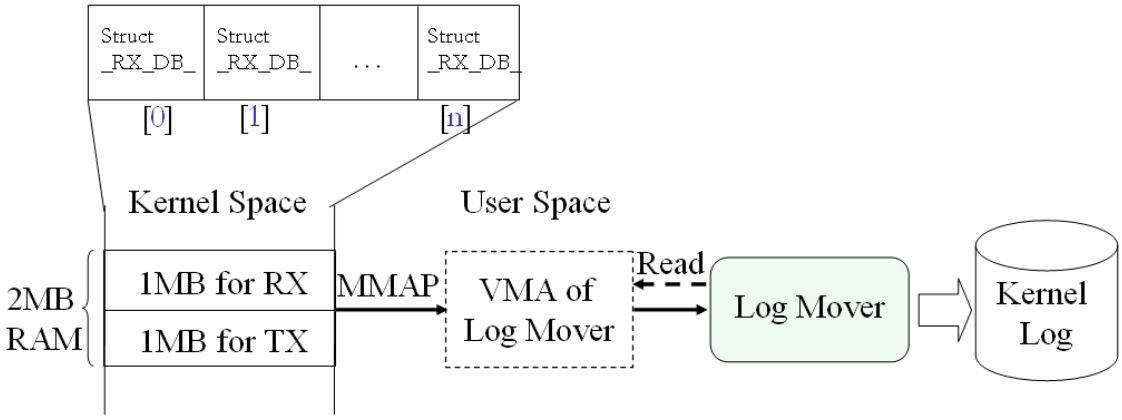



Figure 6-3 核心日誌備份流程圖

## 6.5 工具日誌檔案整合與說明

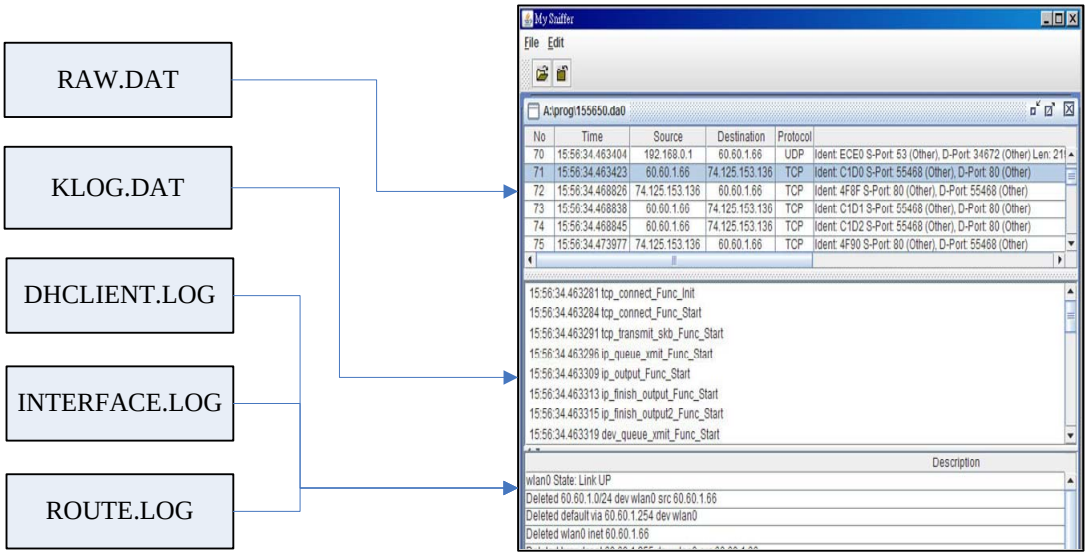



Figure 6-4 工作日誌檔與 UI 工具整合流程圖

本論文共使用 7 種不同格式的工作日誌檔，其功能與用途說明如下：

| 檔案名稱 | 檔案格式 | 說明 |
|---|---|---|
| KLOG.DAT | Binary | Instrument Module 工作日誌 |
| RAW.DAT | Binary | 封包截取器工作日誌 |
| DHCLIENT.LOG | Text | DHCP Client 工作日誌 |
| INTERFACE.LOG | Text | 監控網路介面卡 UP/DOWN/IP 位址用的工作日誌 |
| ROUTE.LOG | Text | 監控 Routing Table 用的工作日誌 |

Table 6-3 工作日誌檔用途說明

KLOG.DAT

本論文工具會在啟動後，自動的將 Instrument Module 所產生的工作日誌備份到 KLOG.DAT 檔案裡面，工作日誌是以二進位格式存入 KLOG.DAT。

RAW.DAT

封包截取器功能有開啟時，會透過 Libpcap 函數庫，將進出 Linux Kernel 的網路封包截取下來，並存入 RAW.DAT 檔案。

DHCLIENT.LOG

本論文會透過修改 DHCP Protocol 的方式來監控 Linux 上 IP 取得的過程。檔案內容請參照 Table 6-9。

INTERFACE.LOG

本論文使用 Netlink 來監控網路介面卡的 UP、DOWN 與 IP Address 的取得與釋放等事件。檔案內容如下所示：

```
15:56:39.531042 [NETWORK] wlan0 State: Link UP
15:56:39.548802 [NETWORK] Deleted wlan0 inet 60.60.1.66
15:56:42.101918 [NETWORK] Assign wlan0 inet 60.60.1.67
```

Table 6-4 INTERFACE.LOG 內容範例

ROUTE.LOG

本論文使用 Netlink 來監控 Linux Kenrel 的 Routing 模組所產生的 Routing Event。檔案內容如下所示：

```
15:56:39.533904 [ROUTING] Deleted 60.60.1.0/24 dev wlan0 src 60.60.1.66
15:56:39.533989 [ROUTING] Deleted default via 60.60.1.254 dev
15:56:39.548802 [NETWORK] Deleted wlan0 inet 60.60.1.66
15:56:43.107429 [ROUTING] default via 60.60.1.254 dev wlan0
```

Table 6-5 ROUTE.LOG 內容範例

此外，當使用者於主選單執行「Generate log file」功能時，本論文工具還會產兩個系統偵錯檔案(KLOG.TXT 與 RAW.TXT)。本論文工具會將 KLOG.DAT 與 RAW.DAT 的檔案內容直譯為有意義的文字檔，以供除錯時使用。此兩個檔案的內容如下所示：

```
00001
Source IP: 60.60.1.66
Source MAC: 00:1F:E1:65:FE:24
Dest IP: 192.168.0.1
Dest MAC: 00:12:CF:38:12:3A
TYPE: 0800
Packet Len: 80
Identification: 394
15:04:34.600935 inet_sendmsg_Func_Init
15:04:34.600936 inet_sendmsg_Func_Start
```

Table 6-6 KLOG.TXT 內容範例

00001 為五位數流水號，每當接收到一筆 Instrument Module 的記錄時，此欄位會自動加‘1’。Soure IP、Dest IP 與 Identification，依序為 IP 表頭中的來源 IP 位址、目的 IP 位址與 Identification 欄位。Source MAC、Dest MAC 與 TYPE 為 Ethernet 的表頭資訊。Packet Len 為 Ethernet 封包長度。其餘資訊為網路封包經過核心網路函式所記錄下來的時間與函式名稱。

```
00001 15:05:13.752677  [TCP] SIP: 60.60.1.254 DIP: 60.60.1.65 Ident: DD9F
00002 15:05:13.752689  [WIFI] Association  Request SA: 96:3c:3c:01:fe:3c
00003 15:05:13.753036  [TCP] SIP: 60.60.1.254 DIP: 60.60.1.65 Ident: DDA0
00004 15:05:13.753039  [WIFI] Association  Request SA: de:3c:3c:01:fe:3c
```

Table 6-7 RAW.TXT 內容範例

00001 為五位數流水號，每當呼叫一次 Libpcap API 以取得一筆網路封包時，此欄位會自動加 '1'。下一個欄位為時間資訊，用來存放接收到此封包時的系統時間。隨後的資訊為 Ethernet 封包的內容解析，當收到一筆 TCP 或 UDP 封包時，會解析出 SIP(來源位址)、DIP(目的位址)、Ident(IP 表頭的 Identification 欄位)、S-Port(來源網路埠號)、D-Port(目的網路埠號)。此時，如果根據這些資訊，可以解析出協議名稱的話，本工具還會對封包內容做進一步的解析。目前的版本，已經可以支援 TCP/IP/ICMP/IGMP/IEEE 802.11 等封包格式，並且能夠依據使用的埠號來分辨出以較常見的協議堆疊，如 TIME、FTP-Data、FTP、DHCP、TFTP、SNMP 等。

## 6.6 DHCP Protocol

本論文需修改 DHCP Client 源始碼以達到監控此協議的運作流程。主要的做法是在 DHCP Protocol state machine 狀態改變時，將事件狀態與系統時間寫入 DHCP 日誌檔中。程式範例如下所示。

```
void AddMsgToLog (char *p)
{
	struct timeval tv1;
	char buf[255];
	char buf2[255];
	int fd= -1;
	gettimeofday(&tv1,NULL);
	ts_print(&tv1,buf);
	sprintf (buf2,"%s [%s] %s\n",buf,devName,p);
	fd = open ("/var/dhclient.log",O_WRONLY|O_APPEND, 0644);
	if (fd == -1)
	fd = open ("/var/dhclient.log",O_WRONLY|O_CREAT|O_TRUNC, 0644);
	write (fd,buf2,strlen(buf2));
	close(fd);
}
```

Table 6-8 新增日誌程式碼範例

於 dhclient.cpp 原始碼中的 dhcpack、dhcpoffer、dhcpnak、send_discover、send_request 等函數內呼叫此 AddMsgToLog 函數，用來將事件發生原因寫入磁碟中儲存。

| | |
|---|---|
| 08:16:44.038266 | [wlan0] DHCPDISCOVER on wlan0 to 255.255.255.255 port 67 interval 4 |
| 08:16:46.065273 | [wlan0] DHCPOFFER from 30.30.1.254 |
| 08:16:46.065426 | [wlan0] DHCPREQUEST on wlan0 to 255.255.255.255 port 67 |
| 08:16:46.089100 | [wlan0] DHCPACK from 30.30.1.254 |

Table 6-9 DHCP 工作日誌內容

## 6.7 Linux 核心網路分析工具功能

此功能主要由 Java 程式語言撰寫而成，用來瀏覽工作日誌檔而使用，程式操作畫面如下所示：

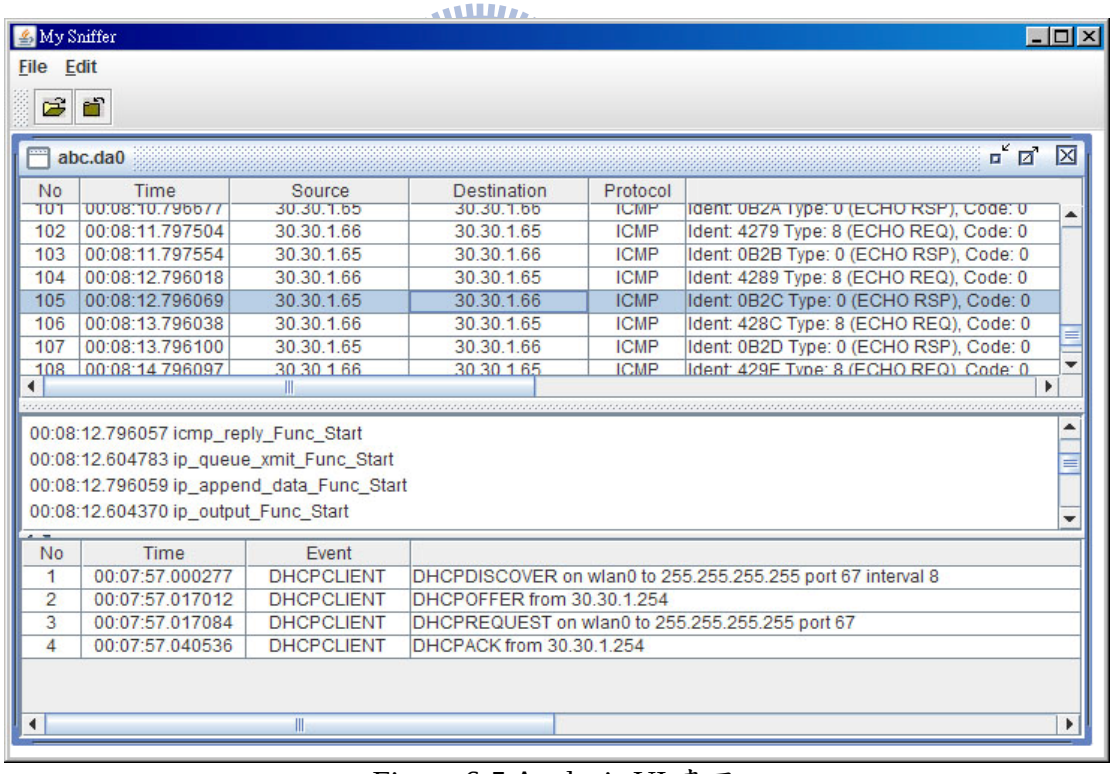
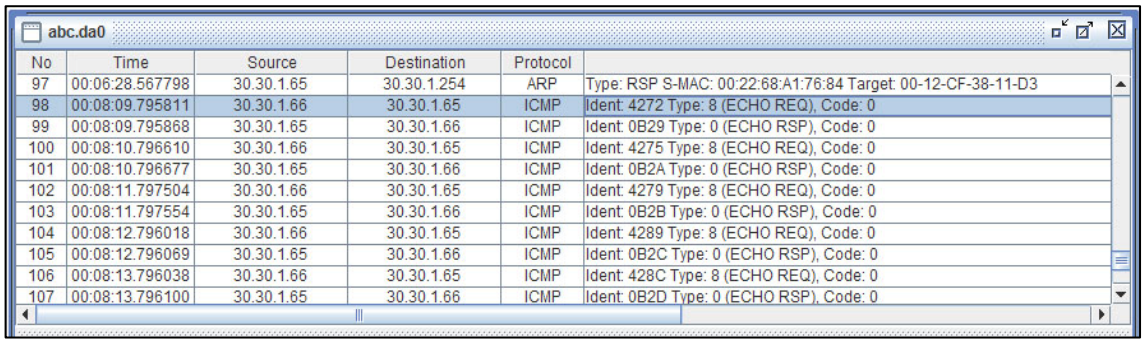



Figure 6-5 Analysis UI 畫面

GUI 介面共分為三大部份，第一部份為 Packet Viewer 功能。主要目的在於瀏覽封包截取器所補獲到得資料封包




| No | Time | Source | Destination | Protocol | |
|---|---|---|---|---|---|
| 97 | 00:06:28.567798 | 30.30.1.65 | 30.30.1.254 | ARP | Type: RSP S-MAC: 00:22:68:A1:76:84 Target: 00-12-CF-38-11-D3 |
| 98 | 00:08:09.795811 | 30.30.1.66 | 30.30.1.65 | ICMP | Ident: 4272 Type: 8 (ECHO REQ), Code: 0 |
| 99 | 00:08:09.795868 | 30.30.1.65 | 30.30.1.66 | ICMP | Ident: 0B29 Type: 0 (ECHO RSP), Code: 0 |
| 100 | 00:08:10.796610 | 30.30.1.66 | 30.30.1.65 | ICMP | Ident: 4275 Type: 8 (ECHO REQ), Code: 0 |
| 101 | 00:08:10.796677 | 30.30.1.65 | 30.30.1.66 | ICMP | Ident: 0B2A Type: 0 (ECHO RSP), Code: 0 |
| 102 | 00:08:11.797504 | 30.30.1.66 | 30.30.1.65 | ICMP | Ident: 4279 Type: 8 (ECHO REQ), Code: 0 |
| 103 | 00:08:11.797554 | 30.30.1.65 | 30.30.1.66 | ICMP | Ident: 0B2B Type: 0 (ECHO RSP), Code: 0 |
| 104 | 00:08:12.796018 | 30.30.1.66 | 30.30.1.65 | ICMP | Ident: 4289 Type: 8 (ECHO REQ), Code: 0 |
| 105 | 00:08:12.796069 | 30.30.1.65 | 30.30.1.66 | ICMP | Ident: 0B2C Type: 0 (ECHO RSP), Code: 0 |
| 106 | 00:08:13.796038 | 30.30.1.66 | 30.30.1.65 | ICMP | Ident: 428C Type: 8 (ECHO REQ), Code: 0 |
| 107 | 00:08:13.796100 | 30.30.1.65 | 30.30.1.66 | ICMP | Ident: 0B2D Type: 0 (ECHO RSP), Code: 0 |

Figure 6-6 Packet Viewer

第二部份為 Kernel Viewer。其主功能用來顯示所有進入到 Linux Kernel 的重要核心函式。

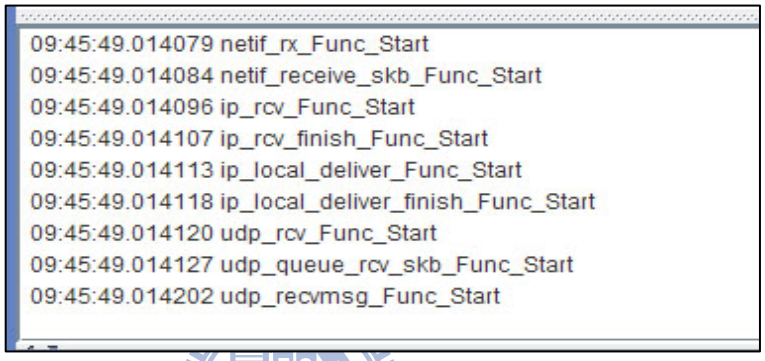



Figure 6-7 Kernel Viewer

第三部份為 Kernel Event Viewer。其主功能用用來監聽重要的 Linux 網路核心事件。

| No | Time | Event | |
|---|---|---|---|
| 1 | 09:47:02.307366 | NETWORK | wlan0 State: Link DOWN |
| 2 | 09:47:14.311999 | ROUTING | Deleted 60.60.1.0/24 dev wlan0 src 60.60.1.64 |
| 3 | 09:47:14.312080 | ROUTING | Deleted default via 60.60.1.254 dev wlan0 |
| 4 | 09:47:14.325777 | NETWORK | Deleted wlan0 inet 60.60.1.64 |
| 5 | 09:47:14.326616 | ROUTING | Deleted broadcast 60.60.1.255 dev wlan0 src 60.60.1.64 |
| 6 | 09:47:14.326662 | ROUTING | Deleted broadcast 60.60.1.0 dev wlan0 src 60.60.1.64 |
| 7 | 09:47:14.326699 | ROUTING | Deleted local 60.60.1.64 dev wlan0 src 60.60.1.64 |
| 8 | 09:47:14.327447 | NETWORK | Deleted wlan0 inet fe80::222:68ff:fea1:7684 |
| 9 | 09:47:14.328256 | ROUTING | Deleted fe80::/64 dev wlan0 |
| 10 | 09:47:14.328286 | ROUTING | Deleted ff00::/8 dev wlan0 |
| 11 | 09:47:43.134751 | NETWORK | wlan0 State: Link UP |
| 12 | 09:47:43.141312 | ROUTING | ff00::/8 dev wlan0 |
| 13 | 09:47:43.141360 | ROUTING | fe80::/64 dev wlan0 |
| 14 | 09:47:43.169129 | ROUTING | ff02::16/128 via ff02::16 dev wlan0 |
| 15 | 09:47:44.097883 | ROUTING | ff02::1:ff65:fe24/128 via ff02::1:ff65:fe24 dev wlan0 |
| 16 | 09:47:44.686404 | NETWORK | Assign wlan0 inet fe80::222:68ff:fea1:7684 |
| 17 | 09:47:45.097730 | ROUTING | ff02::2/128 via ff02::2 dev wlan0 |

Figure 6-8 Kernel Event Viewer

# 第七章 實驗結果與貢獻

本節將介紹為此論文工具所設計的效能分析實驗與使用本工具來對 VOIP Handover 作行為分析實驗。效能分析的實驗目的在於實測出本工具的處理效能。而針對 VOIP Handover 行為來做分析目的，則在於將 PC 當成是一個 VOIP 產品來做評核，除了藉此實驗來了解 Fedora Linux 的 Handover 行為，更希望透過本實驗的實驗結果，能夠找到可以加速 Handover 效能的方向。

## 7.1 系統效能測試

本實驗主要的目的在於測試 Linux 核心於 Patch 前與 Patch 後的網路處理效能，期望的結果為 Patch 後的 Linux Kernel 不能夠對系統效能產生太大的影響。此實驗共分兩個部份,第一部份為使用 Iperf 來做效能測試，第二部份使用 FTP 軟體來做效能測試。

Iperf 測試部份，主要的測試方法為，從 PC2 至 PC1 以 5Mbps 至 1000Mbps 的速度，傳送 UDP 封包持續 30 秒鐘，30 秒鐘過後，再檢查兩部 PC 上的工具日誌檔(KLOG.TXT 與 RAW.TXT)中的資料筆數是否與 PC1 Iperf Client 與 PC2 Iperf Server 執行結束時所顯示的傳送筆數相符。於 PC2 的 Iperf Server 結束時也會於測試結果畫面中顯示出傳送速度與掉包率。透過壓力測試，以了解到本工具的處理效能。詳細步驟與操作方法，說明如下:

(1)將 PC1 與 PC2 設定好固定 IP，PC1：192.168.100.4，PC2：192.168.100.5

(環境配置請參考 Figure 7-1 網路效能測試系統環境配置圖)。

(2)於 PC2 啟動 Iperf Server，使用指令如下

Iperf –s –u –w 2048k –i -1

(3)於 PC1 啟動 Iperf Client，使用指令如下

Iperf –c 192.168.100.5 –u –w 2048k –b [5120k ..204800k] –t 5 –i 1

FTP 測試部份，主要的測試方法為，從 PC2 至 PC1 以 FTP 下載 5MB 至 1000MB 檔案的方式，來測量其傳輸時間與傳輸速率，除了測試 Ethernet 介面外，還必須包含無線網路介面，系統設定上，必須區分開啟本工具的完整功能(Instrument Module 日誌備份+封包截取器)，與開啟 Instrument Module 及日誌備份，以及開啟 Instrument Module 但不開啟日誌備份功能來進行測試。以了解日誌備份對系統效能所造成的影響。系統架設如下所示:

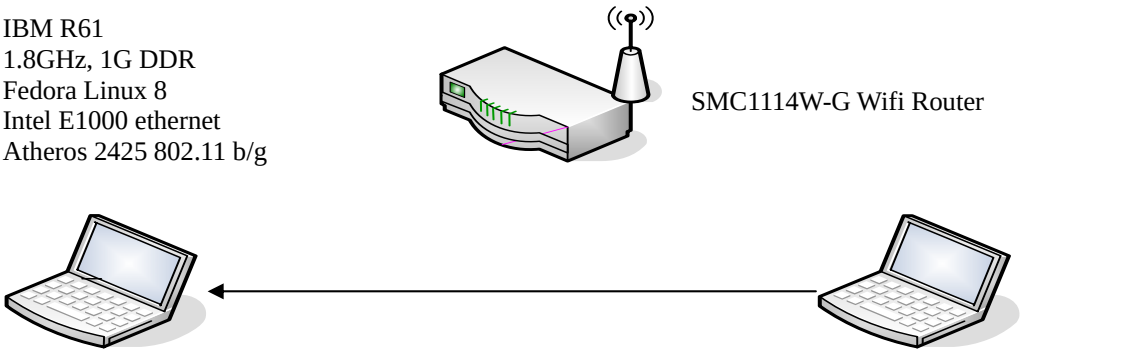



Figure 7-1 網路效能測試系統環境配置圖

## 7.1.1 測試結果

| Speed (Mps) | Patched Linux kernel | | | Original Linux kernel |
|---|---|---|---|---|
| | Profiler+SNIFFER | Profiler only | None | |
| | Loss Rate (%) | Loss Rate (%) | Loss Rate (%) | Loss Rate (%) |
| 5 | 0 | 0 | 0 | 0 |
| 50 | 0 | 0 | 0 | 0 |
| 100 | 0.32 | 0.13 | 0.086 | 0.086 |
| 200 | 44 | 0.91 | 0.83 | 0.34 |
| 400 | 82 | 68 | 18 | 0.37 |
| 500 | 93 | 89 | 64 | 0.45 |
| 1000 | 99 | 99 | 99 | 0.55 |

Table 7-1 Iperf 網路效能測試結果

| Wire FTP Test | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | Linux kernel with patch | | | | | | Original Linux kernel | |
| File Size (MB) | Profiler SNIFFER | | Profiler only | | None | | None | |
| | Time (S) | Speed (MB) | Time (S) | Speed (MB) | Time (S) | Speed (MB) | Time (S) | Speed (MB) |
| 5 | 0.23 | 23 | 0.21 | 24 | 0.21 | 24 | 0.18 | 29 |
| 10 | 0.41 | 26 | 0.39 | 26 | 0.39 | 26 | 0.38 | 27 |
| 50 | 2.02 | 25 | 1.98 | 26 | 1.96 | 26 | 2.1 | 26 |
| 100 | 4.01 | 25 | 3.93 | 26 | 3.92 | 26 | 3.89 | 26 |
| 200 | 8.6 | 24 | 8.5 | 24 | 8.4 | 24 | 8.4 | 24 |
| 400 | 19.1 | 21 | 17.6 | 23 | 17.1 | 23 | 16.3 | 25 |
| 500 | 22.7 | 23 | 20.5 | 25 | 20.4 | 25 | 20.3 | 25 |
| 1000 | 39.4 | 26 | 36.7 | 28 | 36.5 | 28 | 36.4 | 28 |

| Wireless FTP Test | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | Linux kernel with patch | | | | | | Original Linux kernel | |
| File Size (MB) | Profiler SNIFFER | | Profiler only | | None | | None | |
| | Time (S) | Speed (MB) | Time (S) | Speed (MB) | Time (S) | Speed (MB) | Time (S) | Speed (MB) |
| 5 | 3.4 | 1.5 | 3.41 | 1.5 | 3.45 | 1.5 | 3.45 | 1.5 |
| 10 | 6.81 | 1.5 | 6.86 | 1.5 | 6.86 | 1.5 | 6.85 | 1.5 |
| 50 | 34.5 | 1.5 | 39.4 | 1.5 | 35.3 | 1.4 | 35.3 | 1.4 |
| 100 | 70.9 | 1.4 | 69.5 | 1.4 | 69.2 | 1.4 | 69.0 | 1.4 |

Table 7-2 FTP 測試結果

### 7.1.2 結論

從 FTP 網路效能實驗結果得知，隨著傳送時間的拉長，處理效能反而有下降的趨勢，在下載 100MB 以下檔案大小的時候，速度還算穩定，但在 100MB 到 500MB 之間，速度開始下滑。以有線部份的 FTP 檔案傳輸來說，當傳輸 400MB 的檔案大小時，其傳送速度下降至 21Mbps，與未經 Patch 過的 Linux Kernel 相比，速度下降了 4MB。

再對照 IPerf 實驗結果，亦可以看的出來當傳輸速度大於 50Mbps 以後，開始有出現掉包的現象發生。這部份，可以透過更換執行速度更高的主機，或使用存取速度更高的硬碟來做改善，更可以透過更好的備份演算法來加速資料的備份。以目前的程式架構來說，每接收一次 Linux Kernel 日誌，便立刻將資料寫存入磁碟之中，這樣會有磁碟存取過於頻繁的現象發生，做法上並不是非常的好。因此，可以改為每收到 N 筆後，再一次寫入磁碟的方式，相信對速度上的提昇是會有幫助的。

從實驗結果看來，以本工具來評量傳輸速度 50Mps 以下的裝置，並不會有太大的問題的。相對於曹敏峰於[2]中所使用的方法，透過預先配置好的 50000 Byte 核心記憶體來儲存 Instrument Module 日誌，以一筆 60Byte 的本文記錄來說，最多也只能儲存 833 筆記錄，當超過此資料筆數後，會重新覆蓋此儲存區域。簡而言之，當傳輸速度愈高時，這種現象會發生的愈快，因此並不是一個非常好的做法。再者，以本文格式來做為資料儲存格式，不但會浪費磁碟存取空間，在做資料搜尋時，也並不是非常的方便。

## 7.2 VOIP Handover 行為分析

此實驗的主要目的，在於利用本工具來對 VOIP 的 Handover 行為做分析。目的在於觀察 VOIP 的 Handover 行為，以了解到整個 Handover 行為在網路核心所產生的變化。並透過本工具來計算出 Handover 行為發生時的 6 種延遲時間[10]。相關的實驗分析也可以參考[12]。本實驗所使用的測試設備與環境架設如下圖所示：

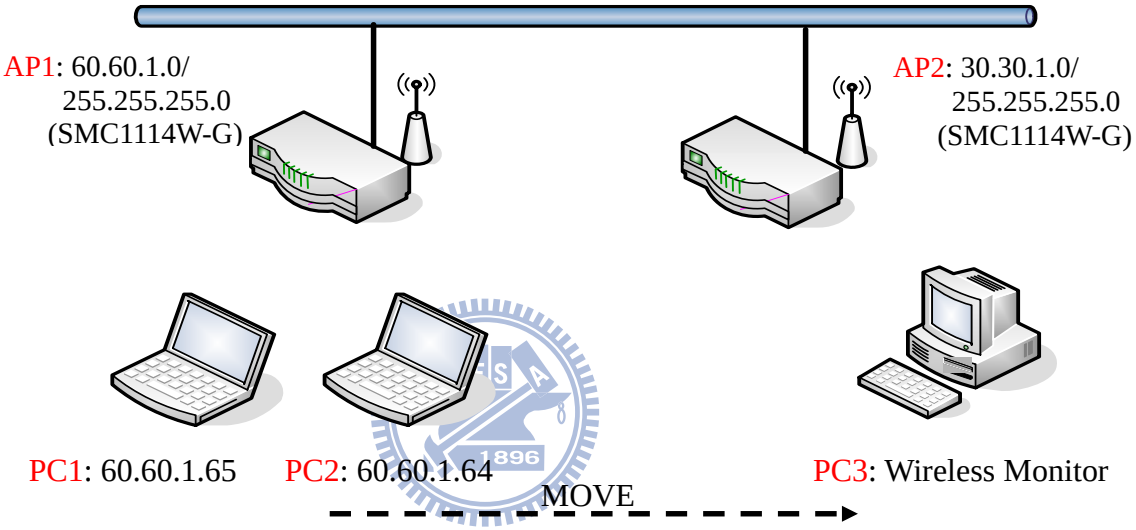



Figure 7-2 VOIP Handover 系統環境配置圖

主要的測試方法為，從 PC1 撥打 VOIP 電話(skype)至 PC2，於通話 60 秒後 Power down AP1，此時可以觀察到 Fedora Linux 會 Handover 至 AP2，並透過本工具來計算 6 種不同階段的延遲時間，用來分析 Fedora Linux 的 Handover 行為。詳細測試步驟與方法如下：

(1) 設定好 PC1 與 PC2，並使其與 AP1 作連線。此時 PC1 與 PC2 皆會拿到相同網段 60.60.1.0/24 的 IP 位址。

(2) 於 PC1、PC2、PC3 上開啟本工具。

(3) 從 PC1 撥打 VOIP 電話(skype)至 PC2，並於通話 60 秒鐘後進入下一步驟。

(4) 關閉 AP1 電源，並觀察 PC1 與 PC2 是否能夠順利地 Handover 至 AP2。

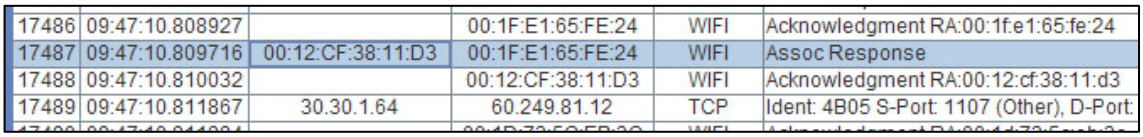
(5) 使用本工具來計算 Handover 所產生的 6 種延遲時間。

### 7.2.1 實驗結果

舉 Association Delay 來說，Association Delay 的計算方式如下：

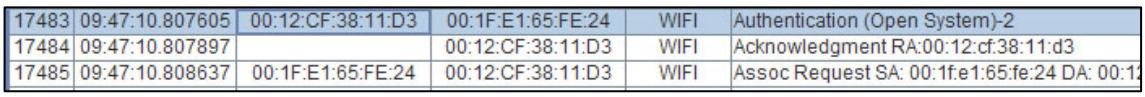
Association Delay = Association Response - Authentication Response

我們可以從 JAVA UI 上的 Packet Sniffer 找到 Association Response 與 Authentication Response 這兩筆封包，再將其封包截取時間做相減即可。



| | | | | | |
|---|---|---|---|---|---|
| 17486 | 09:47:10.808927 | | 00:1F:E1:65:FE:24 | WIFI | Acknowledgment RA:00:1f:e1:65:fe:24 |
| 17487 | 09:47:10.809716 | 00:12:CF:38:11:D3 | 00:1F:E1:65:FE:24 | WIFI | Assoc Response |
| 17488 | 09:47:10.810032 | | 00:12:CF:38:11:D3 | WIFI | Acknowledgment RA:00:12:cf:38:11:d3 |
| 17489 | 09:47:10.811867 | 30.30.1.64 | 60.249.81.12 | TCP | Ident: 4B05 S-Port: 1107 (Other), D-Port: |

Figure 7-3 Association Response 封包顯示畫面



| | | | | | |
|---|---|---|---|---|---|
| 17483 | 09:47:10.807605 | 00:12:CF:38:11:D3 | 00:1F:E1:65:FE:24 | WIFI | Authentication (Open System)-2 |
| 17484 | 09:47:10.807897 | | 00:12:CF:38:11:D3 | WIFI | Acknowledgment RA:00:12:cf:38:11:d3 |
| 17485 | 09:47:10.808637 | 00:1F:E1:65:FE:24 | 00:12:CF:38:11:D3 | WIFI | Assoc Request SA: 00:1f:e1:65:fe:24 DA: 00:1 |

Figure 7-4 Authentication Response 封包顯示畫面

| | PC1 | PC2 |
|---|---|---|
| L2H Detection Method Delay | 32.360017s | 27.628691s |
| Probing and Probe Delay | 6.798874s | 6.787582s |
| Authentication Delay | 1.799973s | 1.683394s |
| Association Delay | 0.002111s | 0.00206s |
| L3H Detection Method Delay | 3.888115s | 3.87256s |
| DHCP Delay | 0.038293s | 0.035146s |
| DAD Delay | N/A | N/A |

Table 7-3 VOIP Handover 行為分析與測試

※因為 Linux PC 並沒有送出 GARP，所以無法計算 DAD 與 IP Configuration Delay，故結果以 N/A 表示。同樣的情況在 Windows PC 上是會發出 GARP 的，所以 Fedora Linux 在這部份還有改善的空間。

### 7.2.2 結論

從本次的實驗發現，當 PC 與 AP 失聯後，Fedora Linux 會優先嘗試與前一次連線成功的 AP 進行連線。這部份花了非常多的時間，跟 Windows PC 比較起來，差異非常的大，因為 Windows PC 能夠在 3 秒內就開始嘗試與其它的 AP 進行網路連線。

從 Probing and Probe Delay 的結果來看，Atheros driver 花了將近 7 秒鐘來判斷出 Link 斷掉了，之後再通知 Linux 核心 Link Down 已經發生。

從 L3H Detection Method Delay 的結果來看，Linux OS 在收到 Association Response 封包後，並沒有立刻呼叫 DHCP Client 協議，有將近 4 秒鐘的延遲，這部份可能得檢查 Linux Network Manager 原始碼才能得知在這 4 秒鐘之內，Linux 在做什麼事情。

從結果來看，Linux 的 Network Manager 並不能將 Handover 處理的非常好，在實驗過程當中，我們發現在 Wireless 斷線後，如果 Ethernet 網卡上連接有網路線的話，Network Manager 就會優先嘗試 Handover 至有線網路，並且放棄對 Wireless 連線做嘗試。我們甚至發現，如果 Ethernet Port 並沒有插上網路線，但是在網路卡有設定固定 IP 的情況底下，還是機會發生 Network Manager 取消與 Wireless AP 嘗試連線的現象發生，這部份的行為跟 Window PC 不太一樣。Network Manager 看起來比較像是一個有線網路與無線網路的保護切換程式，但切換速度似乎又太慢了，這部份的確看來是有改善空間。

## 7.3 貢獻

本篇論文所產生的工具程式已經可以用來整合(1)核心內部通訊協定堆疊之分析(2)核心內部網路事件通知機制(3)封包擷取工具。並且具備了對有線與無線網路裝置的分析能力，透過高效率的核心函式插碼機制，可以深入核心內部做探測。並透過高效率的記憶體映射技術，可以將儲存在核心記憶體內大量的輔助資訊傳送到使用者程式中去做處理，這樣的方式除了可以用來偵測核心內部的網路堆疊，對於感興趣的核心函數，也都可以使用此方法來做探測。對於追蹤核心函數提供了一個很好的應用使用方法。

本工具也具有將 Kernel Instrument Module 日誌、Router Event 日誌、Networking Interface Event 日誌，封包截取器日誌、DHCP Client 日誌等，整合於同一個 GUI 上的功能，讓使用者可以方便的瀏覽任何一筆資料，對於使用上來說是非常方便的。

此外，對於存在於 Kernel 內部的日誌資料，以 Instrument Module 日誌來說，透過 MMAP 將此日誌資料緩衝區映射到使用者程式來進行備份的方法，也是一種很好的應用，解決了日誌資料暫用核心記憶體的問題。之前曹敏峰的做法，只能存放 50000Byte 日誌資料於核心之中，本論文新的做法大大的提昇了處理效率。

透過將 Instrument Module 日誌與封包截取器日誌做整合，可以提供使用者更完整的資訊。雖然以封包截取器來監聽網路封包會消耗大量的系統效能，但因為 Libpcap 函數庫也是透過 MMAP 處理技術，會直接從網卡上的 RING Buffer 將網路封包映射到用戶端程式，也因此讓即時地監聽網路封包、分析封包、分析協議層的行為成為可能。

近年來，隨著網路裝置的大量出現，市場上除了滿足量的需求之外，還必須時時注意品質的提昇，才能夠延長產品的使用壽命。本工具除了消極地可以做為除錯的設備外，更可以提供軟體發展人員用來尋找系統效能低落的重要原因。透過工具的快速分析，將來也可以將這個方法進一步的應用在自動化的產品檢測上，來達到縮減產品的開發時間。

# 第八章 結論與未來工作

## 8.1 結論

隨著網際網路的快速掘起，造就了網路裝置的快速氾濫。如何有效率的來分析網路問題是一個很值的深思的問題。可以說牽一髮而動全身，整個分析方法都必須從上到下做垂直整合。以往在沒有工具輔助的情況底下，開發人員只有不斷的插入 Debug Code 到核心原始碼內去除錯，他所看到的只是一小點而已。而且 Linux 原始碼是如此的龐大，每 Compile 一次都必須花費不少的時間，如果只是很好複製的問題，應該都不難解決。但如果是系統問題，若沒有工具來做輔助，是很難發現問題所在的。

本論文及是從內而外，及從下而上將網路的核心事件做整合。讓使用者可以在單一介面上從一個點看到整個面。讓所有資訊一覽無遺。對於解決系統效能或是找尋系統瓶頸來說，這將會是一個方便好用的工具。

隨著個人電腦效能的不斷提昇，系統處理資訊的速度會變的愈來愈快。本工具就能夠看到更多的資料。也就能夠更快速的過濾各種可能會影響系統效能的真正原因，找到系統內部的錯誤。

## 8.2 未來工作

未來的工作，可以增加智慧型的診斷工具到本工具中，透過增加更多的協議過濾器到系統裡，本工具就能增加對更多問題的處理能理。磁碟存取演算法的效能改善也是一個很重要的議題。現階段所有資料都可以由 Packet Buffer 內直接取出來，所以資訊的吞吐量是非常高的。但開啟封包截取器也會使得 Linux 系統的整體效能受到一定程度的影響。

簡單來說，在關閉封包截取器的情況底下，Kernel 日誌的處理是可以達到很高的處理效能的。但是當封包截取器開啟的情況底下。系統大致上來說只能夠處理 100MB 以下的資訊量。將來，如果有一天，必需要去處理一個存在於 Gigabit Ethernet 的問題。相信就會有捉襟見肘的情況發生。因此這也是一個可以加以改良的方向。

此外最重要是，如何從大量的工作日誌當中，快速的分析出各種延遲時間，並找出這些延遲時間的發生原因，這才是最重要的，有了這些資訊才能提供系統開發者改進的方向，也才能把更高品質的產品服務帶給消費者。

# Reference


[1] Ariane Keller, “Kernel space - user space communication”, Available from: http://people.ee.ethz.ch/~arkeller/linux/k_u_howto.html

[2] Kai-Chen Hsu, "Design and Implementation of a Middleware for Network-Aware Applications", WIN Lab NCTU, 2006.

[3] Min-Feng Tsao, "Design and Implementation of an Efficient and Configurable Instrument Platform for Linux Network Protocol Stack”, WIN Lab NCTU, 2008.

[4] DANIEL P.BOVET， Understanding the LINUX KERNEL, O’Reilly, Aug. 2006.

[5] Wireshark: The World's Most Popular Network Protocol Analyzer, Available from: *http://www.wireshark.org/*

[6] Sniffer Portable: Network Analyzer, available from: http://www.netscout.com/products/sniffer_portable.asp

[7] Fedora Linux, Available from: http://fedoraproject.org/

[8] J. Corbet, A. Rubini, and G. Kroah-Hartman, Linux Device Drivers, 3rd Edition, O’Reilly, Feb. 2005.

[9] John Shapley Gray, Interprocess communication in Linux,Pearson Education Taiwan Ltd.

[10] Yuan-Li Hsu ,”Analysis of Link-layer and Cross-layer Handover Behavior of IEEE 802.11 Wireless Networks” , WIN Lab NCTU, 2009.

[11] CHRISTIAN BENVENUTI, Understanding LINUX NETWORK INTERNALS, O’Reilly, Feb. 2005.

[12] Chia-Chi Hung , "An Empirical Analysis of IEEE 802.11 Wireless Networks and SIP-based VoIP Handovers", WIN Lab NCTU, 2007.