(19)中華民國智慧財產局
(12)發明說明書公開本 (11)公開編號：TW 201206120 A1
(43)公開日： 中華民國 101 (2012) 年 02 月 01 日

(21)申請案號：099125188 (22)申請日： 中華民國 99 (2010) 年 07 月 29 日
(51)Int. Cl.： *H04L12/56 (2006.01)*
(71)申請人：國立交通大學 (中華民國) NATIONAL CHIAO TUNG UNIVERSITY (TW)
新竹市大學路 1001 號
(72)發明人：曾建超 TSENG, CHIEN CHAO (TW) ；李宗鴻 LI, TSUNG HUNG (TW) ；張弘鑫 CHANG, HUNG HSIN (TW)
(74)代理人：蔡清福
申請實體審查：有 申請專利範圍項數：10 項 圖式數：7 共 29 頁

(54)名稱

追蹤網路封包之處理程序的方法

METHOD FOR TRACING PROCESSING PROCEDURE OF NETWORK PACKET

(57)摘要

本發明提出一種追蹤網路封包處理程序的機制，可藉由核心函式存取網路封包資料，追蹤網路封包在核心系統的處理程序。應用在網路通訊裝置中，該機制可以利用網路封包的資料結構，找出存取封包的函式，並在該函式嵌入指令，記錄執行函式與封包內容，追蹤系統內部處理網路封包的行為與時間點，協助開發者分析系統內部的網路行為，釐清與改善網路通訊問題。本發明包含一個函式分析方法，該方法可以找出系統中有存取網路封包資料的函式，並在這些函式中嵌入指令，藉由程式本身執行函式的順序，依序觸發嵌入指令來記錄被執行的函式名稱或代碼。

70：工具原始碼

```
int ip_build_and_send_pkt(struct sk_buff *skb, struct sock *sk,
              __be32 saddr, __be32 daddr, struct ip_options *opt)
{
    struct timeval tv;
    do_gettimeofday(&tv);
    printk("Entry time: [%ld][%ld] function: [%p], skb: [%p]\n",
        tv.tv_sec, tv.tv_usec, __FUNCTION__, skb);

    struct inet_sock *inet = inet_sk(sk);
    struct rtable *rt = skb->rtable;
    struct iphdr *iph;

    /* Build the IP header. */
    skb_push(skb, sizeof(struct iphdr) + (opt ? opt->optlen : 0));
    skb_reset_network_header(skb);
    iph = ip_hdr(skb);
    iph->version  = 4;
    iph->ihl      = 5;
    iph->tos      = inet->tos;
    if (ip_dont_fragment(sk, &rt->u.dst))
        iph->frag_off = htons(IP_DF);
    else
        iph->frag_off = 0;
    iph->ttl      = ip_select_ttl(inet, &rt->u.dst);
    iph->daddr    = rt->rt_dst;
```

70

(19)中華民國智慧財產局
(12)發明說明書公開本 (11)公開編號：TW 201206120 A1
(43)公開日： 中華民國 101 (2012) 年 02 月 01 日

(21)申請案號：099125188 (22)申請日： 中華民國 99 (2010) 年 07 月 29 日
(51)Int. Cl.： *H04L12/56 (2006.01)*
(71)申請人：國立交通大學 (中華民國) NATIONAL CHIAO TUNG UNIVERSITY (TW)
新竹市大學路 1001 號
(72)發明人：曾建超 TSENG, CHIEN CHAO (TW) ；李宗鴻 LI, TSUNG HUNG (TW) ；張弘鑫 CHANG, HUNG HSIN (TW)
(74)代理人：蔡清福
申請實體審查：有 申請專利範圍項數：10 項 圖式數：7 共 29 頁

(54)名稱

追蹤網路封包之處理程序的方法

METHOD FOR TRACING PROCESSING PROCEDURE OF NETWORK PACKET

(57)摘要

本發明提出一種追蹤網路封包處理程序的機制，可藉由核心函式存取網路封包資料，追蹤網路封包在核心系統的處理程序。應用在網路通訊裝置中，該機制可以利用網路封包的資料結構，找出存取封包的函式，並在該函式嵌入指令，記錄執行函式與封包內容，追蹤系統內部處理網路封包的行為與時間點，協助開發者分析系統內部的網路行為，釐清與改善網路通訊問題。本發明包含一個函式分析方法，該方法可以找出系統中有存取網路封包資料的函式，並在這些函式中嵌入指令，藉由程式本身執行函式的順序，依序觸發嵌入指令來記錄被執行的函式名稱或代碼。

70：工具原始碼

```
int ip_build_and_send_pkt(struct sk_buff *skb, struct sock *sk,
              __be32 saddr, __be32 daddr, struct ip_options *opt)
{
    struct timeval tv;
    do_gettimeofday(&tv);
    printk("Entry time: [%ld][%ld] function: [%p], skb: [%p]\n",
        tv.tv_sec, tv.tv_usec, __FUNCTION__, skb);

    struct inet_sock *inet = inet_sk(sk);
    struct rtable *rt = skb->rtable;
    struct iphdr *iph;

    /* Build the IP header. */
    skb_push(skb, sizeof(struct iphdr) + (opt ? opt->optlen : 0));
    skb_reset_network_header(skb);
    iph = ip_hdr(skb);
    iph->version  = 4;
    iph->ihl      = 5;
    iph->tos      = inet->tos;
    if (ip_dont_fragment(sk, &rt->u.dst))
        iph->frag_off = htons(IP_DF);
    else
        iph->frag_off = 0;
    iph->ttl      = ip_select_ttl(inet, &rt->u.dst);
    iph->daddr    = rt->rt_dst;
```

70

201206120

發明專利說明書

（本說明書格式、順序，請勿任意更動，※記號部分請勿填寫）

※申請案號：99125188

※申請日：99.7.29　※IPC分類：H04L 12/56

一、發明名稱：（中文/英文）

追蹤網路封包之處理程序的方法

Method for Tracing Processing Procedure of Network Packet

二、中文發明摘要：

本發明提出一種追蹤網路封包處理程序的機制，可藉由核心函式存取網路封包資料，追蹤網路封包在核心系統的處理程序。應用在網路通訊裝置中，該機制可以利用網路封包的資料結構，找出存取封包的函式，並在該函式嵌入指令，記錄執行函式與封包內容，追蹤系統內部處理網路封包的行為與時間點，協助開發者分析系統內部的網路行為，釐清與改善網路通訊問題。本發明包含一個函式分析方法，該方法可以找出系統中有存取網路封包資料的函式，並在這些函式中嵌入指令，藉由程式本身執行函式的順序，依序觸發嵌入指令來記錄被執行的函式名稱或代碼。

三、英文發明摘要：

This invention presents a new mechanism that can trace the execution sequence of kernel functions that process data network packet and record information of concern. With such

trace, we can analyze the networking behavior, conduct software debugging and optimize performance of networking devices. The underlying idea of the mechanism is that the execution sequence of functions process a network packet can be derived from the sequence the functions access the data structure of the network packet. The proposed mechanism first adopts a function analyzer to identify all functions that refer or handle the data structure of network packets, and then patches instructions in each function identified. At run time, the execution of each patched function will trigger the patched instructions automatically to record the function identity and other information of concern. Because, the patching instructions are executed in a sequence exactly the same as the one of the patched functions, the patching instructions can thus record the execution sequence of the patched functions.

四、指定代表圖：

(一)本案指定代表圖為：第（ 七 ）圖。

(二)本代表圖之元件符號簡單說明：

70：工具原始碼

五、本案若有化學式時，請揭示最能顯示發明特徵的化學式：

[S]

六、發明說明：

【發明所屬之技術領域】

本發明為一種追蹤網路封包之處理程序的方法，尤指一種有關網路通訊裝置、雲端計算之追蹤網路封包的處理程序之方法。

【先前技術】

對於網路通訊裝置而言，界定網路行為異常(例如：傳輸延遲、反應時間以及封包遺漏等問題)的原因是有必要的，此對於日漸普及的連網裝置的開發非常重要。通常網路封包會由系統核心處理，系統核心的分析工具(Linux Kernel Profiler)是用來協助開發者分析系統核心內部的(internal)行為，現有的工具大略可以分為三類：原始工具類(Source Instrumentation)、二進制工具類(Binary Instrumentation)、統計取樣類(Statistical Sampling)，各分類相關的分析工具即如表一所示者。

表一、現有系統核心分析工具的分類

Source Instrumentation	Binary Instrumentation	Statistical Sampling
LTTng	Kprobe	Oprofile
KTAU	Systemtap	
LKST	KernInst	
KFT	KLASY	
Ftrace	Dtrace	

[S]

Kernprof

Source Instrumentation主要是修改系統原始碼(Source Code)，插入能記錄系統內部資訊的指令，以達到分析系統核心內部的(internal)行為。例如Linux追蹤工具箱第二代(Linux Trace Toolkit Next Generation, LTTng)、核心調整及分析公用程式(Kernel Tuning and Analysis Utilities, KTAU)、Linux核心狀態追蹤器(Linux Kernel State Tracer, LKST)等，修改系統中重要的核心函式，此三項技術在核心系統內部對重要的核心函式插入程式碼，記錄下核心事件觸發時間，用來分析核心系統的運作行為，此記錄通常包含上下文切換(context switch)、計時器期滿(timer expired)和系統呼叫(system call)等等。

但是，依然無法詳細追蹤網路封包在核心系統內部處理過程，而且無法自動尋找與嵌入追蹤指令於核心函式序列(sequence)，亦即此些記錄無法協助開發者追蹤網路封包於核心網路通訊協定的處理過程。核心函式追蹤(Kernel Function Trace, KFT)、FTrace和Kernprof等，則進一步修改所有的系統核心函式，記錄核心函式執行順序和執行時間等資訊，此三項技術在核心系統內部所有的核心函式均插入程式碼，完整取得核心系統的運作過程，結果如第一圖所示者。

其中：index指主函式(primary function)的編號、%time指primary function的使用時間比、self指primary function的運行時間、children指primary function呼叫子(child)的運

[S]

行時間、called指被呼叫的總次數、name指primary function的名稱。在實框線內的主函式的編號(index)分別為[20]及[21]，在主函式的上下均有母(Parent)及子(Children)的程式（如以虛線的箭頭所示者）；然而，開發者依然無法利用這些資訊，正確地追蹤核心系統處理封包的函式執行順序；亦即使用者依然無法藉由分析結果，進而有效率地且正確地追蹤封包導向的核心函式序列。

Binary Instrumentation主要是直接修改系統機器碼(Object Code)，將執行流程暫時導到此記錄系統內部資訊的函式，以達到動態分析系統內部的行為。以K探針(Kprobe)為例，開發者需要藉由編譯程式(Compiler)或者反組譯程式(Disassembler)的協助，事先取得目標核心函式的進出點位址，再藉由此類工具協助修改系統機器碼，取得所需資訊。

又Kprobe、KernInst、D追蹤(DTrace)此三項技術主要是修改核心系統的機器碼(Object Code)，以達到動態分析系統內部的行為，使用者藉由Compiler或者Disassembler的協助再修改機器碼，但是使用者需要具有核心系統處理網路封包的相關知識，否則無法正確地追蹤封包導向的核心函式序列，所以依然無法自動尋找與嵌入追蹤指令於核心函式序列，此類技術皆有須在特定裝置及核心版本上運作的缺失。至於二進制工具類之系統跟蹤分析程式(Systemtap)，此技術主要是提供一個介面給使用者，讓使用者藉由此介面簡化使用Kprobe的困難處，使用者只須提

[S]

供目標核心函式的名稱，Systemtap負責找出目標核心函式的進去點位址，並且完成設定Kprobe，但是使用者依然需要具有核心系統處理網路封包的相關知識，否則無法正確地追蹤封包導向的核心函式序列，所以依然無法自動尋找與嵌入追蹤指令於核心函式序列。

另有些工具，如核心階層特徵定向系統(Kernel Level Aspect-oriented System, KLASY)，此技術也是提供一個介面給使用者，讓使用者藉由此介面簡化使用KernInst的困難處，其透過修改後的Compiler(gcc)先對核心程式做處理，再由開發者針對某些函式做追蹤與記錄，使用者可以提供目標資料結構的名稱，KLASY負責找出目標資料結構被處理的指令位址，並且完成設定KernInst，但是此技術所產生的分析資料過於龐大，使用者無法有效率地追蹤封包導向的核心函式序列，且此技術需要特定的compiler協助，對於嵌入式裝置的開發者相當不方便。

亦即開發者必須具有核心系統處理網路封包的相關知識，才可以人工檢視的方式追蹤封包處理程序，然而人工檢視方式不但費時，更容易遺漏或誤查，很難正確有效地追蹤封包在核心系統的函式執行順序，而且Binary Instrumentation的方式之應用性是較差的。至於Statistical Sampling主要是週期性檢查CPU中正在執行的指令(Instruction)，再將記錄結果以統計的方式，呈現給使用者。而就O分析工具(Oprofile)而言，此技術主要是以統計的方式，分析核心系統的行為，週期性檢查指令再

[S]

將記錄結果以統計的方式，呈現給使用者，因此無法完整地且正確地追蹤封包導向的核心函式序列。

此些記錄並非針對處理網路封包的函式做記錄，所以並不適合用來追蹤處理封包的函式執行順序。因此，目前沒有一套工具能夠正確有效地分析與追蹤網路通訊裝置之系統處理封包的函式之執行順序，協助開發者除錯並界定發生傳輸延遲、反應時間以及封包遺漏等問題的原因。此系統核心分析工具(Linux Kernel Profiler)應用於追蹤網路通訊裝置處理封包的函式之執行順序的缺點如下：

1. 無法提供封包導向的函式序列之追蹤記錄，對於想要分析處理封包的函式之執行順序的使用者相當不方便；

2. 必須運作於某種特定的裝置或者系統版本，對於嵌入式裝置的開發相當不方便；以及

3. 使用者必須先具備封包處理與系統實作知識，並依人工檢閱方式植入追蹤指令，過程繁瑣且容易錯誤。

因此，如何改善函式序列之無法提供封包導向的追蹤記錄、必須運作於某種特定的裝置及必須先具備封包處理與系統實作知識之人工檢閱方式等問題，經發明人進行實驗、測試及研究後，終於獲得一種追蹤網路封包之處理程序的方法，除了有效解決無法提供追蹤記錄、運作於特定裝置及先具備實作知識等缺點外，亦能獲致加速網路通訊軟體的開發時程與執行效能之功效。亦即本發明所欲解決的課題即為如何克服存取網路封包的函式不容易找出的

問題，而使得系統核心之處理程序得以被追蹤，以及如何克服只有使用 sk_buff 此一特定資料之原始型態名作為搜尋的基準點之問題，又如何克服要在該函式中設法記錄執行函式與封包內容的問題等。

【發明內容】

本發明為一種追蹤一網路封包(Network Packet)之一處理程序的方法，其中該網路封包係儲存在一資料結構內，且該網路封包在一系統核心內接受該處理程序，該方法包括如下步驟，呼叫一函式以傳遞該資料結構，並據以追蹤該處理程序。

較佳者，該方法的系統為一 Linux，而該資料結構為一 sk_buff，該方法利用該系統核心以存取該網路封包，該方法更包含利用一特定函式以存取該網路封包，且該處理程序為該資料結構之一執行順序。

又按照一主要技術的觀點來看，本發明可以涵蓋到一種追蹤一網路封包之一處理程序的方法，其中該網路封包係在一系統核心內接受該處理程序，該方法包括如下步驟，提供一函式，以處理該網路封包，其中該函式具一參數，定義該參數之一資料型態名，以及藉搜尋該資料型態名，而追蹤該處理程序。

較佳者，該方法的資料型態名係為一搜尋的基準點，而該資料型態名為一 sk_buff 之原始型態名或一變形型態名。

當然，該方法的變形型態名可以為一別名、一客製化

資料型態名或一巢狀資料結構，而該變形型態名係使其與該網路封包之一資料型態產生關聯。

當然，該方法更可以包含從該系統核心之一原始碼中，尋找一存取該網路封包的函式，其中該存取該網路封包的函式為一直接存取該網路封包的函式或一間接存取該網路封包的函式。

較佳者，該方法之直接存取網路封包的函式係利用一已宣告的變數(Global Variable)、一參數型態名、一回傳型態名或一區域定義以直接存取該網路封包。

較佳者，該方法之間接存取網路封包的函式係利用一呼叫存取網路封包之函式(Indirect Caller)或一被存取網路封包之函式呼叫(Indirect Callee)，並藉由一傳址方式而間接得到一指向該網路封包的指標，以存取該網路封包。

若是從另一個可行的角度來看，本發明即為一種追蹤一網路封包之一處理程序的方法，其中該網路封包係在一系統核心內接受該處理程序，該方法包括如下步驟，提供存取該網路封包的一函式，嵌入一追蹤指令於該函式中，開始執行該函式，並進而觸發該追蹤指令，以及記錄該函式之一識別符，俾得以追蹤該處理程序。

當然，該方法的追蹤指令係可以為一工具原始碼(Instrument Source Code)，該識別符為一名稱或一代碼。

本發明經由上述構想的解說，即能看出所運用之追蹤網路封包的處理程序之方法，果能利用呼叫函式以傳遞該資料結構，而據以追蹤該處理程序，並具有定義該參數之

[S]

資料型態名進而藉搜尋該資料型態名以追蹤該處理程序之特色。為了易於說明，本發明得藉由下述之較佳實施例及圖示而更加清楚。

【實施方式】

本發明提出一種新的應用在網路通訊裝置中之追蹤網路封包處理程序的機制，該機制可藉由核心函式存取網路封包資料的順序，追蹤網路封包在核心系統的處理程序。本發明應用在網路通訊裝置中，可以利用網路封包的資料結構，及一個函式分析方法自動找出程式中有存取網路封包資料的函式，並在此些函式中嵌入指令，藉由程式本身執行函式的順序，依序觸發嵌入指令來記錄被執行的函式名稱或代碼與封包內容之相關資訊。本發明可以利用Linux核心系統內部用於管理與儲存網路封包的資料結構(sk_buff)，追蹤核心系統內部處理網路封包的行為與時間點。

利用這些紀錄，協助網路通訊軟體開發者很容易分析系統內部之網路通訊的系統行為，釐清與改善網路通訊(網路應用與核心協定或個別軟體函式)的問題，加速網路通訊軟體的開發時程與執行效能。亦即本發明可以追蹤存取網路封包的函式執行順序、追蹤系統內部處理網路封包的行為與時間點及不限定任何的裝置與系統版本。本案可能應用之產業包括資訊產業、網路通訊產業及雲端計算等，而可能應用之產品則為無線網路設備、手提電腦(Note- book)、PDA、手機(Handset)、網路存取閘道器、

[S]

網路家電與任何連網裝置等。

在此我們將以 Linux 核心系統為例，針對網路封包於 Linux 核心系統內部處理的過程，紀錄量度(measurement)資料，諸如各函式所做之內容改變或各函式之啟動(statring)時間，說明本發明提出的追蹤網路封包處理程序的機制之一個較佳實施(applying)例，此類記錄資訊在開發(developing)網路裝置上具有重要價值。但本發明應用並不侷限於此一環境，只要知道網路封包資料型態名，就可以套用本發明追蹤此系統處理該網路封包的程序。本發明應用於 Linux 核心系統中，利用 Linux 核心系統內部的網路封包之資料結構，追蹤 Linux 核心系統內部處理網路封包的行為與時間點，協助開發者分析 Linux 核心系統內部的網路行為。

本發明於 Linux 核心系統的實施例，主要增加兩個部份，A.追蹤核心系統內部存取網路封包的函式執行順序，以及 B.嵌入指令。且先就 A.部分而言，請參閱第二圖，可見一網路封包的一資料結構 sk_buff，其包括一管理資料(Management Data)及一封包資料儲存(Packet data storage)，本發明利用 Linux 核心系統內部用於管理與儲存網路封包的資料結構，作為尋找的基準點，找出存取網路封包的函式，而在 Linux 核心系統內部用於管理與儲存網路封包的資料結構即為 sk_buff。請參閱第三圖，顯示出一種追蹤一網路封包之一處理程序的方法，其中該網路封包係儲存在該資料結構內，且該網路封包在一系統核心

[S]

內接受該處理程序，該方法包括如下步驟，呼叫一函式以傳遞該資料結構，並據以追蹤該處理程序。而且存取網路封包的函式使用函式呼叫傳遞 sk_buff（如第三圖中的虛框線內所示者），可讓其餘的函式得以存取網路封包，因此本發明可以輕易實施於 Linux 核心系統之中。

該方法的作業系統為一 Linux，而該資料結構為一 sk_buff，該方法利用該系統核心以存取該網路封包，該方法更包含利用一特定函式以存取該網路封包，且該處理程序為該資料結構之一執行順序。

又按照一主要技術的觀點來看，請參閱第四圖，本發明可以涵蓋到一種追蹤網路封包之處理程序的方法，其中該網路封包係在一系統核心內接受該處理程序，該方法包括如下步驟，提供一函式，以處理該網路封包，其中該函式具一參數，定義該參數之一資料型態名，以及藉搜尋該資料型態名，而追蹤該處理程序。當然，此時的方法之資料型態名係可以作為一搜尋的基準點，而本發明利用此函式分析方法，包含一個尋找特定資料之型態名的方法，該特定資料之型態名包含原始型態與各種變形型態名。

亦即從 Linux 核心系統的原始碼之中，在尋找存取網路封包的函式時，不僅可以使用該資料型態名為一 sk_buff 此一特定資料之原始型態名，亦可以改用一變形型態名。請參閱第四圖 A，因為 C 語言擁有別名(Alias)、客製化資料型態(Customized data types)、巢狀資料結構(Nested Structures)等規則，故該方法的變形型態名可以為

一別名 41、一客製化資料型態名 42 或一巢狀資料結構 43，而變形型態名 43 係使其與該網路封包之一資料型態產生關聯。

由於 C 語言的函式可以透過許多種方式存取特定資料，因此本發明的函式分析方法包含一個尋找存取網路封包之函式的方法，該方法包含從該系統核心之一原始碼中，尋找一存取該網路封包的函式，其中該存取該網路封包的函式包含一直接存取該網路封包的函式或一間接存取該網路封包的函式。請參閱第四圖 B，該直接存取網路封包的函式係利用以下四種方式，一全域變數(亦稱為“已宣告的變數”，Global Variable)44、一函式的參數型態名(Parameter Type)45、一回傳型態名(Return Type)46或一區域定義(Local Definition)47，得以直接存取該網路封包。

另一方面，此間接存取特定資料的函式並未利用以上四種方式，請參閱第五圖，該間接存取網路封包的函式係利用一呼叫存取網路封包之函式(Indirect Caller)，以函式 E 呼叫直接存取網路封包之函式 F 為例，E 藉由一傳址(call by reference)方式，而間接從 F 得到一指向該網路封包的指標，因此使其得以存取該網路封包。或是如第六圖所示之一被存取網路封包之函式呼叫(Indirect Callee)，以 H 被直接存取網路封包之函式 G 呼叫為例，H 藉由 call by reference，而間接從 G 取得該指向網路封包的指標，使其因此得以存取該網路封包。

[S]

再就 B.部分而言，請參閱第七圖(其原始函式係位於第三圖)，若是從另一個可行的角度來看，本發明可以利用上述函式分析方法，找出程式中有存取網路封包的函式，並在這些函式中嵌入指令，亦即本發明為一種追蹤網路封包之處理程序的方法，其中該網路封包係在一系統核心內接受該處理程序，該方法包括如下步驟，提供存取該網路封包的一函式，嵌入一追蹤指令（例如：在第七圖中的大虛線框內者即為一工具原始碼(Instrument Source Code)70，是一插入追蹤技術的指令）於該函式中，開始執行該函式，並進而觸發該追蹤指令，以及記錄該函式之一識別符，俾得以追蹤該處理程序。

當然，此時的方法之該追蹤指令係可以為一工具原始碼，此些指令碼的功用係可以記錄下函式的執行順序以及所關心的系統內部資料或資源情況，提供開發者追蹤與分析系統的運作狀態，而該識別符為一名稱或一代碼或其他各式各樣的符號。本發明可以應用於網路裝置的開發，目前網路已大量建置，連網裝置也日漸普及，未來再加上雲端計算產業的發展，裝置連網會更加普遍，本發明可以提升網路通訊裝置的軟體開發效率與通訊效能，產業價值極大。

又本案進行檢索的關鍵字為：packet processing sequence、packet processing procedure、packet trace、packet flow、packet data type 及 kernel function 等。而當檢索之資料庫為美國專利商標局專利

[S]

資料庫檢索系統 (http://www.uspto.gov/)時，其結果如下：

1. "packet processing sequence" AND "packet data type": 0篇

2. "packet processing sequence" AND "kernel function": 0篇

3. "packet processing procedure" AND "packet data type": 0篇

4. "packet processing procedure" AND "kernel function": 0篇

5. "packet flow" AND "packet data type": 9篇

6. "packet flow" AND "kernel function": 7篇

7. "packet trace" AND "packet data type": 0篇

8. "packet trace" AND "kernel function": 0篇

謹將第5及第6次已檢索到之專利資料臚列如下：

1 US 7,453,801

Admission control and resource allocation in a communication system supporting application flows having quality of service requirements

2 US 7,305,511

Providing both wireline and wireless connections to a wireline interface

3 US 7,164,657

Intelligent collaboration across network systems

[S]

4 US 7,136,904

Wireless cable replacement for computer peripherals using a master adapter

5 US 7,127,541

Automatically establishing a wireless connection between adapters

6 US 6,963,955

Scattering and gathering data for faster processing

7 US 6,950,859

Wireless cable replacement for computer peripherals

8 US 6,894,972

Intelligent collaboration across network system

9 US 6,665,495

Non-blocking, scalable optical router architecture and method for routing optical traffic

10 US 7,685,254

Runtime adaptable search processor

11 US 7,631,107

Runtime adaptable protocol processor

12 US 7,627,693

IP storage processor and engine using RDMA

13 US 7,536,462

Memory system for a high performance IP processor

[S]

14 US 7,487,264

High performance IP processor

15 US 7,415,723

Distributed network security system and a hardware processor

16 US 7,376,755

TCP/IP processor and engine using RDMA

又當檢索之資料庫為歐洲專利局專利資料庫檢索系統 (http://ep.espacenet.com/)時，其結果如下：

1. "packet processing sequence" AND "packet data type": 0 篇

2. "packet processing sequence" AND "kernel function": 0 篇

3. "packet processing procedure" AND "packet data type": 0 篇

4. "packet processing procedure" AND "kernel function": 0 篇

5. "packet flow" AND "packet data type": 0 篇

6. "packet flow" AND "kernel function": 0 篇

7. "packet trace" AND "packet data type": 0 篇

8. "packet trace" AND "kernel function": 0 篇

將上述檢索的技術內容與本案進行比對後的結果，顯示出均與本發明無關或不同，並沒有與本發明類似的專利。

[S]

綜上所述，本發明確能以一嶄新的設計，藉由利用呼叫函式以傳遞該資料結構，而據以追蹤該處理程序，並且所定義該參數之資料型態名，能實質獲致藉由搜尋該資料型態名以追蹤該處理程序之功效。故凡熟習本技藝之人士，得任施匠思而為諸般修飾，然皆不脫如附申請專利範圍所欲保護者。

【圖式簡單說明】

第一圖：是習知的原始工具類之 Kernprof 核心函式的追蹤畫面；

第二圖：是本發明的追蹤網路封包之處理程序的方法所利用之資料結構的結構圖；

第三圖：是本發明之使用函式呼叫以傳遞資料結構的較佳實施例之程式設計圖；

第四圖 A：是本發明之使用 C 語言的別名、客製化資料型態及巢狀資料結構等規則之較佳實施例的程式設計圖；

第四圖 B：是直接存取網路封包的函式之全域變數、函式的參數型態名、回傳型態名及區域定義等程式圖；

第五圖：是利用呼叫存取網路封包之函式的程式設計圖；

第六圖：是利用被存取網路封包之函式呼叫的程式設計圖；以及

第七圖：是第三圖之嵌入指令結果的較佳實施例之程式設計圖。

【主要元件符號說明】

41：別名
42：客製化資料型態名
43：巢狀資料結構
44：全域變數
45：參數型態名
46：回傳型態名
47：區域定義
70：工具原始碼

七、申請專利範圍：

1.一種追蹤一網路封包(Network Packet)之一處理程序的方法，其中該網路封包係儲存在一資料結構內，且該網路封包在一系統核心(Kernel)內接受該處理程序，該方法包括如下步驟：

呼叫一函式(function)以傳遞該資料結構，並據以追蹤該處理程序。

2.如申請專利範圍第 1 項所述之方法，其中該系統為一 Linux，而該資料結構為一 sk_buff，該方法利用該系統核心以存取該網路封包，該方法更包含利用一特定函式以存取該網路封包，且該處理程序為該資料結構之一執行順序。

3.一種追蹤一網路封包之一處理程序的方法，其中該網路封包係在一系統核心內接受該處理程序，該方法包括如下步驟：

提供一函式，以處理該網路封包，其中該函式具一參數；

定義該參數之一資料型態名；以及

藉搜尋該資料型態名，而追蹤該處理程序。

4.如申請專利範圍第 3 項所述之方法，其中該資料型態名係為一搜尋的基準點，而該資料型態名為一 sk_buff 之原始型態名或一變形型態名。

5.如申請專利範圍第 4 項所述之方法，其中該變形型態名為一別名、一客製化資料型態名或一巢狀資料結構，而該

[S]

變形型態名係使其與該網路封包之一資料型態產生關聯。

6.如申請專利範圍第3項所述之方法，更包含從該系統核心之一原始碼中，尋找一存取該網路封包的函式，其中該存取該網路封包的函式為一直接存取該網路封包的函式或一間接存取該網路封包的函式。

7.如申請專利範圍第6項所述之方法，其中該直接存取網路封包的函式係利用一已宣告的變數(Global Variable)、一參數型態名、一回傳型態名或一區域定義以直接存取該網路封包。

8.如申請專利範圍第6項所述之方法，其中該間接存取網路封包的函式係利用一呼叫存取網路封包之函式(Indirect Caller)或一被存取網路封包之函式呼叫(Indirect Callee)，並藉由一傳址方式而間接得到一指向該網路封包的指標，以存取該網路封包。

9.一種追蹤一網路封包之一處理程序的方法，其中該網路封包係在一系統核心內接受該處理程序，該方法包括如下步驟：

提供存取該網路封包的一函式；

嵌入一追蹤指令於該函式中；

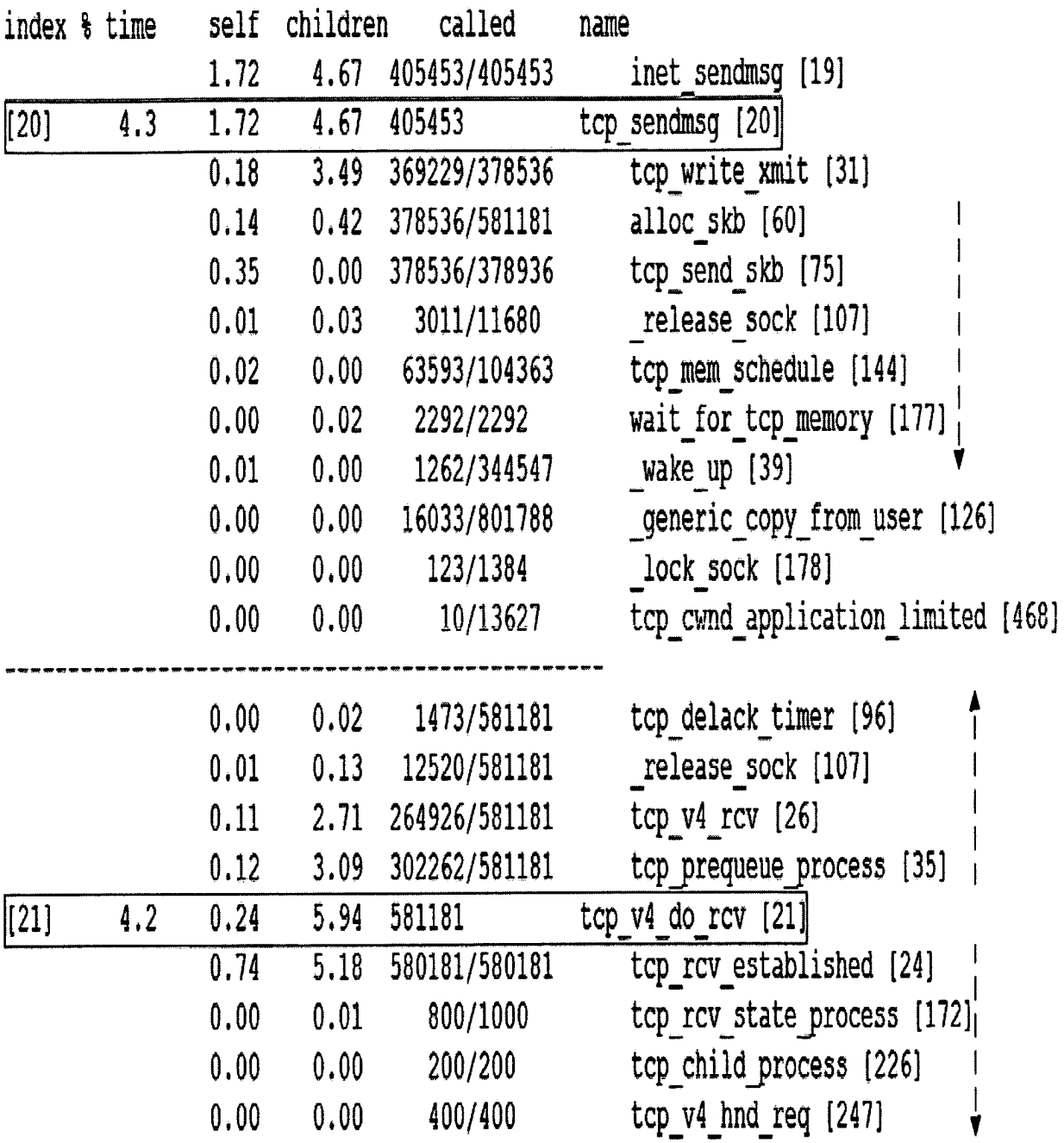
開始執行該函式，並進而觸發該追蹤指令；以及

記錄該函式之一識別符，俾得以追蹤該處理程序。

10.如申請專利範圍第9項所述之方法，其中該追蹤指令係為一工具原始碼(Instrument Source Code)，該識別符為一名稱或一代碼。

[S]

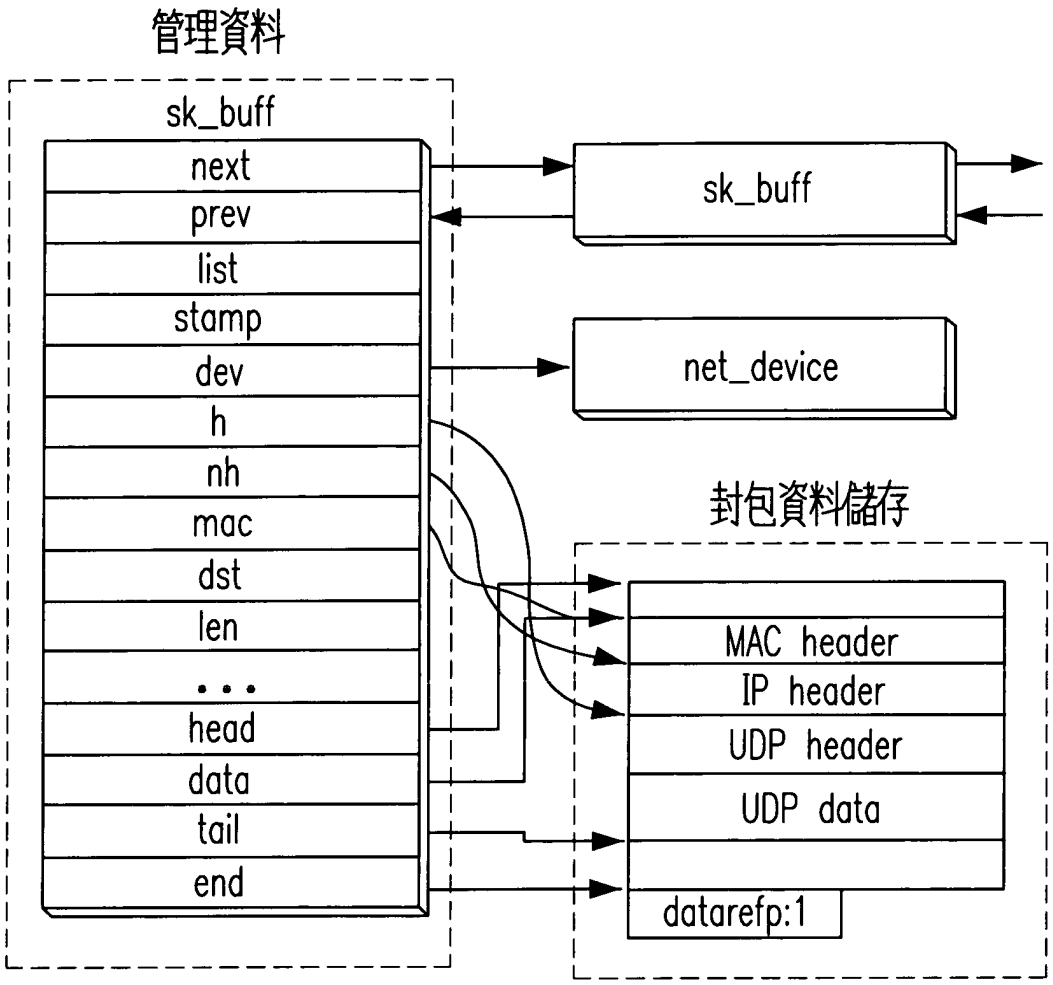
八、圖式：



```
index % time    self  children    called     name
                1.72    4.67  405453/405453      inet_sendmsg [19]
[20]     4.3    1.72    4.67  405453           tcp_sendmsg [20]
                0.18    3.49  369229/378536      tcp_write_xmit [31]
                0.14    0.42  378536/581181      alloc_skb [60]
                0.35    0.00  378536/378936      tcp_send_skb [75]
                0.01    0.03    3011/11680       _release_sock [107]
                0.02    0.00   63593/104363      tcp_mem_schedule [144]
                0.00    0.02    2292/2292        wait_for_tcp_memory [177]
                0.01    0.00    1262/344547      _wake_up [39]
                0.00    0.00   16033/801788      _generic_copy_from_user [126]
                0.00    0.00     123/1384        _lock_sock [178]
                0.00    0.00      10/13627       tcp_cwnd_application_limited [468]
-----------------------------------------------
                0.00    0.02    1473/581181      tcp_delack_timer [96]
                0.01    0.13   12520/581181      _release_sock [107]
                0.11    2.71  264926/581181      tcp_v4_rcv [26]
                0.12    3.09  302262/581181      tcp_prequeue_process [35]
[21]     4.2    0.24    5.94  581181           tcp_v4_do_rcv [21]
                0.74    5.18  580181/580181      tcp_rcv_established [24]
                0.00    0.01     800/1000        tcp_rcv_state_process [172]
                0.00    0.00     200/200         tcp_child_process [226]
                0.00    0.00     400/400         tcp_v4_hnd_req [247]
```

第一圖

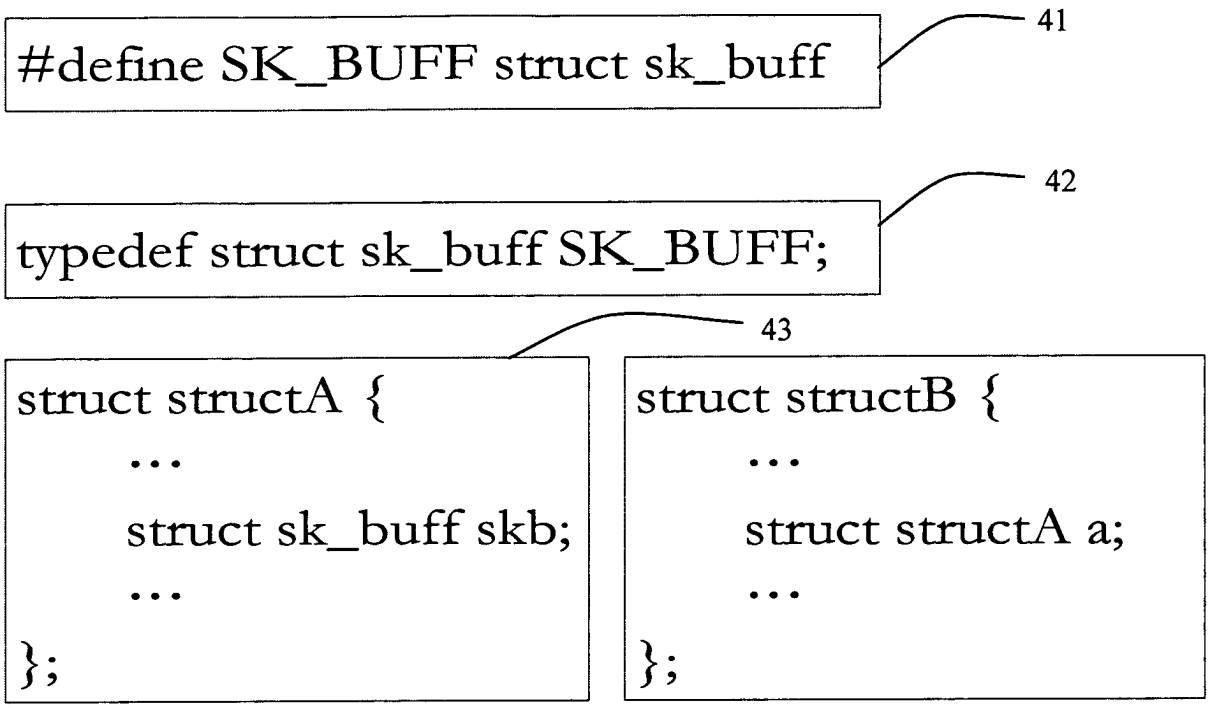
[S]


管理資料
sk_buff
next
prev
list
stamp
dev
h
nh
mac
dst
len
. . .
head
data
tail
end
sk_buff
net_device
封包資料儲存
MAC header
IP header
UDP header
UDP data
datarefp:1

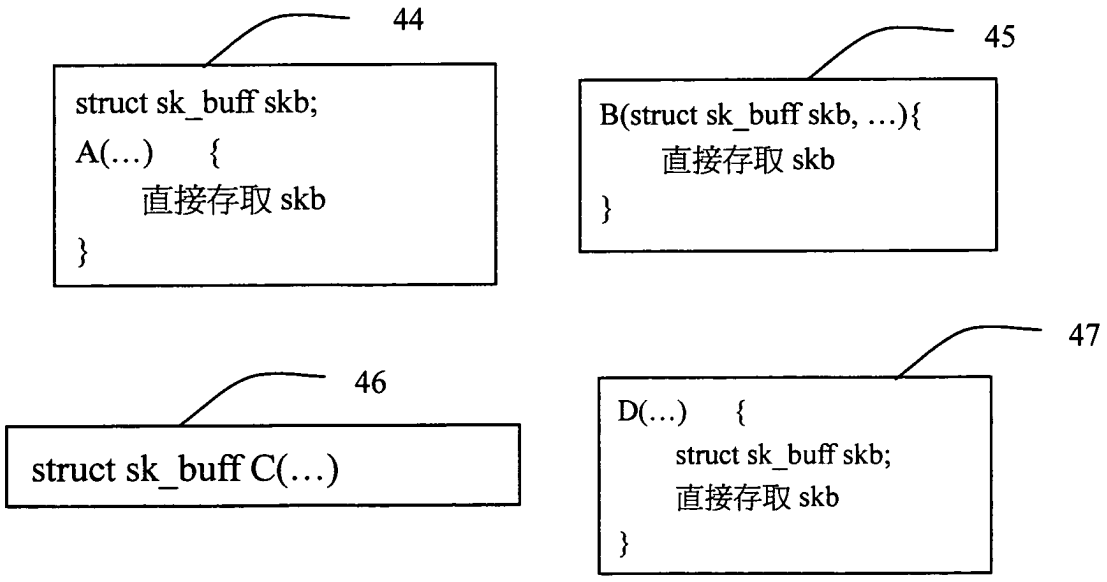
第二圖

```
int ip_build_and_send_pkt(struct sk_buff *skb, struct sock *sk,
              __be32 saddr, __be32 daddr, struct ip_options *opt)
{
    struct inet_sock *inet = inet_sk(sk);
    struct rtable *rt = skb->rtable;
    struct iphdr *iph;

    /* Build the IP header. */
    skb_push(skb, sizeof(struct iphdr) + (opt ? opt->optlen : 0));
    skb_reset_network_header(skb);
    iph = ip_hdr(skb);
    iph->version  = 4;
    iph->ihl      = 5;
    iph->tos      = inet->tos;
    if (ip_dont_fragment(sk, &rt->u.dst))
        iph->frag_off = htons(IP_DF);
    else
        iph->frag_off = 0;
    iph->ttl      = ip_select_ttl(inet, &rt->u.dst);
    iph->daddr    = rt->rt_dst;
    iph->saddr    = rt->rt_src;
    iph->protocol = sk->sk_protocol;
    ip_select_ident(iph, &rt->u.dst, sk);

    if (opt && opt->optlen) {
        iph->ihl += opt->optlen>>2;
        ip_options_build(skb, opt, daddr, rt, 0);
    }

    skb->priority = sk->sk_priority;
    skb->mark = sk->sk_mark;

    /* Send it out. */
    return ip_local_out(skb);
}
```

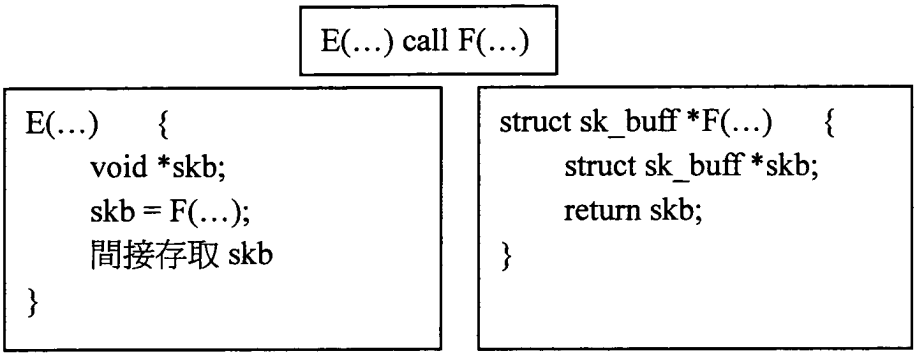
第三圖


41
#define SK_BUFF struct sk_buff
42
typedef struct sk_buff SK_BUFF;
43
struct structA {
...
struct sk_buff skb;
...
};
struct structB {
...
struct structA a;
...
};

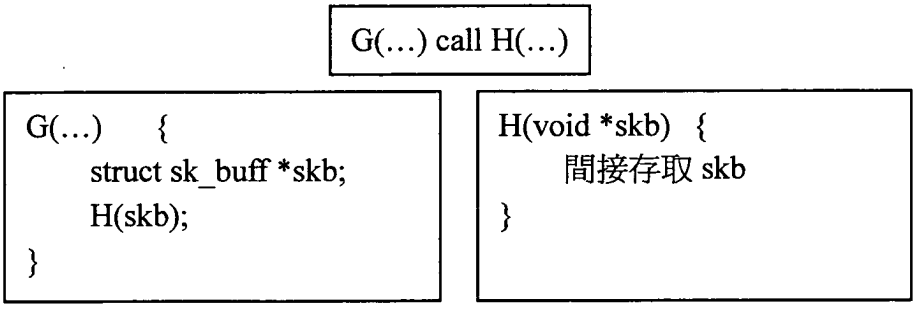
第四圖 A


44
struct sk_buff skb;
A(...) {
直接存取 skb
}
45
B(struct sk_buff skb, ...){
直接存取 skb
}
46
struct sk_buff C(...)
47
D(...) {
struct sk_buff skb;
直接存取 skb
}

第四圖 B


E(…) call F(…)
E(…) {
void *skb;
skb = F(…);
間接存取 skb
}
struct sk_buff *F(…) {
struct sk_buff *skb;
return skb;
}

第五圖


G(…) call H(…)
G(…) {
struct sk_buff *skb;
H(skb);
}
H(void *skb) {
間接存取 skb
}

第六圖

```
int ip_build_and_send_pkt(struct sk_buff *skb, struct sock *sk,
              __be32 saddr, __be32 daddr, struct ip_options *opt)
{
    struct timeval tv;
    do_gettimeofday(&tv);
    printk("Entry time: [%ld][%ld] function: [%p], skb: [%p]\n",
        tv.tv_sec, tv.tv_usec, __FUNCTION__, skb);

    struct inet_sock *inet = inet_sk(sk);
    struct rtable *rt = skb->rtable;
    struct iphdr *iph;

    /* Build the IP header. */
    skb_push(skb, sizeof(struct iphdr) + (opt ? opt->optlen : 0));
    skb_reset_network_header(skb);
    iph = ip_hdr(skb);
    iph->version  = 4;
    iph->ihl      = 5;
    iph->tos      = inet->tos;
    if (ip_dont_fragment(sk, &rt->u.dst))
        iph->frag_off = htons(IP_DF);
    else
        iph->frag_off = 0;
    iph->ttl      = ip_select_ttl(inet, &rt->u.dst);
    iph->daddr    = rt->rt_dst;
```

70

第七圖