那些年，我們忽略的socket參數

原創 187J3X1 2019-08-17 16:45

調試過網絡程序的人大多使用過tcpdump，那你知道tcpdump是如何工作的嗎？

tcpdump這類工具也被稱爲Sniffer，它可以在不影響應用程序正常報文的情況下，將流經網卡的報文**複製**一份給Sniffer，然後經過加工過濾，最後呈現給用戶。

本文不分析tcpdump的具體實現，而只是借tcpdump來揭示一些網絡編程中一個大多數人都容易忽略的一個主題：Socket參數對用戶接收報文的影響...

相信所有接觸過Socket編程的人都應該認識下面這個API

```
#include <sys/socket.h>
sockfd = socket(int socket_family, int socket_type, int protocol);
```

沒錯，它基本是socket編程的第一步，創建一個套接字。他有三個參數，不過又有多少人真的去了解這些參數的意義呢？ 對於TCP或者UDP應用的開發者來說，他們可以很容易地從互聯網上**找(抄)**到這樣的例子：

```
/* 創建TCP socket*/
sockfd = socket(AF_INET, SOCK_STREAM, 0);

/* 創建UDP socket*/
sockfd = socket(AF_INET, SOCK_DGRAM, 0)
```

爲什麼第一個參數要使用AF_INET,爲什麼第二個參數要使用SOCK_STREAM或者SOCK_DGRAM，爲什麼第三個參數要填0？

socket_family

第一個參數表示創建的socket所屬的地址簇或者協議簇，取值以AF或者PF開頭定義在(include\linux\socket.h)，實際使用中並沒有區別(有兩個不同的名字只是因爲是歷史上的設計原因)。最常用的取值有AF_INET,AF_PACKET,AF_UNIX等。AF_UNIX用於主機內部進程間通信，本文暫且不談。AF_INET與AF_PACKET的區別在於使用**前者只能看到IP層以上的東西，而後者可以看到鏈路層的信息**。

什麼意思呢？ 爲了說明這個問題，我們需要知道網絡報文的分類。如下圖所示：Ethernet II幀是應用最爲廣泛的幀類型(當然也有像PPP這樣的其他鏈路幀類型)。Ethernet II幀內部，又可大致分爲IP報文和其他報文。我們熟悉的TCP或者UDP報文都屬於IP報文。

24小時熱門

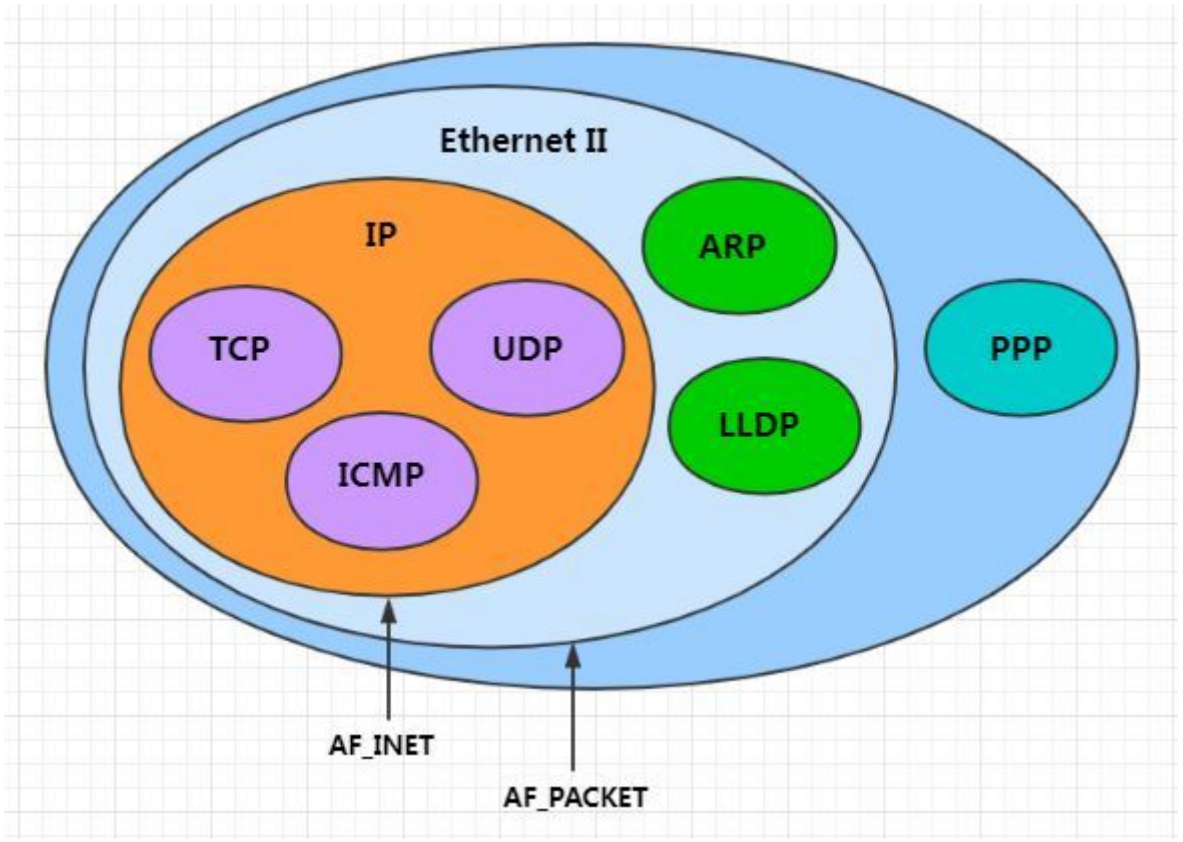
沛星

最新文章

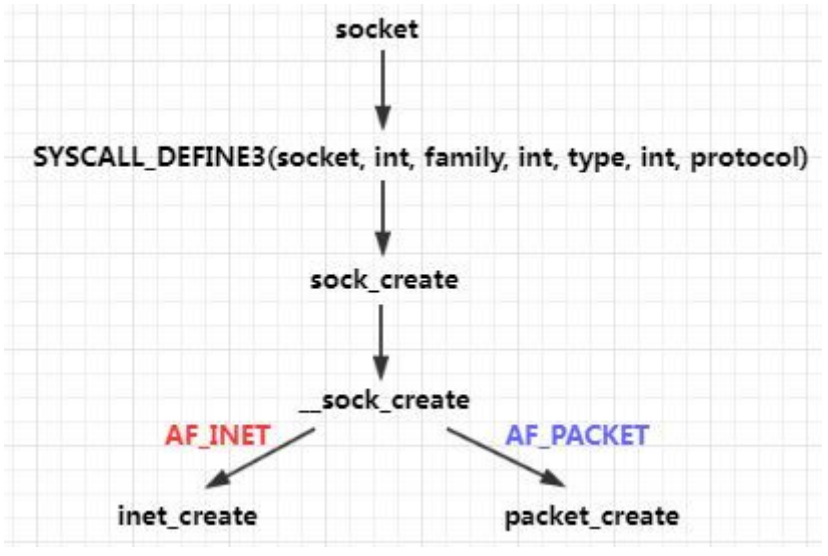
TCP的重傳退
隧道而言, TCP
麼
TCP Metrics--
ation timestar
Linux 路由緩存
Cisco思科網絡
ugin 實現原理

最新評論文

Java 認證考試
豌豆莢APP 行
Studio 3T for
算法註冊機
ZUKEN軟件CR
18破解教程 (
看美劇練口語
dernFamily) S
win10 藍屏分
wpkclnt+1361
Windows7 開
極解決方案
中區約炮 line
d08080 台中砲
雅找美女
心血來潮拆華
算升級)
配置Visual Stu
片機C51代碼編
編輯代碼事半



AF_INET是與IP報文對應的，而AF_PACKET則是與Ethernet II報文對應的。AF_INET創建的套接字稱爲inet socket，而AF_PACKET創建的套接字稱爲packet socket



socket_type & protocol

第一個參數family會影響第二個參數socket_type和第三個參數protocol取值範圍

第二個參數socket_type表示套接字類型。它的取值不多，常見的就以下三種

```
enum sock_type {
    SOCK_STREAM    = 1,     /* stream (connection) socket  */
    SOCK_DGRAM     = 2,     /* datagram (conn.less) socket */
    SOCK_RAW       = 3,     /* raw socket                  */
};
```

第三個參數protocol表示套接字上報文的協議。

對於AF_INET地址簇，protocol的取值範圍是如 **IPPROTO_TCP IPPROTO_UDP IPPROTO_ICMP** 這樣的IP報文協議類型，或者**IPPROTO_IP = 0** 這個特殊值

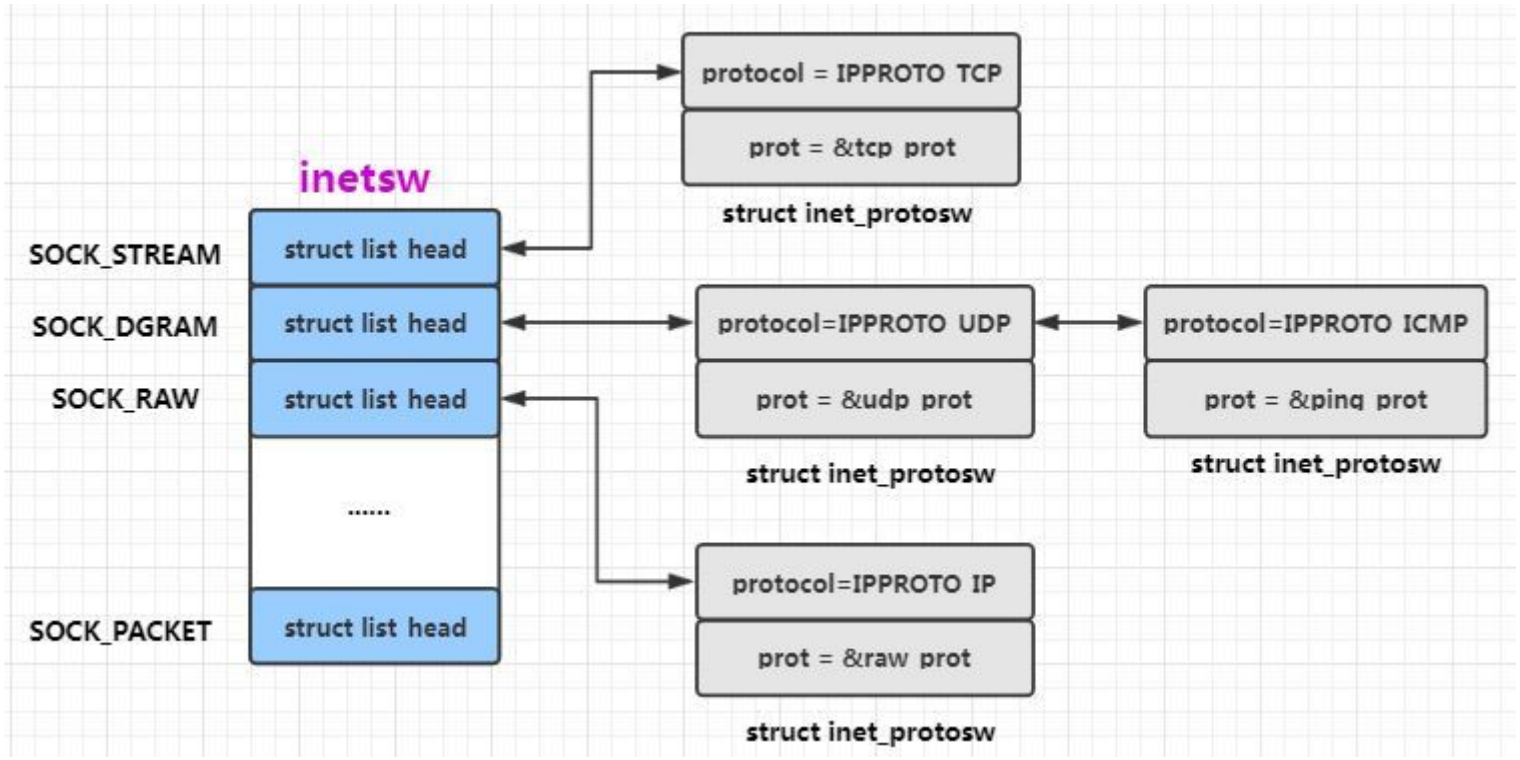
對於AF_PACKET地址簇，protocol的取值範圍是 **ETH_P_IP ETH_P_ARP**這樣的以太幀協議類型。

inet socket的協議開關表

每一個inet socket只能收發一種IP協議類型的報文，這是在套接字創建的時候就決定的(protocol參數)，比如TCP套接字是不能收發UDP報文的，反之也是一樣。並且，protocol的值還受到socket_type的限制，不匹配的取值會導致套接字創建操作會返回失敗。

```
/* 錯誤取值，返回失敗 */
sockfd = socket(AF_INET, SOCK_DGRAM, IPPROTO_TCP);
```

內核通過協議開關表記錄了哪些哪些取值是有效的，inet在初始化時會將支持的協議註冊在協議開關表中的以socket_type爲KEY的鏈表上：



而在創建套接字時，inet_create會在協議開關表中根據socket_type和protocol進行匹配

```
list_for_each_entry_rcu(answer, &inetsw[sock->type], list) {
        err = 0;
        /* Check the non-wild match. */
        if (protocol == answer->protocol) {
            if (protocol != IPPROTO_IP)
                break;
        } else {
            /* Check for the two wild cases. */
            if (IPPROTO_IP == protocol) {
                protocol = answer->protocol;
                break;
            }
            if (IPPROTO_IP == answer->protocol)
                break;
        }
        err = -EPROTONOSUPPORT;
    }
```

IPPROTO_IP的值爲0, 在用戶使用0作爲創建套接字的第三個參數時，會匹配到該鏈表上的第一個協議，這正是創建TCP或者UDP套接字時，第三個參數可以爲0的原因, 0表示由內核自動選擇。..

```
/* 創建TCP socket*/
sockfd = socket(AF_INET, SOCK_STREAM, 0);

/* 創建UDP socket*/
sockfd = socket(AF_INET, SOCK_DGRAM, 0)
```

raw inet socket

對於inet socket來說，一個TCP報文可以這樣分解：

```
packet = IP Header + TCP Header +  Payload
```

如果我們是使用SOCK_STREAM創建的TCP套接字，應用程序在通過send發送數據時，只需要提供Payload就行了，而IP Header和TCP Header則由內核組裝完成。接收方向，應用程序通過recv也只能收到payload

而RAW套接字則爲應用提供了更底層的控制能力

```
int s = socket (AF_INET, SOCK_RAW, IPPROTO_TCP);
```

使用上面的接口可以創建一個更原始的TCP套接字，當我們使用這個套接字發送數據時，需要提供Payload和TCP Header，而IP Header依然由內核協議棧自動組裝。

如果希望手動組裝IP Header，有兩個方法：

第一種是protocol使用IPPROTO_RAW

```
int s = socket (AF_INET, SOCK_RAW, IPPROTO_RAW);
```

第二種是置位IP_HDRINCL的套接字選項。

```
int s = socket (AF_INET, SOCK_RAW, IPPROTO_TCP);

int one = 1;
const int *val = &amp;one;
if (setsockopt (s, IPPROTO_IP, IP_HDRINCL, val, sizeof (one)) &lt; 0)
{
    printf (&quot;Error setting IP_HDRINCL. Error number : %d . Error message : %s \n&quot; ,
errno , strerror(errno));
    exit(0);
}
```

以上兩種方法都是告訴內核，IP Header也由應用程序自己提供。

packet socket

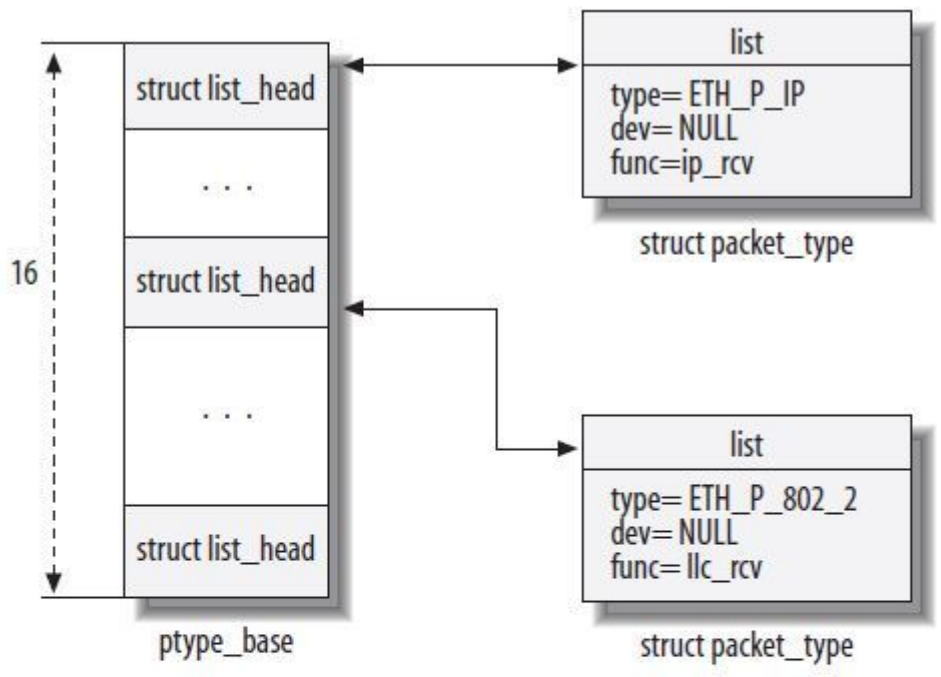
inet socket的控制範圍是IP報文，而packet socket的控制範圍擴大到了以太層報文。

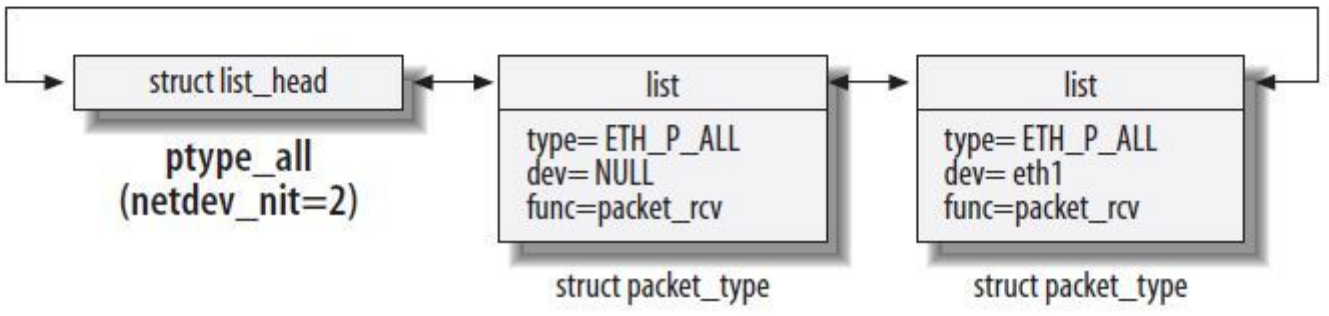
對於inet socket, 第二個參數socket_type只能選擇SOCK_DGRAM、SOCK_RAW或者SOCK_PACKET, protocol則表示支持的網絡層的協議類型。

Protocol Handler

對以太幀來說，不同的網絡層協議類型(比如IP ARP PPPoE)有不同的接收處理函數。在內核中，這就是Protocol Handler。

內核中的Protocl Handler是這樣組織的[注]：





注該patch將Protocl Handler在dev下增加了ptype_all鏈表和ptype_base鏈表

無論網卡是否採用NAPI，內核最終都會調用到__netfi_receive_skb接收報文，這個函數會遍歷ptype_all鏈表上已註冊的handler，然後再遍歷ptype_base特定協議鏈上的所有已註冊的handler

handler的註冊是通過dev_add_pack完成的,如果沒有指定協議(ETH_P_ALL)，該handler就會註冊在ptype_all上(tcpdump默認就會註冊在這裏)，否則根據協議註冊在ptype_base的某條鏈表上。

在報文接收過程中，**同一個skb會被deliver_skb到多個handler**(至少將ptype_all鏈表上的handler走一遍)。

內核啓動時，inet會註冊一個handler，它支持IP協議，**所有AF_INET套接字實際上是共用這樣一個handler**，對應的接收函數是ip_rcv，區分是哪一個套接字的報文是之後的工作。

```
/* net/ipv4/af_inet.c */
static struct packet_type ip_packet_type __read_mostly = {
    .type = cpu_to_be16(ETH_P_IP),
    .func = ip_rcv,
};

static int __init inet_init(void)
{
    // code omitted
    dev_add_pack(&ip_packet_type);
    // code omitted
}
```

而對於AF_PACKET，handler是在packet_create中單獨註冊的，也就是說，**每個AF_PACKET套接字擁有獨立的handler**

```
static int packet_create(struct net *net, struct socket *sock, int protocol,
             int kern)
{
    // code omitted
    po->prot_hook.func = packet_rcv;
    // code omitted
    register_prot_hook(sk);  // 這裏面去 dev_add_pack
}
```

單獨的handler，使得在接收函數packet_rcv的時候，就已經可以知道這是屬於哪一個套接字的數據了。

raw packet socket

對於AF_PACKET來說，一個報文可以這樣分解：

```
packet = Ethernet Header + Payload
```

而SOCK_DGRAM和SOCK_RAW的區別就在於，在接收方向，使用SOCK_DGRAM套接字的應用程序收到的報文已經去除了Ethernet Header，而SOCK_RAW套接字則會保留。

packet socket 與 tcpdump

回到本文最初的問題，tcpdump是如何完成嗅探工作的呢？ 沒錯！它正是使用的packet socket：

- tcpdump作爲sniffer，它不能影響正常的報文收發，因此它需要單獨的protocol handler，這樣內核接收的報文會複製一份後，交給tcpdump
- tcpdump不止能抓取IP報文, 它還可以抓起鏈路層信息或者其他一些非IP報文。

REF

difference-between-pf-inet-sockets-and-pf-packet

data-link-access-and-zero-copy

raw-socket-in-linux

raw-sockets-c-code-linux

tcp linux socket

發表評論

登錄以後才評論...

登 录