用戶態tcpdump 如何實現抓到內核網絡包的?

計算機網絡工程師 今天

以下文章來源於開發內功修煉

開發內功修煉

飛哥有鵝廠、搜狗10 年多的開發工作經驗。通過本號，我把多年中對於性能的一些深度思考分享給大家。

> **來自公眾號:**

大家好，我是飛哥！

今天聊聊大家工作中經常用到的tcpdump。

在網絡包的發送和接收過程中，絕大部分的工作都是在內核態完成的。那麼問題來了，我們常用的運行在用戶態的程序tcpdump 是那如何實現抓到內核態的包的呢？有的同學知道tcpdump 是基於libpcap 的，那麼libpcap 的工作原理又是啥樣的呢。如果讓你裸寫一個抓包程序，你有沒有思路？

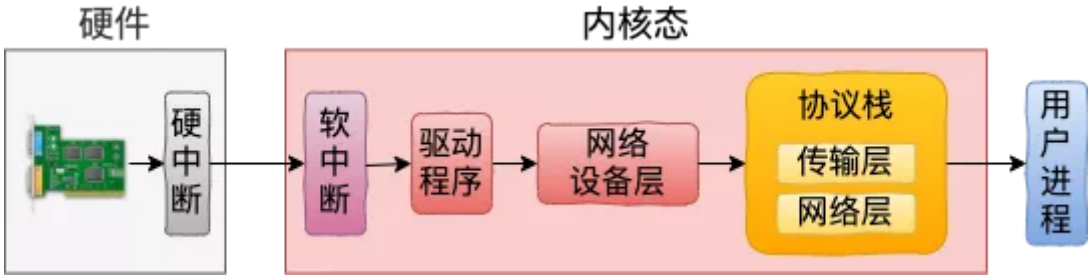
按照飛哥的風格，不搞到最底層的原理咱是不會罷休的。所以我對相關的源碼進行了深入分析。通過本文，你將徹底搞清楚了以下這幾個問題。

- tcpdump 是如何工作的？
- netfilter 過濾的包tcpdump 是否可以抓的到？
- 讓你自己寫一個抓包程序的話該如何下手？

借助這幾個問題，我們來展開今天的探索之旅！

一、網絡包接收過程

在 圖解Linux網絡包接收過程 這個過程我們可以簡單用如下這個圖來表示。



找到tcpdump 抓包點

我們在網絡設備層的代碼裡找到了tcpdump 的抓包入口。在__netif_receive_skb_core 這個函數里會遍歷ptype_all 上的協議。還記得上文中我們提到tcpdump 在ptype_all 上註冊了虛擬協議。這時就能執行的到了。來看函數：

```
//file: net/core/dev.c
static int __netif_receive_skb_core(struct sk_buff *skb, bool pfmemalloc)
{
    ......
    //遍历 ptype_all （tcpdump 在这里挂了虚拟协议）
    list_for_each_entry_rcu(ptype, &ptype_all, list) {
        if (!ptype->dev || ptype->dev == skb->dev) {
            if (pt_prev)
                ret = deliver_skb(skb, pt_prev, orig_dev);
            pt_prev = ptype;
        }
```

```
    }
}
```

在上面函數中遍歷ptype_all，並使用deliver_skb 來調用協議中的回調函數。

```
//file: net/core/dev.c
static inline int deliver_skb(...)
{
 return pt_prev->func(skb, skb->dev, pt_prev, orig_dev);
}
```

對於tcpdump 來說，就會進入packet_rcv 了（後面我們再說為啥是進入這個函數）。這個函數在net/packet/af_packet.c 文件中。

```
//file: net/packet/af_packet.c
static int packet_rcv(struct sk_buff *skb, ...)
{
 __skb_queue_tail(&sk->sk_receive_queue, skb);
 ......
}
```

可見packet_rcv 把收到的skb 放到了當前packet socket 的接收隊列裡了。這樣後面調用recvfrom 的時候就可以獲取到所抓到的包！！

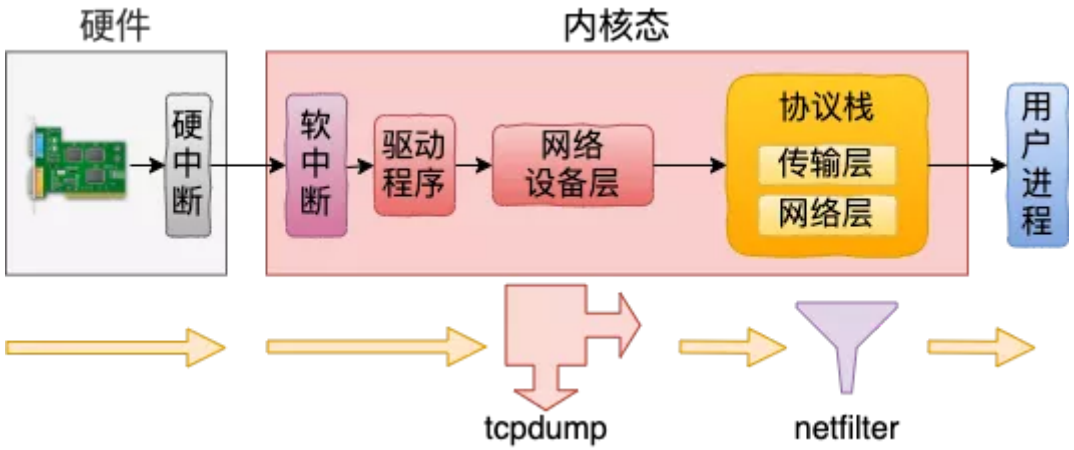
再找netfilter 過濾點

為了解釋我們開篇中提到的問題，這裡我們再稍微到協議層中多看一些。在ip_rcv 中我們找到了一個netfilter 相關的執行邏輯。

```
//file: net/ipv4/ip_input.c
int ip_rcv(...)
{
 ......
 return NF_HOOK(NFPROTO_IPV4, NF_INET_PRE_ROUTING, skb, dev, NULL,
        ip_rcv_finish);
}
```

如果你用NF_HOOK 作為關鍵詞來搜索，還能搜到不少netfilter 的過濾點。不過所有的過濾點都是位於IP 協議層的。

在接收包的過程中，數據包是先經過網絡設備層然後才到協議層的。

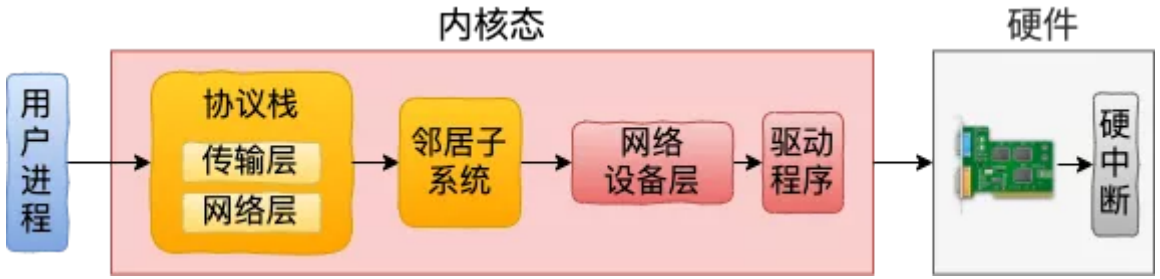


那麼我們開篇中的一個問題就有了答案了。假如我們設置了netfilter 規則，在接收包的過程中，工作在網絡設備層的tcpdump 先開始工作。還沒等netfilter 過濾，tcpdump 就抓到包了！

所以，在接收包的過程中，netfilter 過濾並不會影響tcpdump 的抓包！

二、網絡包發送過程

我們接著再來看網絡包發送過程。在 25 張圖，一萬字，拆解Linux 網絡包發送過程 發送過程可以匯總成簡單的一張圖。



找到netfilter 過濾點

在發送的過程中，同樣是在IP 層進入各種netfilter 規則的過濾。

```
//file: net/ipv4/ip_output.c
int ip_local_out(struct sk_buff *skb)
{
 //执行 netfilter 过滤
 err = __ip_local_out(skb);
}

int __ip_local_out(struct sk_buff *skb)
{
 ......
 return nf_hook(NFPROTO_IPV4, NF_INET_LOCAL_OUT, skb, NULL,
```

```
        skb_dst(skb)->dev, dst_output);
}
```

在這個文件中，還能看到若干處netfilter 過濾邏輯。

找到tcpdump 抓包點

發送過程在協議層處理完畢到達網絡設備層的時候，也有tcpdump 的抓包點。

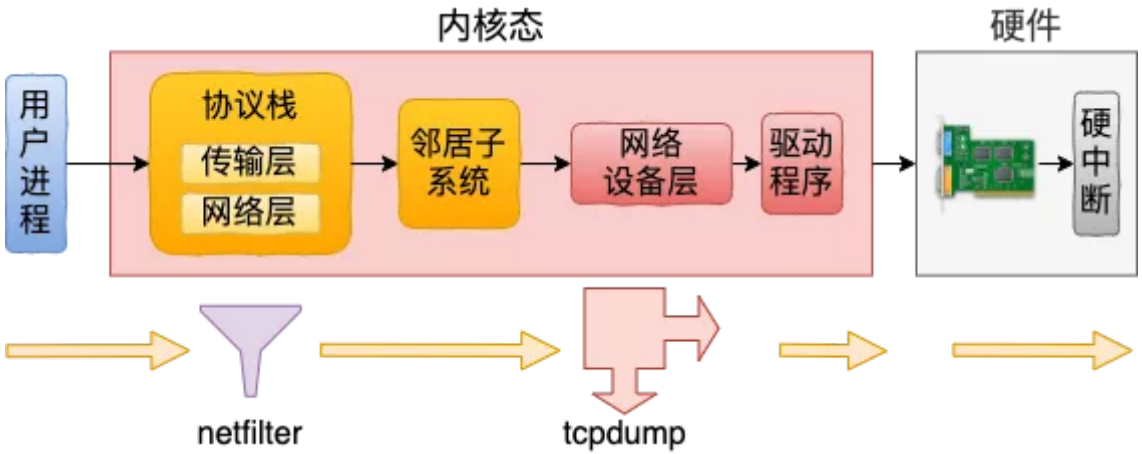
```
//file: net/core/dev.c
int dev_hard_start_xmit(struct sk_buff *skb, struct net_device *dev,
   struct netdev_queue *txq)
{
 ...
 if (!list_empty(&ptype_all))
  dev_queue_xmit_nit(skb, dev);
}

static void dev_queue_xmit_nit(struct sk_buff *skb, struct net_device *dev)
{
 list_for_each_entry_rcu(ptype, &ptype_all, list) {
  if ((ptype->dev == dev || !ptype->dev) &&
      (!skb_loop_sk(ptype, skb))) {
   if (pt_prev) {
    deliver_skb(skb2, pt_prev, skb->dev);
    pt_prev = ptype;
    continue;
   }
```

```
    ......
  }
 }
}
```

在上述代碼中我們看到，在dev_queue_xmit_nit 中遍歷ptype_all 中的協議，並依次調用deliver_skb。這就會執行到tcpdump 掛在上面的虛擬協議。

在網絡包的發送過程中，和接收過程恰好相反，是協議層先處理、網絡設備層後處理。



如果netfilter 設置了過濾規則，那麼在協議層就直接過濾掉了。在下層網絡設備層工作的tcpdump 將無法再捕獲到該網絡包

三、TCPDUMP 啟動

前面兩小節我們說到了內核收發包都通過遍歷ptype_all 來執行抓包的。那麼我們現在來看看用戶態的tcpdump 是如何掛載協議到內ptype_all 上的。

我們通過strace 命令我們抓一下tcpdump 命令的系統調用，顯示結果中有一行socket 系統調用。Tcpdump 秘密的源頭就藏在這行對socket 函數的調用裡。

```
# strace tcpdump -i eth0
socket(AF_PACKET, SOCK_RAW, 768)
......
```

socket 系統調用的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。在Linux 裡，支持很多種協議族，在include/linux/socket.h 中可以找到所有的定義。這裡創建的是packet 類型的socket。

> 協議族和地址族：每一種協議族都有其對應的地址族。比如IPV4 的協議族定義叫PF_INET，其地址族的定義是AF_INET。它們是一一對應的，而且值也完全一樣，所以經常混用。

```
//file: include/linux/socket.h
#define AF_UNSPEC 0
#define AF_UNIX  1 /* Unix domain sockets   */
#define AF_LOCAL 1 /* POSIX name for AF_UNIX */
#define AF_INET  2 /* Internet IP Protocol  */
#define AF_INET6 10 /* IP version 6   */
#define AF_PACKET 17 /* Packet family  */
......
```

另外上面第三個參數768 代表的是ETH_P_ALL，socket.htons(ETH_P_ALL) = 768。

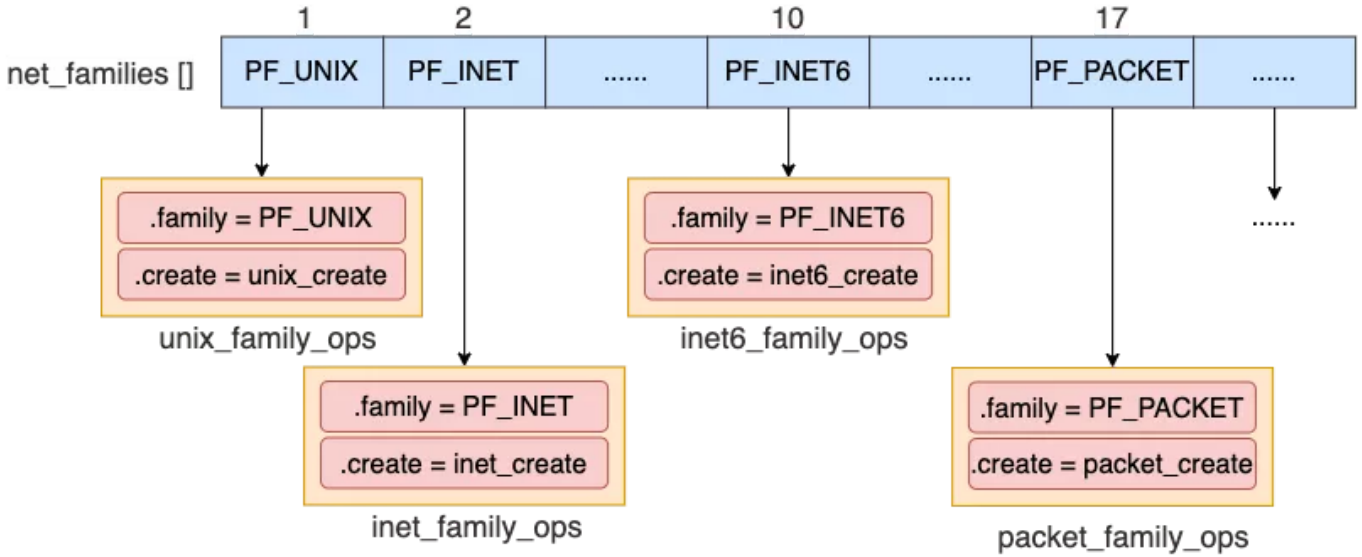
我們來展開看這個packet 類型的socket 創建的過程中都乾了啥，找到socket 創建源碼。

```
//file: net/socket.c
SYSCALL_DEFINE3(socket, int, family, int, type, int, protocol)
{
 ......
 retval = sock_create(family, type, protocol, &sock);
}

int __sock_create(struct net *net, int family, int type, ...)
{
 ......
 pf = rcu_dereference(net_families[family]);
 err = pf->create(net, sock, protocol, kern);
}
```

在__sock_create 中，從net_families 中獲取了指定協議。並調用了它的create 方法來完成創建。

net_families 是一個數組，除了我們常用的PF_INET（ ipv4 ） 外，還支持很多種協議族。比如PF_UNIX、PF_INET6（ipv6）、PF_PACKET等等。每一種協議族在net_families 數組的特定位置都可以找到其family 類型。在這個family 類型裡，成員函數create 指向該協議族的對應創建函數。



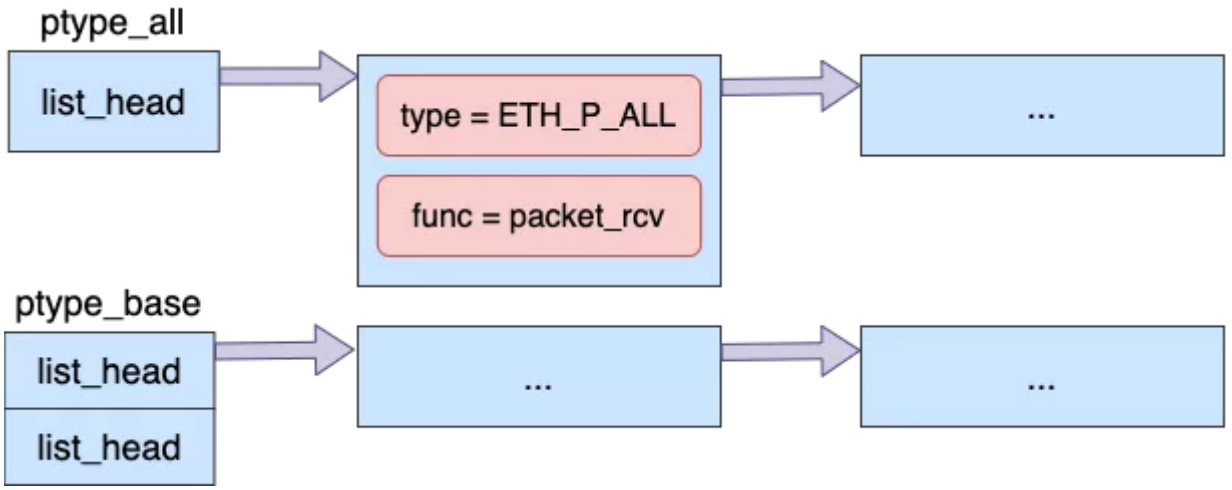
根據上圖，我們看到對於packet 類型的socket，pf->create 實際調用到的是packet_create 函數。我們進入到這個函數中來一探究竟，這是理解tcpdump 工作原理的關鍵！

```
//file: packet/af_packet.c
static int packet_create(struct net *net, struct socket *sock, int protocol,
    int kern)
{
 ...
 po = pkt_sk(sk);
 po->prot_hook.func = packet_rcv;

 //注册钩子
 if (proto) {
  po->prot_hook.type = proto;
  register_prot_hook(sk);
 }
}
```

```
static void register_prot_hook(struct sock *sk)
{
 struct packet_sock *po = pkt_sk(sk);
 dev_add_pack(&po->prot_hook);
}
```

在packet_create 中設置回調函數為packet_rcv，再通過register_prot_hook => dev_add_pack 完成註冊。註冊完後，是在全局協議ptype_all 鍊錶中添加了一個虛擬的協議進來。



我們再來看下dev_add_pack 是如何註冊協議到ptype_all 中的。回顧我們開頭看到的socket 函數調用，第三個參數proto 傳入的是ETH_P_ALL。那dev_add_pack 其實最後是把hook 函數添加到了ptype_all 裡了，代碼如下。

```
//file: net/core/dev.c
void dev_add_pack(struct packet_type *pt)
{
```

```
 struct list_head *head = ptype_head(pt);
 list_add_rcu(&pt->list, head);
}

static inline struct list_head *ptype_head(const struct packet_type *pt)
{
 if (pt->type == htons(ETH_P_ALL))
  return &ptype_all;
 else
  return &ptype_base[ntohs(pt->type) & PTYPE_HASH_MASK];
}
```

> 我們整篇文章都以ETH_P_ALL 為例，但其實有的時候也會有其它情況。在別的情況下可能會註冊協議到ptype_base 裡了，而不是ptype_all。同樣， ptype_base 中的協議也會在發送和接收的過程中被執行到。

總結：tcpdump 啟動的時候內部邏輯其實很簡單，就是在ptype_all 中註冊了一個虛擬協議而已。

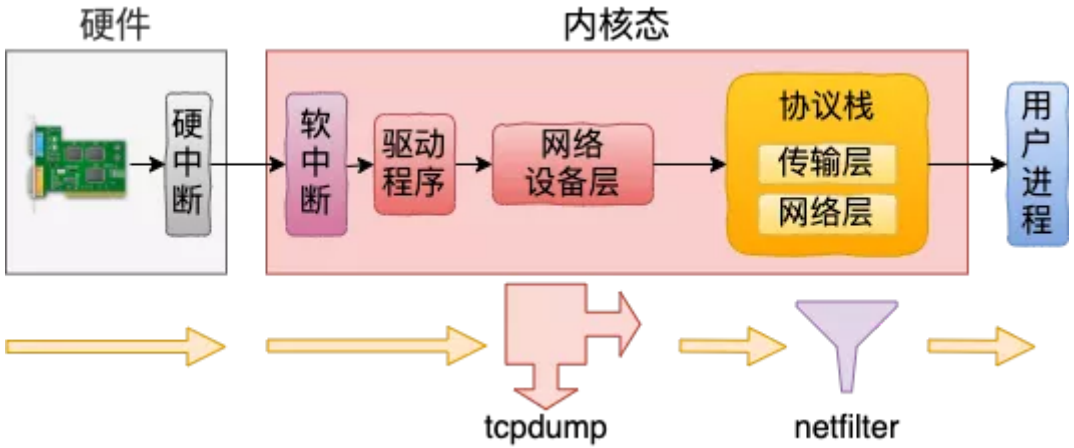
四、總結

現在我們再回頭看開篇提到的幾個問題。

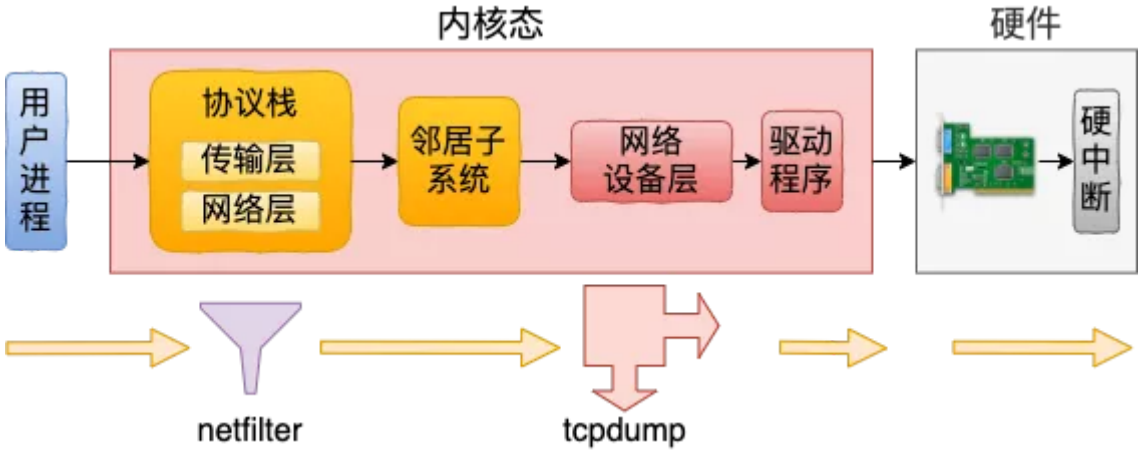
1. tcpdump是如何工作的

用戶態tcpdump 命令是通過socket 系統調用，在內核源碼中用到的ptype_all 中掛載了函數鉤子上去。無論是在網絡包接收過程中，還是在發送過程中，都會在網絡設備層遍歷ptype_all 中的協議，並執行其中的回調。tcpdump 命令就是基於這個底層原理來工作的。

2. netfilter 過濾的包tcpdump是否可以抓的到 在網絡包接收的過程中，由於tcpdump 近水樓台先得月，所以完全可以捕獲到命中 netfilter 過濾規則的包。



但是在發送的過程中，恰恰相反。網絡包先經過協議層，這時候被netfilter 過濾掉的話，底層工作的tcpdump 還沒等看見就啥也沒了。



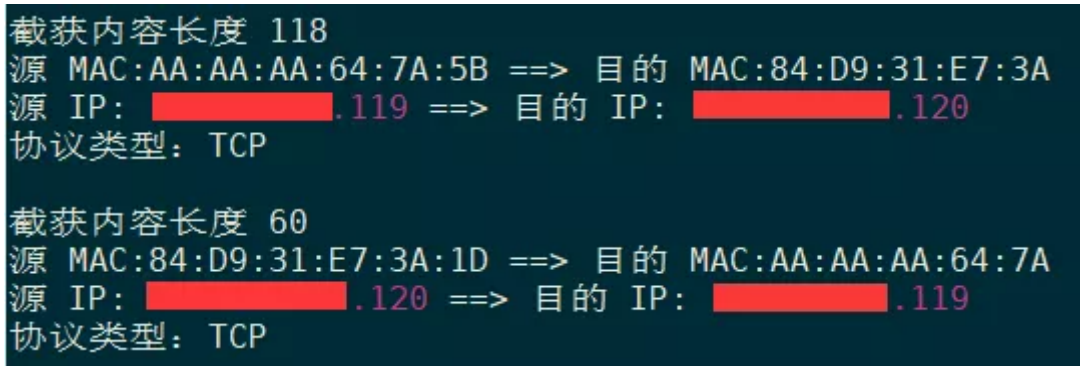
3. 讓你自己寫一個抓包程序的話該如何下手 我用c 寫了一段抓包，並且解析源IP 和目的IP 的簡單demo。

源碼地址：

編譯一下，注意運行需要root 權限。

```
# gcc -o main main.c
# ./main
```

運行結果預覽如下。



--- EOF ---

公眾號

喜歡此內容的人還喜歡

因軟件缺陷，736名員工被起訴，39人被冤假錯判

CSDN

百度C++工程師的那些極限優化（並發篇）

百度Geek說

這題答案會是什麼呢?

程序員的幽默