pcap编程简单介绍

在网络编程开发过程中，我们经常会使用wireshark来对网卡抓包，看看我们程序的收发包情况，wireshark是最流行的网络抓包工具之一，并且是开源代码，能够在github上查看，地址为：

https://github.com/wireshark/wireshark.git。

我自己使用wireshark时间也比较长了，但是并未了解其内部实现机制。这几天在看项目代码时，看到项目使用了wireshark里面dumpcap的程序，所以心血来潮想了解一下wireshark的实现机制。

wireshark能够通过dumpcap程序实现抓包，看了代码之后发现dumpcap原来是使用libpcap实现的，为了巩固一下自己对libpcap的理解，写篇文章简单介绍一下libpcap的使用。内容主要参考tcpdump上对pcap的内容，比原版要简陋很多，想看原版可以过去看一下，地址为：http:/www.tcpdump.org/pcap.html。

1. 首先熟悉一下关键词的缩写

BPF(Berkeley Packet Filter)

DLPI(Data Link Provider Interface)

1. 概览

pcap提供了多个api，当这些api按照一定顺序调用后，最终能够实现获取到网络数据包的过程。基本的过程如下：

(1)首先确认从哪个网卡捕获网络数据。

(2)初始化pcap。在这一步骤中，我们将第一步中获取的网卡名称传入api，告诉api应该捕获哪个网卡，api会返回一个句柄，pcap叫做’session’。

(3)在抓包过程中，由于特殊需要，我们可能只需要监听tcp的报文，或者只监听udp的报文，那么我们需要设置一下过滤条件。

(4)这一步就可以抓包了

(5)程序退出之前，需要释放一些申请的资源。这里主要释放(2)中的句柄。

以上就是pcap实现抓包的步骤，下面会列举各个步骤的api。

1. api介绍

(1)首先我们会确定从哪个网卡实现抓包，可以通过命令行直接将网卡传入到程序，也可以获取到所有网卡，然后挑一个即可。

* 从命令行传入，可以取参数值。main函数提供了argv的参数，网卡名称可以通过argv直接传入程序
* 获取到非loopback的第一个网卡。通过pcap\_lookupdev函数，pcap能够返回第一个非loopback的网卡名称。
* 也可以通过pcap\_findalldevs这个api获取到所有网卡，然后根据需要确定最终需要监听哪个网卡。

(2)获取到网卡后，需要初始化pcap了。我们将(1)中获取的网卡dev参数传入函数，会返回一个句柄，这个句柄又叫pcap的session。是后续设置过滤条件以及抓包的必要条件。

#include <pcap.h>

...

pcap\_t \*handle;

handle = pcap\_open\_live(dev, BUFSIZ, 1, 1000, errbuf);

if (handle == NULL) {

fprintf(stderr, "Couldn't open device %s: %s\n", dev, errbuf);

return(2);

}

初始化也可以通过pcap\_create来创建，这里不多做介绍。

(3)初始化完handle后，需要设置一些过滤条件，比如只筛选port 23的，或者只捕获tcp协议的等等。筛选过滤条件也是有语法存在的，本章不讲筛选的语法。

#include <pcap.h>

...

pcap\_t \*handle; /\* Session handle \*/

char dev[] = "rl0"; /\* Device to sniff on \*/

char errbuf[PCAP\_ERRBUF\_SIZE]; /\* Error string \*/

struct bpf\_program fp; /\* The compiled filter expression \*/

char filter\_exp[] = "port 23"; /\* The filter expression \*/

bpf\_u\_int32 mask; /\* The netmask of our sniffing device \*/

bpf\_u\_int32 net; /\* The IP of our sniffing device \*/

if (pcap\_lookupnet(dev, &net, &mask, errbuf) == -1) {

fprintf(stderr, "Can't get netmask for device %s\n", dev);

net = 0;

mask = 0;

}

if (pcap\_compile(handle, &fp, filter\_exp, 0, net) == -1) {

fprintf(stderr, "Couldn't parse filter %s: %s\n", filter\_exp, pcap\_geterr(handle));

return(2);

}

if (pcap\_setfilter(handle, &fp) == -1) {

fprintf(stderr, "Couldn't install filter %s: %s\n", filter\_exp, pcap\_geterr(handle));

return(2);

}

过滤条件的设置是通过pcap\_compile和pcap\_setfilter两个函数设置的。

pcap\_compile是将filter\_exp的内容编译进fp中，然后再通过pcap\_setfilter设置到handel中。

(4)好了，现在可以读取数据包了。

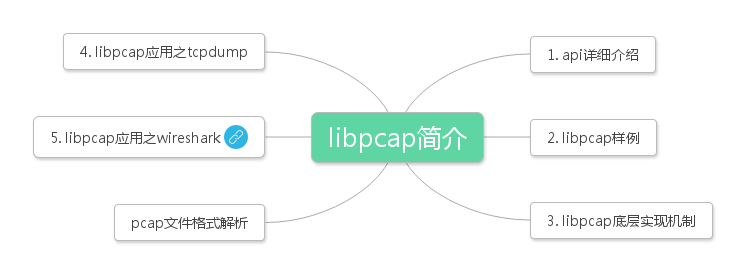
/\* Grab a packet \*/

packet = pcap\_next(handle, &header);

pcap\_next抓获一个数据包后返回，packet是数据包的指针位置，对于tcp/ip协议，packet内容从mac层开始，然后ip层和tcp层。

pcap\_next是抓取一个数据包的api，如果要抓取多个数据包，可以使用pcap\_loop或者pcap\_dispatch。

(5)程序退出前，调用pcap\_close释放掉句柄。

我画了一个简单的libpcap后续相关内容，有兴趣的可以一起研究一下。

我使用的联系代码存放在github上，地址为：

<https://github.com/topmyself/practice/blob/master/C/sniff.c>

libpcap的脑图放在了

<https://www.processon.com/mind/57ad9e1fe4b0f7781f30eb81>