Linux下使用libpcap进行网络抓包并保存到文件

原创 lvjian_ 于 2019-03-18 15:02:21 发布 10764 收藏 81 版权

分类专栏： 网络 文章标签： libpcap linux 网络抓包

libpcap 是一个抓取网络数据报文的C语言函数库，使用这个库可以非常方便的抓取网络上的报文，方便我们分析经过我们设备上的各种报文；

1.libpcap安装

下载文件：libpcap-x.x.x.tar.gz （这个是linux版本，x.x.x代表版本号）

下载地址：http://www.tcpdump.org/#latest-releases

解压(tar -zxvf)

./configure

make

sudo make install

2.使用pcap自动探测网络接口

先通过ifconfig观察一下设备情况：

```
[iona@linux157 PCap]$
[iona@linux157 PCap]$ ifconfig
eth0      Link encap:Ethernet  HWaddr 40:F2:E9:30:2A:44
          inet addr:192.168.22.158  Bcast:192.168.22.255  Mask:255.255.255.0
          inet6 addr: fe80::42f2:e9ff:fe30:2a44/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:638830223 errors:0 dropped:0 overruns:0 frame:0
          TX packets:491090715 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:183114171458 (170.5 GiB)  TX bytes:123366103515 (114.8 GiB)
          Memory:d2200000-d2240000

eth1      Link encap:Ethernet  HWaddr 40:F2:E9:30:2A:45
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)
          Memory:d2240000-d2280000
```


然后开始编写代码，先包含头文件pcap.h，它的路径
在：**/usr/local/include/pcap/pcap.h** 里面有每个类型定义的详细说明；

1）char * pcap_lookupdev(char * errbuf)

这个函数就是用来探测网络接口的，它会返回第一个合适的网络接口字符串指针，如果出错则在errbuf中返回，长度至少是**PCAP_ERRBUF_SIZE。**

pcap.cpp

```
#include <pcap.h>
#include <stdio.h>
#include <stdlib.h>

int main()
{
    char errBuf[PCAP_ERRBUF_SIZE], * devStr;

    devStr = pcap_lookupdev(errBuf);
    if (devStr)
        printf("success: device: %s\n", devStr);
    else
    {
        printf("error: %s\n", errBuf);
        exit(1);
    }

    return 0;
}
```

makefile(这里使用的C++风格，供参考)：

```
all:pcap

pcap:pcap.o
	gcc -g -lstdc++ -lpcap -o pcap pcap.o

clean:
	rm -f pcap pcap.o
```

运行结果：

```
[iona@linux157 src]$ ./pcap
error: no suitable device found
[iona@linux157 src]$ sudo ./pcap
success: device: eth0
```

第一次显示"no suitable device found"是因为权限不够；用sudo执行后就显示出来了；

3.使用pcap_next捕获第一个报文

相关函数说明：

2）pcap_t * pcap_open_live(const char * device, int snaplen, int promisc, int to_ms, char * errbuf)

要捕获报文，在探测到接口之后我们还要打开它，该函数会返回指定接口的pcap_t类型指针，后面的所有操作都要使用这个指针；

参数一：device是第一步探测到的接口的字符串；

参数二：snaplen是对于每个数据包，从开头要抓多少个字节，我们可以设置这个值来只抓每个数据包的头部，而不关心具体的内容。典型的以太网帧长度是1518字节，但其他的某些协议的数据包会更长一点，但任何一个协议的一个数据包长度都必然小于65535个字节；

参数三：promisc指定是否打开混杂模式(Promiscuous Mode)，0表示非混杂模式，任何其他值表示混合模式。如果要打开混杂模式，那么网卡必须也要打开混杂模式，可以使用如下的命令打开eth0混杂模式：
ifconfig eth0 promisc；

参数四：to_ms指定需要等待的毫秒数，超过这个数值后，第3步获取数据包的这几个函数就会立即返回。0表示一直等待直到有数据包到来；

参数五：用来返回错误信息；

3)void pcap_close(pcap_t * p)

释放网络接口，用于关闭pcap_open_live()获取的pcap_t的网络接口对象并释放相关资源；

4)u_char * pcap_next(pcap_t * p, struct pcap_pkthdr * h)

该函数收到一个包就立刻返回，返回值为NULL表示没有收到包；

第一个参数是第2）个函数pcap_open_live返回的指针，第二个参数记录了报文长度等，其结构也可以在pcap.h里查看到；

```
#include <pcap.h>
#include <time.h>
#include <stdio.h>
#include <stdlib.h>

int main()
{
    char errBuf[PCAP_ERRBUF_SIZE], * devStr;

    devStr = pcap_lookupdev(errBuf);
```

```
    if (devStr)
                    printf("success: device: %s\n", devStr);
    else
    {
        printf("error: %s\n", errBuf);
        exit(1);
    }

    /* open a device, wait until a packet arrives */
    pcap_t * device = pcap_open_live(devStr, 65535, 1, 0, errBuf);
    if (!device)
    {
        printf("error: pcap_open_live(): %s\n", errBuf);
        exit(1);
    }

    /* wait a packet to arrive */
    struct pcap_pkthdr packet;
    const u_char * pktStr = pcap_next(device, &packet);

    if (!pktStr)
    {
        printf("did not capture a packet!\n");
        exit(1);
    }

    printf("Packet len:%d, Bytes:%d, Received time:%s\n", packet.len,
        packet.caplen, ctime((const time_t *)&packet.ts.tv_sec));

    for(int i=0; i < packet.len; ++i)
    {
        printf(" %02x", pktStr[i]);
        if ((i + 1) % 16 == 0)
        {
            printf("\n");
        }
    }
    printf("\n");

    pcap_close(device);
    return 0;
}
```

显示结果：

```
[iona@linux157 src]$ sudo ./pcap
[sudo] password for iona:
success: device: eth0
Packet len:60, Bytes:60, Received time:Fri Mar 15 16:25:26 2019

40 f2 e9 30 2a 44 bc 9c 31 a4 2b 1d 08 00 45 00
00 28 3d e1 40 00 7a 06 27 89 ac 12 18 0d c0 a8
16 9e d6 cc 00 16 ee 4b 97 62 77 b2 43 cc 50 10
fc cf ff 8e 00 00 00 00 00 00 00 00
```

4.循环捕获报文

5)int pcap_loop(pcap_t * p, int cnt, pcap_handler callback, u_char * user)

第一个参数是第2）个函数pcap_open_live返回的指针，第二个参数是需要抓的数据包的个数，一旦抓到了cnt个数据包，pcap_loop立即返回。负数的cnt表示pcap_loop永远循环抓包，直到出现错误；第三个参数是一个回调函数指针，形式如下：**void callback(u_char * userarg, const struct pcap_pkthdr * pkthdr, const u_char * packet)** 第一个参数是pcap_loop的最后一个参数，第二个参数是收到的数据包的pcap_pkthdr结构，第三个参数是数据包数据；

6)int pcap_dispatch(pcap_t * p, int cnt, pcap_handler callback, u_char * user)

和pcap_loop()非常类似，它同时还受pcap_open_live()的第4个参数to_ms控制超时返回时间；

```
#include <pcap.h>
#include <time.h>
#include <stdio.h>
#include <stdlib.h>



void processPacket(u_char *arg, const struct pcap_pkthdr *pkthdr, const u_char *packet)
{       int *count = (int *)arg;

    printf("Packet Count: %d\n", ++(*count));
    printf("Received Packet Size: %d\n", pkthdr->len);
    printf("Payload:\n");

    for(int i=0; i < pkthdr->len; ++i)
    {
        printf("%02x ", packet[i]);
        if ((i + 1) % 16 == 0)
        {
            printf("\n");
        }
```

```
    }       printf("\n\n");
    return;
}

int main()
{
    char errBuf[PCAP_ERRBUF_SIZE], * devStr;

    devStr = pcap_lookupdev(errBuf);
    if (devStr)
        printf("success: device: %s\n", devStr);
    else
    {
        printf("error: %s\n", errBuf);
        exit(1);
    }

    /* open a device, wait until a packet arrives */
    pcap_t * device = pcap_open_live(devStr, 65535, 1, 0, errBuf);
    if (!device)
    {
        printf("error: pcap_open_live(): %s\n", errBuf);
        exit(1);
    }

    int count = 0;
    /*Loop forever & call processPacket() for every received packet.*/
    pcap_loop(device, -1, processPacket, (u_char *)&count);

    pcap_close(device);
    return 0;
}
```

结果（一部分）：

```
[iona@linux157 src]$ sudo ./pcap
success: device: eth0
Packet Count: 1
Received Packet Size: 60
Payload:
40 f2 e9 30 2a 44 bc 9c 31 a4 2b 1d 08 00 45 00
00 28 6d 5b 40 00 7a 06 f8 0e ac 12 18 0d c0 a8
16 9e ee 21 00 16 c4 26 17 fd 63 ec 7a 74 50 10
3f cb 2b e7 00 00 00 00 00 00 00 00

Packet Count: 2
Received Packet Size: 346
Payload:
bc 9c 31 a4 2b 1d 40 f2 e9 30 2a 44 08 00 45 10
01 4c ee 5c 40 00 40 06 af d9 c0 a8 16 9e ac 12
18 0d 00 16 ee 21 63 ec 7a 74 c4 26 17 fd 50 18
00 8d 9c a4 00 00 7c e8 ff 01 cb a8 69 23 0a b0
27 26 fe f3 90 ef 6a bb 13 ea 2a f2 b3 28 3b 93
7d c9 2d c7 62 83 39 a3 3c 67 bd c9 ba 8f e0 73
2d 5c 00 7f f0 f7 9d b4 e2 f7 a8 cf 92 46 41 3f
50 2f dd c7 57 02 4a 9d 0f 7c 45 01 85 5c 31 d8
e7 a7 96 18 db b6 f1 de fc 84 18 11 da 6d 4b af
2e 86 84 fb 80 e6 d5 57 47 03 b0 9b d8 cb bb 8e
8b 49 9d 7c 48 f0 7e af b4 c6 af bd f0 8c d4 78
```


5.过滤数据包

libpcap捕获报文是把经过的报文复制一份，当接口上报文数量很大时，抓取报文非常的占用系统资源，通过过滤数据包来捕获数据包，既可以过滤掉我们不想要的报文，又可以提高效率；

设置过滤条件，就是过滤表达式；

举例：

src host 192.168.1.177，dst port 80，not tcp等；

7）int pcap_compile(pcap_t * p, struct bpf_program * fp, char * str, int optimize, bpf_u_int32 netmask)

fp是一个传出参数，存放编译后的bpf，str是过滤表达式，optimize是否需要优化过滤表达式，metmask简单设置为0即可；

8）int pcap_setfilter(pcap_t * p, struct bpf_program * fp)

fp就是前一步pcap_compile()的第二个参数；

```
#include <pcap.h>
#include <time.h>
#include <stdio.h>
#include <stdlib.h>



void processPacket(u_char *arg, const struct pcap_pkthdr *pkthdr, const u_char *packet)
{       int *count = (int *)arg;
```

```
    printf("Packet Count: %d\n", ++(*count));
  printf("Received Packet Size: %d\n", pkthdr->len);      printf("Payload:\n");

    for(int i=0; i < pkthdr->len; ++i)
    {
        printf("%02x ", packet[i]);
        if ((i + 1) % 16 == 0)
        {
            printf("\n");
        }
    }
    printf("\n\n");
    return;
}

int main()
{
    char errBuf[PCAP_ERRBUF_SIZE], * devStr;

    devStr = pcap_lookupdev(errBuf);
    if (devStr)
        printf("success: device: %s\n", devStr);
    else
    {
        printf("error: %s\n", errBuf);
        exit(1);
    }

    /* open a device, wait until a packet arrives */
    pcap_t * device = pcap_open_live(devStr, 65535, 1, 0, errBuf);
    if (!device)
    {
        printf("error: pcap_open_live(): %s\n", errBuf);
        exit(1);
    }

    /* construct a filter */
    struct bpf_program filter;
    pcap_compile(device, &filter, "tcp", 1, 0);
    pcap_setfilter(device, &filter);

    int count = 0;
    /*Loop forever & call processPacket() for every received packet.*/
    pcap_loop(device, 30, processPacket, (u_char *)&count);

    pcap_close(device);
    return 0;
```

```
}
```

6.把数据包保存到文件

捕获到数据包之后，通常就是对数据包的分析，具体的报文分析方法要依据网络协议详细分类并展开处理，这里不做讨论。我们可以暂时把捕获到的数据包保存到文件中，稍后再由分析报文的程序或者工具来具体分析；libpcap库提供了保存为pcap类型的函数，非常方便，保存之后就可以用Wireshark直接打开了，如果想保存为纯粹的数据包，我们也可以用C语言的文件操作，直接把数据包以二进制的形式保存到文件中；

9）pcap_dumper_t *pcap_dump_open(pcap_t *p, const char *file)

函数返回pcap_dumper_t类型的指针，file是文件名，可以是绝对路径，例如：/home/iona/packet.pcap；

10）void pcap_dump_close(pcap_dumper_t *p);

用来关闭pcap_dump_open打开的文件，入参是pcap_dump_open返回的指针；

11）int pcap_dump_flush(pcap_dumper_t *p)

刷新缓冲区，把捕获的数据包从缓冲区真正拷贝到文件；

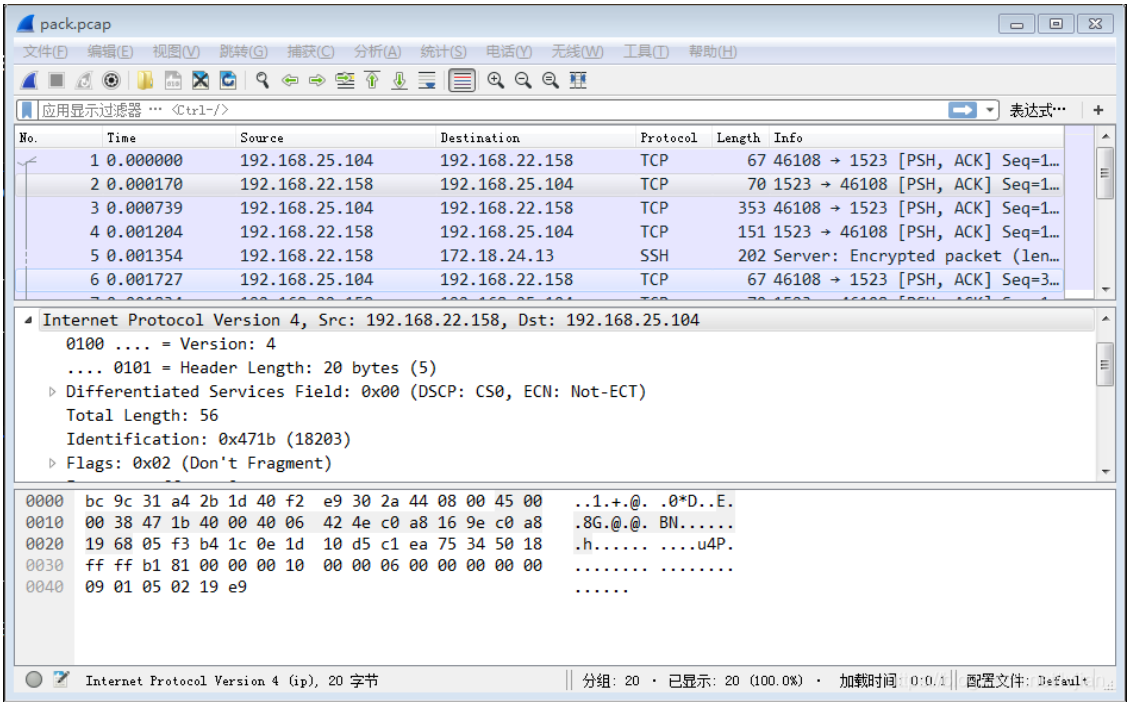
12）void pcap_dump(u_char * userarg, const struct pcap_pkthdr * pkthdr, const u_char * packet)

输出数据到文件，与pcap_loop的第二个参数回调函数**void callback(u_char * userarg, const struct pcap_pkthdr * pkthdr, const u_char * packet)** 形式完全相同，可以直接当pcap_loop的第二个参数；

```
#include <pcap.h>
#include <time.h>
#include <stdio.h>
#include <stdlib.h>


void processPacket(u_char *arg, const struct pcap_pkthdr *pkthdr, const u_char *packet)
{       pcap_dump(arg, pkthdr, packet);
    printf("Received Packet Size: %d\n", pkthdr->len);
    return;
}

int main()
{
    char errBuf[PCAP_ERRBUF_SIZE], * devStr;
```

```
    devStr = pcap_lookupdev(errBuf);
    if (devStr)
        printf("success: device: %s\n", devStr);
    else
    {
        printf("error: %s\n", errBuf);
        exit(1);
    }

    /* open a device, wait until a packet arrives */
    pcap_t * device = pcap_open_live(devStr, 65535, 1, 0, errBuf);
    if (!device)
    {
        printf("error: pcap_open_live(): %s\n", errBuf);
        exit(1);
    }

    /*open pcap write output file*/
    pcap_dumper_t* out_pcap;
    out_pcap  = pcap_dump_open(device,"pack.pcap");

    /*Loop forever & call processPacket() for every received packet.*/
    pcap_loop(device, 20, processPacket, (u_char *)out_pcap);

    /*flush buff*/
    pcap_dump_flush(out_pcap);

    pcap_dump_close(out_pcap);
    pcap_close(device);
    return 0;
}
```

抓包结果：



把数据包保存为二进制文件与保存为pcap文件很相似，只要把pcap_dump相关操作换成FILE相关操作就可以了，和libpcap库函数没什么关系，就不再描述了（上传的代码中有示例）；

相关的资料和代码都传上来了：

https://download.csdn.net/download/lvjian_/11022128