

个人资料



httpw

访问：406722次

积分：4511分

排名：第1697名

原创：110篇 转载：5篇

译文：0篇 评论：138条

文章搜索

文章分类

Ubuntu/Linux (15)

Eclipse (2)

Android (11)

LAMP (8)

C/C++ (8)

WebOS (3)

Embed/ARM (17)

Matlab (1)

JavaEE (1)

Network (0)

Ruby (1)

Qt/Symbian (9)

GCC/Makefile (1)

iPhone/iPad/XCode/Mac/cocos2d

Unity3D (5)

Work (0)

文章存档

2012年10月 (1)

2012年09月 (18)

2012年08月 (10)

2012年07月 (10)

2012年06月 (9)

展开

有奖征资源，博文分享有内涵 社区问答：非端iOS测试指南 专访阿里陶辉 2014 CSDN博文大赛 10月微软MVP申请

libpcap使用

分类：C/C++ 2012-04-28 18:34 22601人阅读 评论(45) 收藏 举报

struct server 网络 tcp c makefile

libpcap使用

libpcap是一个网络数据包捕获函数库，功能非常强大，Linux下著名的tcpdump就是以其为基础的。今天我们利用它来完成一个我们自己的网络嗅探器(sniffer)

首先先介绍一下本次实验的环境：
Ubuntu 11.04，IP：192.168.1.1，广播地址：192.168.1.255，子网掩码：255.255.255.0
可以使用下面的命令设置：
sudo ifconfig eth0 192.168.1.1 broadcast 192.168.1.255 netmask 255.255.255.0

```
hutao@hutao-VirtualBox:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:9c:ff:b1
          inet addr:192.168.1.1  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe9c:ffb1/64  Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:740 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2593 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:803534 (803.5 KB)  TX bytes:248368 (248.3 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128  Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:48 errors:0 dropped:0 overruns:0 frame:0
          TX packets:48 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:3568 (3.5 KB)  TX bytes:3568 (3.5 KB)

hutao@hutao-VirtualBox:~$
```

1.安装
在<http://www.tcpdump.org/>下载libpcap（tcpdump的源码也可以从这个网站下载）
解压
./configure

阅读排行

zlib的安装与使用	(25641)
libpcap使用	(22597)
android socket通信（上）	(15357)
黑苹果Mac系统快捷键修	(15277)
申请iOS开发者证书	(13521)
发布应用到AppStore	(11132)
iPod Touch 4刷机教程	(10922)
黑苹果安装Mac OS X Lic	(10422)
Unity3d与iOS的交互（1	(8458)
XCode发布App到调试机	(8357)

评论排行

libpcap使用	(45)
Unity3d与iOS的交互（1	(16)
android socket通信（下	(9)
申请iOS开发者证书	(6)
zlib的安装与使用	(6)
黑苹果Mac系统快捷键修	(4)
android socket通信（上	(3)
Android emulator中的sy	(3)
Unity3d之坦克大战（二	(3)
使用WinCE 6.0模拟器	(3)

推荐文章

最新评论

Linux使用蓝牙	xiaoxiaokun888: 你好请问运行sudo cat >/dev/rfcomm0 这步的时候，是手机已经点击蓝牙设备列表进行...
libpcap使用	sgchen: 真的很赞，楼主努力
libpcap使用	liujiangai: 楼主写的很全面，可是我运行test3的时候不知道是什么原因ping 192.168.1.10后，在...
豆瓣FM客户端使用说明	zodiac1111: 不知道有没有源代码，在debian+mate桌面环境下显示的有点小问题。
zlib的安装与使用	水木先生: 可以解压.rar格式的文件吗？
android socket通信（上）	shengno1: 谢谢开源，学习了；
涛涛写的豆瓣FM Linux客户端1.C	callofdutyops: 非常感谢！但是问题和二楼的一样.....
android socket通信（下）	ffightingg: 楼主的initClientSocket方法需要在子线程中去完成，不然会报错
Matlab之print,fprint,fscanf,disp等	果断注册: 写得很好，受教了，谢谢
Unity3d与iOS的交互（1）	哒咯网络: 我的这个函数跟你生成的不一样啊 没有controller这个参数啊 着急

make
sudo make install

2.使用

安装好libpcap后，我们要使用它啦，先写一个简单的程序，并介绍如何使用libpcap库编译它：

Makefile:

```
[plain]
01. all: test.c
02.      gcc -g -Wall -o test test.c -lpcap
03.
04. clean:
05.      rm -rf *.o test
```

其后的程序的Makefile均类似，故不再重复

test1.c

```
[cpp]
01. #include <pcap.h>
02. #include <stdio.h>
03.
04. int main()
05. {
06.     char errBuf[PCAP_ERRBUF_SIZE], * device;
07.
08.     device = pcap_lookupdev(errBuf);
09.
10.     if(device)
11.     {
12.         printf("success: device: %s\n", device);
13.     }
14.     else
15.     {
16.         printf("error: %s\n", errBuf);
17.     }
18.
19.     return 0;
20. }
```

可以成功编译，不过运行的时候却提示找不到libpcap.so.1，因为libpcap.so.1默认安装到了/usr/local/lib下，我们做一个符号链接到/usr/lib/下即可：

```
hutao@hutao-VirtualBox:~/test$ ./test
./test: error while loading shared libraries: libpcap.so.1: cannot open share
d object file: No such file or directory
hutao@hutao-VirtualBox:~/test$ ldd test
linux-gate.so.1 => (0x00706000)
libpcap.so.1 => not found
libc.so.6 => /lib/i386-linux-gnu/libc.so.6 (0x00bb8000)
/lib/ld-linux.so.2 (0x00501000)
hutao@hutao-VirtualBox:~/test$ whereis libpcap
libpcap: /usr/lib/libpcap.so /usr/lib/libpcap.a /usr/local/lib/libpcap.so /us
r/local/lib/libpcap.a
hutao@hutao-VirtualBox:~/test$ sudo ln -s /usr/local/lib/libpcap.so.1 /usr/li
b/libpcap.so.1
hutao@hutao-VirtualBox:~/test$ ldd test
linux-gate.so.1 => (0x0036d000)
libpcap.so.1 => /usr/lib/libpcap.so.1 (0x00a72000)
libc.so.6 => /lib/i386-linux-gnu/libc.so.6 (0x00d25000)
/lib/ld-linux.so.2 (0x007cf000)
hutao@hutao-VirtualBox:~/test$
```

运行test的时候输出"no suitable device found", 原因是我们没有以root权限运行, root权限运行后就正常了:

```
hutao@hutao-VirtualBox:~/test$ make
gcc -g -Wall -o test test.c -lpcap
hutao@hutao-VirtualBox:~/test$ ./test
error: no suitable device found
hutao@hutao-VirtualBox:~/test$ sudo ./test
success: device: eth0
hutao@hutao-VirtualBox:~/test$
```

下面开始正式讲解如何使用libpcap:

首先要使用libpcap, 我们必须包含pcap.h头文件, 可以在/usr/local/include/pcap/pcap.h找到, 其中包含了每个类型定义的详细说明。

1. 获取网络接口

首先我们需要获取监听的网络接口:

我们可以手动指定或让libpcap自动选择, 先介绍如何让libpcap自动选择:

char * pcap_lookupdev(char * errbuf)

上面这个函数返回第一个合适的网络接口的字符串指针, 如果出错, 则errbuf存放出错信息字符串, errbuf至少应该是PCAP_ERRBUF_SIZE个字节长度的。注意, 很多libpcap函数都有这个参数。

pcap_lookupdev()一般可以在跨平台的, 且各个平台上的网络接口名称都不相同的情况下使用。

如果我们手动指定要监听的网络接口, 则这一步跳过, 我们在第二步中将要监听的网络接口字符串硬编码在pcap_open_live里。

2. 释放网络接口

在操作为网络接口后，我们应该要释放它：

void pcap_close(pcap_t * p)

该函数用于关闭pcap_open_live()获取的pcap_t的网络接口对象并释放相关资源。

3.打开网络接口

获取网络接口后，我们需要打开它：

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

上面这个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。

第一个参数是第一步获取的网络接口字符串，可以直接使用硬编码。

第二个参数是对于每个数据包，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。

第三个参数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

第四个参数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。

第五个参数是存放出错信息的数组。

4.获取数据包

打开网络接口后就已经开始监听了，那如何知道收到了数据包呢？有下面3种方法：

a)

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

如果返回值为NULL，表示没有抓到包

第一个参数是第2步返回的pcap_t类型的指针

第二个参数是保存收到的第一个数据包的pcap_pkthdr类型的指针

pcap_pkthdr类型的定义如下：

```
[cpp]
01. struct pcap_pkthdr
02. {
03.     struct timeval ts;    /* time stamp */
04.     bpf_u_int32 caplen;   /* length of portion present */
05.     bpf_u_int32 len;      /* length this packet (off wire) */
06. };
```

注意这个函数只要收到一个数据包后就会立即返回

b)

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

第一个参数是第2步返回的pcap_t类型的指针

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，它必须是如下的形式：

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，当收到足够数量的包后pcap_loop会调用callback回调函数，同时将pcap_loop()的user参数传递给它

第二个参数是收到的数据包的pcap_pkthdr类型的指针

第三个参数是收到的数据包数据

c)

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

这个函数和pcap_loop()非常类似，只是在超过to_ms毫秒后就会返回(to_ms是pcap_open_live()的第4个参数)

例子:

test2:

```
[cpp]
01.  #include <pcap.h>
02.  #include <time.h>
03.  #include <stdlib.h>
04.  #include <stdio.h>
05.
06.  int main()
07.  {
08.      char errBuf[PCAP_ERRBUF_SIZE], * devStr;
09.
10.      /* get a device */
11.      devStr = pcap_lookupdev(errBuf);
12.
13.      if(devStr)
14.      {
15.          printf("success: device: %s\n", devStr);
16.      }
17.      else
18.      {
19.          printf("error: %s\n", errBuf);
20.          exit(1);
21.      }
22.
23.      /* open a device, wait until a packet arrives */
24.      pcap_t * device = pcap_open_live(devStr, 65535, 1, 0, errBuf);
25.
26.      if(!device)
27.      {
28.          printf("error: pcap_open_live(): %s\n", errBuf);
29.          exit(1);
30.      }
31.
32.      /* wait a packet to arrive */
33.      struct pcap_pkthdr packet;
34.      const u_char * pktStr = pcap_next(device, &packet);
35.
36.      if(!pktStr)
37.      {
38.          printf("did not capture a packet!\n");
39.          exit(1);
40.      }
41.
42.      printf("Packet length: %d\n", packet.len);
43.      printf("Number of bytes: %d\n", packet.caplen);
44.      printf("Recieved time: %s\n", ctime((const time_t *)&packet.ts.tv_sec));
45.
46.      pcap_close(device);
47.
48.      return 0;
49.  }
```

The image shows two terminal windows from a virtual machine named 'hutao@hutao-VirtualBox'. The top window shows the output of a program that successfully captured a packet on the 'eth0' interface. The output is: 'success: device: eth0', 'Packet length: 42', 'Number of bytes: 42', and 'Recieved time: Fri Apr 27 21:55:52 2012'. The bottom window shows the output of a 'ping' command to '192.168.1.10', which returns: 'PING 192.168.1.10 (192.168.1.10) 56(84) bytes of data.'.

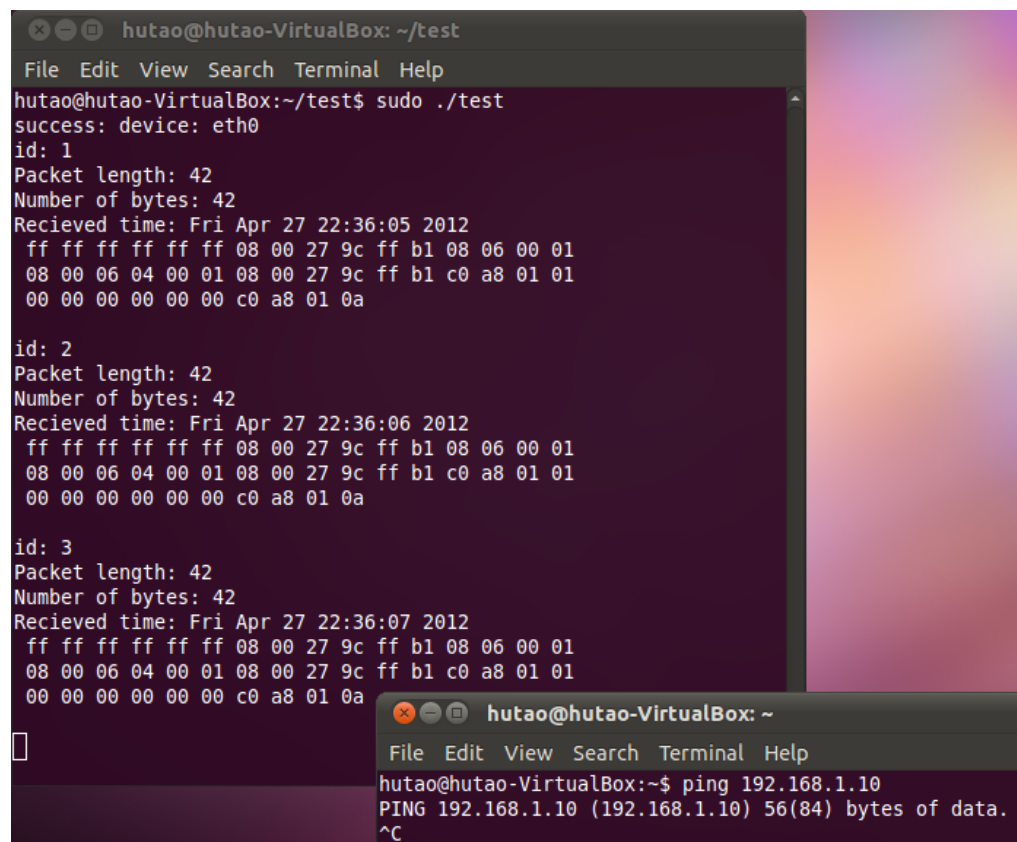
打开两个终端，先ping 192.168.1.10，由于我们的ip是192.168.1.1，因此我们可以收到广播的数据包，另一个终端运行test，就会抓到这个包。

下面的这个程序会把收到的数据包内容全部打印出来，运行方式和上一个程序一样：

test3:

```
[cpp]
01. #include <pcap.h>
02. #include <time.h>
03. #include <stdlib.h>
04. #include <stdio.h>
05.
06. void getPacket(u_char * arg, const struct pcap_pkthdr * pkthdr, const u_char * packet)
07. {
08.     int * id = (int *)arg;
09.
10.     printf("id: %d\n", ++(*id));
11.     printf("Packet length: %d\n", pkthdr->len);
12.     printf("Number of bytes: %d\n", pkthdr->caplen);
13.     printf("Recieved time: %s", ctime((const time_t *)&pkthdr->ts.tv_sec));
14.
15.     int i;
16.     for(i=0; i<pkthdr->len; ++i)
17.     {
18.         printf(" %02x", packet[i]);
19.         if( (i + 1) % 16 == 0 )
20.         {
21.             printf("\n");
22.         }
23.     }
24.
25.     printf("\n\n");
26. }
27.
28. int main()
29. {
30.     char errBuf[PCAP_ERRBUF_SIZE], * devStr;
31.
32.     /* get a device */
33.     devStr = pcap_lookupdev(errBuf);
34.
35.     if(devStr)
36.     {
37.         printf("success: device: %s\n", devStr);
38.     }
39.     else
```

```
40.     {
41.         printf("error: %s\n", errBuf);
42.         exit(1);
43.     }
44.
45.     /* open a device, wait until a packet arrives */
46.     pcap_t * device = pcap_open_live(devStr, 65535, 1, 0, errBuf);
47.
48.     if(!device)
49.     {
50.         printf("error: pcap_open_live(): %s\n", errBuf);
51.         exit(1);
52.     }
53.
54.     /* wait loop forever */
55.     int id = 0;
56.     pcap_loop(device, -1, getPacket, (u_char*)&id);
57.
58.     pcap_close(device);
59.
60.     return 0;
61. }
```



```
hutao@hutao-VirtualBox: ~/test
File Edit View Search Terminal Help
hutao@hutao-VirtualBox:~/test$ sudo ./test
success: device: eth0
id: 1
Packet length: 42
Number of bytes: 42
Recieved time: Fri Apr 27 22:36:05 2012
ff ff ff ff ff ff 08 00 27 9c ff b1 08 06 00 01
08 00 06 04 00 01 08 00 27 9c ff b1 c0 a8 01 01
00 00 00 00 00 00 c0 a8 01 0a

id: 2
Packet length: 42
Number of bytes: 42
Recieved time: Fri Apr 27 22:36:06 2012
ff ff ff ff ff ff 08 00 27 9c ff b1 08 06 00 01
08 00 06 04 00 01 08 00 27 9c ff b1 c0 a8 01 01
00 00 00 00 00 00 c0 a8 01 0a

id: 3
Packet length: 42
Number of bytes: 42
Recieved time: Fri Apr 27 22:36:07 2012
ff ff ff ff ff ff 08 00 27 9c ff b1 08 06 00 01
08 00 06 04 00 01 08 00 27 9c ff b1 c0 a8 01 01
00 00 00 00 00 00 c0 a8 01 0a

hutao@hutao-VirtualBox: ~
File Edit View Search Terminal Help
hutao@hutao-VirtualBox:~$ ping 192.168.1.10
PING 192.168.1.10 (192.168.1.10) 56(84) bytes of data.
^C
```

从上图可以看出, 如果我们没有按Ctrl+c, test会一直抓到包, 因为我们将pcap_loop()设置为永远循环

由于ping属于icmp协议, 并且发出icmp协议数据包之前必须先通过arp协议获取目的主机的mac地址, 因此我们抓到的包是arp协议的, 而arp协议的数据包长度正好是42字节(14字节的以太网帧头+28字节的arp数据)。具体内容请参考相关网络协议说明。

5.分析数据包

我们既然已经抓到数据包了，那么我们要开始分析了，这部分留给读者自己完成，具体内容可以参考相关的网络协议说明。在本文的最后，我会示范性的写一个分析arp协议的sniffer，仅供参考。**要特别注意一点，网络上的数据是网络字节顺序的，因此分析前需要转换为主机字节顺序(ntohs()函数)。**

6.过滤数据包

我们抓到的数据包往往很多，如何过滤掉我们不感兴趣的数据包呢？

几乎所有的操作系统(BSD, AIX, Mac OS, Linux等)都会在内核中提供过滤数据包的方法，**主要都是基于BSD Packet Filter(BPF)结构的。libpcap利用BPF来过滤数据包。**

过滤数据包需要完成3件事：

- a) 构造一个过滤表达式**
- b) 编译这个表达式**
- c) 应用这个过滤器**

a)

BPF使用一种类似于汇编语言的语法书写过滤表达式，不过libpcap和tcpdump都把它封装成更高级且更容易的语法了，具体可以man tcpdump，以下是一些例子：

src host 192.168.1.177

只接收源ip地址是192.168.1.177的数据包

dst port 80

只接收tcp/udp的目的端口是80的数据包

not tcp

只接收不使用tcp协议的数据包

tcp[13] == 0x02 and (dst port 22 or dst port 23)

只接收SYN标志位置位且目标端口是22或23的数据包（tcp首部开始的第13个字节）

icmp[icmptype] == icmp-echoreply or icmp[icmptype] == icmp-echo

只接收icmp的ping请求和ping响应的数据包

ether dst 00:e0:09:c1:0e:82

只接收以太网mac地址是00:e0:09:c1:0e:82的数据包

ip[8] == 5

只接收ip的ttl=5的数据包（ip首部开始的第8个字节）

b)

构造完过滤表达式后，我们需要编译它，使用如下函数：

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

fp：这是一个传出参数，存放编译后的bpf

str：过滤表达式

optimize：是否需要优化过滤表达式

netmask：简单设置为0即可

c)

最后我们需要应用这个过滤表达式：

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

第二个参数fp就是前一步pcap_compile()的第二个参数

应用完过滤表达式之后我们便可以使用pcap_loop()或pcap_next()等抓包函数来抓包了。

下面的程序演示了如何过滤数据包，我们只接收目的端口是80的数据包：

test4.c

```
[cpp]
01.  #include <pcap.h>
02.  #include <time.h>
03.  #include <stdlib.h>
04.  #include <stdio.h>
05.
06.  void getPacket(u_char * arg, const struct pcap_pkthdr * pkthdr, const u_char * packet)
07.  {
08.      int * id = (int *)arg;
09.
10.      printf("id: %d\n", ++(*id));
11.      printf("Packet length: %d\n", pkthdr->len);
12.      printf("Number of bytes: %d\n", pkthdr->caplen);
13.      printf("Recieved time: %s", ctime((const time_t *)&pkthdr->ts.tv_sec));
14.
15.      int i;
16.      for(i=0; i<pkthdr->len; ++i)
17.      {
18.          printf(" %02x", packet[i]);
19.          if( (i + 1) % 16 == 0 )
20.          {
21.              printf("\n");
22.          }
23.      }
24.
25.      printf("\n\n");
26.  }
27.
28.  int main()
29.  {
30.      char errBuf[PCAP_ERRBUF_SIZE], * devStr;
31.
32.      /* get a device */
33.      devStr = pcap_lookupdev(errBuf);
34.
35.      if(devStr)
36.      {
37.          printf("success: device: %s\n", devStr);
38.      }
39.      else
40.      {
41.          printf("error: %s\n", errBuf);
```

```

42.     exit(1);
43. }
44.
45. /* open a device, wait until a packet arrives */
46. pcap_t * device = pcap_open_live(devStr, 65535, 1, 0, errBuf);
47.
48. if(!device)
49. {
50.     printf("error: pcap_open_live(): %s\n", errBuf);
51.     exit(1);
52. }
53.
54. /* construct a filter */
55. struct bpf_program filter;
56. pcap_compile(device, &filter, "dst port 80", 1, 0);
57. pcap_setfilter(device, &filter);
58.
59. /* wait loop forever */
60. int id = 0;
61. pcap_loop(device, -1, getPacket, (u_char*)&id);
62.
63. pcap_close(device);
64.
65. return 0;
66. }

```

在下面的这一个例子中，客户机通过tcp的9732端口连接服务器，发送字符'A'，之后服务器将'A'+1即'B'返回给客户机，具体实现可以参

考：<http://blog.csdn.net/httpw/article/details/7519964>

服务器的ip是192.168.56.101，客户机的ip是192.168.56.1

服务器：

```

hutao@hutao-VirtualBox:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:9c:ff:b1
          inet addr:192.168.56.101  Bcast:192.168.56.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe9c:ffbf/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:2524 errors:0 dropped:0 overruns:0 frame:0
          TX packets:4454 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:3154176 (3.1 MB)  TX bytes:383872 (383.8 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:378 errors:0 dropped:0 overruns:0 frame:0
          TX packets:378 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:31516 (31.5 KB)  TX bytes:31516 (31.5 KB)

hutao@hutao-VirtualBox:~$

```

Makefile:

```

[plain]
01. all: tcp_client.c tcp_server.c
02.     gcc -g -Wall -o tcp_client tcp_client.c
03.     gcc -g -Wall -o tcp_server tcp_server.c
04.
05. clean:
06.     rm -rf *.o tcp_client tcp_server

```

tcp_server:

```
[cpp]
01. #include <sys/types.h>
02. #include <sys/socket.h>
03. #include <netinet/in.h>
04. #include <arpa/inet.h>
05. #include <unistd.h>
06. #include <stdlib.h>
07. #include <stdio.h>
08.
09. #define PORT 9832
10. #define SERVER_IP "192.168.56.101"
11.
12. int main()
13. {
14.     /* create a socket */
15.     int server_sockfd = socket(AF_INET, SOCK_STREAM, 0);
16.
17.     struct sockaddr_in server_addr;
18.     server_addr.sin_family = AF_INET;
19.     server_addr.sin_addr.s_addr = inet_addr(SERVER_IP);
20.     server_addr.sin_port = htons(PORT);
21.
22.     /* bind with the local file */
23.     bind(server_sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr));
24.
25.     /* listen */
26.     listen(server_sockfd, 5);
27.
28.     char ch;
29.     int client_sockfd;
30.     struct sockaddr_in client_addr;
31.     socklen_t len = sizeof(client_addr);
32.     while(1)
33.     {
34.         printf("server waiting:\n");
35.
36.         /* accept a connection */
37.         client_sockfd = accept(server_sockfd, (struct sockaddr *)&client_addr, &len);
38.
39.         /* exchange data */
40.         read(client_sockfd, &ch, 1);
41.         printf("get char from client: %c\n", ch);
42.         ++ch;
43.         write(client_sockfd, &ch, 1);
44.
45.         /* close the socket */
46.         close(client_sockfd);
47.     }
48.
49.     return 0;
50. }
```

tcp_client:

```
[cpp]
01. #include <sys/types.h>
02. #include <sys/socket.h>
03. #include <netinet/in.h>
04. #include <arpa/inet.h>
05. #include <unistd.h>
06. #include <stdlib.h>
07. #include <stdio.h>
08.
09. #define PORT 9832
10. #define SERVER_IP "192.168.56.101"
11.
12. int main()
13. {
14.     /* create a socket */
15.     int sockfd = socket(AF_INET, SOCK_STREAM, 0);
16.
17.     struct sockaddr_in address;
18.     address.sin_family = AF_INET;
19.     address.sin_addr.s_addr = inet_addr(SERVER_IP);
20.     address.sin_port = htons(PORT);
```

```

21.
22.     /* connect to the server */
23.     int result = connect(sockfd, (struct sockaddr *)&address, sizeof(address));
24.     if(result == -1)
25.     {
26.         perror("connect failed: ");
27.         exit(1);
28.     }
29.
30.     /* exchange data */
31.     char ch = 'A';
32.     write(sockfd, &ch, 1);
33.     read(sockfd, &ch, 1);
34.     printf("get char from server: %c\n", ch);
35.
36.     /* close the socket */
37.     close(sockfd);
38.
39.     return 0;
40. }

```

运行方法如下，首先在服务器上运行tcp_server，然后运行我们的监听器，然后在客户机上运行tcp_client，注意，**我们可以先清空arp缓存，这样就可以看到整个通信过程（包括一开始的arp广播）**

在客户机上运行下列命令来清空记录服务器的arp缓存：

sudo arp -d 192.168.56.101

arp -a后发现已经删除了记录服务器的arp缓存

抓包的结果如下所示，由于包太多了，无法全部截图，因此我把所有内容保存在下面的文本中了：

```

hutao@hutao-VirtualBox:~/test3$ sudo ./test
success: device: eth0
id: 1
Packet length: 60
Number of bytes: 60
Recieved time: Sat Apr 28 19:57:50 2012
ff ff ff ff ff ff 0a 00 27 00 00 00 08 06 00 01
08 00 06 04 00 01 0a 00 27 00 00 00 c0 a8 38 01
00 00 00 00 00 00 c0 a8 38 65 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
id: 2
Packet length: 42
Number of bytes: 42
Recieved time: Sat Apr 28 19:57:50 2012
0a 00 27 00 00 00 08 00 27 9c ff b1 08 06 00 01
08 00 06 04 00 02 08 00 27 9c ff b1 c0 a8 38 65
0a 00 27 00 00 00 c0 a8 38 01
id: 3
Packet length: 74
Number of bytes: 74
Recieved time: Sat Apr 28 19:57:50 2012
08 00 27 9c ff b1 0a 00 27 00 00 00 08 00 45 00
00 3c d4 af 40 00 40 06 74 55 c0 a8 38 01 c0 a8
38 65 8e 20 26 68 79 e1 63 8c 00 00 00 00 a0 02
39 08 d4 13 00 00 02 04 05 b4 04 02 08 0a 00 14
b7 23 00 00 00 00 01 03 03 06
id: 4
Packet length: 74

```

全部的包如下：

[plain]

```
01. hutao@hutao-VirtualBox:~/test3$ sudo ./test
02. success: device: eth0
03. id: 1
04. Packet length: 60
05. Number of bytes: 60
06. Recieved time: Sat Apr 28 19:57:50 2012
07. ff ff ff ff ff ff 0a 00 27 00 00 00 08 06 00 01
08. 08 00 06 04 00 01 0a 00 27 00 00 00 c0 a8 38 01
09. 00 00 00 00 00 00 c0 a8 38 65 00 00 00 00 00 00
10. 00 00 00 00 00 00 00 00 00 00 00 00
11.
12. id: 2
13. Packet length: 42
14. Number of bytes: 42
15. Recieved time: Sat Apr 28 19:57:50 2012
16. 0a 00 27 00 00 00 08 00 27 9c ff b1 08 06 00 01
17. 08 00 06 04 00 02 08 00 27 9c ff b1 c0 a8 38 65
18. 0a 00 27 00 00 00 c0 a8 38 01
19.
20. id: 3
21. Packet length: 74
22. Number of bytes: 74
23. Recieved time: Sat Apr 28 19:57:50 2012
24. 08 00 27 9c ff b1 0a 00 27 00 00 00 08 00 45 00
25. 00 3c d4 af 40 00 40 06 74 55 c0 a8 38 01 c0 a8
26. 38 65 8e 20 26 68 79 e1 63 8c 00 00 00 00 a0 02
27. 39 08 d4 13 00 00 02 04 05 b4 04 02 08 0a 00 14
28. b7 23 00 00 00 00 01 03 03 06
29.
30. id: 4
31. Packet length: 74
32. Number of bytes: 74
33. Recieved time: Sat Apr 28 19:57:50 2012
34. 0a 00 27 00 00 00 08 00 27 9c ff b1 08 00 45 00
35. 00 3c 00 00 40 00 40 06 49 05 c0 a8 38 65 c0 a8
36. 38 01 26 68 8e 20 b6 c4 e6 e5 79 e1 63 8d a0 12
37. 38 90 f1 e5 00 00 02 04 05 b4 04 02 08 0a 00 57
38. a1 2c 00 14 b7 23 01 03 03 05
39.
40. id: 5
41. Packet length: 66
42. Number of bytes: 66
43. Recieved time: Sat Apr 28 19:57:50 2012
44. 08 00 27 9c ff b1 0a 00 27 00 00 00 08 00 45 00
45. 00 34 d4 b0 40 00 40 06 74 5c c0 a8 38 01 c0 a8
46. 38 65 8e 20 26 68 79 e1 63 8d b6 c4 e6 e6 80 10
47. 00 e5 fb c1 00 00 01 01 08 0a 00 14 b7 24 00 57
48. a1 2c
49.
50. id: 6
51. Packet length: 67
52. Number of bytes: 67
53. Recieved time: Sat Apr 28 19:57:50 2012
54. 08 00 27 9c ff b1 0a 00 27 00 00 00 08 00 45 00
55. 00 35 d4 b1 40 00 40 06 74 5a c0 a8 38 01 c0 a8
56. 38 65 8e 20 26 68 79 e1 63 8d b6 c4 e6 e6 80 18
57. 00 e5 ba b7 00 00 01 01 08 0a 00 14 b7 25 00 57
58. a1 2c 41
59.
60. id: 7
61. Packet length: 66
62. Number of bytes: 66
63. Recieved time: Sat Apr 28 19:57:50 2012
64. 0a 00 27 00 00 00 08 00 27 9c ff b1 08 00 45 00
65. 00 34 47 cb 40 00 40 06 01 42 c0 a8 38 65 c0 a8
66. 38 01 26 68 8e 20 b6 c4 e6 e6 79 e1 63 8e 80 10
67. 01 c5 f1 dd 00 00 01 01 08 0a 00 57 a1 2e 00 14
68. b7 25
69.
70. id: 8
71. Packet length: 67
72. Number of bytes: 67
73. Recieved time: Sat Apr 28 19:57:50 2012
74. 0a 00 27 00 00 00 08 00 27 9c ff b1 08 00 45 00
75. 00 35 47 cc 40 00 40 06 01 40 c0 a8 38 65 c0 a8
76. 38 01 26 68 8e 20 b6 c4 e6 e6 79 e1 63 8e 80 18
77. 01 c5 f1 de 00 00 01 01 08 0a 00 57 a1 2e 00 14
78. b7 25 42
```

```
79.
80. id: 9
81. Packet length: 66
82. Number of bytes: 66
83. Recieved time: Sat Apr 28 19:57:50 2012
84. 0a 00 27 00 00 00 08 00 27 9c ff b1 08 00 45 00
85. 00 34 47 cd 40 00 40 06 01 40 c0 a8 38 65 c0 a8
86. 38 01 26 68 8e 20 b6 c4 e6 e7 79 e1 63 8e 80 11
87. 01 c5 f1 dd 00 00 01 01 08 0a 00 57 a1 2e 00 14
88. b7 25
89.
90. id: 10
91. Packet length: 66
92. Number of bytes: 66
93. Recieved time: Sat Apr 28 19:57:50 2012
94. 08 00 27 9c ff b1 0a 00 27 00 00 00 08 00 45 00
95. 00 34 d4 b2 40 00 40 06 74 5a c0 a8 38 01 c0 a8
96. 38 65 8e 20 26 68 79 e1 63 8e b6 c4 e6 e7 80 10
97. 00 e5 fb bc 00 00 01 01 08 0a 00 14 b7 25 00 57
98. a1 2e
99.
100. id: 11
101. Packet length: 66
102. Number of bytes: 66
103. Recieved time: Sat Apr 28 19:57:50 2012
104. 08 00 27 9c ff b1 0a 00 27 00 00 00 08 00 45 00
105. 00 34 d4 b3 40 00 40 06 74 59 c0 a8 38 01 c0 a8
106. 38 65 8e 20 26 68 79 e1 63 8e b6 c4 e6 e7 80 11
107. 00 e5 fb bb 00 00 01 01 08 0a 00 14 b7 25 00 57
108. a1 2e
109.
110. id: 12
111. Packet length: 66
112. Number of bytes: 66
113. Recieved time: Sat Apr 28 19:57:50 2012
114. 0a 00 27 00 00 00 08 00 27 9c ff b1 08 00 45 00
115. 00 34 47 ce 40 00 40 06 01 3f c0 a8 38 65 c0 a8
116. 38 01 26 68 8e 20 b6 c4 e6 e8 79 e1 63 8f 80 10
117. 01 c5 f1 dd 00 00 01 01 08 0a 00 57 a1 2e 00 14
118. b7 25
119.
120. id: 13
121. Packet length: 66
122. Number of bytes: 66
123. Recieved time: Sat Apr 28 19:57:50 2012
124. 08 00 27 9c ff b1 0a 00 27 00 00 00 08 00 45 00
125. 00 34 d4 b4 40 00 40 06 74 58 c0 a8 38 01 c0 a8
126. 38 65 8e 20 26 68 79 e1 63 8f b6 c4 e6 e8 80 10
127. 00 e5 fb b9 00 00 01 01 08 0a 00 14 b7 26 00 57
128. a1 2e
```

仔细研究即可发现服务器与客户机是如何通过tcp通信的。

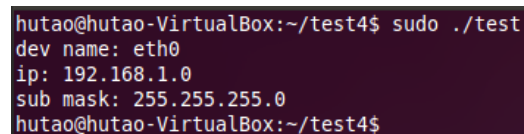
下面的这个程序可以获取eth0的ip和子网掩码等信息：

test5:

```
[cpp]
01. #include <stdio.h>
02. #include <stdlib.h>
03. #include <pcap.h>
```

```
04. #include <errno.h>
05. #include <netinet/in.h>
06. #include <arpa/inet.h>
07.
08. int main()
09. {
10.     /* ask pcap to find a valid device for use to sniff on */
11.     char * dev; /* name of the device */
12.     char errbuf[PCAP_ERRBUF_SIZE];
13.     dev = pcap_lookupdev(errbuf);
14.
15.     /* error checking */
16.     if(!dev)
17.     {
18.         printf("pcap_lookupdev() error: %s\n", errbuf);
19.         exit(1);
20.     }
21.
22.     /* print out device name */
23.     printf("dev name: %s\n", dev);
24.
25.     /* ask pcap for the network address and mask of the device */
26.     bpf_u_int32 netp; /* ip */
27.     bpf_u_int32 maskp; /* subnet mask */
28.     int ret; /* return code */
29.     ret = pcap_lookupnet(dev, &netp, &maskp, errbuf);
30.
31.     if(ret == -1)
32.     {
33.         printf("pcap_lookupnet() error: %s\n", errbuf);
34.         exit(1);
35.     }
36.
37.     /* get the network address in a human readable form */
38.     char * net; /* dot notation of the network address */
39.     char * mask; /* dot notation of the network mask */
40.     struct in_addr addr;
41.
42.     addr.s_addr = netp;
43.     net = inet_ntoa(addr);
44.
45.     if(!net)
46.     {
47.         perror("inet_ntoa() ip error: ");
48.         exit(1);
49.     }
50.
51.     printf("ip: %s\n", net);
52.
53.     /* do the same as above for the device's mask */
54.     addr.s_addr = maskp;
55.     mask = inet_ntoa(addr);
56.
57.     if(!mask)
58.     {
59.         perror("inet_ntoa() sub mask error: ");
60.         exit(1);
61.     }
62.
63.     printf("sub mask: %s\n", mask);
64.
65.     return 0;
66. }
```

结果如图:



```
hutao@hutao-VirtualBox:~/test4$ sudo ./test
dev name: eth0
ip: 192.168.1.0
sub mask: 255.255.255.0
hutao@hutao-VirtualBox:~/test4$
```


int pcap_lookupnet(const char * device, bpf_u_int32 * netp, bpf_u_int32 * maskp, char * errbuf)
可以获取指定设备的ip地址，子网掩码等信息
netp：传出参数，指定网络接口的ip地址
maskp：传出参数，指定网络接口的子网掩码
pcap_lookupnet()失败返回-1
我们使用inet_ntoa()将其转换为可读的点分十进制形式的字符串

本文的绝大部分来源于libpcap的官方文档：[libpcapHakin9LuisMartinGarcia.pdf](#)，可以在官网下载，文档只有9页，不过很详细，还包括了数据链路层，网络层，传输层，应用层等的分析。很好！

更多参考可以man pcap

最后为了方便大家，本文的所有代码和上述的pdf文档都一并上传上来了：
<http://download.csdn.net/detail/httpw/4264686>

完成！


更多 10

上一篇 编译ucosii
下一篇 ffmpeg的编译与使用




主题推荐 网络协议 操作系统 汇编语言 以太网 virtualbox

猜你在找

- | | |
|------------------------------|------------------|
| moto & google笔试题目-STL/C++面试题 | GDB 查看死锁 |
| C语言对xml文件的读写操作 | “抄袭事件”判决书 |
| vim显示行号、语法高亮、自动缩进的设置 | 见过最好的git入门教程 |
| linux C语言获取系统内存信息 | 使用STL的hash_map要点 |
| TCP拥塞控制算法内核实现剖析（九） | Java内部类总结（吐血之作） |

	面包板 ¥22.68/PCS		加速度计 ¥1.00/个		阿里巴巴批发 ¥500.00/片		pic编程器 ¥120.00/件	1 2 3
---	-------------------	---	-----------------	--	---------------------	---	---------------------	-------------

查看评论

- 44楼 [sgchen](#) 2014-05-31 08:52发表
-  真的很赞，楼主努力
- 43楼 [liujiangai](#) 2014-05-26 20:11发表
-  楼主写的很全面，可是我运行test3的时候不知道是什么原因ping 192.168.1.10后，在另一个终端运行显示success: device: eth0后就不在出现任何东西，这是什么原因啊？？谢谢。。。
- 42楼 [宇宙书童涛涛](#) 2014-04-04 22:41发表
-  正好在学c++和网络tcp通讯方面的东东, 楼主你这个用来上手真是强大！顶了

41楼 [likeyiyy](#) 2014-04-02 18:45发表



谢谢。

40楼 [话题在绕弯](#) 2014-03-21 09:28发表



很给力，详细而流程清晰。 不错不错

39楼 [tianyashuibin](#) 2014-02-27 10:45发表



谢谢楼主分享，最近正要做抓包协议分析。

38楼 [非非主流](#) 2014-01-16 11:49发表



楼主你好，我在ubuntu下运行编译好的第二个例子的时候，使用“./test2”会出现设备名“dbus-system”，并且也有相应的包；使用“sudo ./test2”会显示设备名“eth0”，但是出现“did not capture packet”。这是怎么回事？还有楼主运行的时候有没有联网？

前面我严格按照你的步骤来，配置我的eth0的ip，mask等；
我的网卡接口有3个，eth0，wlan0，lo；

37楼 [_离子](#) 2013-12-21 15:55发表



楼主讲的很细致！
ret = pcap_lookupnet(dev, &netp, &maskp, errbuf);
第二个参数是网络号，并非IP。

36楼 [阿赢](#) 2013-12-17 11:09发表



mac地址过滤关键字写错，应该是ether，不是ehter

35楼 [Michael_Xin_CV](#) 2013-10-30 23:26发表



狂赞！ 楼主好强大~

34楼 [farways](#) 2013-10-18 12:37发表



真好，正需要。

33楼 [Xiaohua_C](#) 2013-10-13 15:37发表



感谢！

32楼 [jingtingkang](#) 2013-10-10 10:11发表



不错，写的非常到位！

31楼 [finder_cs](#) 2013-06-26 17:05发表



一定要打开网卡的混杂模式，不知道为什么用ioctl代码打开混杂模式不对。还是谢谢楼主。

30楼 [lynnhua_](#) 2013-06-24 16:21发表



太帅了

29楼 [yarsen](#) 2013-06-16 16:21发表



楼主你好，我将来从事ipv6方面的工作，现在需要学习下libpcap，libnet等库。我看了下分享在download上的资料，那应该是本杂志的吧？如何获取了。
另，在学习网络安全方面有何建议？比如学习的规划步骤的。
谢谢

28楼 [blooney](#) 2013-05-20 19:00发表



谢谢楼主的详细介绍，学习了。
请教一个问题，用这个libpcap循环打印以太网帧的话，为什么没有最前边8个字节的前导码呢？

27楼 [windzmy](#) 2013-05-17 10:25发表



很好很强大，转一个。晚上回去仔细研究。

26楼 [nemo2011](#) 2013-05-06 20:30发表



int pcap_lookupnet()获得的是网关的地址。不是device的ip地址。

25楼 [Dayin_mao](#) 2013-04-24 09:48发表

赞一个~



24楼 [hellostar1912](#) 2013-04-21 10:55发表



最近在研究抓包的问题，楼主这篇文档帮助很大，谢谢

23楼 [laifu_ma](#) 2013-04-12 14:54发表



研究libpcap中，学习。。。

22楼 [RabbitToDancing](#) 2013-04-11 16:30发表



博主。我想问一下，可以用libcap抓到802.11的管理帧吗？在mac层抓包可以吗？应该怎么弄呀。

21楼 [sjtupeitao](#) 2013-03-28 15:24发表



学习了。。。

20楼 [xyiaiguozhe](#) 2013-03-21 20:21发表



学习了

19楼 [war189](#) 2013-03-13 13:40发表



好东西

18楼 [yuanxiaowu](#) 2013-01-18 16:51发表



不错，学习了！！

17楼 [luckywang1103](#) 2013-01-02 20:56发表



写的好，学习了！

16楼 [wangjc_strive](#) 2012-12-18 10:08发表



学习了，很好的文章，谢谢楼主
还有个问题，就是在获取网络接口，pcap_lookupdev()函数的地方，我希望手动设置我希望的网络接口该怎么实现呢？在网上找了一下没有找到，希望楼主能解答一下，小弟不胜感激！

Re: [LinuxMan](#) 2012-12-26 09:54发表



回复imei857：你指的是不是eth1这样？别的网卡

只需要做个字符指针就行了。

15楼 [zhangyouneng](#) 2012-12-17 17:20发表



学习了，很不错

14楼 [typiloong](#) 2012-12-17 16:30发表



受益良多，感谢楼主！

13楼 [zhouwu_linux](#) 2012-12-07 08:28发表



非常感谢~~！还有没有的libnet的实例呢，如果把能libnet结合起来的实例就最好不过的！！

12楼 [haoshen](#) 2012-12-06 22:50发表



学习了，大牛一个

11楼 [tx1990](#) 2012-12-05 09:48发表



谢谢，直接使用了你的代码，哈哈

10楼 [柏杨1314](#) 2012-11-27 15:57发表



楼主能不能赐教一下，在winpcap驱动下开发抓包工具的方法？怎么使用Examples-pcap里的例子，在vc2008中，是怎么把Examples-pcap中的；例子引用到vc里面的，求解啊，愿高手不吝赐教.....

9楼 [zhiying678](#) 2012-11-22 22:18发表



楼主好棒 谢谢 哦哦

8楼 [ggygz](#) 2012-11-15 09:41发表



非常感谢楼主。
初学者，看了man page 和一些介绍，都没收获，唯独楼主这篇有讲解又有例子，非常靠谱。

7楼 [oracle_1010](#) 2012-11-02 13:57发表



还是想请教一下，过滤表达式该如何过滤http协议层呢

6楼 [xiaohongbo2009](#) 2012-10-10 16:27发表



谢谢楼主分享
受教了

5楼 [Guccang](#) 2012-09-04 13:38发表



博主威武

4楼 [hg982104747](#) 2012-08-29 15:19发表



很好，很详细，谢谢楼主了。看英文的官方文档真心不习惯啊

3楼 [zqm175899960](#) 2012-08-21 23:12发表



好强大，转了，谢谢博主

2楼 [左眼看到鬼](#) 2012-08-13 10:24发表



好强大。。。转载了，谢谢博主

1楼 [x123niuniuniu](#) 2012-05-16 17:23发表



学习了，很强大！

您还没有登录,请[\[登录\]](#)或[\[注册\]](#)

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

核心技术类目

[全部主题](#) [Java](#) [VPN](#) [Android](#) [iOS](#) [ERP](#) [IE10](#) [Eclipse](#) [CRM](#) [JavaScript](#) [Ubuntu](#) [NFC](#)
[WAP](#) [jQuery](#) [数据库](#) [BI](#) [HTML5](#) [Spring](#) [Apache](#) [Hadoop](#) [.NET](#) [API](#) [HTML](#) [SDK](#)
[IIS](#) [Fedora](#) [XML](#) [LBS](#) [Unity](#) [Splashtop](#) [UML](#) [components](#) [Windows Mobile](#) [Rails](#)
[QEMU](#) [KDE](#) [Cassandra](#) [CloudStack](#) [FTC](#) [coremail](#) [OPhone](#) [CouchBase](#) [云计算](#) [iOS6](#)
[Rackspace](#) [Web App](#) [SpringSide](#) [Maemo](#) [Compuware](#) [大数据](#) [aptech](#) [Perl](#) [Tornado](#) [Ruby](#)
[Hibernate](#) [ThinkPHP](#) [Spark](#) [HBase](#) [Pure](#) [Solr](#) [Angular](#) [Cloud Foundry](#) [Redis](#) [Scala](#)
[Django](#) [Bootstrap](#)