

实验1 数据包嗅探和伪造

王美珍

mzwang@hust.edu.cn

QQ: 64205973

主要内容

- **1 发送和接收报文**
- **2 监听（嗅探）报文**
- **3 伪造报文**
- **4 Scapy Vs. C**
- **5 字节序和校验和**

实验环境（**for**自己电脑安装虚拟机）

- Ubuntu Seed虚拟机下载地址：
 - QQ群空间
- 虚拟机软件：vmware (15.5.0及兼容版本)
+ vmware tools
- ubuntu系统的用户密码
 - 普通用户： seed 密码: dees
 - 超级用户： root 密码： seedubuntu
- 实验采用一个虚拟机，多个容器来完成

实验环境(**for** 实践教学平台)

■ 实践教学平台登录**URL**: <https://222.20.126.111:8443>,

■ 虚拟机管理方式:

1) 通过实践平台web方式下VRC连接;

2) 直接ssh连接, 此种情况下, 需要安装openvpn (2.5.x), 通过openvpn访问虚拟机。Openvpn的配置文件、登录账号以及地址, 私信跟我联系。

■ 虚拟机系统的用户密码

普通用户: ubuntu 密码:123456

设置seed账号的密码: `sudo passwd seed`

设置root账号的密码: `sudo passwd`

实验环境(**for** 实践教学平台)

- 创建虚拟机镜像（用**root**用户操作）：
- `#tar -cpf /home/seed/buildrpm.tar --directory=/ --exclude=proc --exclude=sys --exclude=dev --exclude=run /`
- `#cat /home/seed/buildrpm.tar | docker import - seedubuntu`

□ 官网容器操作参考手册：

<https://github.com/seed-labs/seed-labs/blob/master/manuals/docker/docker.md>

docker容器的使用

□ 容器查看

- `docker ps -a`, 可以看到已有一个server (没有的话可以自己创建)

□ 容器创建

- `docker run -it --name=user --hostname=user --privileged "seedubuntu" /bin/bash`

□ 容器启用/停止

- `docker start/stop 容器名`

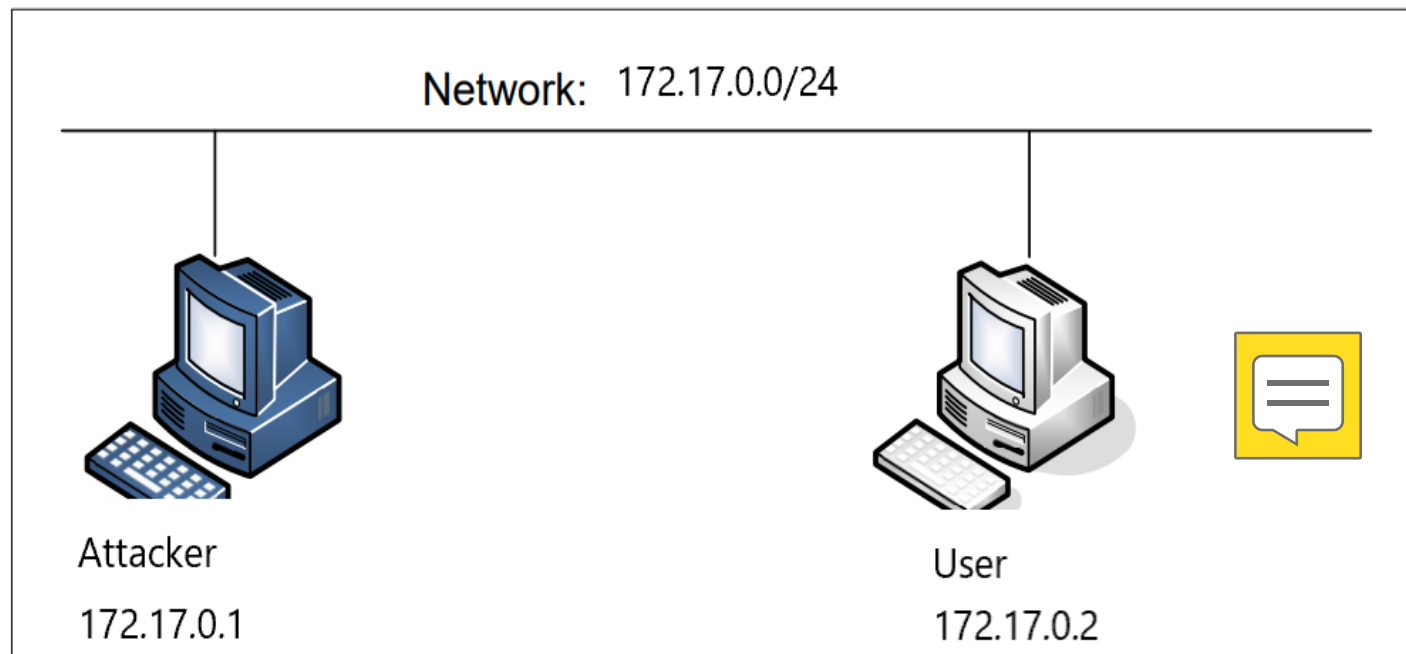
□ 进入容器的命令行

- `docker exec -it 容器名 /bin/bash`

□ 虚拟机和容器之间互拷数据

- `sudo docker cp 文件名 容器名:/路径`
 - `sudo docker cp 容器名:/路径/文件名 /路径/文件名`
-

2.1 网络环境搭建

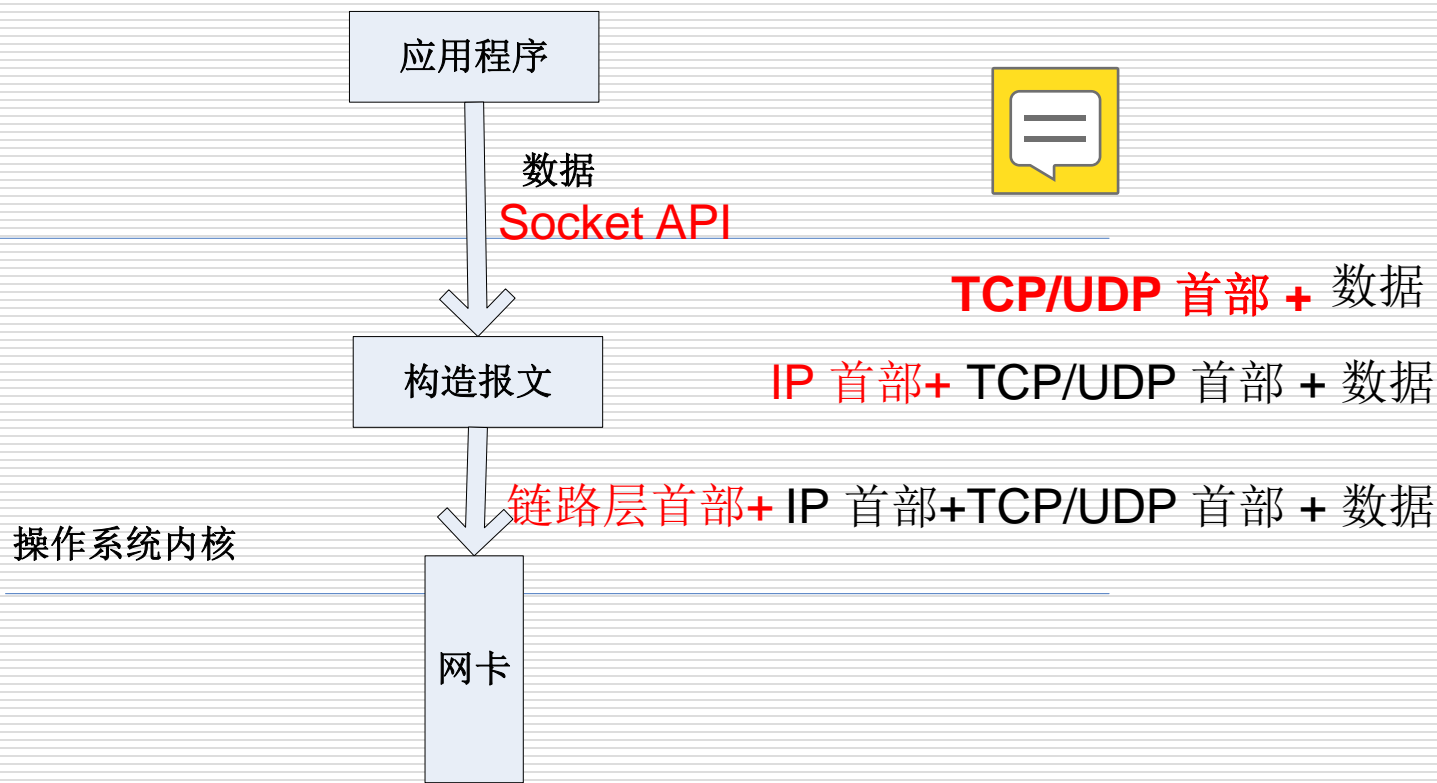


实验环境截图



1.1 socket发送报文

□ Socket发送报文



1.1 socket发送报文(python)

□ 示例：UDP 客户端

```
#!/usr/bin/env python3
```

```
import socket
```

```
IP = "127.0.0.1"
```

```
PORT = 9090
```

```
data = b'Hello, World!'
```

```
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
```

```
sock.sendto(data, (IP, PORT))
```

测试结果：本机用nc在9090端口上进行监听

```
[03/23/22]seed@VM:~$ nc -l -v 9090
Listening on [0.0.0.0] (family 0, port 9090)
Hello, World!
```

1.1 socket发送报文（C语言）

```
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h>
#include <arpa/inet.h>

void main()
{
    struct sockaddr_in dest_info;
    char * data = "Hello World!";

    //create a network socket
    int sock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);

    memset((char *)&dest_info, 0 ,sizeof(dest_info));
    dest_info.sin_family = AF_INET;
    dest_info.sin_addr.s_addr = inet_addr("127.0.0.1");
    dest_info.sin_port = htons(9090);

    sendto(sock, data, strlen(data), 0 , (struct sockaddr*)&dest_info, sizeof(dest_info));
    close(sock);
}
```

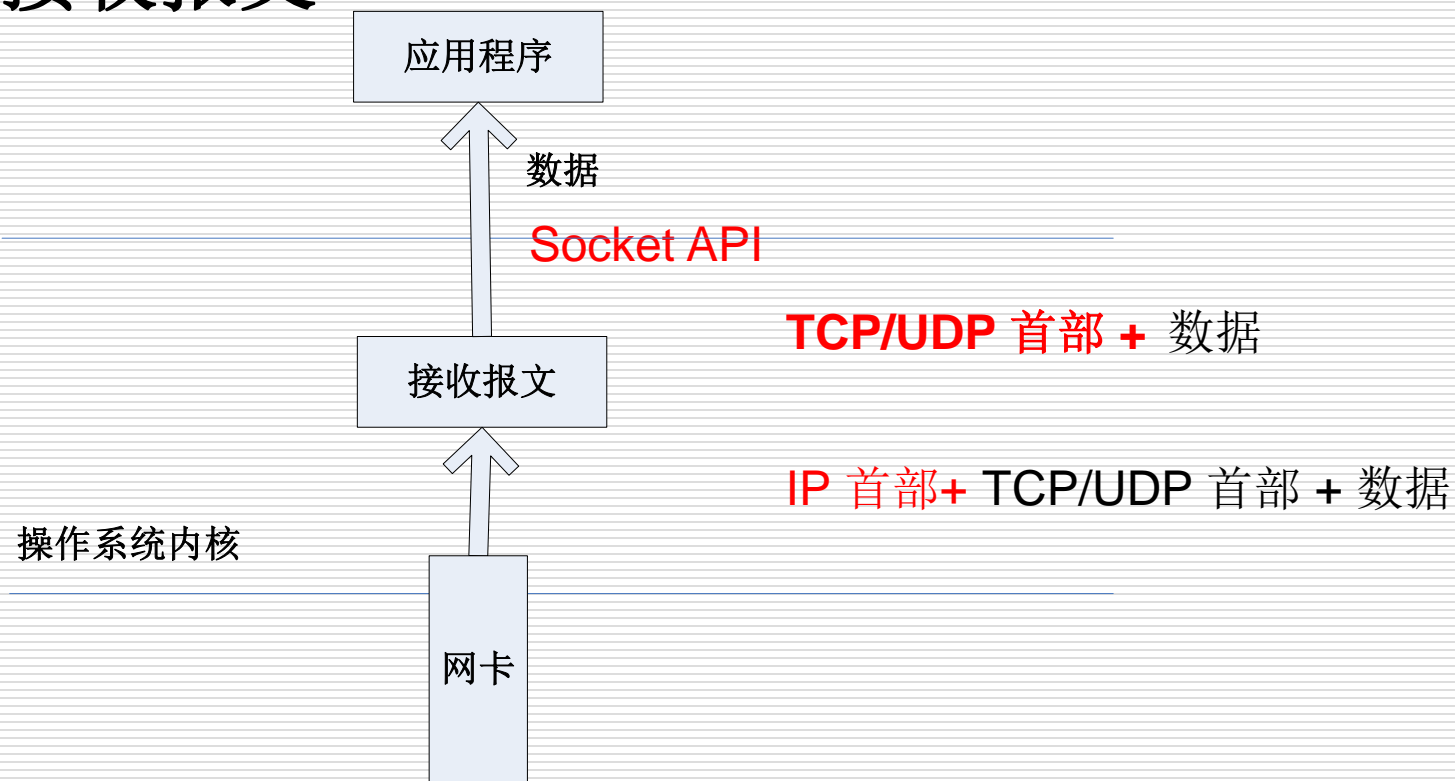


```
}
[03/23/22]seed@VM:~$
[03/23/22]seed@VM:~$ nc -luv 9090
Listening on [0.0.0.0] (family 0, port 9090)
Hello World! 
```

```
[03/23/22]seed@VM:~$ man inet_addr
[03/23/22]seed@VM:~$ vi udp_c.c.
[03/23/22]seed@VM:~$ vi udp_c.c
[03/23/22]seed@VM:~$ gcc -o udp_c udp_c.c
[03/23/22]seed@VM:~$ ./udp_c
[03/23/22]seed@VM:~$ ./udp_c
[03/23/22]seed@VM:~$
```

1.2 socket接收报文

□ 接收报文



1.2 socket接收报文 (python)

```
#!/usr/bin/env python3
import socket

IP = "0.0.0.0"
PORT = 9090

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((IP,PORT))

while True:
    data,(ip,port) = sock.recvfrom(1024)
    print("Sender: {} and Port: {}".format(ip, port))
    print("Received message: {}".format(data))
```



```
[03/23/22]seed@VM:~$
[03/23/22]seed@VM:~$ nc -u 127.0.0.1 9090
it's a test
```

```
[03/23/22]seed@VM:~$
[03/23/22]seed@VM:~$ python3 udp_s.py
Sender: 127.0.0.1 and Port: 40977
Received message: b"it's a test\n"
```

1.2 socket接收报文 (C)

```
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h>
#include <arpa/inet.h>

void main()
{
    struct sockaddr_in server;
    struct sockaddr_in client;
    int clientlen;
    char buf[1500];

    //create a network socket
    int sock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);

    memset((char *)&server, 0, sizeof(server));
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = htonl(INADDR_ANY);
    server.sin_port = htons(9090);

    if( bind(sock, (struct sockaddr *)&server, sizeof(server)) < 0){
        printf("Error on binding");
        close(sock);
        return;
    }

    while(1){
        bzero(buf,1500);
        recvfrom(sock, buf, 1500-1, 0, (struct sockaddr *)&client, &clientlen);
        printf("received message: %s\n",buf);
    }

    close(sock);
}
```

```
[03/23/22]seed@VM:~$ vi udp_s.c
[03/23/22]seed@VM:~$ ./udp_s
received message: hello

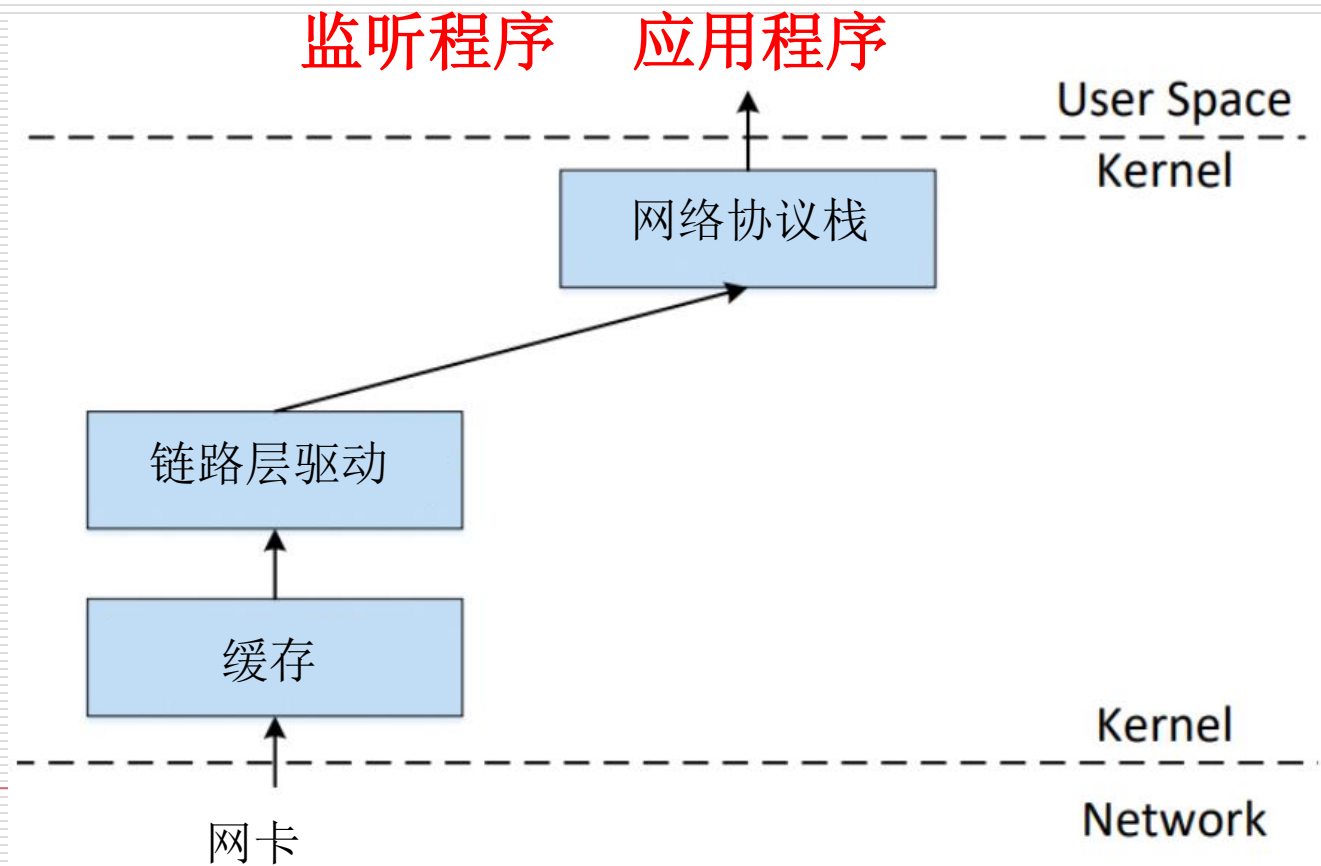
received message: hahaha
```

```
C
[03/23/22]seed@VM:~$ nc -u 127.0.0.1 9090
hello
hahaha
```

2 报文嗅探

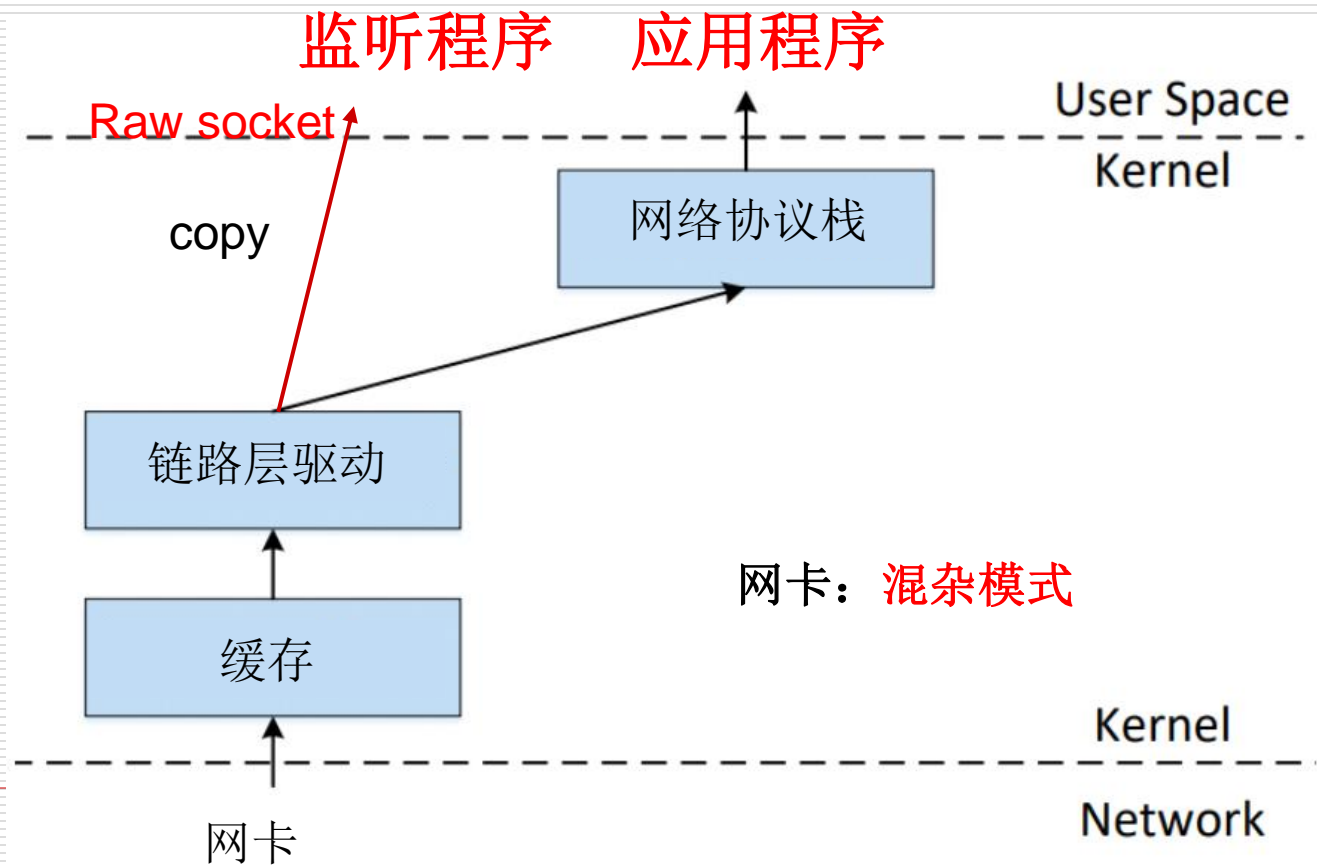


□ 报文如何从底层接收上来？



2 如何获得报文的拷贝？

□ Raw Socket



2.1 用原始套接字进行报文捕获


```
int main() {
    int PACKET_LEN = 512;
    char buffer[PACKET_LEN];
    struct sockaddr saddr;
    struct packet_mreq mr;

    // Create the raw socket
    int sock = socket(AF_PACKET, SOCK_RAW, htons(ETH_P_ALL));

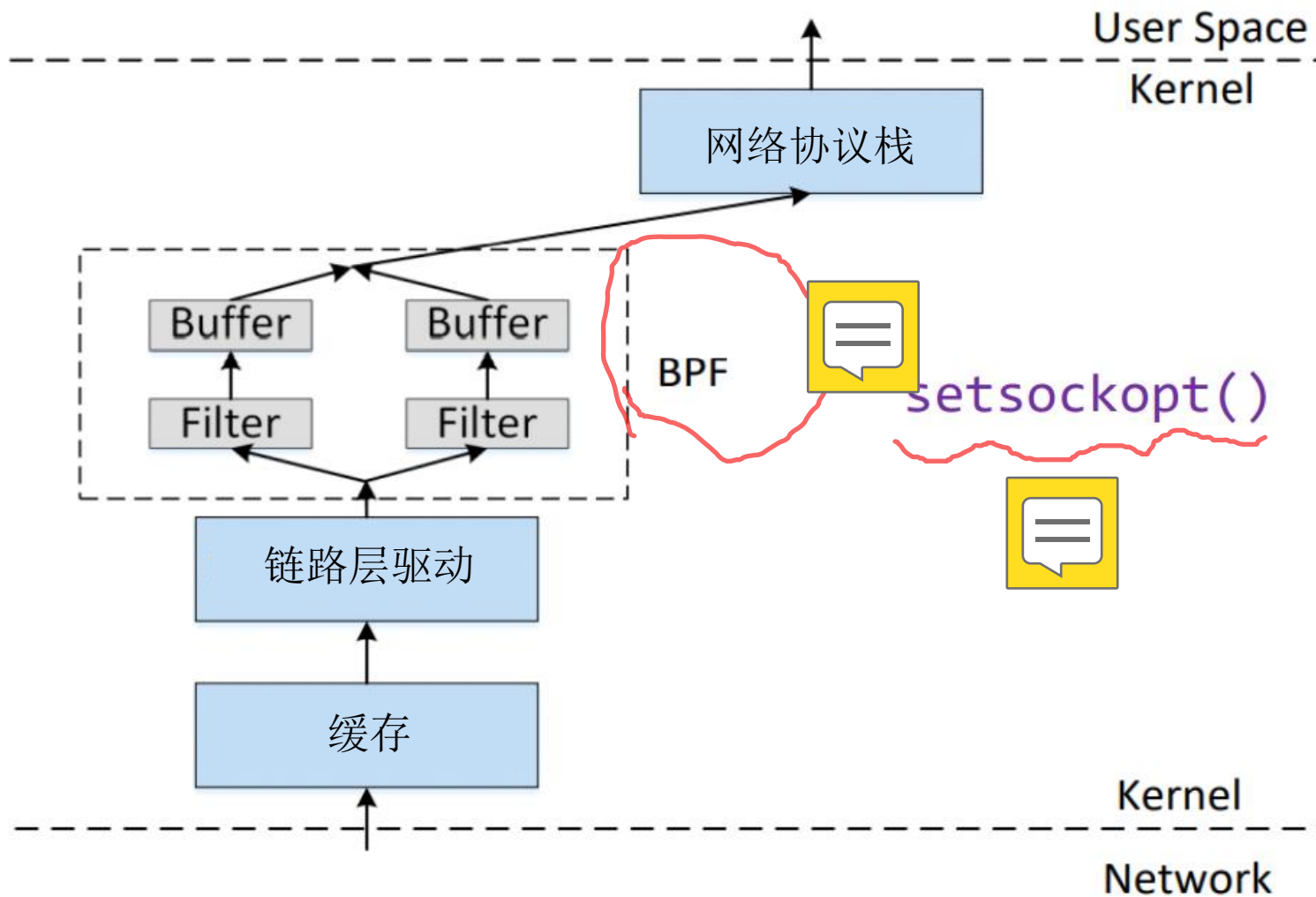
    // Turn on the promiscuous mode.
    mr.mr_type = PACKET_MR_PROMISC;
    setsockopt(sock, SOL_PACKET, PACKET_ADD_MEMBERSHIP, &mr, sizeof(mr));

    // Getting captured packets
    while (1) {
        int data_size = recvfrom(sock, buffer, PACKET_LEN, 0,
                                &saddr, (socklen_t*)sizeof(saddr));
        if(data_size) printf("Got one packet\n");
    }

    close(sock);
    return 0;
}
```



2.1 过滤出不想要的报文



2.2 PCAP: Packet Capture API

- ❑ 最开始来源于**tcpdump**
 - ❑ 多平台支持:
 - Linux: libpcap
 - Windows: winpcap 和 npcap
 - ❑ **C**语言写的，实现语言实现封装
 - ❑ 基于**pcap**的工具:
 - Wireshark, tcpdump, scapy, McAfee, nmap, snort
-

2.2 用pcap写的监听程序

```
int main()
{
    pcap_t *handle;
    char errbuf[PCAP_ERRBUF_SIZE];
    struct bpf_program fp;
    char filter_exp[] = "udp or icmp ";
    bpf_u_int32 net;

    // Step 1: Open live pcap session on NIC with interface name
    handle = pcap_open_live("enp0s3", 8192, 1, 1000, errbuf);

    // Step 2: Compile filter_exp into BPF psuedo-code
    pcap_compile(handle, &fp, filter_exp, 0, net);
    pcap_setfilter(handle, &fp);

    // Step 3: Capture packets
    pcap_loop(handle, -1, got_packet, NULL);

    pcap_close(handle); //Close the handle
    return 0;
}
```



编译: gcc -o sniff sniff.c -lpcap




2.2 用pcap写的监听程序（续）

```
void got_packet(u_char *args, const struct pcap_pkthdr *header, const u_char *packet)
{
    printf("\nGot a packet\n");
    struct ethheader *eth=(struct ethheader *)packet;

    if(ntohs(eth->ether_type) == 0x800) IP协议
    {
        struct ipheader *ip = (struct ipheader *)(packet + sizeof(struct ethheader));
        printf("    From: %s\n",inet_ntoa(ip->iph_sourceip));
        printf("    To: %s\n",inet_ntoa(ip->iph_destip));

        switch(ip->iph_protocol) {
            case IPPROTO_TCP:
                printf("    Protocol: TCP\n");
                break;
            case IPPROTO_UDP:
                printf("    Protocol: UDP\n");
                break;
            case IPPROTO_ICMP:
                printf("    Protocol: ICMP\n");
                break;
            default:
                printf("    Protocol: Others\n");
                break;
        }
    }
}
```



2.2 Pcap过滤器的例子

- ❑ **dst host 10.0.2.5:**只捕获目的ip为10.0.2.5的数据包
 - ❑ **src host 10.0.2.6:**只捕获源ip为10.0.2.6的数据包
 - ❑ **host 10.0.2.6 and src port 9090:** 只捕获源或目的为10.0.2.6，并且源端口为9090的数据包
 - ❑ **proto tcp:** 只捕获tcp数据包
-

2.3 Python + scapy

□ **Scapy**安装

- `sudo apt install python-scapy`

□ **Python**程序中加载**scapy**模块

- `from scapy.all import *`
-

2.3 Sniffer 例程1

```
#!/usr/bin/python3  
from scapy.all import *  
pkt = sniff(iface='docker0',  
            filter='icmp or udp',  
            count=10)  
pkt.summary()
```

2.3 Sniffer 例程2

□ 执行回调函数

```
#!/usr/bin/python3

from scapy.all import *

def process_packet(pkt):
    #hexdump(pkt)
    pkt.show()
    print("-----")

f = 'udp and dst portrange 50-55 or icmp'

sniff(iface='enp0s3', filter = f, prn=process_packet)
```

2.3 显示报文的不同方式

❖ Using hexdump()

```
>>> hexdump(pkt)
0000  52 54 00 12 35 00 08 00 27 77 2E C3 08 00 45 00  RT..5...'w....E.
0010  00 54 F2 29 40 00 40 01 2C 68 0A 00 02 08 08 08  .T.)@.@.,h.....
0020  08 08 08 00 98 01 10 C7 00 02 B8 66 65 5E 3A 6D  .....fe^:m
0030  0C 00 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15  .....
0040  16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25  ..... !"#$.%
0050  26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35  &'()*+,-./012345
0060  36 37                                           67
```

❖ Using pkt.show()

```
>>> pkt.show()
###[ Ethernet ]###
  dst      = 52:54:00:12:35:00
  src      = 08:00:27:77:2e:c3
  type     = IPv4
###[ IP ]###
  version  = 4
  ihl      = 5
  ...
  proto    = icmp
  chksum   = 0x3c9a
  src      = 10.0.2.8
  dst      = 8.8.8.8
  \options \
###[ ICMP ]###
  type     = echo-request
  code     = 0
  chksum   = 0x6905
  id       = 0x107a
  seq      = 0x2
###[ Raw ]###
  load     = '\x90ee^\x91\xb7\ ...'
```

2.3 理解scapy的层次

□ 通过层堆叠

```
>>> pkt
<Ether type=IPv4 |<IP frag=0 proto=udp |<UDP |<Raw load='hello' |>>>>
```

```
>>> pkt.payload
<IP frag=0 proto=udp |<UDP |<Raw load='hello' |>>>
```

```
>>> pkt.payload.payload
<UDP |<Raw load='hello' |>>
```

```
>>> pkt.payload.payload.payload
<Raw load='hello' |>
```

2.3 访问层

□ 检查层的类型

```
<Ether type=IPv4 |<IP frag=0 proto=udp |<UDP |<Raw load='hello' |>>>>
```

```
>>> pkt.haslayer(UDP)
```

```
True
```

```
>>> pkt.haslayer(TCP)
```

```
0
```

```
>>> pkt.haslayer(Raw)
```

```
True
```

```
>>> pkt[UDP]
```

```
<UDP |<Raw load='hello' |>>
```

```
>>> pkt.getlayer(UDP)
```

```
<UDP |<Raw load='hello' |>>
```

```
>>> pkt[Raw]
```

```
<Raw load='hello' |>
```

```
>>> pkt[Raw].load
```

```
b'hello'
```

□ 访问层

2.3 Sniffer 例子3

```
#!/usr/bin/python3

from scapy.all import *

def process_packet(pkt):
    if pkt.haslayer(IP):
        ip = pkt[IP]
        print("IP: {} --> {}".format(ip.src, ip.dst))

    if pkt.haslayer(TCP):
        tcp = pkt[TCP]
        print("    TCP  port: {} --> {}".format(tcp.sport, tcp.dport))

    elif pkt.haslayer(UDP):
        udp = pkt[UDP]
        print("    UDP  port: {} --> {}".format(udp.sport, udp.dport))

    elif pkt.haslayer(ICMP):
        icmp = pkt[ICMP]
        print("    ICMP type: {}".format(icmp.type))

    else:
        print("    Other protocol")

sniff(iface='enp0s3', filter='ip', prn=process_packet)
```

2.3 获得协议类的信息

□ 获得属性名字

- ls(IP)

- ls(TCP)

- ..

□ 获得方法名字

- help(IP)

- help(TCP)

2.3 任务1：用scapy监听报文

```
#!/usr/bin/python3
from scapy.all import *

print("SNIFFING PACKETS.....")

def print_pkt(pkt):
    print("Source IP:", pkt[IP].src)
    print("Destination IP:", pkt[IP].dst)
    print("Protocol:", pkt[IP].proto)
    print("\n")

pkt = sniff(filter='ip',prn=print_pkt)
```

filter='ip dst 10.0.2.46'
filter='ip dst 10.0.2.46 or tcp
port 23'

- ❑ **sudo python sniff.py**
- ❑ 是否能捕获到其它主机的流量？
- ❑ 试一试其它过滤器

■ <http://biot.com/capstats/bpf.html>

3 报文伪造

□ 发送报文（python）

```
#!/usr/bin/python3
```

```
import socket
```

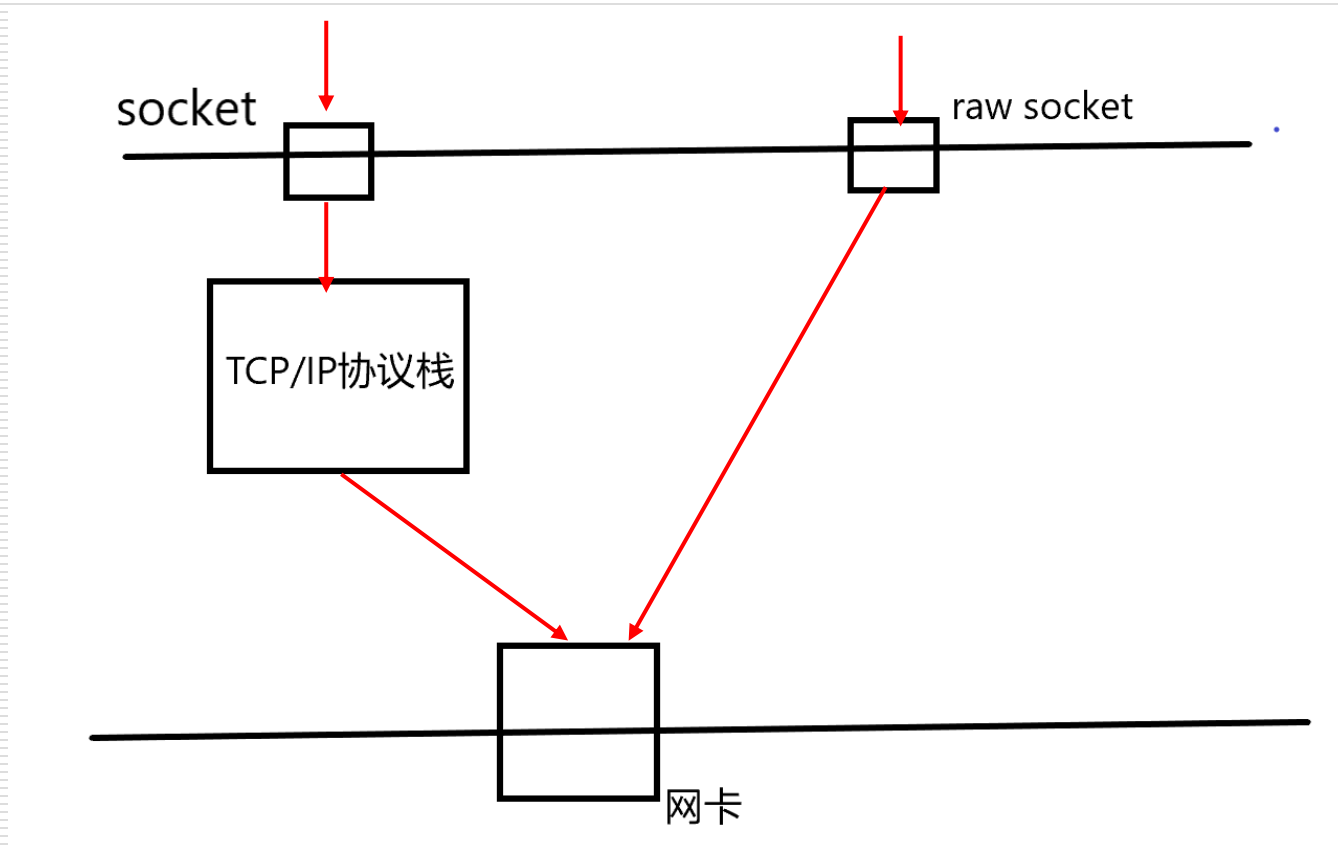
```
IP    = "127.0.0.1"
```

```
PORT = 9090
```

```
data = b'Hello, World!'
```

```
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)  
sock.sendto(data, (IP, PORT))
```

3.1 用原始套接字伪造报文



3.1 原始套接字发送报文

```
int sd;
struct sockaddr_in sin;
char buffer[1024]; // You can change the buffer size

/* Create a raw socket with IP protocol. The IPPROTO_RAW parameter
 * tells the system that the IP header is already included;

 * this prevents the OS from adding another IP header. */
sd = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);
if(sd < 0) {
    perror("socket() error"); exit(-1);
}

/* This data structure is needed when sending the packets
 * using sockets. Normally, we need to fill out several
 * fields, but for raw sockets, we only need to fill out
 * this one field */
sin.sin_family = AF_INET;

// Here you can construct the IP packet using buffer[]
//   - construct the IP header ...
//   - construct the TCP/UDP/ICMP header ...
//   - fill in the data part if needed ...
// Note: you should pay attention to the network/host byte order.

/* Send out the IP packet.
 * ip_len is the actual size of the packet. */
if(sendto(sd, buffer, ip_len, 0, (struct sockaddr *)&sin,
        sizeof(sin)) < 0) {
    perror("sendto() error"); exit(-1);
}
```

3.2 用scapy伪造报文

❑ 伪造ICMP报文

```
#!/usr/bin/python3
from scapy.all import *

print("SENDING SPOOFED ICMP PACKET.....")
ip = IP(src="1.2.3.4", dst="93.184.216.34")
icmp = ICMP()
pkt = ip/icmp
pkt.show()
send(pkt, verbose=0)
```

❑ 伪造UDP报文

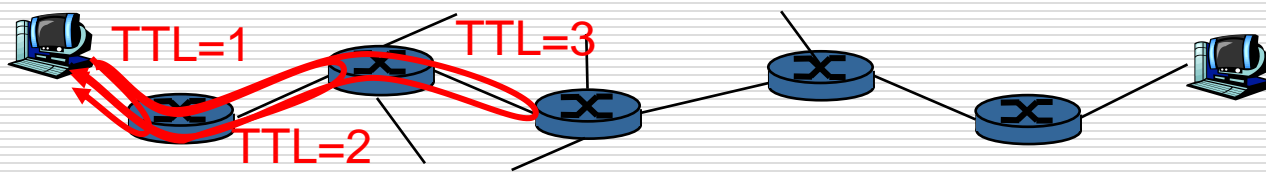
```
#!/usr/bin/python3
from scapy.all import *

print("SENDING SPOOFED UDP PACKET.....")
ip = IP(src="1.2.3.4", dst="10.0.2.69") # IP Layer
udp = UDP(sport=8888, dport=9090)      # UDP Layer
data = "Hello UDP!\n"                  # Payload
pkt = ip/udp/data
pkt.show()
send(pkt, verbose=0)
```

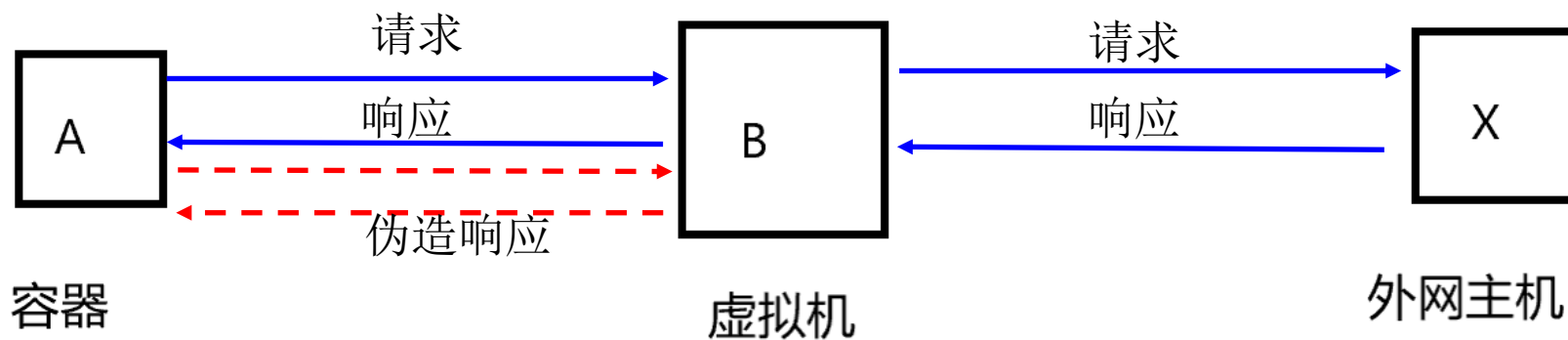
3.2 Scapy构造报文实验：路由追踪

□ **Traceroute 程序**：为从源端到目的地的因特网端到端路径上的路由器提供时延计量方法。对所有到目的地路径上的路由器 i ：

- 发送分组, 设定 $TTL=i$
- 路由器 i 将向发送者返回ICMP差错信息(**TTL超时**)
- 发送者计算发送分组和收到响应之间的时间间隔



3.3 监听请求伪造回应



3.3 Sniff-and-spoof例子

```
def spoof_pkt(pkt):  
    if ICMP in pkt and pkt[ICMP].type == 8:  ← echo request  
        print("Original Packet.....")  
        print("Source IP : ", pkt[IP].src)  
        print("Destination IP :", pkt[IP].dst)  
  
        ip = IP(src=pkt[IP].dst, dst=pkt[IP].src, ihl=pkt[IP].ihl)  
        ip.ttl = 99  
        icmp = ICMP(type=0, id=pkt[ICMP].id, seq=pkt[ICMP].seq)  
  
        if pkt.haslayer(Raw):  ← echo reply  
            data = pkt[Raw].load  
            newpkt = ip/icmp/data  
        else:  
            newpkt = ip/icmp  
  
        print("Spoofed Packet.....")  
        print("Source IP : ", newpkt[IP].src)  
        print("Destination IP :", newpkt[IP].dst)  
  
        send(newpkt, verbose=0)  
  
sniff(filter='icmp and src host 10.0.2.7', prn=spoof_pkt)
```

任务3:

- 在虚拟机上运行
 - **Python sniff_spoof_icmp.py**
 - 在**docker**里面运行:
 - ping 1.2.3.4
 - ping 8.8.8.8
 - ping 不存在的地址
- 是否都能收到回应?
-

4 Scapy 和 C比较

- 缺省值
 - 包（字节数组）对象/结构
 - 性能（实验）
 - Scapy: 发包106个/秒
 - C: 发包4000个/秒
 - 代码量
-

5 字节顺序

□ 大端（**big-endian**）和小端（**little-endian**）

- 大端字节顺序意味着先从大的字节保存，即大的字节放在内存的低端；
- 小端字节序意味着先从小的字节保存，即小的字节放在内存的低端；

0x87654321

在内存中的存放

0x1003	0x87
0x1002	0x65
0x1001	0x43
0x1000	0x21

小端

0x1003	0x21
0x1002	0x43
0x1001	0x65
0x1000	0x87

大端

5 字节顺序转换函数

- ❑ **htons():** 无符号短整型数从主机字节序→网络字节序
 - ❑ **htonl():** 无符号长整型数从主机字节序→网络字节序
 - ❑ **ntohs():** 无符号短整型数从网络字节序→主机字节序
 - ❑ **ntohl():** 无符号短整型数从网络字节序→主机字节序
-

6. 实验任务

- 按照指导手册进行实验，完成问题，在微助教平台提交
 - 设置过滤规则，嗅探数据包；
 - 构造报文，实现路由探测（可以程序发包，wireshark分析ICMP差错报告，记录路径上的路由器）
 - 监听并伪造报文（ping 不存在的地址如1.2.3.4，能伪造回应）
 - C语言和python分别实现
-

补充:

- 拷贝文件到容器（虚拟机上运行）
 - `sudo docker cp 文件名 容器名:/路径`
- 从容器拷贝文件到虚拟机（虚拟机上运行）
 - `sudo docker cp 容器名:/路径/文件名 /路径/文件名`

常见错误:

1. bind错误: 提示address in use

说明绑定的端口被使用了, 需要停掉原来的进程, 或者用一个新的端口号

可以通过`sudo netstat -naup` 查看udp端口使用的进程

`sudo netstat -natp` 查看tcp端口使用的进程

2. Sniff监听失败, 提示找不到设备

`sniff`的`iface`参数指定错误, 因为例子代码跟你的主机的接口名字不一样。可以`ifconfig`查看本机的接口, 跟容器通信的接口为`docker0`, 跟外网通信的接口为`enxxxxx`

3. 发送报文, 对方未接收到

可能是发送报文的地址填写错误, “127.0.0.1”是指的本机地址, 即发送报文的地址是到自己

4. C语言实现监听代码运行以后出现 “segmentation fault”

可能是过滤器的语法错误, 可以用 “`tcpdump -i 网卡名 过滤规则`” 验证过滤器的语法

5. 运行py脚本, 提示 “operation not permitted”

权限不够, 需要`sudo` 执行

6. 明明已经安装了scapy, 但是运行py脚本, 提示找不到scapy模块

请用普通用户执行 (不要用`root`用户, 提示为`$`, 而不是`#`), `sudo`执行

❑ 容器里面抓包命令tcpdump

```
root@3f796f9d8299:/# tcpdump
ERROR: ld.so: object '/home/seed/lib/boost/libboost_program_options.so.1.64.0' from LD_PRELOAD cannot be preloaded (cannot open shared object file): ignored.
ERROR: ld.so: object '/home/seed/lib/boost/libboost_filesystem.so.1.64.0' from LD_PRELOAD cannot be preloaded (cannot open shared object file): ignored.
ERROR: ld.so: object '/home/seed/lib/boost/libboost_system.so.1.64.0' from LD_PRELOAD cannot be preloaded (cannot open shared object file): ignored.
tcpdump: error while loading shared libraries: libcrypto.so.1.0.0: cannot open shared object file: Permission denied
root@3f796f9d8299:/#
```

解决办法（在容器里执行以下命令）：

`mv /usr/sbin/tcpdump /usr/bin/tcpdump`

`ln -s /usr/bin/tcpdump /usr/sbin/tcpdump`
