**计算机网络实验报告**

洪涛

U201717066

软工1705

目录

[一． 实验目的 2](#_Toc43284627)

[二． 软件功能需求描述 3](#_Toc43284628)

[三． 流量统计软件的设计 3](#_Toc43284629)

[1. 协议栈分析 3](#_Toc43284630)

[2. 协议处理 3](#_Toc43284631)

[3. 流量统计 3](#_Toc43284632)

[四． 流量统计软件的实现 3](#_Toc43284633)

[1. 头文件设计： 4](#_Toc43284634)

[2. 具体定义 4](#_Toc43284635)

[3. 实验结果 4](#_Toc43284636)

[五． 总结 10](#_Toc43284637)

## 实验目的

随着计算机网络技术的飞速发展，网络为社会经济做出越来越多的贡献，可以说计算机网络的发展已经成为现代社会进步的一个重要标志。但同时，计算机犯罪、黑客攻击、病毒入侵等恶性事件也频频发生。网络数据包捕获、监听与分析技术是网络安全维护的一个基本技术同时也是网络入侵的核心手段。通过基于网络数据包的截获和协议分析，对网络上传输的数据包进行捕获，可以获取网络上传输的非法信息，对维护网络安全起到重大作用。

本次实验的主要目的有：

1. 理解协议在通信中的作用；
2. 掌握抓包软件的开发；
3. 掌握协议解析的编程方法。
4. 利用所学的知识，制作一款流量统计软件。

## 软件功能需求描述

软件功能：

1. 利用winpcap捕获数据包，并可根据要求进行数据包过滤。
2. 根据IP协议，解析每个数据包的PCI，展示其在不同网络层次所使用的协议结构和具体信息。
3. 根据IP地址，统计源自该IP地址的流量，即捕获到的数据包的数量。

## 流量统计软件的设计

### 协议栈分析

由于TCP/IP协议采用分层的结构，这样在传输数据时，在网络数据的发送端是一个封装的过程，而在数据接收端则是分解的过程。

数据封装过程如图3.1所示

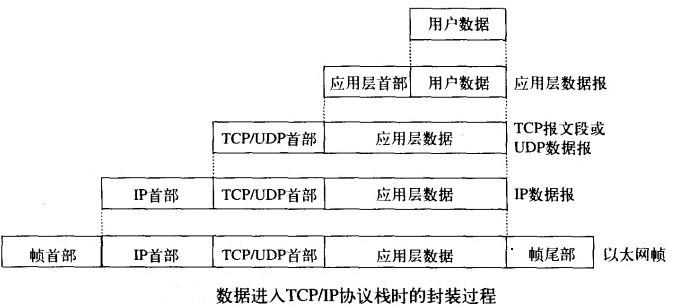


图3.1

而对接收到的包进行解析，就是上述封装的逆过程。

* 1. 以太网协议

在以太网的发展过程中，有很多种帧格式，其中以太网Ⅱ格式应用最为广泛，现在几乎是以太网的标准，它是由RFC894所定义。其帧格式如图3.2所示。

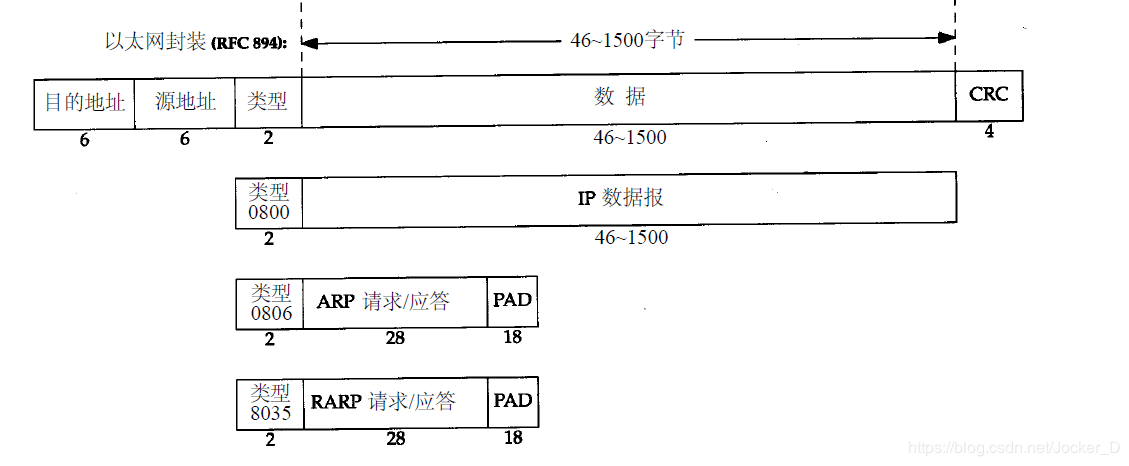


图3.2

* 1. ARP协议

ARP/RARP协议是进行IP地址和MAC地址相互转换的协议，网络通信中，链路层使用MAC地址进行实际的数据通信，而在网络层使用IP地址进行机器定位寻址。APR协议把IP地址映射为MAC地址，RARP相反。格式如图3.3所示。

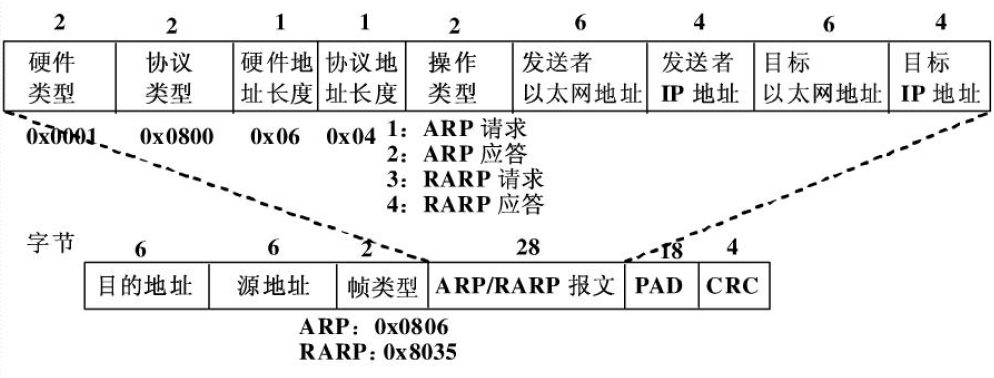


图3.3

* 1. IP协议

IP协议是Internet的核心协议，它工作在网络层，提供了不可靠无连接的数据传送服务。协议格式如图3.4所示。

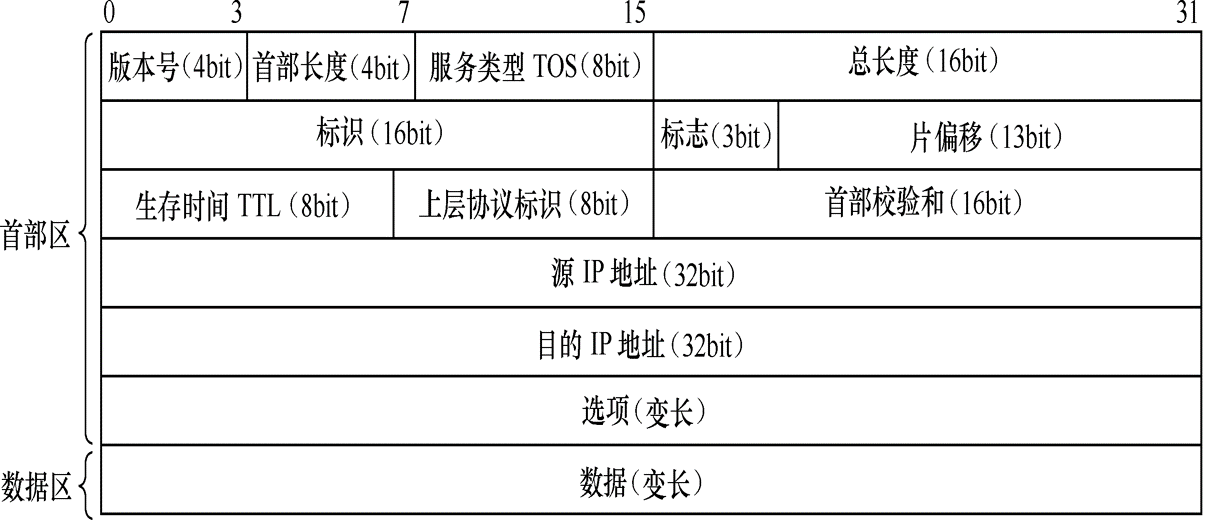


图3.4

* 1. ICMP协议

Internet控制信息协议(Internet Control Message Protocol),它提供了很多Internet的信息描述服务，能够检测网络的运行状况，通知协议有用的网络状态信息。ICMP是基于IP协议的，ICMP协议格式如图3.5 所示。

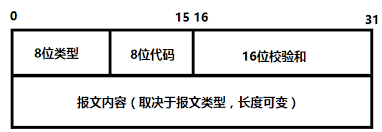


图3.5

* 1. TCP协议

TCP协议是基于连接的可靠的协议，它负责发收端的协定，然后保持正确可靠的数据传输服务。它是在IP协议上运行的，而IP无连接的协议，所以TCP丰富了IP协议的功能，使它具有可靠的传输服务。TCP协议格式如图3.6所示。

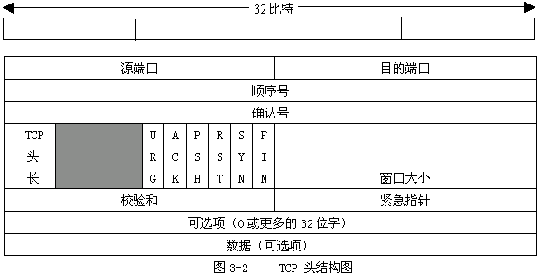


图3.6

* 1. UDP协议

用户数据报协议UDP是在IP协议上的传输层协议，它提供了无连接的协议服务，它在IP协议基础上提供了端口的功能，这样既可让应用程序进行通信了。UDP协议格式如图3.7所示。

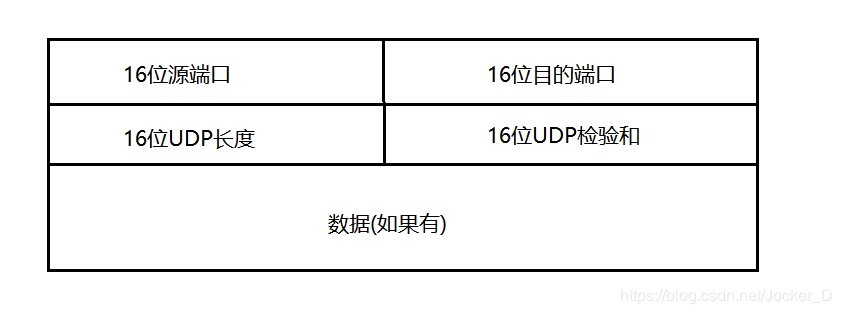


图3.7

### 2. 协议处理

Winpcap系统处理流程图3.8：

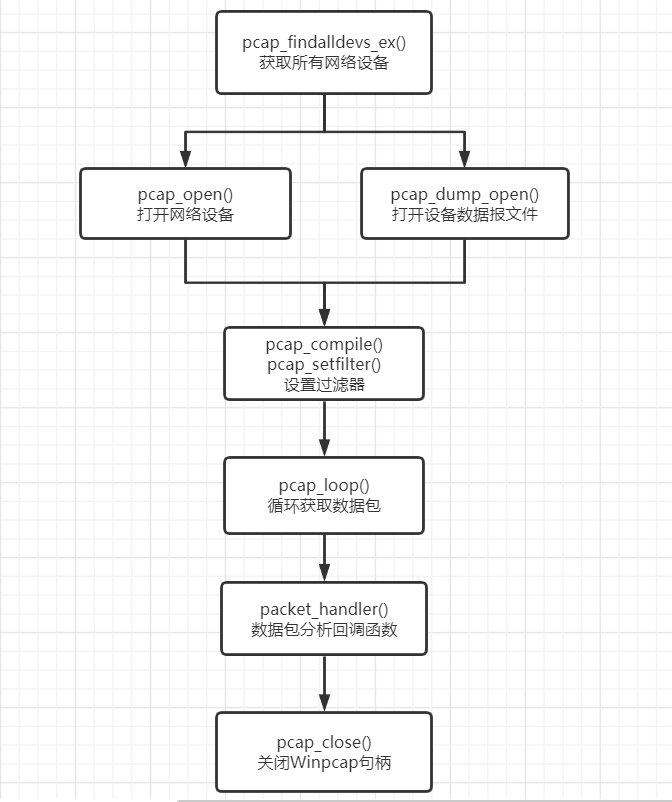


图3.8

协议分析流程图3.9：

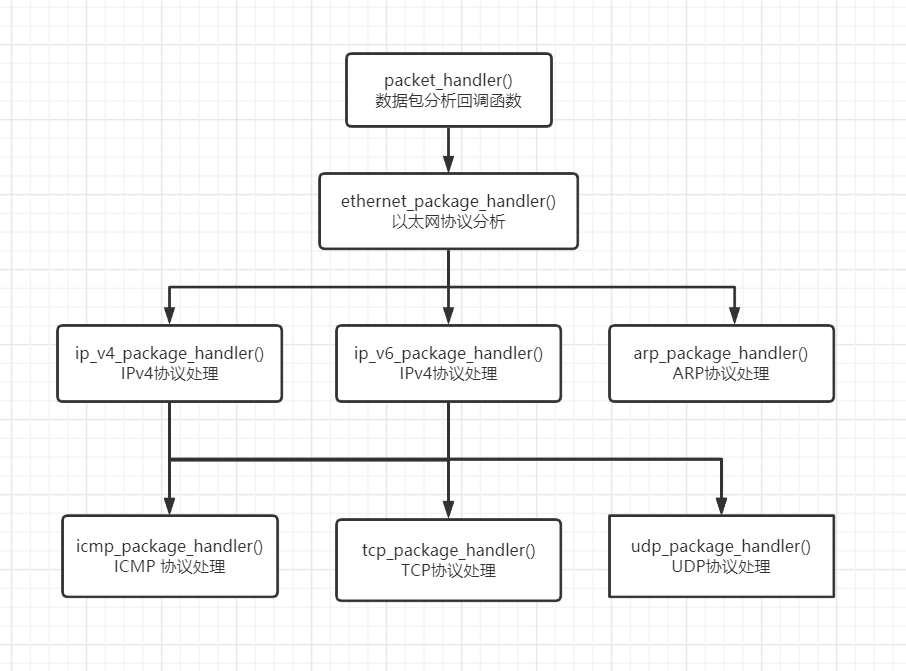


图3.9

Ethernet\_package\_handler()：用来解析以太网帧

IP\_v4\_package\_handler()：用来解析IPv4数据报

IP\_v6\_package\_handler()：用来解析IPv6数据报

Arp\_package\_handler()：用来解析ARP协议

ICMP\_package\_handler()：用来解析ICMP协议

Tcp\_package\_handler()：用来解析TCP报文段

Udp\_package\_handler()：用来解析UDP报文段

### 3. 流量统计

流量统计过程如3.10所示

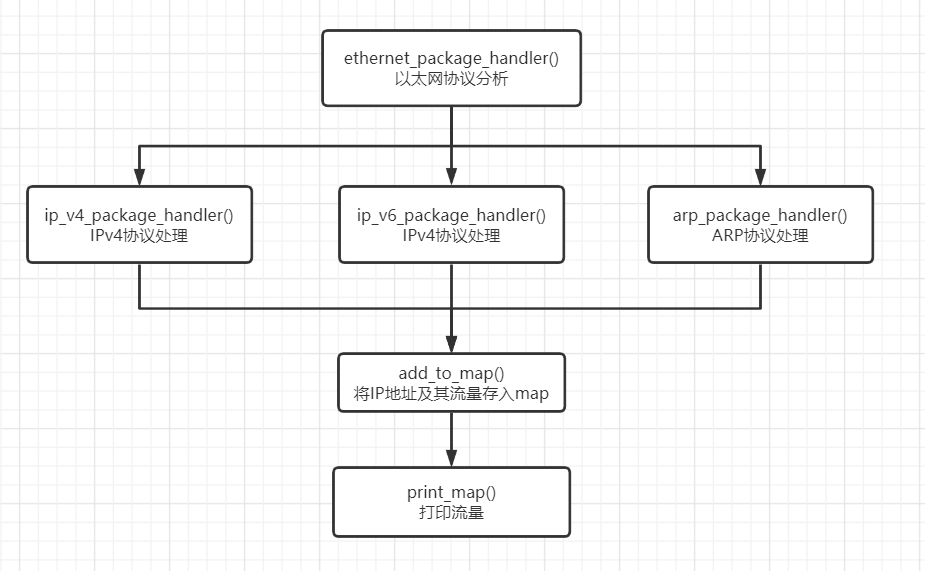


图3.10

在对网络层协议进行解析时，以IP地址为键，流量为值存入map中，如果该IP值已经存在，则流量加一，最后的值即表示该IP的流量

## 流量统计软件的实现

本软件采用C++实现，在VS 2017上编写

### 头文件设计：

#ifndef \_PAC\_ANA\_H

#define \_PAC\_ANA\_H

#ifdef \_MSC\_VER

/\*

\* we do not want the warnings about the old deprecated and unsecure CRT functions

\* since these examples can be compiled under \*nix as well

\*/

#define \_CRT\_SECURE\_NO\_WARNINGS

#endif

/\*set the environment head files\*/

#define WIN32

#pragma comment (lib, "ws2\_32.lib") //load ws2\_32.dll

/\*set the C++ head files\*/

#include <iostream>

#include <stdio.h>

#include <map>

#include <string>

#include <iomanip>

#include <sstream>

/\*set the wpcap head files\*/

#include "pcap.h"

#include <WinSock2.h>

#define DIVISION "--------------------"

#define B\_DIVISION "==================="

/\* 4 bytes IP address \*/

typedef struct ip\_v4\_address ip\_v4\_address;

/\* 16 bytes IP address \*/

typedef struct ip\_v6\_address ip\_v6\_address;

/\*8 bytes MAC addresss\*/

typedef struct mac\_address mac\_address;

/\*ethernet header\*/

typedef struct ethernet\_header ethernet\_header;

/\* IPv4 header \*/

typedef struct ip\_v4\_header ip\_v4\_header;

/\*IPv6 header\*/

typedef struct ip\_v6\_header ip\_v6\_header;

/\*arp header\*/

typedef struct arp\_header arp\_header;

/\*TCP header\*/

typedef struct tcp\_header tcp\_header;

/\* UDP header\*/

typedef struct udp\_header udp\_header;

/\*ICMP header\*/

typedef struct icmp\_header icmp\_header;

/\* prototype of the packet handler \*/

void packet\_handler(u\_char \*param, const struct pcap\_pkthdr \*header, const u\_char \*pkt\_data);

/\*analysis the ethernet packet\*/

void ethernet\_package\_handler(u\_char \*param, const struct pcap\_pkthdr \*header, const u\_char \*pkt\_data);

/\*analysis the IPv4 packet\*/

void ip\_v4\_package\_handler(u\_char \*param, const struct pcap\_pkthdr \*header, const u\_char \*pkt\_data);

/\*analysis the IPv6 packet\*/

void ip\_v6\_package\_handler(u\_char \*param, const struct pcap\_pkthdr \*header, const u\_char \*pkt\_data);

/\*analysis the arp packet\*/

void arp\_package\_handler(u\_char\* param, const struct pcap\_pkthdr \*header, const u\_char \*pkt\_data);

/\*analysis the udp packet\*/

void udp\_package\_handler(u\_char\* param, const struct pcap\_pkthdr \*header, const u\_char \*pkt\_data);

/\*analysis the tcp packet\*/

void tcp\_package\_handler(u\_char\* param, const struct pcap\_pkthdr \*header, const u\_char \*pkt\_data);

/\*analysis the icmp packet\*/

void icmp\_package\_handler(u\_char\* param, const struct pcap\_pkthdr \*header, const u\_char \*pkt\_data);

/\*count the package with c++ std::map\*/

void add\_to\_map(std::map<std::string, int> &counter, ip\_v4\_address ip);

void add\_to\_map(std::map<std::string, int> &counter, ip\_v6\_address ip);

/\*print the map info\*/

void print\_map(std::map<std::string, int> counter);

#endif // !\_PAC\_ANA\_H

### 具体定义

#### Winpcap通用部分：

int main()

{

pcap\_if\_t \*alldevs;

pcap\_if\_t \*d;

int inum;

int i = 0;

int pktnum;

pcap\_t \*adhandle;

char errbuf[PCAP\_ERRBUF\_SIZE];

u\_int netmask = 0xffffff;;

struct bpf\_program fcode;

if (pcap\_findalldevs(&alldevs, errbuf) == -1)

{

fprintf(stderr, "Error in pcap\_findalldevs: %s\n", errbuf);

exit(1);

}

for (d = alldevs; d; d = d->next)

{

cout << ++i << "." << d->name;

if (d->description)

cout << d->description << endl;

else

cout << " (No description available)" << endl;

}

if (i == 0)

{

cout << "\nNo interfaces found! Make sure WinPcap is installed." << endl;

return -1;

}

cout << "Enter the interface number (1-" << i << "): ";

cin >> inum;

if (inum < 1 || inum > i)

{

cout << "\nInterface number out of range." << endl;

pcap\_freealldevs(alldevs);

return -1;

}

for (d = alldevs, i = 0; i < inum - 1; d = d->next, i++);

if ((adhandle = pcap\_open\_live(d->name, // name of the device

65536, // portion of the packet to capture.

// 65536 grants that the whole packet will be captured on all the MACs.

1, // promiscuous mode (nonzero means promiscuous)

1000, // read timeout

errbuf // error buffer

)) == NULL)

{

fprintf(stderr, "\nUnable to open the adapter. %s is not supported by WinPcap\n", d->name);

pcap\_freealldevs(alldevs);

return -1;

}

cout << "listening on " << d->description << "...." << endl;

pcap\_freealldevs(alldevs);

if (pcap\_compile(adhandle, &fcode, "ip or arp", 1, netmask) < 0)

{

fprintf(stderr, "\nUnable to compile the packet filter. Check the syntax.\n");

pcap\_close(adhandle);

return -1;

}

if (pcap\_setfilter(adhandle, &fcode) < 0)

{

fprintf(stderr, "\nError setting the filter.\n");

pcap\_close(adhandle);

return -1;

}

cout << "please input the num of packets you want to catch(0 for keeping catching): ";

cin >> pktnum;

cout << endl;

pcap\_loop(adhandle, pktnum, packet\_handler, NULL);

pcap\_close(adhandle);

getchar();

return 0;

}

#### 用结构体表示每个协议结构

struct ip\_v4\_address

{

u\_char byte1;

u\_char byte2;

u\_char byte3;

u\_char byte4;

};

struct ip\_v6\_address

{

u\_short part1;

u\_short part2;

u\_short part3;

u\_short part4;

u\_short part5;

u\_short part6;

u\_short part7;

u\_short part8;

};

struct mac\_address

{

u\_char byte1;

u\_char byte2;

u\_char byte3;

u\_char byte4;

u\_char byte5;

u\_char byte6;

};

struct ethernet\_header

{

mac\_address des\_mac\_addr;

mac\_address src\_mac\_addr;

u\_short type;

};

struct ip\_v4\_header

{

u\_char ver\_ihl; // Version (4 bits) + Internet header length (4 bits)

u\_char tos; // Type of service

u\_short tlen; // Total length

u\_short identification; // Identification

u\_short flags\_fo; // Flags (3 bits) + Fragment offset (13 bits)

u\_char ttl; // Time to live

u\_char proto; // Protocol

u\_short checksum; // Header checksum

ip\_v4\_address src\_ip\_addr; // Source address

ip\_v4\_address des\_ip\_addr; // Destination address

u\_int op\_pad; // Option + Padding

};

struct ip\_v6\_header

{

u\_int32\_t ver\_trafficclass\_flowlabel;

u\_short payload\_len;

u\_char next\_head;

u\_char ttl;

ip\_v6\_address src\_ip\_addr;

ip\_v6\_address dst\_ip\_addr;

};

struct arp\_header

{

u\_short hardware\_type;

u\_short protocol\_type;

u\_char hardware\_length;

u\_char protocol\_length;

u\_short operation\_code;

mac\_address source\_mac\_addr;

ip\_v4\_address source\_ip\_addr;

mac\_address des\_mac\_addr;

ip\_v4\_address des\_ip\_addr;

};

struct tcp\_header

{

u\_short sport;

u\_short dport;

u\_int sequence;

u\_int acknowledgement;

u\_char offset;

u\_char flags;

u\_short windows;

u\_short checksum;

u\_short urgent\_pointer;

};

struct udp\_header

{

u\_short sport; // Source port

u\_short dport; // Destination port

u\_short len; // Datagram length

u\_short checksum; // Checksum

};

struct icmp\_header

{

u\_char type;

u\_char code;

u\_short checksum;

u\_short id;

u\_short sequence;

};

#### 解析协议的实现：

/\* Callback function invoked by libpcap for every incoming packet \*/

void packet\_handler(u\_char \*param, const struct pcap\_pkthdr \*header, const u\_char \*pkt\_data)

{

struct tm \*ltime;

char timestr[16];

time\_t local\_tv\_sec;

/\* convert the timestamp to readable format \*/

local\_tv\_sec = header->ts.tv\_sec;

ltime = localtime(&local\_tv\_sec);

strftime(timestr, sizeof timestr, "%H:%M:%S", ltime);

cout << B\_DIVISION << "time:" << timestr << ","

<< header->ts.tv\_usec << " len:" << header->len << B\_DIVISION<<endl;

ethernet\_package\_handler(param, header, pkt\_data);

}

void ethernet\_package\_handler(u\_char \*param, const struct pcap\_pkthdr \*header, const u\_char \*pkt\_data)

{

ethernet\_header\* eh = (ethernet\_header\*)pkt\_data;

cout << DIVISION << "以太网协议分析结构" << DIVISION << endl;

u\_short type = ntohs(eh->type);

cout << "类型：0x" << hex << type;

cout << setbase(10);

switch (type)

{

case 0x0800:

cout << " (IPv4)" << endl;

break;

case 0x86DD:

cout << "(IPv6)" << endl;

break;

case 0x0806:

cout << " (ARP)" << endl;

break;

case 0x0835:

cout << " (RARP)" << endl;

default:

break;

}

cout << "目的地址：" << int(eh->des\_mac\_addr.byte1) << ":"

<< int(eh->des\_mac\_addr.byte2) << ":"

<< int(eh->des\_mac\_addr.byte3) << ":"

<< int(eh->des\_mac\_addr.byte4) << ":"

<< int(eh->des\_mac\_addr.byte5) << ":"

<< int(eh->des\_mac\_addr.byte6) << endl;

cout << "源地址：" << int(eh->src\_mac\_addr.byte1) << ":"

<< int(eh->src\_mac\_addr.byte2) << ":"

<< int(eh->src\_mac\_addr.byte3) << ":"

<< int(eh->src\_mac\_addr.byte4) << ":"

<< int(eh->src\_mac\_addr.byte5) << ":"

<< int(eh->src\_mac\_addr.byte6) << endl;

switch (type)

{

case 0x0800:

ip\_v4\_package\_handler(param, header, pkt\_data);

break;

case 0x0806:

arp\_package\_handler(param, header, pkt\_data);

break;

case 0x86DD:

ip\_v6\_package\_handler(param, header, pkt\_data);

break;

default:

break;

}

cout << endl << endl;

}

void arp\_package\_handler(u\_char \*param, const struct pcap\_pkthdr \*header, const u\_char \*pkt\_data)

{

arp\_header\* ah;

ah = (arp\_header\*)(pkt\_data + 14);

cout << DIVISION << "ARP协议分析结构" << DIVISION << endl;

u\_short operation\_code = ntohs(ah->operation\_code);

cout << "硬件类型：" << ntohs(ah->hardware\_type) << endl;

cout << "协议类型：0x" << hex << ntohs(ah->protocol\_type) << endl;

cout << setbase(10);

cout << "硬件地址长度：" << int(ah->hardware\_length) << endl;

cout << "协议地址长度：" << int(ah->protocol\_length) << endl;

switch (operation\_code)

{

case 1:

cout << "ARP请求协议" << endl;

break;

case 2:

cout << "ARP应答协议" << endl;

break;

case 3:

cout << "ARP请求协议" << endl;

break;

case 4:

cout << "RARP应答协议" << endl;

break;

default:

break;

}

cout << "源IP地址："

<< int(ah->source\_ip\_addr.byte1) << "."

<< int(ah->source\_ip\_addr.byte2) << "."

<< int(ah->source\_ip\_addr.byte3) << "."

<< int(ah->source\_ip\_addr.byte4) << endl;

cout << "目的IP地址："

<< int(ah->des\_ip\_addr.byte1) << "."

<< int(ah->des\_ip\_addr.byte2) << "."

<< int(ah->des\_ip\_addr.byte3) << "."

<< int(ah->des\_ip\_addr.byte4) << endl;

add\_to\_map(counter, ah->source\_ip\_addr);

print\_map(counter);

}

void ip\_v4\_package\_handler(u\_char \*param, const struct pcap\_pkthdr \*header, const u\_char \*pkt\_data)

{

ip\_v4\_header \*ih;

ih = (ip\_v4\_header \*)(pkt\_data + 14); //14 measn the length of ethernet header

cout << DIVISION << "IPv4协议分析结构" << DIVISION << endl;

cout << "版本号：" << ((ih->ver\_ihl & 0xf0) >> 4) << endl;

cout << "首部长度：" << (ih->ver\_ihl & 0xf) << "("

<< ((ih->ver\_ihl & 0xf)<<2) << "B)" << endl;

cout << "区别服务：" << int(ih->tos) << endl;

cout << "总长度：" << ntohs(ih->tlen) << endl;

cout << "标识：" << ntohs(ih->identification) << endl;

cout << "标志：" << ((ih->flags\_fo & 0xE000) >> 12) << endl;

cout << "片偏移：" << (ih->flags\_fo & 0x1FFF) << "("

<< ((ih->flags\_fo & 0x1FFF) << 3) << "B)" <<endl;

cout << "生命周期：" << int(ih->ttl) << endl;

cout << "协议：";

switch (ih->proto)

{

case 6:

cout << "TCP" << endl;

break;

case 17:

cout << "UDP" << endl;

break;

case 1:

cout << "ICMP" << endl;

break;

default:

cout << endl;

break;

}

cout << "校验和：" << ntohs(ih->checksum) << endl;

cout << "源IP地址："

<< int(ih->src\_ip\_addr.byte1) << "."

<< int(ih->src\_ip\_addr.byte2) << "."

<< int(ih->src\_ip\_addr.byte3) << "."

<< int(ih->src\_ip\_addr.byte4) << endl;

cout << "目的IP地址："

<< int(ih->des\_ip\_addr.byte1) << "."

<< int(ih->des\_ip\_addr.byte2) << "."

<< int(ih->des\_ip\_addr.byte3) << "."

<< int(ih->des\_ip\_addr.byte4) << endl;

switch (ih->proto)

{

case 6:

tcp\_package\_handler(param, header, pkt\_data);

break;

case 17:

udp\_package\_handler(param, header, pkt\_data);

break;

case 1:

icmp\_package\_handler(param, header, pkt\_data);

break;

default:

break;

}

add\_to\_map(counter, ih->src\_ip\_addr);

print\_map(counter);

}

void ip\_v6\_package\_handler(u\_char \*param, const struct pcap\_pkthdr \*header, const u\_char \*pkt\_data)

{

ip\_v6\_header \*ih;

ih = (ip\_v6\_header \*)(pkt\_data + 14); //14 measn the length of ethernet header

int version = (ih->ver\_trafficclass\_flowlabel & 0xf0000000) >> 28;

int traffic\_class = ntohs((ih->ver\_trafficclass\_flowlabel & 0x0ff00000) >> 20);

int flow\_label = ih->ver\_trafficclass\_flowlabel & 0x000fffff;

cout << "版本号：" << version << endl;

cout << "通信量类：" << traffic\_class << endl;

cout << "流标号：" << flow\_label << endl;

cout << "有效载荷：" << ntohs(ih->payload\_len) << endl;

cout << "下一个首部：" << int(ih->next\_head) << endl;

cout << "跳数限制：" << int(ih->ttl) << endl;

cout << "源IP地址："

<< int(ih->src\_ip\_addr.part1) << ":"

<< int(ih->src\_ip\_addr.part2) << ":"

<< int(ih->src\_ip\_addr.part3) << ":"

<< int(ih->src\_ip\_addr.part4) << ":"

<< int(ih->src\_ip\_addr.part5) << ":"

<< int(ih->src\_ip\_addr.part6) << ":"

<< int(ih->src\_ip\_addr.part7) << ":"

<< int(ih->src\_ip\_addr.part8) << endl;

cout << "目的IP地址："

<< int(ih->dst\_ip\_addr.part1) << ":"

<< int(ih->dst\_ip\_addr.part2) << ":"

<< int(ih->dst\_ip\_addr.part3) << ":"

<< int(ih->dst\_ip\_addr.part4) << ":"

<< int(ih->dst\_ip\_addr.part5) << ":"

<< int(ih->dst\_ip\_addr.part6) << ":"

<< int(ih->dst\_ip\_addr.part7) << ":"

<< int(ih->dst\_ip\_addr.part8) << endl;

switch (ih->next\_head)

{

case 6:

tcp\_package\_handler(param, header, pkt\_data);

break;

case 17:

udp\_package\_handler(param, header, pkt\_data);

break;

case 58:

icmp\_package\_handler(param, header, pkt\_data);

break;

default:

break;

}

add\_to\_map(counter, ih->src\_ip\_addr);

print\_map(counter);

}

void udp\_package\_handler(u\_char\* param, const struct pcap\_pkthdr \*header, const u\_char \*pkt\_data)

{

udp\_header \*uh;

uh = (udp\_header \*)(pkt\_data + 20 + 14);

cout << DIVISION << "UDP协议分析结构" << DIVISION << endl;

cout << "源端口：" << ntohs(uh->sport) << endl;

cout << "目的端口：" << ntohs(uh->dport) << endl;

cout << "长度：" << ntohs(uh->len) << endl;

cout << "检验和：" << ntohs(uh->checksum) << endl;

}

void tcp\_package\_handler(u\_char\* param, const struct pcap\_pkthdr \*header, const u\_char \*pkt\_data)

{

tcp\_header\* th;

th = (tcp\_header\*)(pkt\_data + 14 + 20);

cout << DIVISION << "TCP协议分析结构" << DIVISION << endl;

cout << "源端口：" << ntohs(th->sport) << endl;

cout << "目的端口：" << ntohs(th->dport) << endl;

cout << "序号：" << ntohl(th->sequence) << endl;

cout << "确认号：" << ntohl(th->acknowledgement) << endl;

cout << "数据偏移：" << ((th->offset & 0xf0) >> 4) << "("

<< ((th->offset & 0xf0) >> 2) << "B)"<< endl;

cout << "标志：" ;

if (th->flags & 0x01)

{

cout << "FIN ";

}

if (th->flags & 0x02)

{

cout << "SYN ";

}

if (th->flags & 0x04)

{

cout << "RST ";

}

if (th->flags & 0x08)

{

cout << "PSH ";

}

if (th->flags & 0x10)

{

cout << "ACK ";

}

if (th->flags & 0x20)

{

cout << "URG ";

}

cout << endl;

cout << "窗口：" << ntohs(th->windows) << endl;

cout << "检验和：" << ntohs(th->checksum) << endl;

cout << "紧急指针：" << ntohs(th->urgent\_pointer) << endl;

}

void icmp\_package\_handler(u\_char\* param, const struct pcap\_pkthdr \*header, const u\_char \*pkt\_data)

{

icmp\_header\* ih;

ih = (icmp\_header\*)(pkt\_data + 14 + 20);

cout << DIVISION << "ICMP协议分析结构" << DIVISION << endl;

cout << "ICMP类型：" << ih->type;

switch (ih->type)

{

case 8:

cout << "ICMP回显请求协议" << endl;

break;

case 0:

cout << "ICMP回显应答协议" << endl;

break;

default:

break;

}

cout << "ICMP代码：" << ih->code << endl;

cout << "标识符：" << ih->id << endl;

cout << "序列码：" << ih->sequence << endl;

cout << "ICMP校验和：" << ntohs(ih->checksum) << endl;

}

void add\_to\_map(map<string, int> &counter, ip\_v4\_address ip)

{

string ip\_string;

int amount = 0;

map<string,int>::iterator iter;

ip\_string = to\_string(ip.byte1) + "."

+ to\_string(ip.byte2) + "."

+ to\_string(ip.byte3) + "."

+ to\_string(ip.byte4);

iter = counter.find(ip\_string);

if (iter != counter.end())

{

amount = iter->second;

}

counter.insert\_or\_assign(ip\_string, ++amount);

}

void add\_to\_map(map<string, int> &counter, ip\_v6\_address ip)

{

string ip\_string;

int amount = 0;

map<string, int>::iterator iter;

ip\_string = to\_string(ip.part1) + ":"

+ to\_string(ip.part2) + ":"

+ to\_string(ip.part3) + ":"

+ to\_string(ip.part4) + ":"

+ to\_string(ip.part5) + ":"

+ to\_string(ip.part6) + ":"

+ to\_string(ip.part7) + ":"

+ to\_string(ip.part8);

iter = counter.find(ip\_string);

if (iter != counter.end())

{

amount = iter->second;

}

counter.insert\_or\_assign(ip\_string, ++amount);

}

void print\_map(map<string, int> counter)

{

map<string, int>::iterator iter;

cout << DIVISION << "流量统计" << DIVISION << endl;

cout << "IP" << setfill(' ')<<setw(45) << "流量" << endl;

for (iter = counter.begin(); iter != counter.end(); iter++)

{

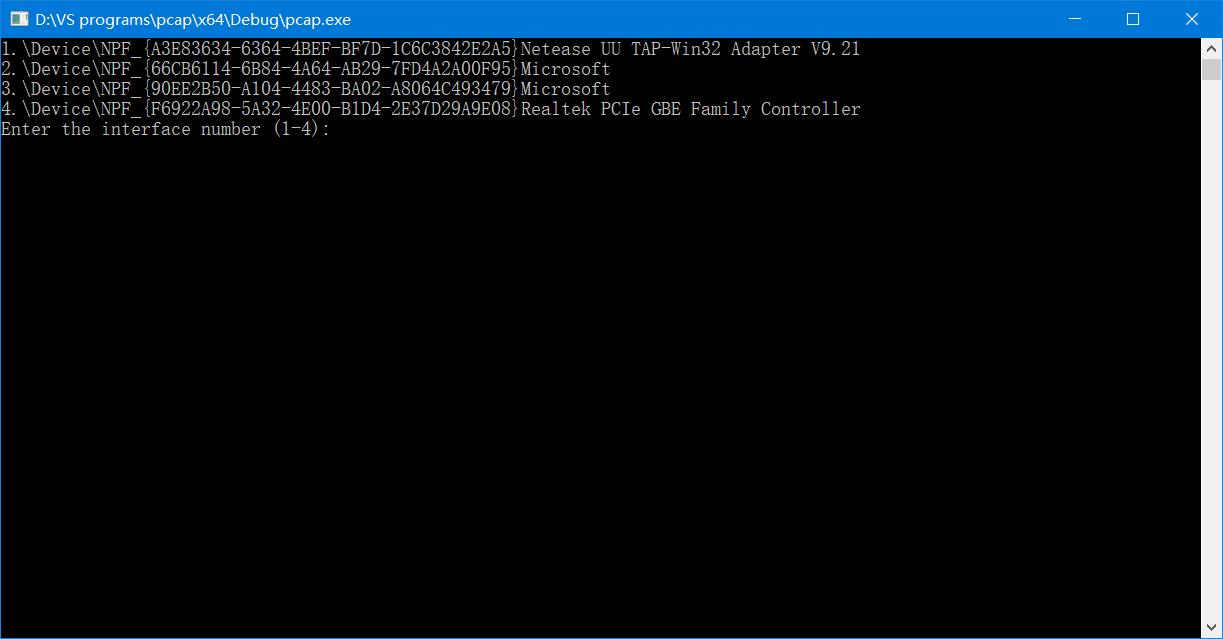
cout << iter->first << setfill('.') << setw(45-iter->first.length()) << iter->second<<endl;

}

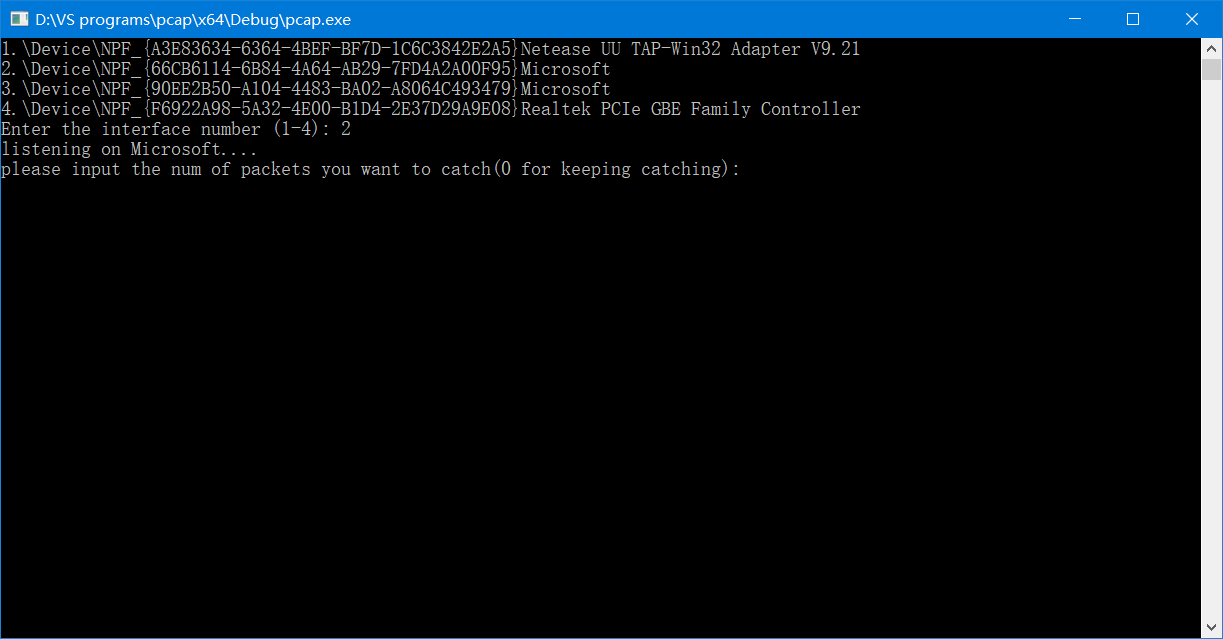
}

### 实验结果

* + - 1. 选择设备



* + - 1. 输入要捕获的数据报数目，0表示一直捕获



* + - 1. ARP协议解析即流量统计如图3.11所示

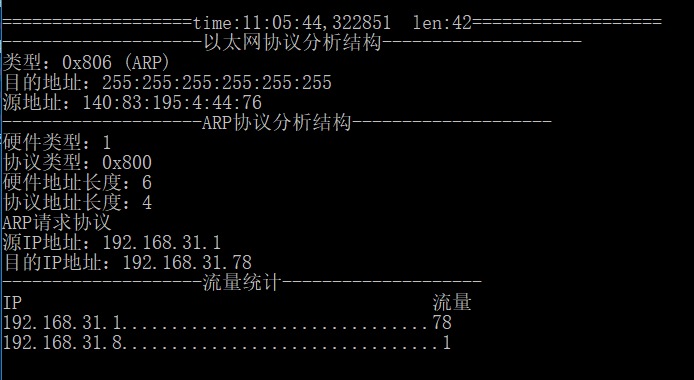


图3.11

* + - 1. IP协议解析即流量统计如图3.12所示

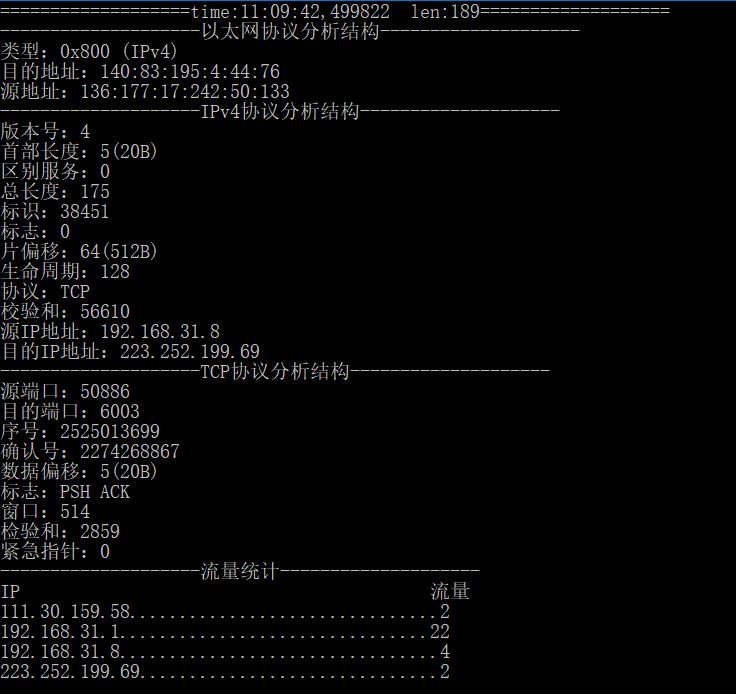


图3.12

* + - 1. UDP协议解析即流量统计如果3.13所示

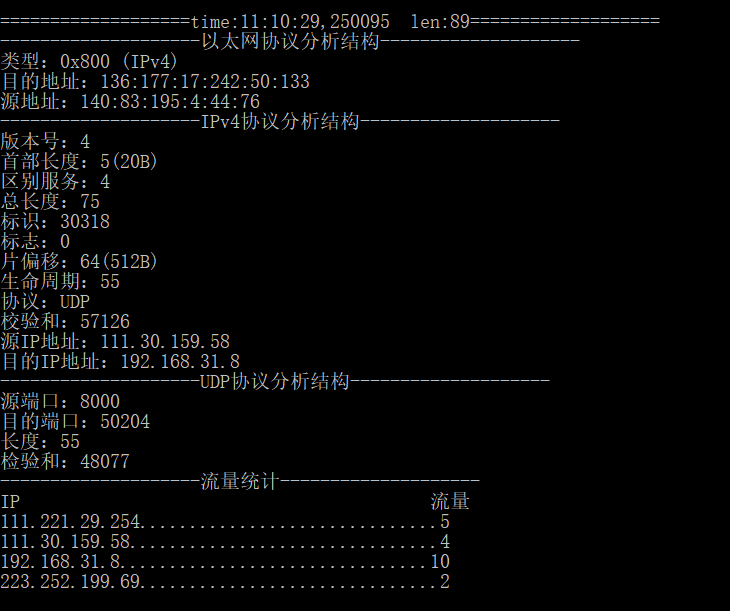


图3.13

* + - 1. ICMP协议即流量统计如图3.14所示

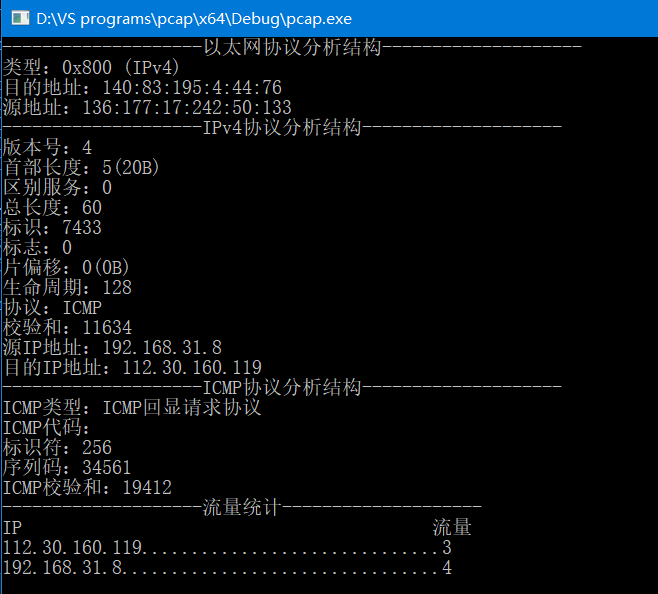
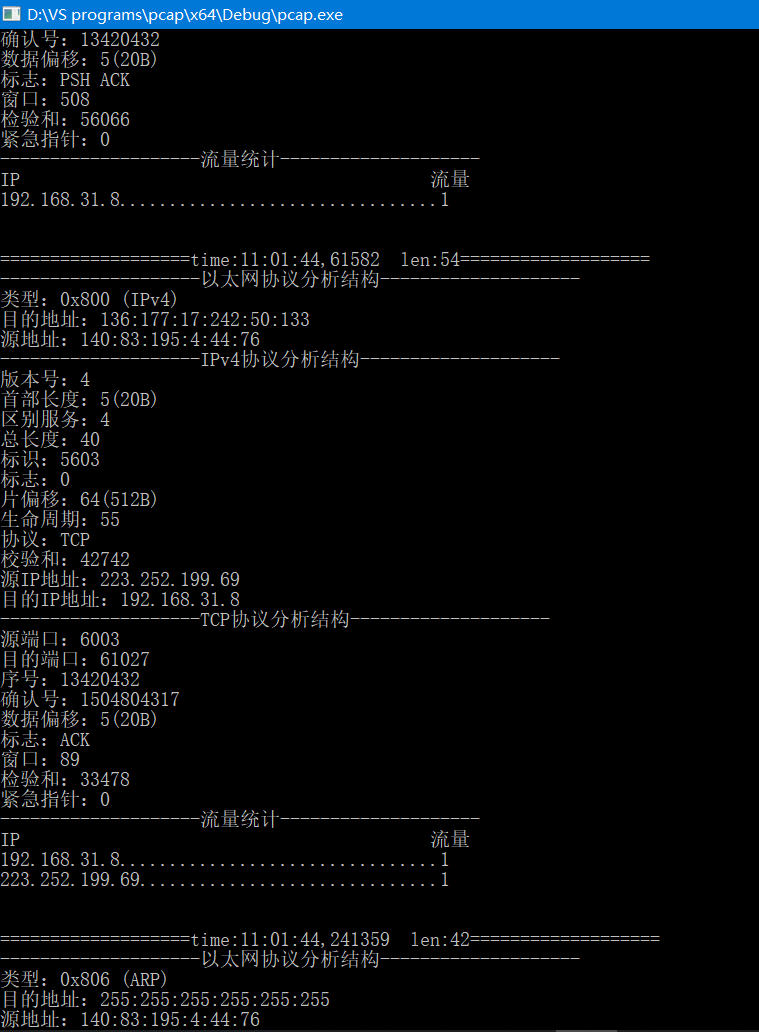
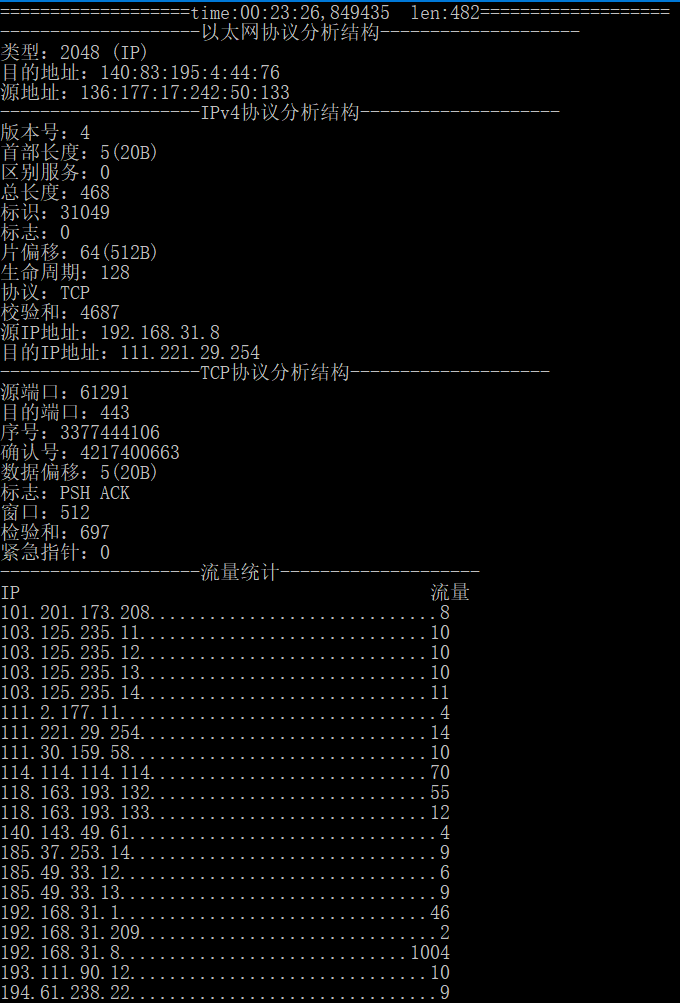


图3.14

* + - 1. 最终结果如果3.15所示





## 总结

计算机网络是一门非常基本但又非常重要的一门课程，掌握不同计算机的不同进程是如何进行信息交互的，数据是如何在信道中传输，以及TCP/IP中各种协议规则，这有利于我们在需要网络通信的软件编程中更加游刃有余，而本次实验的抓包加深了我对协议结构的理解，同时掌握了基本的抓包、过滤、解析的方法，有利于我们发现程序中隐蔽的错误，提高网络通信的安全性。