# checksum算法与实现

实验名称：checksum算法与实现

学时安排：2课时 指导老师：李赞

实验类别：验证型、设计型 实验要求：1人1组

学号： 姓名： 班级：

## 一、实验目的和任务

1. 本实验要求掌握checksum算法原理与代码实现。

## 二、实验设备介绍

B701实验室基本配置环境如下：

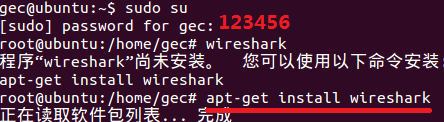
1.软件环境： win10操作系统，VMware workstation，ubuntu18。

## 三、实验内容和步骤

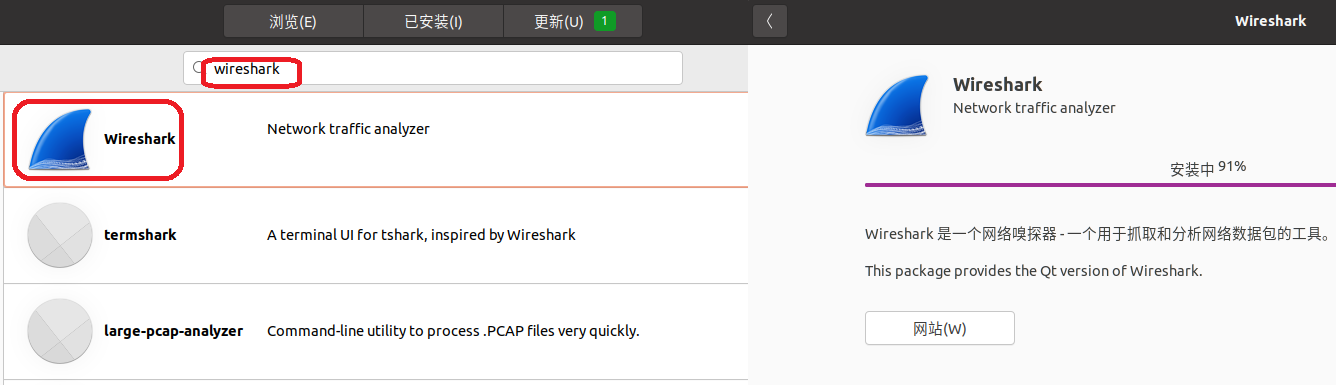
校验字段是通过算法对特定的字节序列进行运算生成的数值，该数值随字节序列一同进行网络传输，在接收端对字节序列重新执行校验算法，可校验数据传输过程中字节序列是否发生变化。本实验首先通过抓包软件查看实际网络包中的checksum字段的值，然后提供了checksum算法的实现。

### 3.1 WireShark捕获数据包分析checksum字段

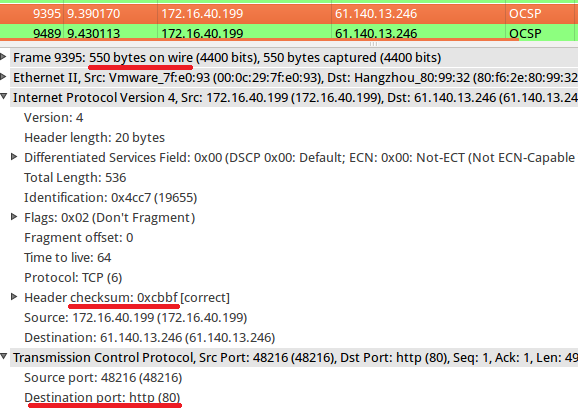
wireshark是linux平台一个非常方便的抓取网络数据包的软件，利用它可以让我们分析已有的网络数据包的格式和构成。如果Ubuntu没有安装wireshark软件，可选择命令在线安装wireshark。配置ubuntu的外网IP:172.16.56.XX，设置好网卡桥接，通过联网方式安装wireshark软件。大多网包都具有checksum字段。



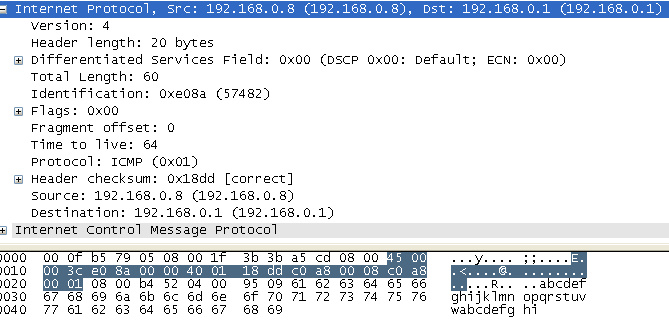
也可以在软件中心查找wireshark，进行在线安装。



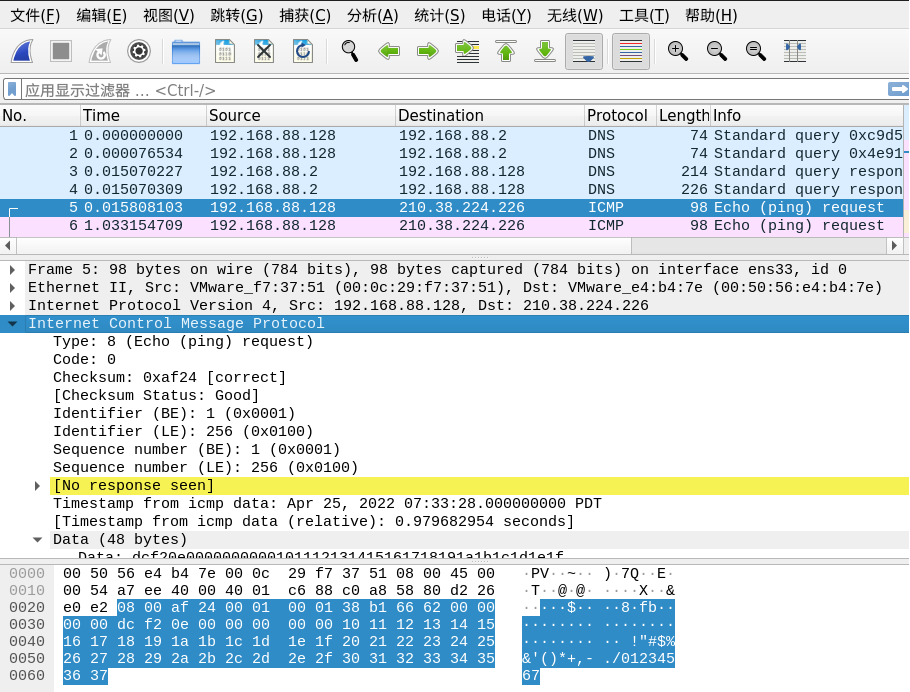
下面是在ubuntu中抓取的http数据包的截图。



下面是捕获的ICMP网络包的截图。

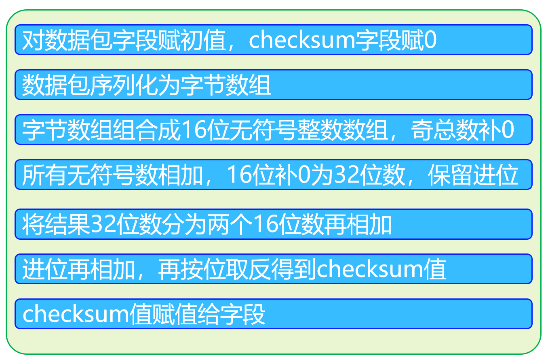
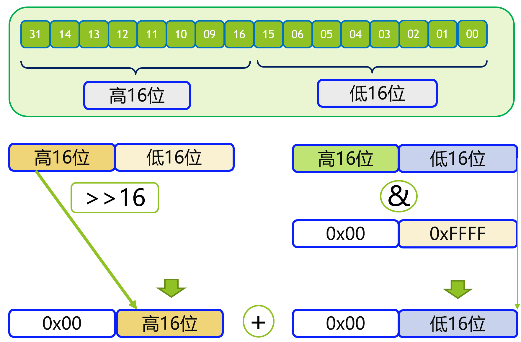


下图是来自ubuntu20抓取的ICMP数据包，



### 3.2 CheckSum算法的实现

校验和Checksum算法的运算是以2字节为一组(16位)进行运算，运算方法是移位和与操作，其运算效率很高，且硬件实现方便，并且该函数最大的特点是生成与检验是相同的。其运算过程如下图所示。

checksum函数的实现如下，通过抓取的包获得原始数据，运行函数产生校验值，其中校验数据即来自抓包。

#include "stdio.h"

unsigned short csum(unsigned short \*addr, int count);

unsigned char data[] =

{

0x45,0x00,0x00,0x54,

0x00,0x00,0x40,0x00,

0x40,0x01,0x00,0x00,//0x18,0xdd

0xc0,0xa8,0x03,0x1a,

0x0e,0xd7,0xb1,0x27

};//这是一个IP头部分

int main(void)

{

printf("the check sum is 0x%x\n",csum((unsigned short \*)data,sizeof(data)));

return 0;

}

unsigned short csum(unsigned short \*addr, int count)

{

/\* Compute Internet Checksum for "count" bytes

\* beginning at location "addr".

\*/

long sum = 0;

unsigned short tmp = 0;

while( count > 1 )

{

/\* This is the inner loop \*/

sum += (long)(\* ((unsigned short \*)addr++));

count -= 2;

}

/\* Add left-over byte, if any \*/

if( count > 0 )

{

sum += \* (unsigned char \*) addr;

}

/\* Fold 32-bit sum to 16 bits \*/

while (sum>>16)

{

sum = (sum & 0xffff) + (sum >> 16);

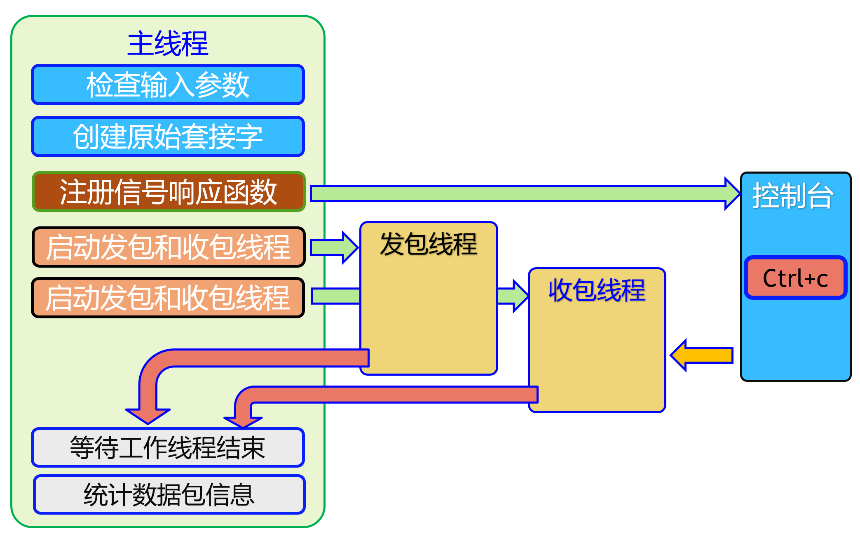
}

return ~sum;

}

### 3.3 原始套接字构建ICMP网络包实现ping功能

下图说明了实现ping命令的程序结构。



下面的表列出了ping函数使用的函数说明。

|  |  |
| --- | --- |
| 函数名 |  |
| icmp\_cksum | 校验和计算函数。 |
| icmp\_pack | 设置ICMP报头 |
| icmp\_unpack | 解析接收的ICMP包信息。 |
| icmp\_tvsub | 计算每个包发送和接收的时间差。 |
| icmp\_send | 发送ICMP请求包线程。 |
| icmp\_recv | 接收ICMP回复包线程。 |
| icmp\_statistics | 对ICMP包统计。 |
| icmp\_sigint | 响应键盘中断信号。 |

/\*ping.c\*/

#include <sys/socket.h>

#include <netinet/in.h>

#include <netinet/ip.h>

#include <netinet/ip\_icmp.h>

#include <unistd.h>

#include <signal.h>

#include <arpa/inet.h>

#include <errno.h>

#include <sys/time.h>

#include <stdio.h>

#include <string.h> /\*bzero\*/

#include <netdb.h>

#include <pthread.h>

/\*保存已经发送包的状态值\*/

typedef struct pingm\_pakcet{

struct timeval tv\_begin; /\*发送的时间\*/

struct timeval tv\_end; /\*接收到的时间\*/

short seq; /\*序列号\*/

int flag; /\*1，表示已经发送但没有接收到回应包0，表示接收到回应包\*/

}pingm\_pakcet;

static pingm\_pakcet pingpacket[128];

static pingm\_pakcet \*icmp\_findpacket(int seq);

static unsigned short icmp\_cksum(unsigned char \*data, int len);

static struct timeval icmp\_tvsub(struct timeval end,struct timeval begin);

static void icmp\_statistics(void);

static void icmp\_pack(struct icmp \*icmph, int seq, struct timeval \*tv, int length );

static int icmp\_unpack(char \*buf,int len);

static void \*icmp\_recv(void \*argv);

static void \*icmp\_send(void \*argv);

static void icmp\_sigint(int signo);

static void icmp\_usage();

#define K 1024

#define BUFFERSIZE 72 /\*发送缓冲区大小\*/

static char send\_buff[BUFFERSIZE];

static char recv\_buff[2\*K]; /\*为防止接收溢出，接收缓冲区设置大一些\*/

static struct sockaddr\_in dest; /\*目的地址\*/

static int rawsock = 0; /\*发送和接收线程需要的socket描述符\*/

static pid\_t pid=0; /\*进程PID\*/

static int alive = 0; /\*是否接收到退出信号\*/

static short packet\_send = 0; /\*已经发送的数据包有多少\*/

static short packet\_recv = 0; /\*已经接收的数据包有多少\*/

static char dest\_str[80]; /\*目的主机字符串\*/

static struct timeval tv\_begin, tv\_end,tv\_interval;

/\*本程序开始发送、结束和时间间隔\*/

static void icmp\_usage()

{

/\*ping加IP地址或者域名\*/

printf("ping aaa.bbb.ccc.ddd\n");

}

/\*主程序\*/

int main(int argc, char \*argv[])

{

struct hostent \* host = NULL;

struct protoent \*protocol = NULL;

char protoname[]= "icmp";

unsigned long inaddr = 1;

int size = 128\*K;

/\*参数是否数量正确\*/

if(argc < 2)

{

icmp\_usage();

return -1;

}

/\*获取协议类型ICMP\*/

protocol = getprotobyname(protoname);

if (protocol == NULL)

{

perror("getprotobyname()");

return -1;

}

/\*复制目的地址字符串\*/

memcpy(dest\_str, argv[1], strlen(argv[1])+1);

memset(pingpacket, 0, sizeof(pingm\_pakcet) \* 128);

/\*socket初始化\*/

rawsock = socket(AF\_INET, SOCK\_RAW, protocol->p\_proto);

if(rawsock < 0)

{

perror("socket");

return -1;

}

/\*为了与其他进程的ping程序区别，加入pid\*/

pid = getuid();

/\*增大接收缓冲区，防止接收的包被覆盖\*/

setsockopt(rawsock, SOL\_SOCKET, SO\_RCVBUF, &size, sizeof(size));

bzero(&dest, sizeof(dest));

/\*获取目的地址的IP地址\*/

dest.sin\_family = AF\_INET;

/\*输入的目的地址为字符串IP地址\*/

inaddr = inet\_addr(argv[1]);

if(inaddr == INADDR\_NONE)

{

/\*输入的是DNS地址\*/

host = gethostbyname(argv[1]);

if(host == NULL)

{

perror("gethostbyname");

return -1;

}

/\*将地址复制到dest中\*/

memcpy((char \*)&dest.sin\_addr, host->h\_addr, host->h\_length);

}

else /\*为IP地址字符串\*/

{

memcpy((char\*)&dest.sin\_addr, &inaddr, sizeof(inaddr));

}

/\*打印提示\*/

inaddr = dest.sin\_addr.s\_addr;

printf("PING %s (%ld.%ld.%ld.%ld) 56(84) bytes of data.\n",

dest\_str,

(inaddr&0x000000FF)>>0,

(inaddr&0x0000FF00)>>8,

(inaddr&0x00FF0000)>>16,

(inaddr&0xFF000000)>>24);

/\*截取信号SIGINT，将icmp\_sigint挂接上\*/

signal(SIGINT, icmp\_sigint);

alive = 1; /\*初始化为可运行\*/

pthread\_t send\_id, recv\_id; /\*建立两个线程，用于发送和接收\*/

int err = 0;

err = pthread\_create(&send\_id, NULL, icmp\_send, NULL); /\*发送\*/

if(err < 0)

{

return -1;

}

err = pthread\_create(&recv\_id, NULL, icmp\_recv, NULL); /\*接收\*/

if(err < 0)

{

return -1;

}

/\*等待线程结束\*/

pthread\_join(send\_id, NULL);

pthread\_join(recv\_id, NULL);

/\*清理并打印统计结果\*/

close(rawsock);

icmp\_statistics();

return 0;

}

/\*CRC16校验和计算icmp\_cksum

参数：

data:数据

len:数据长度

返回值：

计算结果，short类型

\*/

static unsigned short icmp\_cksum(unsigned char \*data, int len)

{

int sum=0; /\*计算结果\*/

int odd = len & 0x01; /\*是否为奇数\*/

/\*将数据按照2字节为单位累加起来\*/

while( len & 0xfffe) {

sum += \*(unsigned short\*)data;

data += 2;

len -=2;

}

/\*判断是否为奇数个数据，若ICMP报头为奇数个字节，会剩下最后一字节\*/

if( odd) {

unsigned short tmp = ((\*data)<<8)&0xff00;

sum += tmp;

}

sum = (sum >>16) + (sum & 0xffff); /\*高低位相加\*/

sum += (sum >>16) ; /\*将溢出位加入\*/

return ~sum; /\*返回取反值\*/

}

/\*设置ICMP报头\*/

static void icmp\_pack(struct icmp \*icmph, int seq, struct timeval \*tv, int length )

{

unsigned char i = 0;

/\*设置报头\*/

icmph->icmp\_type = ICMP\_ECHO; /\*ICMP回显请求\*/

icmph->icmp\_code = 0; /\*code值为0\*/

icmph->icmp\_cksum = 0; /\*先将cksum值填写0，便于之后的cksum计算\*/

icmph->icmp\_seq = seq; /\*本报的序列号\*/

icmph->icmp\_id = pid &0xffff; /\*填写PID\*/

for(i = 0; i< length; i++)

icmph->icmp\_data[i] = i;

/\*计算校验和\*/

icmph->icmp\_cksum = icmp\_cksum((unsigned char\*)icmph, length);

}

/\*解压接收到的包，并打印信息\*/

static int icmp\_unpack(char \*buf,int len)

{

int iphdrlen;

struct ip \*ip = NULL;

struct icmp \*icmp = NULL;

int rtt;

ip=(struct ip \*)buf; /\*IP头部\*/

iphdrlen=ip->ip\_hl\*4; /\*IP头部长度\*/

icmp=(struct icmp \*)(buf+iphdrlen); /\*ICMP段的地址\*/

len-=iphdrlen;

/\*判断长度是否为ICMP包\*/

if( len<8)

{

printf("ICMP packets\'s length is less than 8\n");

return -1;

}

/\*ICMP类型为ICMP\_ECHOREPLY并且为本进程的PID\*/

if( (icmp->icmp\_type==ICMP\_ECHOREPLY) && (icmp->icmp\_id== pid) )

{

struct timeval tv\_internel,tv\_recv,tv\_send;

/\*在发送表格中查找已经发送的包，按照seq\*/

pingm\_pakcet\* packet = icmp\_findpacket(icmp->icmp\_seq);

if(packet == NULL)

return -1;

packet->flag = 0; /\*取消标志\*/

tv\_send = packet->tv\_begin; /\*获取本包的发送时间\*/

gettimeofday(&tv\_recv, NULL); /\*读取此时间，计算时间差\*/

tv\_internel = icmp\_tvsub(tv\_recv,tv\_send);

rtt = tv\_internel.tv\_sec\*1000+tv\_internel.tv\_usec/1000;

/\*打印结果，包含

\* ICMP段长度

\* 源IP地址

\* 包的序列号

\* TTL

\* 时间差

\*/

printf("%d byte from %s: icmp\_seq=%u ttl=%d rtt=%d ms\n",

len,

inet\_ntoa(ip->ip\_src),

icmp->icmp\_seq,

ip->ip\_ttl,

rtt);

packet\_recv ++; /\*接收包数量加1\*/

}

else

{

return -1;

}

return 0;

}

/\*计算时间差time\_tvsub

参数：

end，接收到的时间

begin，开始发送的时间

返回值：

使用的时间

\*/

static struct timeval icmp\_tvsub(struct timeval end,struct timeval begin)

{

struct timeval tv;

/\*计算差值\*/

tv.tv\_sec = end.tv\_sec - begin.tv\_sec;

tv.tv\_usec = end.tv\_usec - begin.tv\_usec;

/\*如果接收时间的usec值小于发送时的usec值，从usec域借位\*/

if(tv.tv\_usec < 0)

{

tv.tv\_sec --;

tv.tv\_usec += 1000000;

}

return tv;

}

/\*发送ICMP回显请求包\*/

static void\* icmp\_send(void \*argv)

{

/\*保存程序开始发送数据的时间\*/

gettimeofday(&tv\_begin, NULL);

while(alive)

{

int size = 0;

struct timeval tv;

gettimeofday(&tv, NULL); /\*当前包的发送时间\*/

/\*在发送包状态数组中找一个空闲位置\*/

pingm\_pakcet \*packet = icmp\_findpacket(-1);

if(packet)

{

packet->seq = packet\_send; /\*设置seq\*/

packet->flag = 1; /\*已经使用\*/

gettimeofday( &packet->tv\_begin, NULL); /\*发送时间\*/

}

icmp\_pack((struct icmp \*)send\_buff, packet\_send, &tv, 64 );

/\*打包数据\*/

size = sendto (rawsock, send\_buff, 64, 0, /\*发送给目的地址\*/

(struct sockaddr \*)&dest, sizeof(dest) );

if(size <0)

{

perror("sendto error");

continue;

}

packet\_send++; /\*计数增加\*/

/\*每隔1s，发送一个ICMP回显请求包\*/

sleep(1);

}

}

/\*接收ping目的主机的回复\*/

static void \*icmp\_recv(void \*argv)

{

/\*轮询等待时间\*/

struct timeval tv;

tv.tv\_usec = 200;

tv.tv\_sec = 0;

fd\_set readfd;

/\*当没有信号发出一直接收数据\*/

while(alive)

{

int ret = 0;

FD\_ZERO(&readfd);

FD\_SET(rawsock, &readfd);

ret = select(rawsock+1,&readfd, NULL, NULL, &tv);

switch(ret)

{

case -1:

/\*错误发生\*/

break;

case 0:

/\*超时\*/

break;

default:

{

/\*接收数据\*/

int size = recv(rawsock, recv\_buff,sizeof(recv\_buff),

0);

if(errno == EINTR)

{

perror("recvfrom error");

continue;

}

/\*解包，并设置相关变量\*/

ret = icmp\_unpack(recv\_buff, size);

if(ret == -1)

{

continue;

}

}

break;

}

}

}

/\*查找一个合适的包位置

\*当seq为-1时，表示查找空包

\*其他值表示查找seq对应的包\*/

static pingm\_pakcet \*icmp\_findpacket(int seq)

{

int i=0;

pingm\_pakcet \*found = NULL;

/\*查找包的位置\*/

if(seq == -1) /\*查找空包的位置\*/

{

for(i = 0;i<128;i++)

{

if(pingpacket[i].flag == 0)

{

found = &pingpacket[i];

break;

}

}

}

else if(seq >= 0) /\*查找对应seq的包\*/

{

for(i = 0;i<128;i++)

{

if(pingpacket[i].seq == seq)

{

found = &pingpacket[i];

break;

}

}

}

return found;

}

/\*打印全部ICMP发送接收统计结果\*/

static void icmp\_statistics(void)

{

long time = (tv\_interval.tv\_sec \* 1000 )+ (tv\_interval.tv\_usec/1000);

printf("--- %s ping statistics ---\n",dest\_str); /\*目的IP地址\*/

printf("%d packets transmitted, %d received, %d%% packet loss, time %ldms\n",

packet\_send, /\*发送\*/

packet\_recv, /\*接收\*/

(packet\_send-packet\_recv)\*100/packet\_send, /\*丢失百分比\*/

time); /\*时间\*/

}

/\*终端信号处理函数SIGINT\*/

static void icmp\_sigint(int signo)

{

alive = 0; /\*告诉接收和发送线程结束程序\*/

gettimeofday(&tv\_end, NULL); /\*读取程序结束时间\*/

tv\_interval = icmp\_tvsub(tv\_end, tv\_begin); /\*计算一下总共所用时间\*/

return;

}

## 四、程序调试中遇到的问题和解决过程及运行结果

4.1使用wireshark抓取数据包，编写相关程序算出checksum值，并与抓取的数据包进行验证。截图上传。

4.2配置虚拟机IP地址，运行原始套接字程序，将运行结果截屏后上传到学习通平台。