**计算机网络编程**

**实验报告**

**班级：07111707**

**组长：1120171189 崔程远**

**成员：1120172149 吴沁璇**

**1120172153 张澈**

**1120172163 王晓媛**

**1120172733 张鉴昊**

**1120172765 曾煜瑾**

**1120173326 曾紫飞**

**北京理工大学**

**计算机学院**

**2020年5月**

**第五章 实验4 IP首部校验和的计算程序**

**1. 实验目的**

了解IP首部的格式，掌握首部校验和的计算方法。

1. **实验内容**

从wireshark中拷贝出一个IPv4数据报首部的16进制表示，放入配置文件，程序读入IP数据报后，计算首部的校验和，和wireshark产生的进行比较。

配置文件关键要点：

IPHeader=4500003CB53040008006E251C0A80168D83AC8EE

程序运行屏幕输出要点：

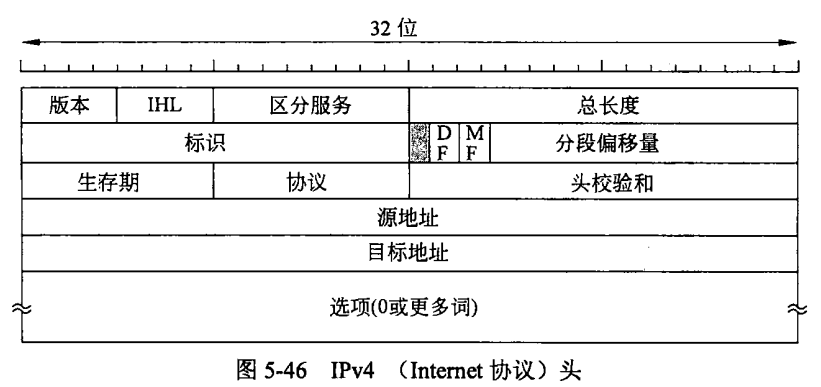
程序首先显示IP数据报首部的十六进制值

然后显示IP数据报首部每个字段的值（十进制），IP地址字段显示点分十进制记法

计算显示首部校验和

1. **实验原理**

IP数据报首部的格式为：

·版本（version）：指IP的版本，目前广泛使用的版本号为4。

·IHL：指明首部到底有多长，IHL的最小值为5，表明首部没有可选项，最大值为15，把首部的长度限制为最大60字节，因此选项字段最多为40字节。

·区分服务（Differentiated services）：前6位用来标记数据包的服务类别，后2位用来携带显式拥塞通知信息。

·总长度（Total length）：该数据报所有内容的长度，最大长度是65535个字节。

·标识（Identification）：用途是让目标主机确定一个新到达的分段属于哪一个数据报，同一个数据报的所有段包含同样的标识值。

·DF（Don't Fragment）：只有当DF = 0时才允许分片。

·MF（More Fragment）：MF = 1表示后面还有分片，MF = 0表示这已经是若干数据报片中的最后一个。

·片段偏移量（Fragment offset）：指明该段在当前数据报中的位置，除了数据报最后一个段外，其他所有段的长度必须是8字节的倍数。

·生存期（Time to live）：数据报在网络中可通过的路由器数的最大值，标识分组在网络中的寿命，以确保分组不会永远在网路中循环。路由器在转发分组前，先把TTL减1，若TTL被减为0，则该分组必须丢弃。

·协议（Protocol）：指出此分组携带的数据使用何种协议，即分组的数据部分应交给哪个传输层协议，如TCP、UDP等。其中值为6表示TCP，值为17表示UDP。

·头检验和（Header checksum）：校验算法的执行过程是这样的：当数据到达时，所有的16位（半字）累加起来，然后再取结果的补码。该算法的目的是到达数据包的头检验和计算结果应该为0.

·源地址（Source address）和目标地址（Destination address）字段表示源网络接口和目标网络接口的IP地址。

校验和的计算方法：

当发送IP包时，需要计算IP报头的校验和：

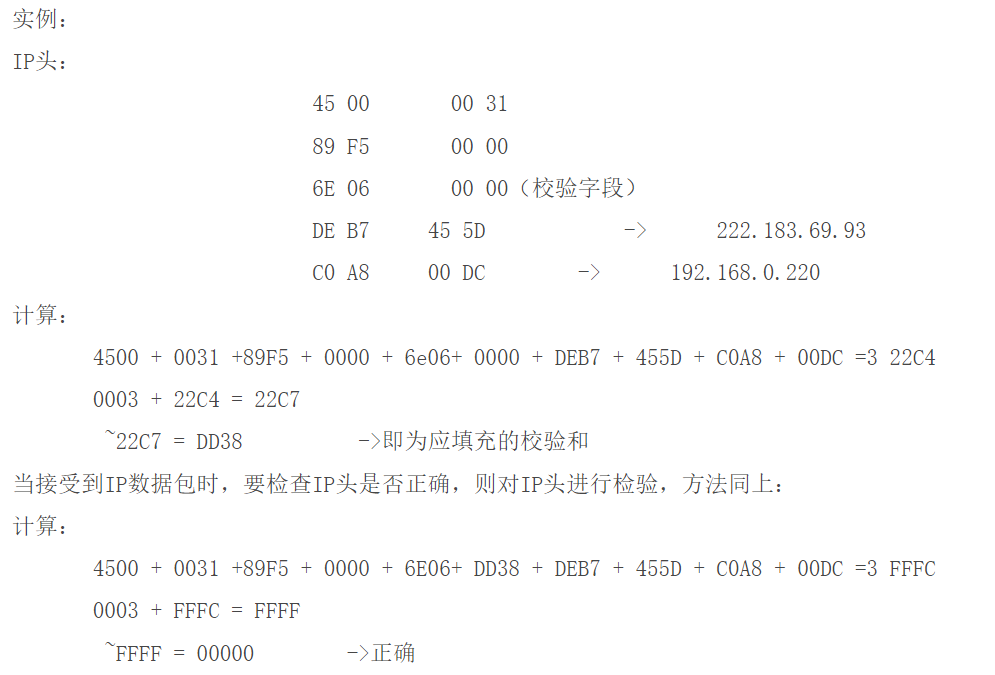
1、把校验和字段置为0；

2、对IP头部中的每16bit进行二进制求和；

3、如果和的高16bit不为0，则将和的高16bit和低16bit反复相加，直到和的高16bit为0，从而获得一个16bit的值；

4、将该16bit的值取反，存入校验和字段。

当接收IP包时，需要对报头进行确认，检查IP头是否有误，算法同上2、3步，然后判断取反的结果是否为0，是则正确，否则有错。



1. **实验环境**

|  |  |  |
| --- | --- | --- |
| 语言 | 集成开发环境 | 编译器 |
| C++ | Visual Studio 2017 | gcc version 4.8.1 |
| Java | Eclipse 2019 | java version "1.8.0\_65" |
| Python | Pycharm 2017 | Python 3.7.0 |

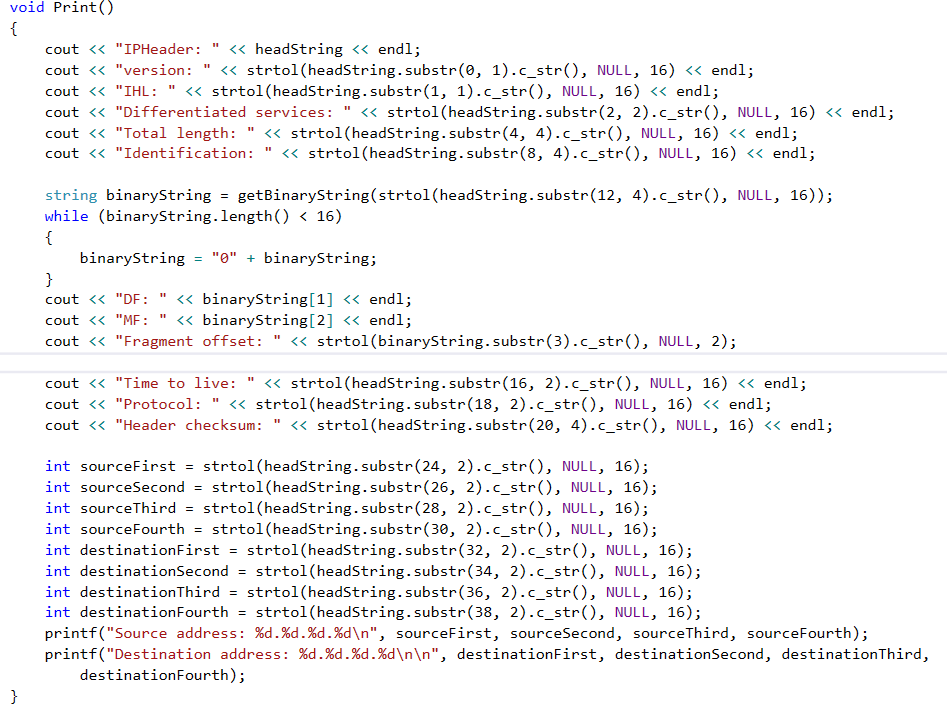
1. **实验步骤**

三份代码的结构和输出完全一致，均采用面向对象的构造方法，以C++代码为例进行分析。

·定义全局变量为IP首部

private:

string headString = "4500003CB53040008006E251C0A80168D83AC8EE";

·显示IP数据报首部每个字段的值（十进制），IP地址字段显示点分十进制记法

·计算校验和

void calculateCheckSum()

{

int a[9][4];

int pos = 0, num = 0;

while (pos < headString.length())

{

if (pos == 20)

{

pos += 4;

continue;

}

int sub = 0;

if (isalpha(headString[pos])) {

sub = headString[pos] - 'A' + 10;

}

else if (isdigit(headString[pos]))

{

sub = headString[pos] - '0';

}

a[num / 4][num % 4] = sub;

num++;

pos++;

}

int b[4];

string checkSumString = "";

int more = 0;

for (int i = 3; i >= 0; i--)

{

int sum = more;

for (int j = 0; j <= 8; j++)

{

sum += a[j][i];

}

b[i] = sum % 16;

more = sum / 16;

}

b[3] += more;

for (int i = 0; i < 4; i++)

{

b[i] = 15 - b[i];

if (b[i] <= 9)

{

checkSumString.append(1, (char)(b[i] + '0'));

}

else

{

checkSumString.append(1, (char)(b[i] - 10 + 'A'));

}

}

cout << "Calculating checksum: " << checkSumString;

if (!checkSumString.compare(headString.substr(20, 4)))

{

cout << ", is identical with Header checksum." << endl;

}

else

{

cout << ", is not identical with Header checksum." << endl;

}

}

·主函数

int main()

{

CheckSum operation;

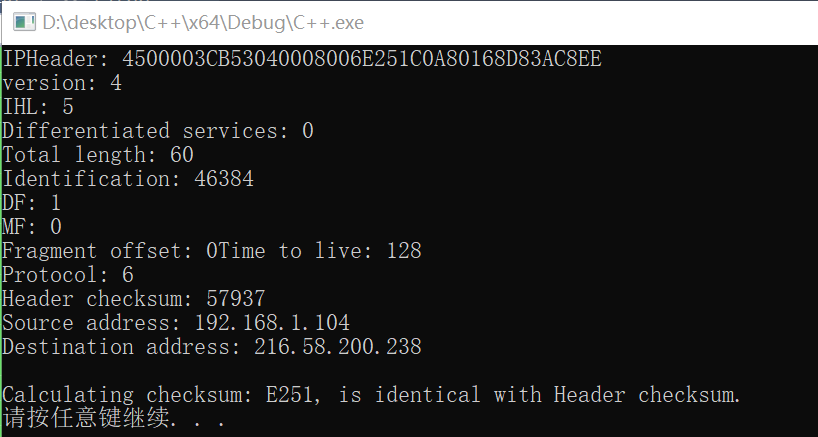
operation.Print();

operation.calculateCheckSum();

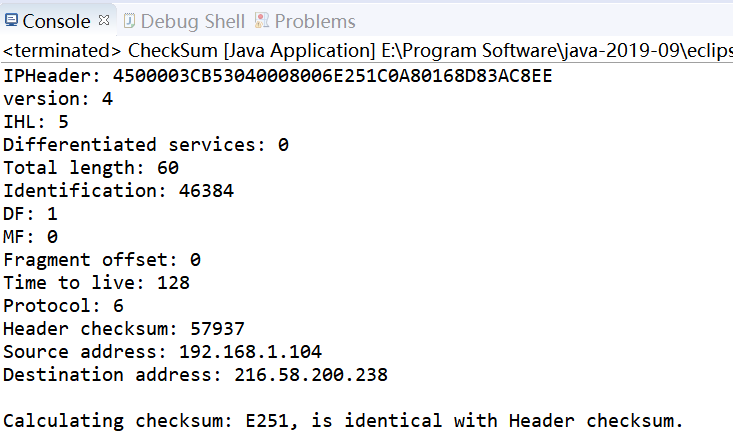
}

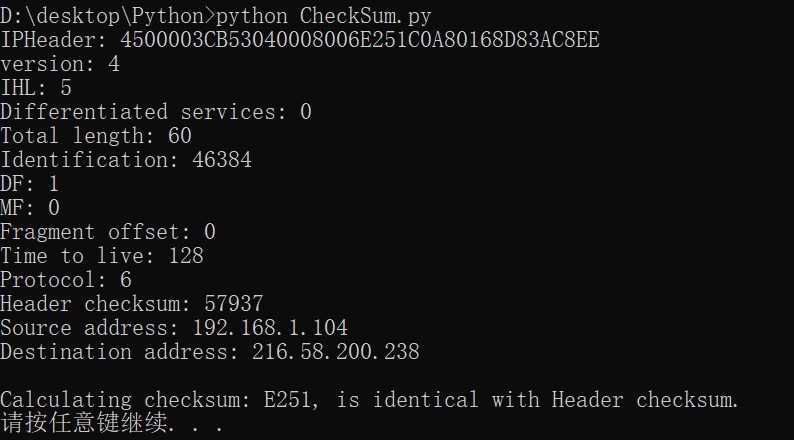
1. **实验结果**

**·C++**

****

**·Java**

**·Python**



1. **实验总结**

这个实验也挺简单的，首部校验和就是把除了校验和字段的其他共9个16进制数字进行进位加法，然后取反，我采取的方法是提取16进制数字到一个9\*4的二维整型数组，进行10进制加法，然后将结果转换为16进制数。这个实验中可能的挑战是二进制、十六进制字符串和整数的转换需要调用一些函数，三种语言的函数不一样，需要查阅一些资料。