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

1120172736 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。

2. 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

3. 实验原理

IP 数据报首部的格式为：

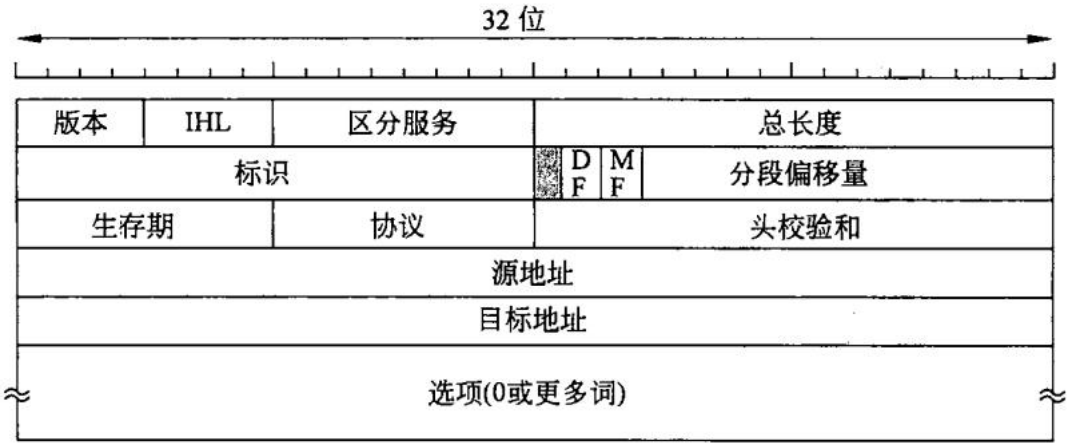


图 5-46　IPv4 （Internet 协议）头

- 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。

实例：

IP头：

45 00　　00 31
89 F5　　00 00
6E 06　　00 00（校验字段）
DE B7　　45 5D　　->　　222.183.69.93
C0 A8　　00 DC　　->　　192.168.0.220

计算：

4500 + 0031 +89F5 + 0000 + 6e06+ 0000 + DEB7 + 455D + C0A8 + 00DC =3 22C4
0003 + 22C4 = 22C7
~22C7 = DD38　　->即为应填充的校验和

当接受到IP数据包时，要检查IP头是否正确，则对IP头进行检验，方法同上：

计算：

4500 + 0031 +89F5 + 0000 + 6E06+ DD38 + DEB7 + 455D + C0A8 + 00DC =3 FFFC
0003 + FFFC = FFFF
~FFFF = 00000　　->正确

4. 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

5. 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

```
void Print()
{
    cout << "IPHeader: " << headString << endl;
    cout << "version: " << strtol(headString.substr(0, 1).c_str(), NULL, 16) << endl;
    cout << "IHL: " << strtol(headString.substr(1, 1).c_str(), NULL, 16) << endl;
    cout << "Differentiated services: " << strtol(headString.substr(2, 2).c_str(), NULL, 16) << endl;
    cout << "Total length: " << strtol(headString.substr(4, 4).c_str(), NULL, 16) << endl;
    cout << "Identification: " << strtol(headString.substr(8, 4).c_str(), NULL, 16) << endl;

    string binaryString = getBinaryString(strtol(headString.substr(12, 4).c_str(), NULL, 16));
    while (binaryString.length() < 16)
    {
        binaryString = "0" + binaryString;
    }
    cout << "DF: " << binaryString[1] << endl;
    cout << "MF: " << binaryString[2] << endl;
    cout << "Fragment offset: " << strtol(binaryString.substr(3).c_str(), NULL, 2);

    cout << "Time to live: " << strtol(headString.substr(16, 2).c_str(), NULL, 16) << endl;
    cout << "Protocol: " << strtol(headString.substr(18, 2).c_str(), NULL, 16) << endl;
    cout << "Header checksum: " << strtol(headString.substr(20, 4).c_str(), NULL, 16) << endl;

    int sourceFirst = strtol(headString.substr(24, 2).c_str(), NULL, 16);
    int sourceSecond = strtol(headString.substr(26, 2).c_str(), NULL, 16);
    int sourceThird = strtol(headString.substr(28, 2).c_str(), NULL, 16);
    int sourceFourth = strtol(headString.substr(30, 2).c_str(), NULL, 16);
    int destinationFirst = strtol(headString.substr(32, 2).c_str(), NULL, 16);
    int destinationSecond = strtol(headString.substr(34, 2).c_str(), NULL, 16);
    int destinationThird = strtol(headString.substr(36, 2).c_str(), NULL, 16);
    int destinationFourth = strtol(headString.substr(38, 2).c_str(), NULL, 16);
    printf("Source address: %d.%d.%d.%d\n", sourceFirst, sourceSecond, sourceThird, sourceFourth);
    printf("Destination address: %d.%d.%d.%d\n\n", destinationFirst, destinationSecond, destinationThird,
        destinationFourth);
}
```

• 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)
            {
```

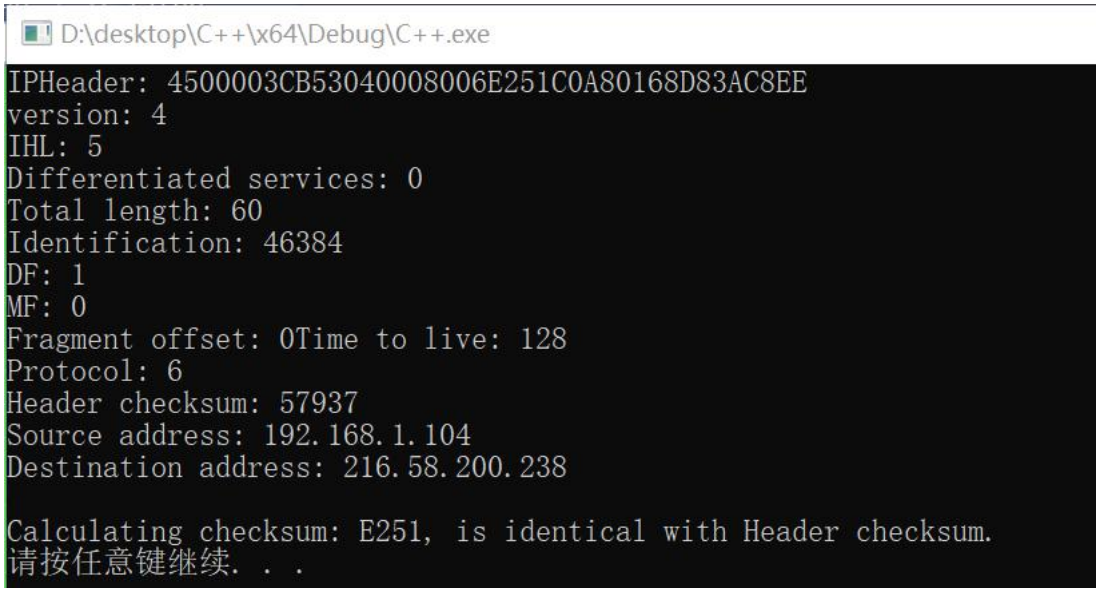
```
                checkSumString.append(1, (char)(b[i] + '0'));
            }
            else
            {
                checkSumString.append(1, (char)(b[i] - 10 + 'A'));
            }
        }
        cout << "Calculating checksum: " << checkSumString;
        if (!checkSumString.compare(headString.substr(20, 4)))
        {
            cout << ", is identical with Header checksum." << endl;
        }
        else
        {
            cout << ", is not identical with Header checksum." << endl;
        }
    }
```

• 主函数

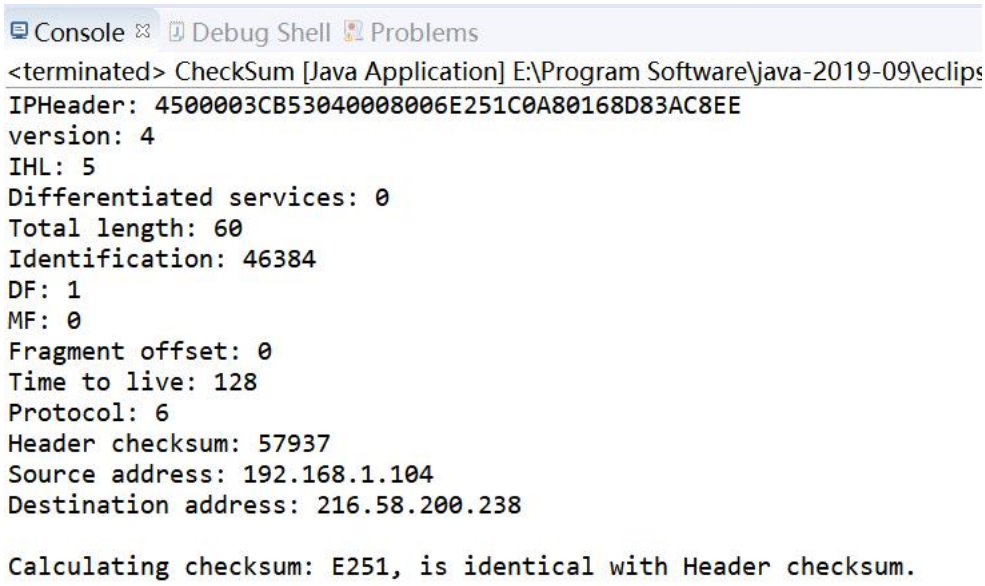
```
int main()
{
    CheckSum operation;
    operation.Print();
    operation.calculateCheckSum();
}
```

6. 实验结果

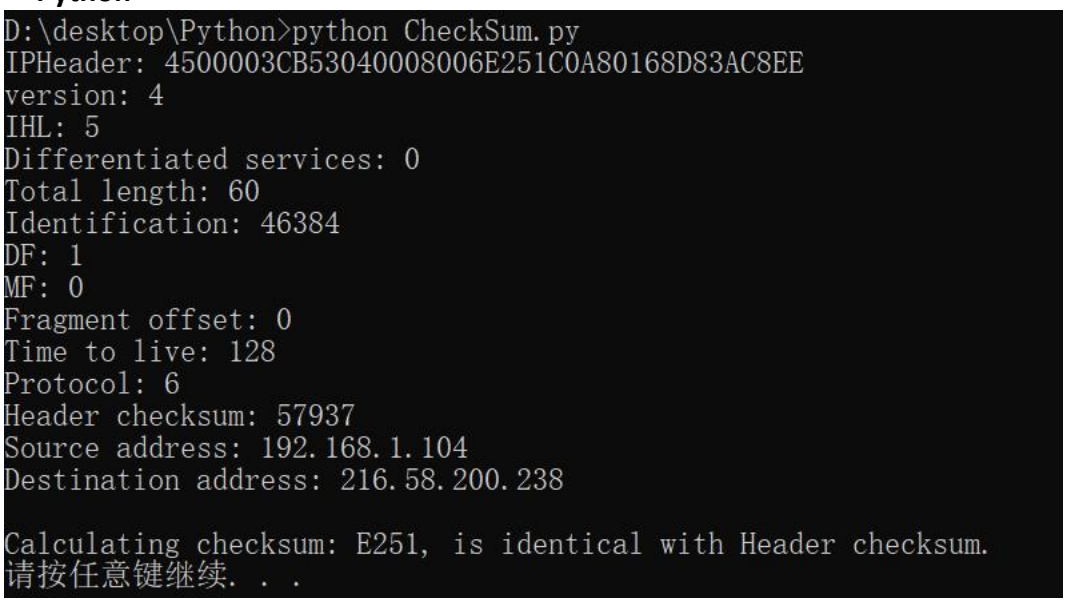
• C++



• **Java**



• **Python**



• **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。