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

第三章 实验 2　分别实现零比特填充和字节填充

1. 实验目的

模拟数据链路层中成帧的零比特填充和字节填充，加深对这两种成帧方法的掌握和理解。

2. 实验内容

比特填充配置文件关键要点：
待发送的数据信息二进制比特串（32 位）
InfoString1=0110XXXXXX11111111111XXXXXXXX110
帧起始和结束标志二进制比特串
FlagString=01111110

比特填充程序运行屏幕输出关键要点：
屏幕显示帧起始标志、帧数据信息和帧结束表示
比特填充，显示比特填充后的发送帧
显示比特删除后的接收帧

字节填充配置文件关键要点：
待发送的数据信息十六进制串（64 位）
InfoString1=347D7E807E40AA7D
帧起始和结束标志十六进制串
FlagString=7E

字节填充程序运行屏幕输出关键要点：
屏幕显示帧起始标志、帧数据信息和帧结束表示
字节填充，显示字节填充后的发送帧
显示字节删除后的接收帧

3. 实验原理

• 零比特填充

零比特填充法允许数据帧中包含任意个数的比特，也允许每个字符的编码包含任意个数的比特。它使用一个特定的比特模式，即 01111110 来标志一帧的开始和结束。为了不使信息位中出现的比特流 01111110 被误判为帧的首尾标志，发送方的数据链路层在信息位中遇到 5 个连续的“1”时，将自动在其后面插入一个“0”；而接受方做该过程的逆操作，即每收到 5 个连续的“1”时，自动删除后面紧跟的“0”，以恢复原信息。

• 字节填充

字节填充让每个帧用一些特殊的字节作为开始和结束，这些特殊字节通常都相同，称为标志字节，两个连续的标志字节代表了一帧的结束和下一帧的开始。然而当标志字节出现在数据中时，会严重干扰到帧的分界，发送方通过在数据链路层中偶尔出现的每个标志字节的前面插入一个特殊的转移字节（ESC）就可以解决这个问题。因此，只要看到数据中标志字节的前面有没有转移字节，就可以把作为帧分界符的标志字节与数据中的标志字节区分开来，接收方的数据链路层在将数据传递给网络层之前必须删除转移字节。

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

零比特填充和字节填充的各三份代码的结构完全一致，均采用面向对象的构造方法，首先初始化一个类。在 Send 函数中初始化数据信息二进制比特串，然后加入开始定界符和结束定界符，并进行零比特填充或字节填充，返回发送帧。发送帧传入 Receive 函数，在 Receive 函数进行比特删除或者字节删除，但保留开始定界符和结束定界符，作为比特删除后或者字节删除后的接收帧，打印出来。下面以 C++代码为例进行代码分析：

一、零比特填充

• 全局变量

```
private:
    string flagString = "01111110";
    string insertString = "11111";
```

表示起始标志/结束标志，每遇到一个 insertString 在其后插入一个 0。

• Send 函数

```
string Send()
    {
        string dataStr = "01101111111111111111111111111110";

        string sendFrame = flagString + dataStr + flagString;
        int flagLen = flagString.length();
        int sendFrameLen = sendFrame.length();
        cout << "帧起始标志：" << sendFrame.substr(0, flagLen) << endl;
        cout <<"帧数据信息：" << sendFrame.substr(flagLen, sendFrameLen
- flagLen - flagLen) << endl;
        cout <<"帧结束标志：" << sendFrame.substr(sendFrameLen - flagLen)
<< endl;

        int insertLen = insertString.length();
        for (int i = flagLen; i < (sendFrame.length() - flagLen); i++)
        {
            if (!sendFrame.substr(i, insertLen).compare(insertString))
            {
                sendFrame = sendFrame.substr(0, i + insertLen) + "0" +
sendFrame.substr(i + insertLen);
                i = i + insertLen;
            }
        }

        cout << "比特填充后的发送帧：" << sendFrame << endl << endl;
```

```
        return sendFrame;
    }
```
• Receive 函数
```
void Receive(string receiveFrame)
    {
        int flagLen = flagString.length();
        int insertLen = insertString.length();
        for (int i = flagLen; i < (receiveFrame.length() - flagLen); i++)
        {
            if (!receiveFrame.substr(i,
insertLen).compare(insertString))
            {
                receiveFrame = receiveFrame.substr(0, i + insertLen) +
receiveFrame.substr(i + insertLen + 1);
                i = i + insertLen - 1;
            }
        }
        cout << "比特删除后的接收帧：" << receiveFrame << endl;
    }
```
• 主函数
```
int main()
{
    BitStuffing operation;
    string frameStr = operation.Send();
    operation.Receive(frameStr);

}
```

二、字节填充

• 全局变量
```
private:
    string flagString = "7E";
    string insertString = "1B";
```
表示起始标志/结束标志，每遇到一个标志位在其前面插入一个“1B”（ESC）。

• Send 函数
```
string Send()
    {
        string dataStr = "347D7E807E40AA7D";
        string sendFrame = flagString + dataStr + flagString;
        int flagLen = flagString.length();
        int sendFrameLen = sendFrame.length();
        cout << "帧起始标志：" << sendFrame.substr(0, flagLen) << endl;
        cout <<"帧数据信息：" << sendFrame.substr(flagLen, sendFrameLen
- flagLen - flagLen) << endl;
```

```
        cout <<"帧结束标志："<< sendFrame.substr(sendFrameLen - flagLen)
<< endl;

        int insertLen = insertString.length();
        for (int i = flagLen; i < (sendFrame.length() - flagLen); i++)
        {
            if (!sendFrame.substr(i, flagLen).compare(flagString))
            {
                sendFrame = sendFrame.substr(0, i) + insertString +
sendFrame.substr(i);
                i = i + insertLen + flagLen;
            }
        }

        cout << "字节填充后的发送帧：" << sendFrame << endl << endl;
        return sendFrame;
    }
```

• Receive 函数

```
void Receive(string receiveFrame)
    {
        int flagLen = flagString.length();
        int insertLen = insertString.length();
        for (int i = flagLen; i < (receiveFrame.length() - flagLen); i++)
        {
            if (!receiveFrame.substr(i, flagLen).compare(flagString))
            {
                receiveFrame = receiveFrame.substr(0, i - insertLen) +
receiveFrame.substr(i);
                i = i + flagLen;
            }
        }
        cout << "字节删除后的接收帧：" << receiveFrame << endl;
    }
```

• 主函数

```
int main()
{
    ByteStuffing operation;
    string frameStr = operation.Send();
    operation.Receive(frameStr);
}
```

6. 运行结果

一、零比特填充

Send 函数中将 01101111111111111111111111111110 作为待发送的数据信息。

• C++

```
D:\desktop\C++>BitStuffing.exe
Frame start flag: 01111110
Frame data info: 011011111111111111111111111111110
Frame end flag: 01111110
Send frame after BitStuffing: 0111111001101111101111101111101111101111011001111110

Receive frame after deleting bit: 0111111001101111111111111111111111111111001111110
请按任意键继续. . .
```

• Java

```
Console  Debug Shell  Problems
<terminated> BitStuffing [Java Application] E:\Program Software\java-2019-09\eclipse\jre\bin\javaw.exe (2(
Frame start flag: 01111110
Frame data info: 011011111111111111111111111111110
Frame end flag: 01111110
Send frame after BitStuffing: 0111111001101111101111101111101111101111011001111110

Receive frame after deleting bit: 0111111001101111111111111111111111111111001111110
```

• Python

```
D:\desktop\Python>python BitStuffing.py
Frame start flag: 01111110
Frame data info: 011011111111111111111111111111110
Frame end flag: 01111110
Send frame after BitStuffing: 0111111001101111101111101111101111101111011001111110

Receive frame after deleting bit: 0111111001101111111111111111111111111111001111110
```

二、字节填充

待发送的数据信息为 347D7E807E40AA7D

• C++

```
D:\desktop\C++>ByteStuffing.exe
Frame start flag: 7E
Frame data info: 347D7E807E40AA7D
Frame end flag: 7E
Send frame after ByteStuffing: 7E347D1B7E801B7E40AA7D7E

Receive frame after deleting byte: 7E347D7E807E40AA7D7E
请按任意键继续. . .
```

• Java

```
Console  Debug Shell  Problems
<terminated> ByteStuffing [Java Application] E:\Program Software\java-201
Frame start flag: 7E
Frame data info: 347D7E807E40AA7D
Frame end flag: 7E
Send frame after ByteStuffing: 7E347D1B7E801B7E40AA7D7E

Receive frame after deleting byte: 7E347D7E807E40AA7D7E
```

• Python

```
D:\desktop\Python>python ByteStuffing.py
Frame start flag: 7E
Frame data info: 347D7E807E40AA7D
Frame end flag: 7E
Send frame after ByteStuffing: 7E347D1B7E801B7E40AA7D7E

Receive frame after deleting byte: 7E347D7E807E40AA7D7E
```

7. 实验总结

这个实验相对来说挺简单的，通过遍历出字符子串然后在其前面插入转义字符实现字节填充，或者在其后面插入零比特实现零比特填充。通过这个实验，我觉得零比特填充的性能优于字节填充，因为字节填充的转义字节也有可能在原始数据中出现使得帧定界可能有偏差。