     
**重庆大学课程项目报告**  


项目名称：网络协议数据的封装与解封装

学 院： 计算机学院

# 网络协议设计思想

本项目将以教材讲解的5层协议的体系结构为基础，通过C语言模拟每一层数据（除物理层）的封装和解封装。5层协议的体系结构如图1所示。由于物理层是单纯的0101…比特流，我们不需要进行模拟，所以本项目模拟的层次只包括前4层：数据链路层、网络层、运输层和应用层。

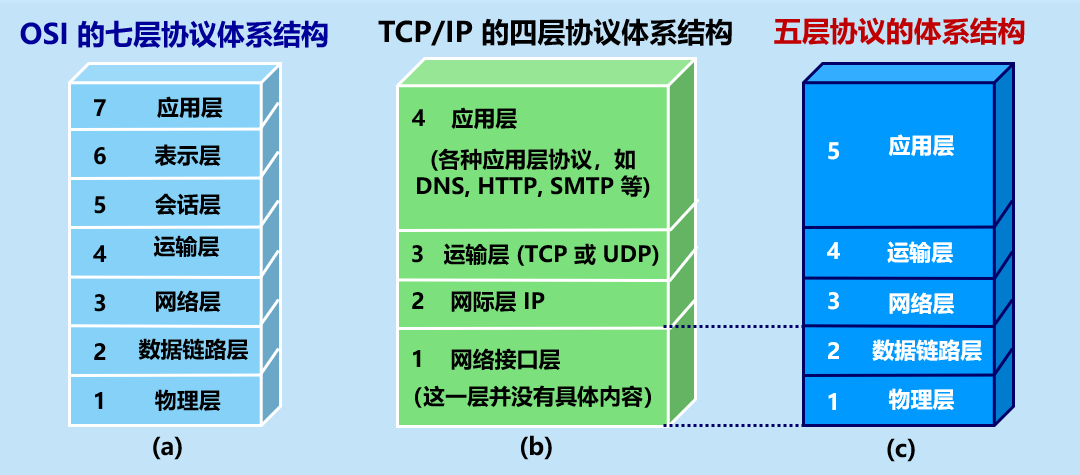


图1 三种体系结构对比图

下面将自底向上逐层介绍本项目所采用的每一层具体协议和格式。

## 数据链路层设计

本项目在数据链路层采用了以太网（IEEE 802.3）的MAC帧格式，MAC帧结构如图2所示。

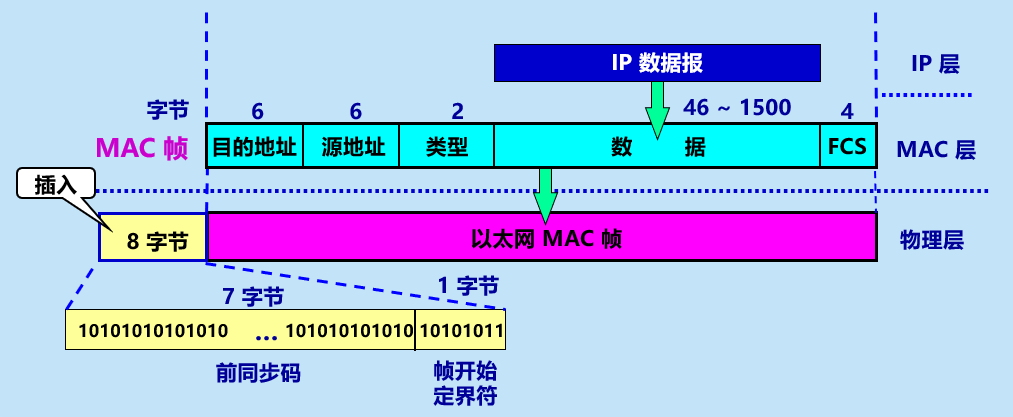


图2 MAC帧格式示意图

从图中可以看出MAC帧由以下几个部分组成：

1. 前同步码和帧起始定界符（Preamble）：用于同步接收方的时钟和标识个MAC帧的开始。

2. 目标MAC地址（Destination MAC Address）：指示接收帧的目标设备的MAC地址。

3. 源MAC地址（Source MAC Address）：指示发送帧的源设备的MAC地址。

4. 类型字段（Type）：指示上一层使用的是什么协议。

5. 数据字段（Data）：包含要传输的实际数据。

6. 帧校验序列（Frame Check Sequence）：用于检测传输过程中是否发生了错误。

由于我们模拟传输的过程不需要进行时钟同步，所以模拟过程中不会使用前同步码和帧起始定界符，也就是说我们封装的MAC只具有目标地址、源地址、类型字段、数据字段和帧检测字段五个部分。它们各自的长度在图中已经标识出来了。

## 网络层设计

本项目在网络层采用了IPv4协议，封装的数据格式也是按照IP数据报的格式进行封装的，IP数据报的格式如图3所示。

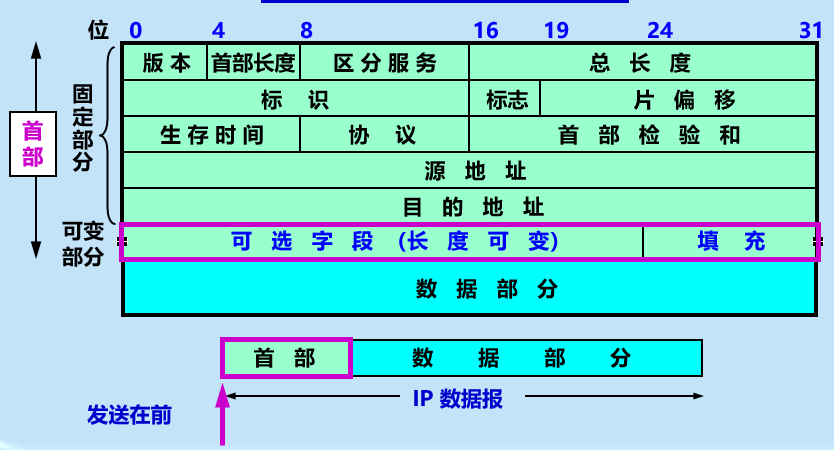


图3 IP数据报格式示意图

IPv4数据包的格式如下：

1. 版本（Version）：指示使用的IPv4协议版本，设置为4。

2. 头部长度（Header Length）：指示IPv4头部的长度，以32位字长为单位。

3. 服务类型（Type of Service）：用于指定数据包的优先级和服务质量要求。

4. 总长度（Total Length）：指示整个IPv4数据包的长度，包括头部和数据部分。

5. 标识（Identification）：用于唯一标识一个数据包，用于分片和重组。

6. 标志（Flags）：用于指示数据包是否可以分片以及分片的相关信息。

7. 片偏移（Fragment Offset）：指示分片数据包在原始数据包中的位置。

8. 生存时间（Time to Live）：指示数据包在网络中的最大生存时间，用于防止数据包在网络中无限循环。

10. 协议（Protocol）：指示封装在IPv4数据包中的上层协议，如TCP、UDP等。

11. 头部校验和（Header Checksum）：用于检测IPv4头部是否出现错误。

12. 源IP地址（Source IP Address）：指示发送数据包的源主机的IP地址。

13. 目标IP地址（Destination IP Address）：指示接收数据包的目标主机的IP地址。

14. 选项（Options）：可选字段，用于提供额外的功能或控制。

在本项目的模拟过程中，由于我们设计的模拟较为简单，就没有设置第14项选项字段，默认该字段不存在。其余部分都按照上述内容实现了。

## 运输层设计

本项目在运输层选择了无连接的UDP协议，它提供了一种简单的、不可靠的数据传输服务。UDP协议较为简单，只是在IP的数据报服务之上增加了很少的一点功能。UDP的格式如图4所示。

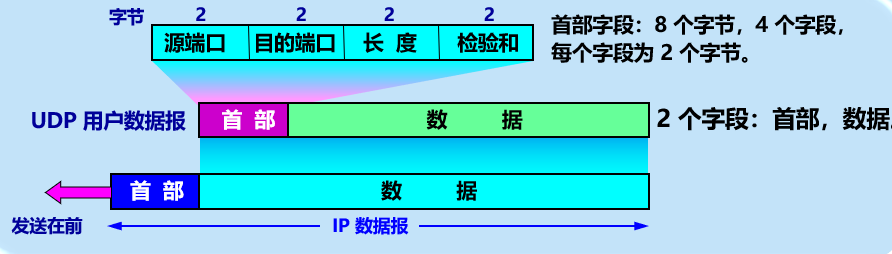


图4 UDP格式示意图

从图中可以看出UDP的具体格式如下：

1. 源端口号（Source Port）：指示发送端的端口号。

2. 目标端口号（Destination Port）：指示接收端的端口号。

3. 长度（Length）：指示UDP数据报的长度，包括头部和数据部分。

4. 校验和（Checksum）：用于检测UDP数据报是否出现错误。

5. 数据（Data）：包含要传输的实际数据。

UDP除了数据之外的4个字段都是2字节的长度，实现起来也较为简单。

## 应用层设计

我们在应用层设计了一个用于进行聊天的协议。我们参考了QQ类应用软件的格式，一条文字消息需要由消息内容、消息接受者、消息发送者以及时间戳构成。除了这些在显示时需要的内容，我们还需要一些其他的附件信息来保证协议传输的质量。我们将我们的协议命名为ChatRoom Protocol（CRP）。

### CRP语法

CRP具体的格式如下所示：

1. 版本号（version）：2字节，使用的CRP协议的版本，协议可能会更新，需要更改版本号。

2. 请求类型（request）：2字节，用于区别是何种类型的信息。

3. 编码方式（encode）：2字节，指明聊天消息内容的编码方式，以便对方解码。

4. 状态（state）：2字节，用于表明消息的状态。

5. 发送者署名长度（slen）：2字节，发送者署名的长度。

6. 接受者署名长度（rlen）：2字节，接受者署名的长度。

7. 消息内容长度（mlen）：2字节，消息内容的长度。

8. 时间戳（time）：24字节，由发送主机自动在发送时为这条消息打上时间戳。

9. 发送者（sender）：用于表示发送者的昵称。

10.接受者（receiver）：用于表示接受者的昵称。

11.消息内容（message）：用于传输消息的内容。

12.总长度（length）：消息的总长度。

### CRP语义

在本项目中，控制信息含义如下：

1. 版本号默认为第一版，即0x0001。

2. 请求类型默认为Chat，即聊天。

3. 编码方式默认为UTF-8。

4. 状态默认为200，表示消息成功传输。

### CRP同步

接受方如果接受到消息，会先解读版本号，然后按照对应的版本号来解读这一条消息。下一步是查看请求类型，如果类型为Chat，就按照聊天消息的格式依次解读发送者，接受者，以及消息内容，然后转发给对应的接受者。如果是请求类消息，比如说删除群组，或者加入群组，就由服务器判断这一条请求是否可接受，然后依次执行。

# 各层次网络封装与解封装实现

下面将按照结构体、封装函数、解封装函数的顺序，依次讲解我们的程序编码功能。

## 各层次结构体定义

我们将结构体的定义全部放到了protocols.h这个头文件中，下面依次展示每一层的结构体定义。

### 数据链路层结构体

数据链路层对应的结构体图5所示，由于我们采用的是以太网（802.3）帧结构，所以共需要4个结构，目标地址（dst\_mac）、源地址（src\_mac）、类型（type）和校验和（crc）。

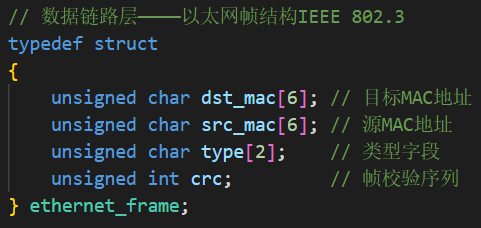


图5 以太网帧结构结构体

### 网络层IP数据报结构体

网络层对应的结构体图6所示，由于我们采用的是IP数据报结构，所以共需要12个字段，在这里我们不包括拓展首部。



图6 网络层IP数据报结构体

### 运输层结构体

我们在运输层使用了UDP协议，UDP协议较为简单，只有4个字段，结构体的定义如下所示：

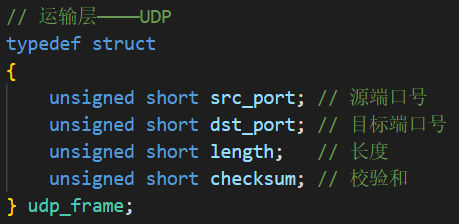


图7 运输层UDP结构体

### 应用层CRP结构体

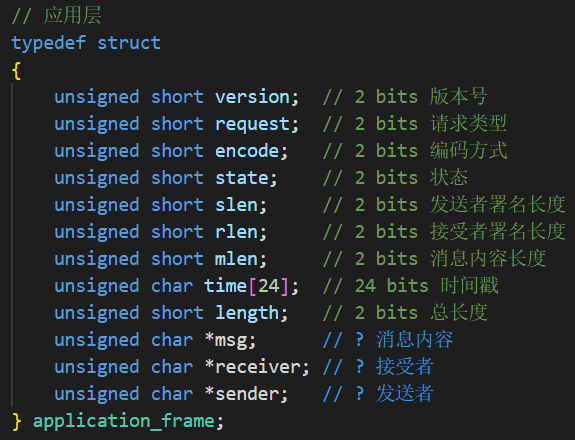
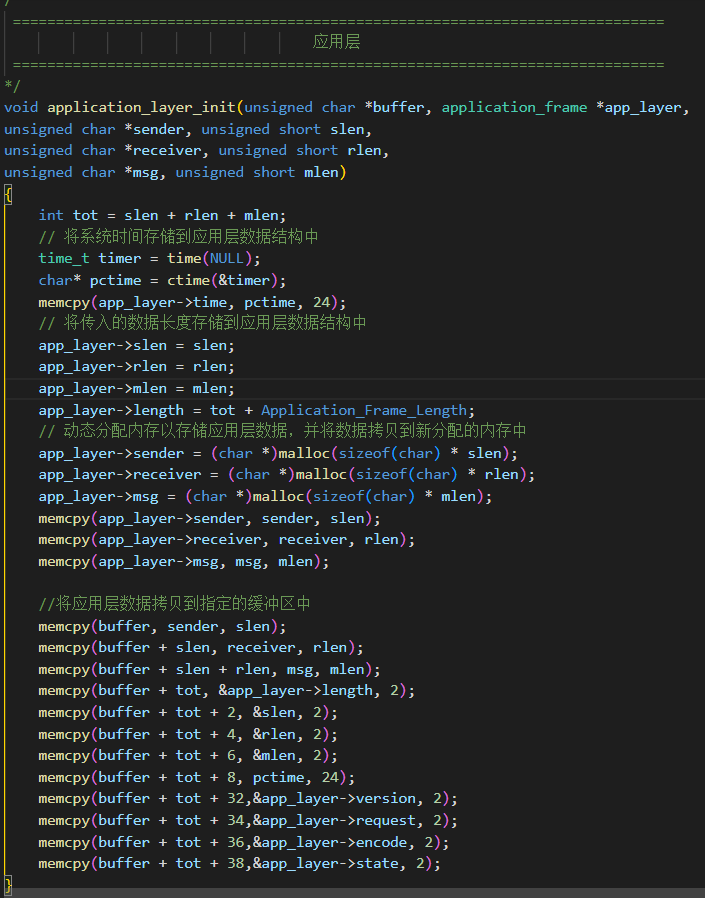


图8 应用层CRP协议结构体

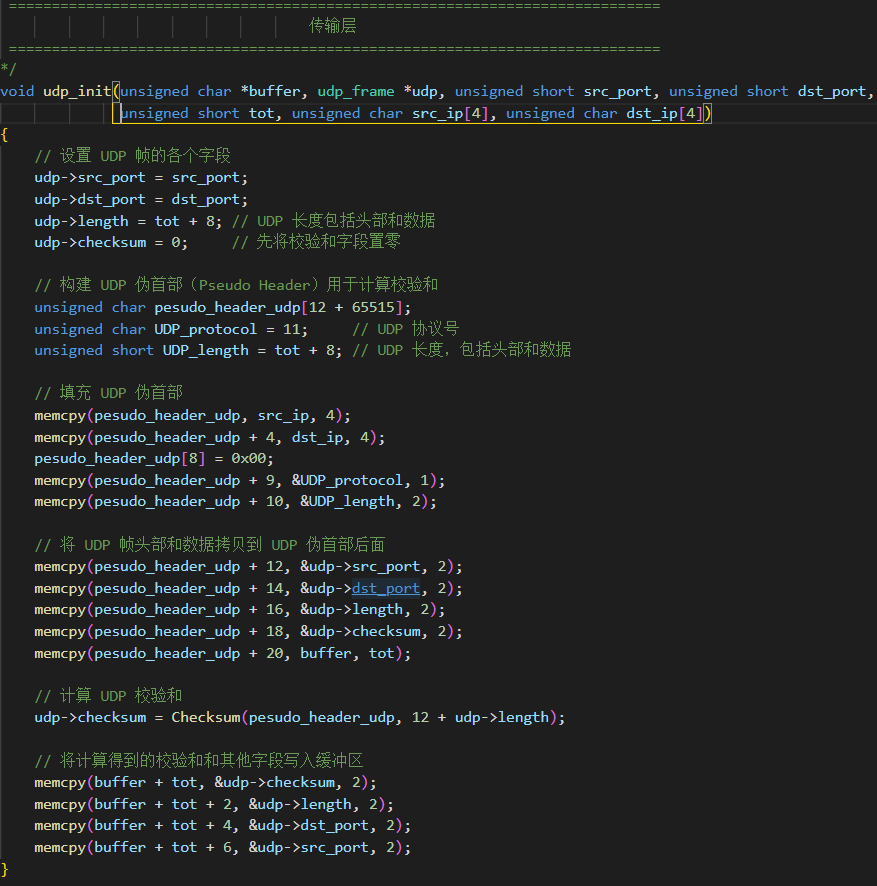
## 封装函数实现

我们将封装的函数全部放入了encode.c这个文件中。下面依次展示每一层封装的函数实现。由于封装的过程是从高层到底层，所以我们这里的封装函数的展示顺序也是从应用层到数据链路层。

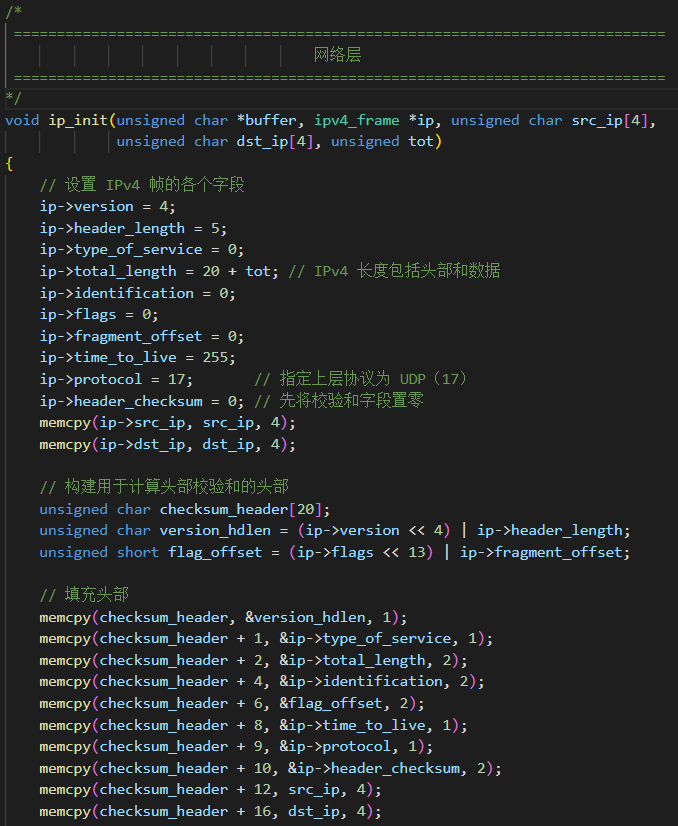
### 应用层封装函数

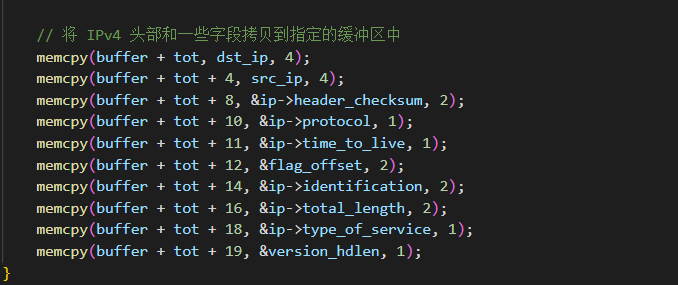


### 运输层封装函数

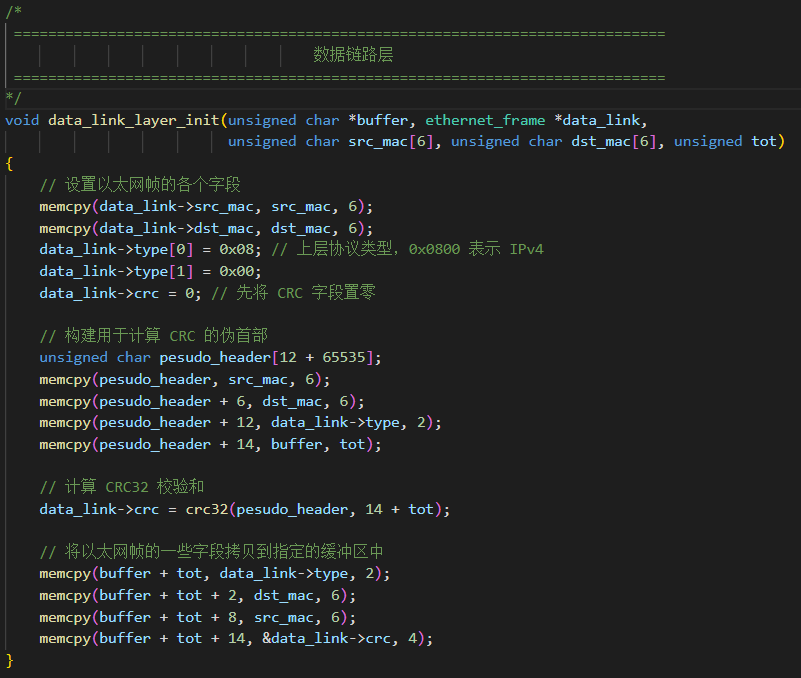


### 网络层封装函数





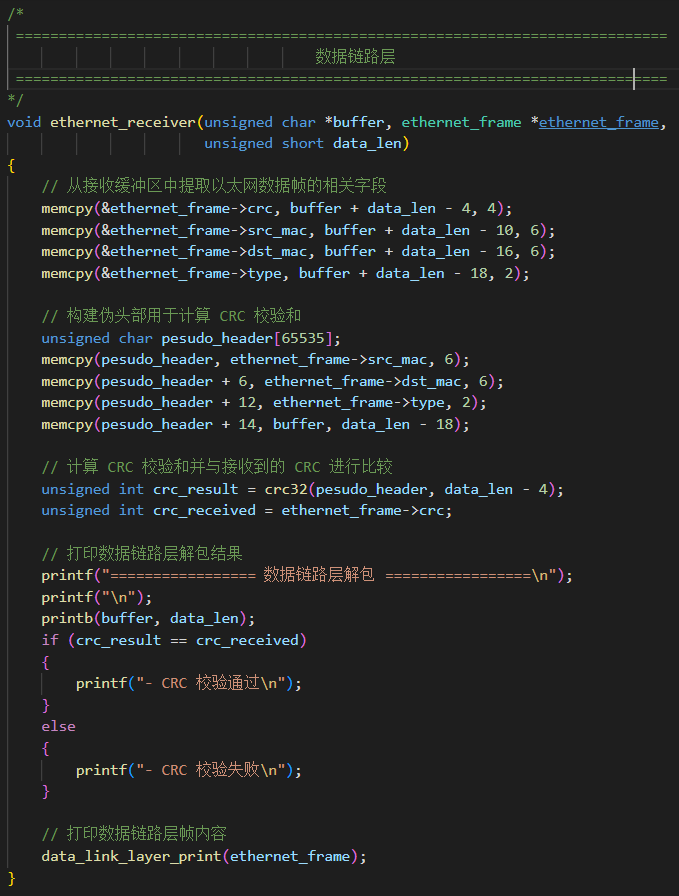
### 数据链路层封装函数



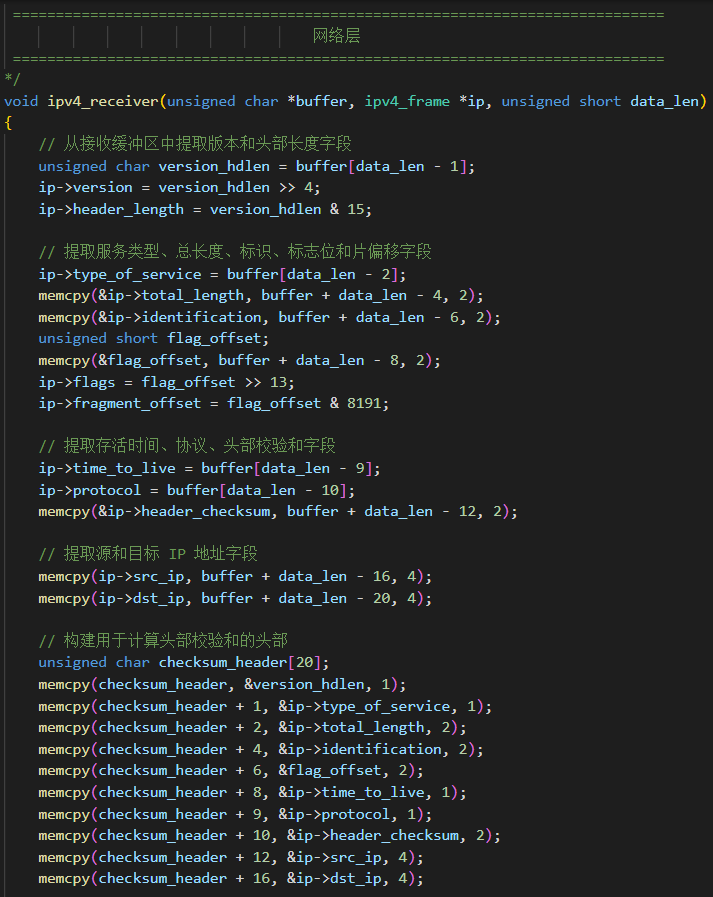
## 解封装函数实现

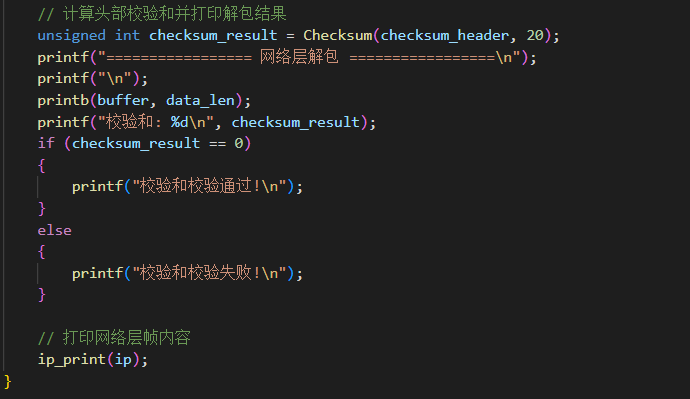
我们将解封装的函数全部放入了decode.c这个文件中。下面依次展示每一层封装的函数实现。由于解封装的过程是从底层到高层，所以我们这里的解封装函数的展示顺序更好与封装函数的顺序相反。

### 数据链路层解封装函数

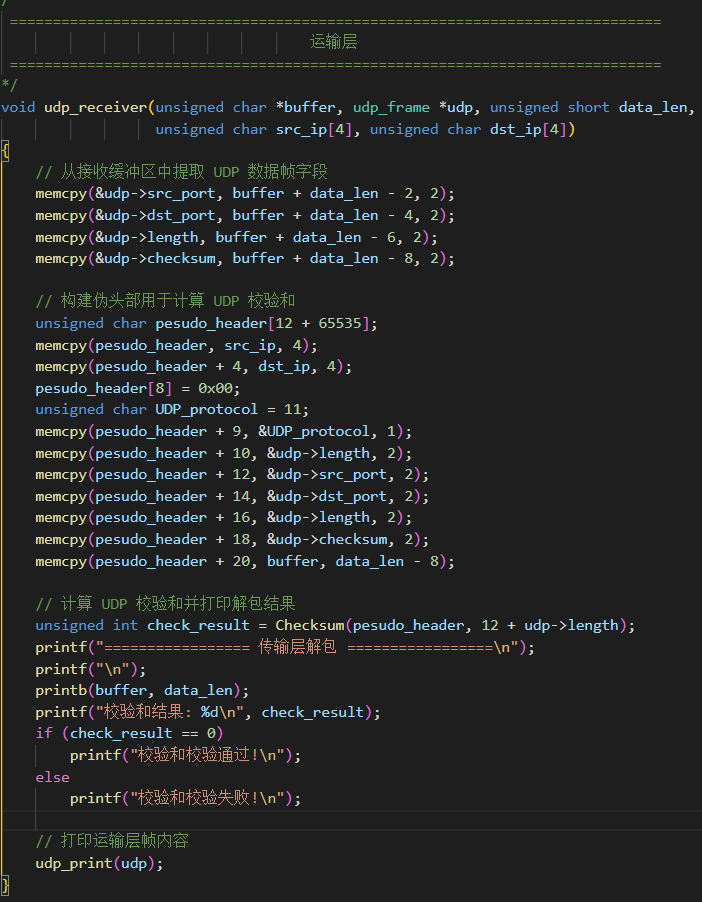


### 网络层解封装





### 运输层解封装



### 应用层解封装



# 代码运行结果展示

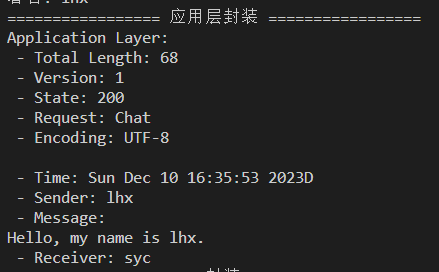
下面将展示我们的项目运行代码结果，我们需要依次运行encode.c和decode.c两个文件，然后按照命令行的提示输入聊天到内容，以完成信息的封装。

## encode.c文件运行结果

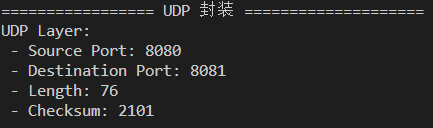
运行encode.c文件后会先提示我们需要输入聊天信息，接受者和署名。效果如下图所示：



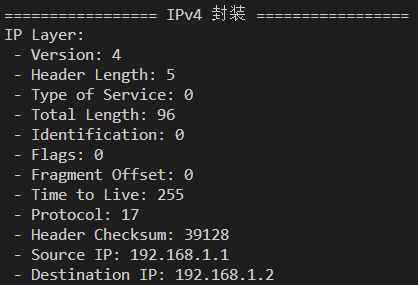
在署名输入完成后敲击回车即可开始信息的封装与发送。下面逐层展示信息的封装结果：



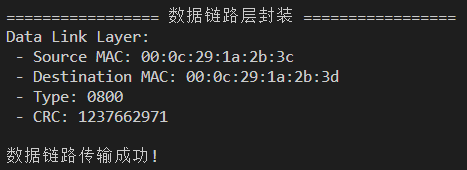
应用层封装展示了我们填入每个字段的内容。可以看到这里主机读取系统时间为我们的消息打上了时间戳。



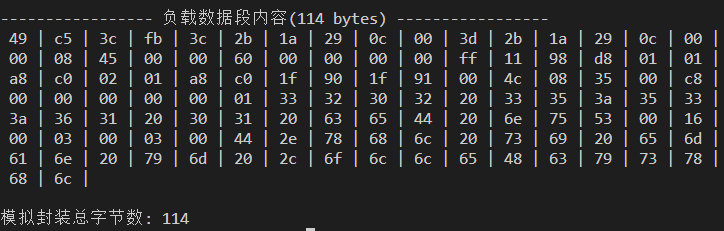
运输层封装为我们的信息设置了端口，在这里程序还计算了校验和为2101。



接下来是网络层的封装，这里展示了填入网络层结构体的各个字段。这里的IP地址是我们模拟设置的。



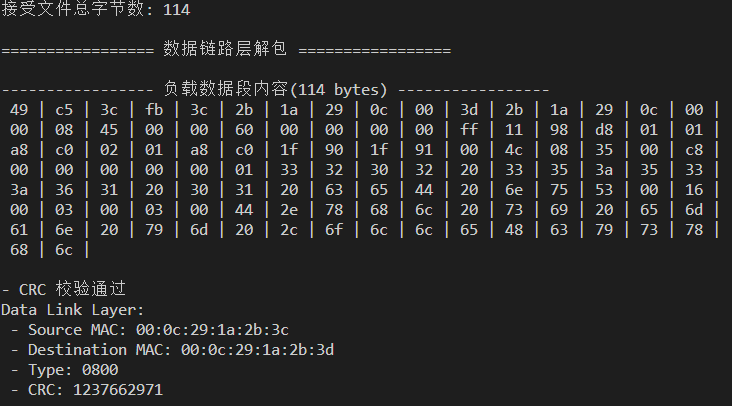
在数据链路层完成封装后我们的信息就可以开始传输了，我们模拟传输过程使用的是通过二进制方式写入txt文件中，然后再通过decode.c文件来读取txt文件中的内容来完成信息的模拟传输，写入二进制文件的内容如下所示：



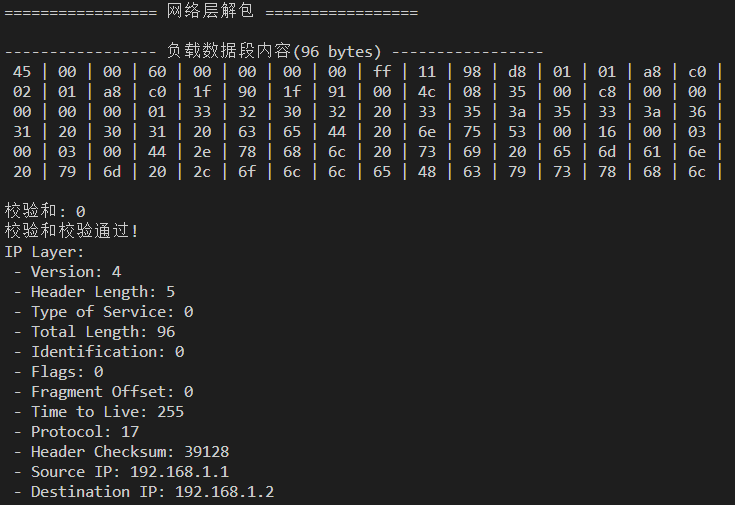
至此我们完成了信息的封装和传输，下面需要编译并运行decode.c文件来对我们传输的数据进行解封装。

## decode.c文件运行结果

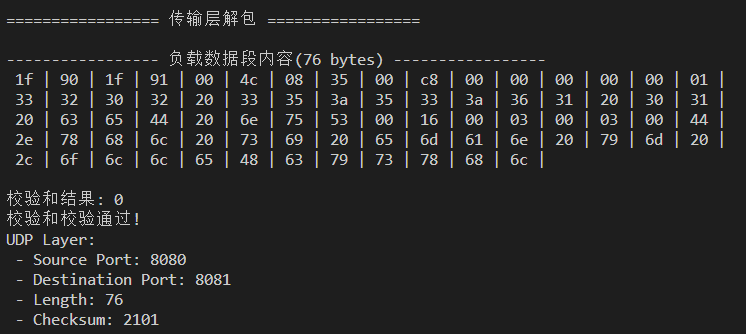
首先是数据链路层的解封装，这里展示了我们从txt文件中读取到的二进制比特流。这与我们封装时写入的二进制比特流一致。



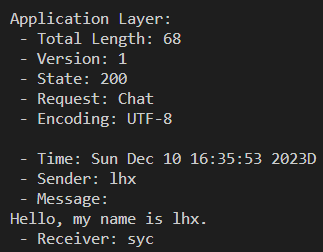
接着是网络层的解封装过程，我们可以看到解封出来的字段的内容。



然后是运输层的解封装。



最后是应用层的解封装。



在应用层把我们传输的信息以聊天的形式展示在这里。与我们的语气效果一致，表明我们模拟的消息传输过程成功。

# 项目总结与体会

通过这个项目，我们深入理解了网络协议在不同层次的封装与解封装过程。这不仅加深了我们对网络协议栈的理解，还提高了我们在C语言编程方面的实践能力。同时，设计并实现应用层的协议也锻炼了我们对协议设计的能力。通过演示实验，我们能够清晰地展示每个协议层次的工作过程，加深了我们对网络通信的整体认识。这个项目为我们今后在网络编程和协议设计方面的学习奠定了坚实的基础。