

1. 整体设计

本项目使用 c 语言完成了对数据链路层，网络层，运输层的协议的数据单元封装和解封装。数据链路层采用 **PPP 协议**、网络层参考 **IPv4 数据报格式**、运输层参考 **UDP 用户数据报** 实现。

2. 文件结构

```
----define_struct.h: 定义结构体的代码(各个协议的格式)

----framecheck_utils.h: 这进行检错的, ppp生成FCS, ip和udp生成检验和

----printf_utils.h: 各层封装或解封后的输出

----send.c: 封装部分

    可直接运行, 输入数据后, 会有结果输出, 并将数据写在testmsg.txt这个文件(封装好的数据)

----receive.c: 解封装部分

    运行后, 从testmsg.txt文件中读取send.c存放的数据, 并输出结果
```

3. 程序流程

本项目分为封装和解封装两部分，其互为逆过程。

封装部分中，由于未进行应用层的实现，首先获取用户需要传输的数据，然后将其按照 UDP 数据报->IPv4 数据报->PPP 帧格式的顺序进行封装，最后将封装结束后的数据存入 testmsg.txt 文件，同时以 16 进制的格式打印出写入的数据，方便后续进行对比核验。

解封装部分即从 testmsg.txt 中读取数据后，进行封装的逆过程。每一层均有详细的字段输出。

4. 实现过程

本节主要介绍协议格式、校验码的实现、封装和解封装的具体过程。

4.1 协议格式定义

本项目格式定义参考计算机网络第七版书。

4.1.1 PPP 帧格式

PPP 帧协议格式定义如下：

```

/**
 * define data_link_layer struct
 * based on PPP protocol
 */
typedef struct {
    unsigned char head_flag; //头部标志字段
    unsigned char addr;      //地址字段
    unsigned char ctrl;      //控制字段
    unsigned char protoc[2]; //协议
    unsigned char payload[1500]; //信息部分
    unsigned int fcs;        //帧校验序列
    unsigned char tail_flag; //尾部标志字段
} PPPFrame;

```

4.1.2 IPv4 数据报格式

```

/**
 * define internet_layer struct
 * based on IPv4 protocol
 */

typedef struct {
    unsigned char version; //版本号: 4bits
    unsigned char header_len; //头部长度: 4bits
    unsigned char distin_service; //区分服务: 8bits
    unsigned short total_len; //总长度: 16bits

    unsigned short identification; //标识: 16bits
    unsigned char flags; //标志: 3bits
    unsigned char frag_offset; //偏移: 13bits

    unsigned char ttl; //生存时间: 8bits
    unsigned char protocol; //协议: 8bits
    unsigned short checksum; //首部检验和: 16bits

    unsigned char src_ip[4]; //源IP地址: 32bits
    unsigned char dst_ip[4]; //目的IP地址: 32bits
} IPv4_header;

```

4.1.3 UDP 数据报格式

```

/**
 * define transport_layer struct
 * based on UDP protocol
 */

typedef struct {
    unsigned short src_port; //源端口: 2字节
    unsigned short dst_port; //目的端口: 2字节
    unsigned short length; //数据长度: 2字节
    unsigned short checksum; //校验和: 2字节
}udp_header;

```

4.2 校验和

4.2.1 CRC 校验和

CRC 校验和用于计算 PPP 帧格式中的 FCS 部分。

该函数接受两个参数，`data` 为待计算的字节数组指针，`data_len` 为字节数组的长度。函数返回计算得到的 CRC16 校验值。

首先初始化校验码 `crc` 为 `0xFFFF`。然后通过遍历字节数组 `data`，将每个字节与校验码进行异或操作。接下来，利用 CRC 计算方法，对校验码进行 8 位二进制运算。如果校验码的最高位为 1，则将校验码左移一位后与 `0x1021` 进行异或操作；如果最高位为 0，则直接左移一位。最后，返回计算得到的校验码。

```
//计算CRC16校验值
unsigned int crc16(unsigned char* data, unsigned int data_len){
    unsigned int crc = 0xFFFF;
    int i;
    for(i = 0; i < data_len; i++){
        crc ^= data[i] << 8;
        int j;
        for(j = 0; j < 8; j++){
            if(crc & 0x8000){
                crc = (crc << 1) ^ 0x1021;
            }
            else {
                crc << 1;
            }
        }
    }
    return crc;
}
```

4.2.2 二进制反码求和再求反码

```
unsigned short Checksum(unsigned char* buffer, unsigned short size){
    unsigned int sum = 0;
    unsigned short cksum;
    unsigned char lower, higher;
    unsigned short temp;
    if(size%2 == 1) {
        higher= buffer[size-1];
        lower = 0x00;
        temp = higher;
        temp = (temp << 8) + lower;
        sum += temp;
        size--;
    }
    while(size > 1) {
        lower = buffer[size-1];
        higher = buffer[size - 2];
        temp = higher;
        temp = (temp << 8) + lower;
        sum += temp;
        size -= 2;
    }
}
```

```

cksum = (sum >> 16) + (sum & 0xffff);
cksum = ~cksum;
cksum = (cksum>>8) + (cksum << 8);
return cksum;

```

该校验方法用于 IP 数据报和 UDP 数据报的检验。

4.3 封装

4.3.1 主函数

首先定义或初始化一些必要的信息，如 ppp 帧首部，IPv4 数据报头部和 udp 用户数据报协议头部。然后定义 input_data 用于接收用户输入数据，buffer 用于在各个层中流动的数据。自定义发送方的 ip 地址和端口号。

```

int main(){
    //define struct
    PPPFrame ppp_frame;
    IPv4_header ipv4;
    udp_header udp;

    // transport process
    char input_data[100];           //用户输入的数据
    unsigned char buffer[2000];    //在五层体系中流动的数据
    unsigned char src_ip[4] = {192,168,1,100}; //源IP地址
    unsigned char dst_ip[4] = {192,168,1,200}; //目的IP地址
    unsigned short src_port = 8090; //源端口
    unsigned short dst_port = 8091; //目的端口

```

获取到用户希望传输的数据，并将 input_data 拷贝到 buffer 数组中。

```

    unsigned short dst_port = 8091; //目的端口
    printf("begining.....\nplease input the data you will transfer: \n");
    //获取用户输入
    fgets(input_data,sizeof(input_data),stdin);
    unsigned short input_data_len = strlen(input_data) - 1;

    printf("您输入的数据为: %s",input_data);
    printf("数据长度为: %u\n",input_data_len);

    memcpy(&buffer[PPP_BACK_CONST], input_data, input_data_len);

```

- 进行 UDP 数据报封装

```

printf("***** Transport Layer is Packing *****\n");
transport_layer_init(&buffer[0], &udp, src_port, dst_port, src_ip, dst_ip, input_data_len);
printf_transport_layer(&udp);

```

- 进行 IP 数据报封装

```

printf("***** Network Layer is Packing *****\n");
network_layer_init(&buffer[0], &ipv4, src_ip, dst_ip, input_data_len+8);
printf_network_layer(&ipv4);

```


- 进行 PPP 帧格式封装

```
printf("***** Data Link Layer is Packing *****\n");
data_link_layer_init(&buffer[0],&ppp_frame,input_data_len+8+20);
printf_data_link_layer(&ppp_frame);
```

- 将封装后的数据格式打印出来

```
int i;
for (i = input_data_len+8+20+8 - 1; i >= 0; i--) {
    if ((input_data_len+8+20+8 - 1 - i) % 16 == 0) {
        printf("\n");
    }
    printf(" %02x |", buffer[i]);
}
```

- 将封装后的数据格式用文件存储起来，方便解封装

```
FILE *output_file;
output_file = fopen("./testmsg.txt","wb+");
if(output_file==NULL) {
    printf("File open error: can't open the file!\n");
}
fwrite(buffer,sizeof(char),input_data_len+8+20+8,output_file);
fclose(output_file);
printf("File writes done!\n");
return 0;
```

4.3.2 UDP 数据报格式封装

首先获取到用户信息，具体信息见代码注释。

```
header->src_port = src_port; //源端口
header->dst_port = dst_port; //目的端口
header->length = len_of_data + 8; //数据长度
header->checksum = 0; //校验和
```

接着进行首部校验。添加 12 字节的伪首部(源 ip 地址，目的 ip 地址，长度等信息)加上 udp 8 字节的首部进行计算。

```
unsigned char pseudo_header[12+65515]; //保存伪首部，用作计算校验和
unsigned short udp_len = len_of_data + 8;
memcpy(&pseudo_header[0], src_ip, 4); //源地址
memcpy(&pseudo_header[4], dst_ip, 4); //目的地址
pseudo_header[8] = 0x00;
pseudo_header[9] = 17;
memcpy(&pseudo_header[10],&udp_len, 2);

memcpy(&pseudo_header[12],&header->src_port, 2); //源端口
memcpy(&pseudo_header[14], &header->dst_port,2); //目的端口
memcpy(&pseudo_header[16],&header->length, 2); //长度
memcpy(&pseudo_header[18], &header->checksum, 2); //校验和

memcpy(&pseudo_header[20],&buffer[PPP_BACK_CONST], len_of_data); //UDP的长度
```

最后将得到的数据信息按照数据报格式依次存入到 buffer 中提供给网络层。

```
memcpy(&buffer[PPP_BACK_CONST+len_of_data], &header->checksum, 2);
memcpy(&buffer[PPP_BACK_CONST+len_of_data+2], &header->length, 2);
memcpy(&buffer[PPP_BACK_CONST+len_of_data+4], &header->dst_port, 2);
memcpy(&buffer[PPP_BACK_CONST+len_of_data+6], &header->src_port, 2);
```

4.3.3 IPv4 数据报格式封装

首先获取基本的首部信息：

```
void network_layer_init(unsigned char* buffer, IPv4_header* header, unsigned
    header->version = 4;          //版本号为4
    header->header_len = 5;        //首部长度为5个字节
    header->distin_service = 0;    //区分服务为0
    header->total_len = 20 + len_of_data; //总长度为20加上数据部分
    header->identification = 0;   //标识为0
    header->flags = 0;             //标志为0
    header->frag_offset = 0;       //设置分片偏移字段为0
    header->ttl = 64;               //生存时间为64
    header->protocol = 17;          //上层使用的是UDP协议
    header->checksum = 0;           //头部校验和为0
    int i;
    for(i=0;i<=3;i++){
        header->src_ip[i] = src_ip[i];
        header->dst_ip[i] = dst_ip[i];
    }
}
```

然后对 ip 数据报进行首部校验。

```
//将版本和首部长度拼接成一个字节
unsigned char version_and_header_len = (header->version << 4) | header->header_len;
//将标志和片偏移拼接成两个字节
unsigned short flags_and_frag_offset = (header->flags << 13) | header->frag_offset;
//存放校验和
unsigned char checksum_all[20];

memcpy(&checksum_all[0], &version_and_header_len, 1); //复制版本和首部长度
memcpy(&checksum_all[1], &header->distin_service, 1); //复制区分服务
memcpy(&checksum_all[2], &header->total_len, 2);      //复制总长度
memcpy(&checksum_all[4], &header->identification, 2); //复制标识
memcpy(&checksum_all[6], &flags_and_frag_offset, 2);  //复制标志和片偏移
memcpy(&checksum_all[8], &header->ttl, 1);             //复制生存时间
memcpy(&checksum_all[9], &header->protocol, 1);        //复制协议
memcpy(&checksum_all[10], &header->checksum, 2);       //复制校验和
memcpy(&checksum_all[12], src_ip, 4);                  //复制源IP地址
memcpy(&checksum_all[16], dst_ip, 4);                  //复制目的IP地址
//计算头部校验和
header->checksum = Checksum(checksum_all, 20);
```

然后将数据写入 buffer 数组传递给数据链路层。

```

memcpy(&buffer[PPP_BACK_CONST + len_of_data],dst_ip,4); //目的ip
memcpy(&buffer[PPP_BACK_CONST + len_of_data + 4],src_ip,4); //原始ip
memcpy(&buffer[PPP_BACK_CONST + len_of_data + 8],&header->checksum,2); //校验和
memcpy(&buffer[PPP_BACK_CONST + len_of_data + 10],&header->protocol,1); //协议
memcpy(&buffer[PPP_BACK_CONST + len_of_data + 11],&header->ttl,1); //生存时间
memcpy(&buffer[PPP_BACK_CONST + len_of_data + 12],&header->flags_and_frag_offset,2); //标志和片偏移
memcpy(&buffer[PPP_BACK_CONST + len_of_data + 14],&header->identification,2); //标识
memcpy(&buffer[PPP_BACK_CONST + len_of_data + 16],&header->total_len,2); //总长度
memcpy(&buffer[PPP_BACK_CONST + len_of_data + 18],&header->distin_service,1); //区分服务
memcpy(&buffer[PPP_BACK_CONST + len_of_data + 19],&header->version_and_header_len,1); //版本和首部长度

```

4.3.4 PPP 帧格式封装

整体流程类同以上，具体信息如封装的每个部分的信息可见注释。

```

void data_link_layer_init(unsigned char* buffer, PPPFrame* frame, unsigned short len_of_data){
    const unsigned char F = 0x7E; //标志字段
    const unsigned char A = 0xFF; //地址字段
    const unsigned char C = 0x03; //控制字段
    frame->head_flag = frame->tail_flag = F;
    frame->addr = A;
    frame->ctrl = C;
    frame->protoc[0] = 0x00; frame->protoc[1] = 0x21; //协议
    frame->fcs = 0;
    //下面是用于计算帧检验序列：cal_fcs_copy=协议(2字节)+数据部分：计算fcs，在解封判断时应只包含这两部分
    unsigned char cal_fcs_copy[2+65532];
    memcpy(&cal_fcs_copy[0],&frame->addr,1);
    memcpy(&cal_fcs_copy[1],&frame->ctrl,1);
    memcpy(&cal_fcs_copy[2],&frame->protoc,2);
    memcpy(&cal_fcs_copy[4],&buffer[PPP_BACK_CONST],len_of_data);
    frame->fcs = crc16(cal_fcs_copy,4+len_of_data);
    // buffer[0] = frame->head_flag; //头部字段
    // buffer[1] = frame->addr; //地址字段
    // buffer[2] = frame->ctrl; //控制字段
    buffer[0] = frame->tail_flag;
    memcpy(&buffer[1],&frame->fcs,2);

    memcpy(&buffer[PPP_BACK_CONST + len_of_data], &frame->protoc,2); //协议
    memcpy(&buffer[PPP_BACK_CONST + len_of_data + 2], &frame->ctrl, 1);
    memcpy(&buffer[PPP_BACK_CONST + len_of_data + 3], &frame->addr, 1);
    memcpy(&buffer[PPP_BACK_CONST + len_of_data + 4], &frame->head_flag, 1);
}

```

4.4 解封装

4.4.1 主函数

首先读取 testmsg.txt 文件中的数据：

```

FILE *receive_file = fopen("testmsg.txt","r");
if(receive_file == NULL){
    printf("open file error\n");
    return 0;
}
unsigned char buffer[65535]; //存放接受的数据
int receive_data_len = 0;
while(!feof(receive_file)){
    buffer[receive_data_len++] = getc(receive_file);
}
fclose(receive_file);
receive_data_len--;

```


接着将文件中的数据打印出来，与封装过程的数据进行对比是否一致。

```
printf("***** Full Payload Received. *****");
int i;
for(i = receive_data_len-1; i >= 0; i--){
    if((receive_data_len - 1 - i) % 16 == 0){
        printf("\n");
    }
    printf("%02x |", buffer[i]);
}
printf("\n\n");
```

最后按照 PPP->IP->UDP 的顺序对读取的数据进行解封装：

```
data_link_layer_receive(&buffer[0],&ppp_frame,receive_data_len);
network_layer_receive(&buffer[0],&ipv4,receive_data_len - 5);
transport_layer_receiver(&buffer[0],&udp,ipv4.src_ip, ipv4.dst_ip,receive_data_len - 25);
```

4.4.2 PPP 帧格式的解封装

首先将 buffer 中的已封装的 PPP 首部的数据提取出来。

```
memcpy(&frame->head_flag, &buffer[len_of_data-1],1);
memcpy(&frame->addr, &buffer[len_of_data-2],1);
memcpy(&frame->ctrl, &buffer[len_of_data-3],1);
memcpy(&frame->protoc, &buffer[len_of_data-5],2);
memcpy(&frame->tail_flag, &buffer[0],1);
memcpy(&frame->fcs, &buffer[1],2);
```

接着利用 FCS 计算校验和，并根据结果与原始 FCS 进行对比，若相同则通过检验，反之则没有通过检验。同时打印收到的 PPP 帧内的首部和尾部信息。

```
//计算FCS
unsigned char cal_fcs_copy[2+65532];
memcpy(&cal_fcs_copy[0],&frame->addr,1);
memcpy(&cal_fcs_copy[1],&frame->ctrl,1);
memcpy(&cal_fcs_copy[2],&frame->protoc,2);
memcpy(&cal_fcs_copy[4],&buffer[PPP_BACK_CONST],len_of_data - 5 - PPP_BACK_CONST);
//int i;

frame->fcs = crc16(cal_fcs_copy,len_of_data - PPP_BACK_CONST - 1);
unsigned char temp[3];
memcpy(&temp[1],&frame->fcs,2);

if(temp[1]==buffer[1] && temp[2] == buffer[2]){
    printf("CHECK PASS!\n");
}
else{
    printf("CHECK ERROR!\n");
}
printf_data_link_layer(frame);
```


4.4.3 IP 数据报的解封装

通过 PPP 解封装后，进行网络层的解封装。首先通过控制读取 buffer 数据的位置，将 IP 首部相关信息读取出来。

```
void network_layer_receive(unsigned char* buffer, IPv4_header* ipv4, unsigned short len_of_data){
    unsigned char version_and_headerlen = buffer[len_of_data - 1];
    ipv4->version = version_and_headerlen >> 4;    //版本
    ipv4->header_len = version_and_headerlen & 0x0F; //首部长度
    ipv4->distin_service = buffer[len_of_data - 2]; //区分服务
    memcpy(&ipv4->total_len, &buffer[len_of_data - 4],2); //总长度
    memcpy(&ipv4->identification, &buffer[len_of_data - 6],2); //标识

    unsigned short flags_and_frag_offset;
    memcpy(&flags_and_frag_offset, &buffer[len_of_data - 8],2);
    ipv4->flags = flags_and_frag_offset >> 13;    //标志
    ipv4->frag_offset = flags_and_frag_offset & 0x1FFF; //片偏移
    ipv4->ttl = buffer[len_of_data - 9]; //生存时间
    ipv4->protocol = buffer[len_of_data - 10]; //协议
```

接着对得到的数据做首部校验。若校验和为 0，则通过检验，反之则出现差错。同时打印 IPv4 数据报首部信息。

```
//检验首部
unsigned char checksum_all[20];
memcpy(&checksum_all[0],&version_and_headerlen,1); //复制版本和首部长度
memcpy(&checksum_all[1],&ipv4->distin_service,1); //复制区分服务
memcpy(&checksum_all[2],&ipv4->total_len,2); //复制总长度
memcpy(&checksum_all[4],&ipv4->identification,2); //复制标识
memcpy(&checksum_all[6],&flags_and_frag_offset,2); //复制标志和片偏移
memcpy(&checksum_all[8],&ipv4->ttl,1); //复制生存时间
memcpy(&checksum_all[9],&ipv4->protocol,1); //复制协议
memcpy(&checksum_all[10],&ipv4->checksum,2); //复制校验和
memcpy(&checksum_all[12], &ipv4->src_ip,4); //复制源IP地址
memcpy(&checksum_all[16], &ipv4->dst_ip,4); //复制目的IP地址

unsigned int checksum_cal = Checksum(checksum_all,20);
printf("checksum cal: %d\n",checksum_cal);
if(checksum_cal == 0){
    printf("checksum is right\n");
}
else{
    printf("checksum is wrong\n");
}
printf_network_layer(ipv4);
```

4.4.4 UDP 数据报的解封装

首先通过控制从 buffer 读取数据的位置将 UDP 首部信息读取出来。

```
void transport_layer_receiver(unsigned char* buffer, udp_header* udp, unsigned char src_ip[4], unsigned char dst_ip[4])
{
    memcpy(&udp->src_port, &buffer[len_of_data - 2], 2); //源端口
    memcpy(&udp->dst_port, &buffer[len_of_data - 4], 2); //目的端口
    memcpy(&udp->length, &buffer[len_of_data - 6], 2); //长度
    memcpy(&udp->checksum, &buffer[len_of_data - 8], 2); //校验和

    unsigned char pseudo_header[12+65510]; //保存伪首部, 用作计算校验和
    memcpy(&pseudo_header[0], src_ip, 4); //源地址
    memcpy(&pseudo_header[4], dst_ip, 4); //目的地址
    pseudo_header[8] = 0x00;
    pseudo_header[9] = 17;
    memcpy(&pseudo_header[10], &udp->length, 2);
    memcpy(&pseudo_header[12], &udp->src_port, 2);
    memcpy(&pseudo_header[14], &udp->dst_port, 2);
    memcpy(&pseudo_header[16], &udp->length, 2);
    memcpy(&pseudo_header[18], &udp->checksum, 2);
    memcpy(&pseudo_header[20], &buffer[PPP_BACK_CONST], len_of_data - 8 - PPP_BACK_CONST);
}
```

然后构造伪首部进行校验, 检查校验和是否为 0; 若为 0, 则通过; 反之则不通过。同时收到的 UDP 首部信息打印出来。

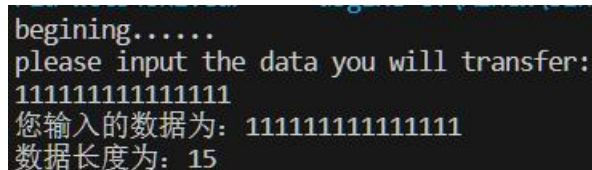
```
unsigned char pseudo_header[12+65510]; //保存伪首部, 用作计算校验和
memcpy(&pseudo_header[0], src_ip, 4); //源地址
memcpy(&pseudo_header[4], dst_ip, 4); //目的地址
pseudo_header[8] = 0x00;
pseudo_header[9] = 17;
memcpy(&pseudo_header[10], &udp->length, 2);
memcpy(&pseudo_header[12], &udp->src_port, 2);
memcpy(&pseudo_header[14], &udp->dst_port, 2);
memcpy(&pseudo_header[16], &udp->length, 2);
memcpy(&pseudo_header[18], &udp->checksum, 2);
memcpy(&pseudo_header[20], &buffer[PPP_BACK_CONST], len_of_data - 8 - PPP_BACK_CONST);

unsigned int checksum_cal = Checksum(pseudo_header, 12+udp->length);
printf("checksum cal: %d\n", checksum_cal);
if(checksum_cal == 0){
    printf("checksum is right\n");
}
else{
    printf("checksum is wrong\n");
}
printf_transport_layer(udp);
```

5. 运行结果

5.1 封装

- 数据输入（用户准备传输的数据）



```
begining.....
please input the data you will transfer:
111111111111111
您输入的数据为: 111111111111111
数据长度为: 15
```

- UDP 数据报封装结果

```
***** Transport Layer is Packing *****
-----This is transport layer(UDP)-----
*****SrcPort: 8090
*****DstPort: 8091
*****Length: 0023
*****Checksum: 55694
```

- IP 数据报封装结果

```
***** Network Layer is Packing *****
-----This is network layer(IPv4)-----
*****Version: 4
*****HeaderLen: 5
*****DestinService: 0
*****TotalLen: 43
*****Identification: 0
*****Flags: 0
*****FragOffset: 0
*****TTL: 64
*****Protocal: 17
*****Checksum: 28875
*****SrcIP: 192.168.1.100
*****DstIP: 192.168.1.200
```

- PPP 帧格式封装结果

```
***** Data Link Layer is Packing *****
-----This is data link layer(PPP)-----
*****Address: ff
*****Control: 03
*****Protocol: 0021
*****CRC: 4294926111
```

- 文件中存储的封装后的数据信息（使用十六进制输出）

```
7e | ff | 03 | 21 | 00 | 45 | 00 | 00 | 2b | 00 | 00 | 00 | 00 | 40 | 11 | 70 |
cb | 64 | 01 | a8 | c0 | c8 | 01 | a8 | c0 | 1f | 9a | 1f | 9b | 00 | 17 | d9 |
8e | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 |
5f | 1f | 7e | File writes done!
```

可以看到传输层、网络层和数据链路层首部固定信息均得到了正确输出，数据也成功写入了文件之中。

5.2 解封装

- 读取文件中的信息（获取到传输的信息）

```
***** Full Payload Received. *****
7e | ff | 03 | 21 | 00 | 45 | 00 | 00 | 2b | 00 | 00 | 00 | 00 | 40 | 11 | 70 |
cb | 64 | 01 | a8 | c0 | c8 | 01 | a8 | c0 | 1f | 9a | 1f | 9b | 00 | 17 | d9 |
8e | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 31 |
5f | 1f | 7e |
```

可以看到读入的数据和写入的数据完全一致。

- PPP 帧格式的解封装结果

```
CHECK PASS!  
-----This is data link layer(PPP)-----  
*****Address:  ff  
*****Control:  03  
*****Protocol: 0021  
*****CRC:      4294926111
```

可以看到数据链路层通过了检验，并且地址、控制、协议等字段也和封装过程的保持一致。

- IP 数据报解封装结果

```
checksum cal: 0  
checksum is right  
-----This is network layer(IPv4)-----  
*****Version:  4  
*****HeaderLen: 5  
*****DistinService:  0  
*****TotalLen:  43  
*****Identification: 0  
*****Flags:  0  
*****FragOffset:  0  
*****TTL:  64  
*****Protocal:  17  
*****checksum:  28875  
*****SrcIP:  192.168.1.100  
*****DstIP:  192.168.1.200
```

可以看到网络层的首部校验和为 0，说明通过了校验。同时版本、协议、IP 地址等也得到了正确的解析和输出。

- UDP 数据报解封装结果

```
checksum cal: 0  
checksum is right  
-----This is transport layer(UDP)-----  
*****SrcPort:  8090  
*****DstPort:  8091  
*****Length:  0023  
*****Checksum: 55694
```

可以看到传输层的首部校验和为 0，说明通过了校验。同时源端口、目的端口等也得到了正确的解析和输出。

通过对封装的信息和解封装的信息的对比可以看出，已实现了对数据链路层，网络层，运输层的协议的数据单元封装和解封装，并得到了正确的结果输出。