

计算机网络平时作业

数据链路层与网络层的封装与解封装



学 生：

学 号：

任课教师： 李双庆

专 业： 计算机科学与技术

班 级： 卓越 1 班

目录

1. 工作分工	3
2. 处理逻辑描述	4
2.1 数据链路层	4
2.1.1 网络帧的封装:	4
2.1.2 网络帧的解封装:	4
2.2 web 层	4
2.2.1 IP 数据包的封装:	4
2.2.2 IP 数据包的解封装:	5
3.封装与解封装关键代码	6
3.1 数据链路层	6
3.1.1 封装帧	6
3.1.2 解封装帧	7
3.2web 层	8
3.2.1 封装 IP 数据报	8
3.2.2 解封装 IP 数据报	9
4 运行结果	11
4.1 结果说明	11
5.附录	13

1. 工作分工

- : 1.数据链路层解封装实现代码
 - 2.网络层解封装实现代码
 - 3. CRC32 校验计算方法头文件编写
 - 4.报告编写以及排版美化
- : 1.数据链路层封装实现代码
 - 2.网络层封装实现代码
 - 3.数据类型头文件编写
 - 4.报告编写以及排版美化

2. 处理逻辑描述

2.1 数据链路层

2.1.1 网络帧的封装：

首先创建一个 `FrameHeader` 结构体，并填写目的 MAC 地址、源 MAC 地址和类型字段。将 `FrameHeader` 和 IP 数据包封装后的内容一起复制到新的数组中，形成封装的网络帧。最后返回封装后的网络帧的长度。

2.1.2 网络帧的解封装：

首先从网络帧中提取 `FrameHeader` 信息，包括目的 MAC、源 MAC 和类型，并查找帧定界符，并验证数据长度。然后检查帧的校验和，若校验和不正确则丢弃。之后提取实际的帧数据部分，并解析其中的 MAC 地址和类型信息，最后返回解封装后的数据。

2.2 web 层

2.2.1 IP 数据包的封装：

首先，创建一个 `Ipv4Header` 结构体，并初始化版本、首部长度、服务类型、总长度、标识、片偏移、TTL、协议等字段。计算 IP 校验和。然后，创建一个 `ICMPHeader` 结构体，并初始化类型、代码、校验和、ID、序列号。计算 ICMP 校验和。将这些头部信息和实际的数据一起复制到一个新的数组中，形成封装后的 IP 数据包。最后返回封装后的数据包的长度。

2.2.2 IP 数据包的解封装：

首先从网络帧中提取 Ipv4Header 和 ICMPHeader 信息。对 IP 和 ICMP 的校验和进行检查，如果校验和不正确，则丢弃该数据包。然后提取实际的数据部分，并从 IP 数据包中提取出原始的应用数据。最后提供相关的 IP 和 ICMP 信息，如源 IP、目的 IP、ICMP 类型等。



3.封装与解封装关键代码

3.1 数据链路层

3.1.1 封装帧

```

1.  int enpackFrame(unit_u8* src_data, int data_len, unit_u8* frame) {
2.      if (data_len == 0) return 0;
3.
4.      printf("===== Frame Encapsulation =====\n");
5.
6.      // Minimum frame check
7.      unit_u8 data[MAXSIZE];
8.      memcpy(data, src_data, data_len);
9.      if (data_len < MINSIZE) {
10.         memset(data + data_len, 0x00, MINSIZE - data_len);
11.         data_len = MINSIZE;
12.     }
13.
14.     printf("Minimum Frame Size Check: Data Length = %d\n", data_len);
15.
16.     FrameHeader fHeader;
17.     fHeader.leader = 0xabaaaaaaaaaaaa; // unit_u64 in little-endian format
18.
19.     // Get MAC addresses and type field
20.     getMACAddress(fHeader.desMAC, "Enter destination MAC address: ");
21.     getMACAddress(fHeader.srcMAC, "Enter source MAC address: ");
22.     getTypeField(fHeader.type);
23.
24.     // Encapsulation
25.     memcpy(frame, &fHeader, SizeFrameHeader);
26.     memcpy(frame + SizeFrameHeader, data, data_len);
27.
28.     int frame_len = SizeFrameHeader + data_len;
29.     return frame_len;
30. }

```

3.1.2 解封装帧

```

1.  int unpackFrame(unit_u8 *frame, int frame_len, unit_u8 *frame_data) {
2.      if (frame_len == 0) return 0;
3.
4.      printf("===== Frame Decapsulation =====\n");
5.
6.      FrameHeader fHeader;
7.      memcpy(&fHeader, frame, SizeFrameHeader);
8.
9.      // Extract frame boundary and data length
10.     int p;
11.     for (p = 0; p < frame_len; p++) {
12.         if (frame[p] == 0xab) {
13.             p++;
14.             break;
15.         }
16.     }
17.
18.     int data_len = frame_len - p - 6 - 6 - 2;
19.
20.     memcpy(frame_data, frame + SizeFrameHeader, data_len);
21.
22.     // Output extracted information
23.     printf("Destination MAC: ");
24.     display(fHeader.desMAC, 6, 1);
25.     printf("Source MAC: ");
26.     display(fHeader.srcMAC, 6, 1);
27.     printf("Type: ");
28.     display(fHeader.type, 2, 1);
29.
30.     return data_len;
31. }

```

3.2web 层

3.2.1 封装 IP 数据报

```
1.  int enpackIP(unit_u8 *data, int data_len, unit_u8 *datagram) {
2.      if (data_len == 0) return 0;
3.
4.      printf("===== IP Encapsulation =====\n");
5.      static u_int sequenceNum = 0;
6.      sequenceNum += 1;
7.
8.      Ipv4Header ipheader;
9.      ipheader.version_ihl = 0x45; // IPv4, IHL = 5
10.     ipheader.diffserve = 0x00;
11.     ipheader.tot_len = 20 + 8 + data_len; // 20 for IP header, 8 for ICMP header, and data Length
12.     ipheader.id = 0x1000;
13.     ipheader.frag_off = 0x0000;
14.     ipheader.ttl = 128;
15.     ipheader.protocol = 0x01; // ICMP
16.     ipheader.check = 0;
17.     ipheader.saddr = inet_addr("192.168.43.34"); // Source IP
18.     ipheader.daddr = htonl(inet_addr("220.181.38.251")); // Destination IP (Baidu)
19.
20.     // Calculate checksum
21.     ipheader.check = CalChecksum((unit_u8 *)&ipheader, SizeIpv4Header, 0);
22.     printf("IP Header Checksum: %x\n", ipheader.check);
23.     ipheader.check = htons((unit_u16)ipheader.check);
24.
25.     ICMPHeader icmpheader;
26.     icmpheader.type = 0x08;
27.     icmpheader.code = 0x0;
28.     icmpheader.check = 0;
29.     icmpheader.id = 0x0001;
30.     icmpheader.sequence = sequenceNum;
31.
32.     unit_u8 checkdata[MAXSIZE];
33.     memcpy(checkdata, (unit_u8 *)&icmpheader, SizeICMPHeader);
34.     memcpy(checkdata + SizeICMPHeader, data, data_len);
35.     icmpheader.check = CalChecksum(checkdata, SizeICMPHeader + data_len, 0);
36.     printf("ICMP Header Checksum: %x\n", icmpheader.check);
```



```

37.     icmpheader.check = htons icmpheader.check);
38.
39.     // Construct datagram
40.     memcpy(datagram, &ipheader, SizeIPv4Header);
41.     memcpy(datagram + SizeIPv4Header, &icmpheader, SizeICMPHeader);
42.     memcpy(datagram + SizeIPv4Header + SizeICMPHeader, data, data_len);
43.
44.     int datagram_len = SizeIPv4Header + SizeICMPHeader + data_len;
45.     return datagram_len;
46. }

```

3.2.2 解封装 IP 数据报

```

1.  int unpackIP(unit_u8 *frame_data, int frame_data_len, unit_u8 *datagram_data) {
2.      if (frame_data_len == 0) return 0;
3.
4.      printf("===== IP Decapsulation =====\n");
5.
6.      IPv4Header ipheader;
7.      ICMPHeader icmpheader;
8.      int datagram_data_len = frame_data_len - SizeIPv4Header - SizeICMPHeader;
9.
10.     memcpy(&ipheader, frame_data, SizeIPv4Header);
11.     memcpy(&icmpheader, frame_data + SizeIPv4Header, SizeICMPHeader);
12.     memcpy(datagram_data, frame_data + SizeIPv4Header + SizeICMPHeader, datagram_data_len);
13.
14.     // Check IP checksum
15.     ipheader.check = CalChecksum((unit_u8 *)&ipheader, SizeIPv4Header, 0);
16.     if (ipheader.check != 0) {
17.         printf("ERROR: IP checksum invalid!\n");
18.         return 0;
19.     }
20.
21.     // Check ICMP checksum
22.     unit_u8 checkdata[MAXSIZE];
23.     memcpy(checkdata, (unit_u8 *)&icmpheader, SizeICMPHeader);
24.     memcpy(checkdata + SizeICMPHeader, datagram_data, datagram_data_len);
25.     icmpheader.check = CalChecksum(checkdata, SizeICMPHeader + datagram_data_len, 0);
26.     if (icmpheader.check != 0) {
27.         printf("ERROR: ICMP checksum invalid!\n");

```

```
28.         return 0;
29.     }
30.
31.     // Output results
32.     struct in_addr saddr, daddr;
33.     memcpy(&saddr, &ipheader.saddr, 4);
34.     memcpy(&daddr, &ipheader.daddr, 4);
35.     printf("Source IP: %s\n", inet_ntoa(saddr));
36.     printf("Destination IP: %s\n", inet_ntoa(daddr));
37.
38.     printf("ICMP Type: %d\n", icmpheader.type);
39.     printf("ICMP Code: %d\n", icmpheader.code);
40.
41.     // Output Data
42.     printf("Data: %s\n", datagram_data);
43.
44.     return datagram_data_len;
45. }
```

4 运行结果

4.1 结果说明

```

Data length: 24
===== IP Encapsulation =====
IP Header Checksum: 6a1f
ICMP Header Checksum: 45ee
Datagram length: 52
45 00 34 00 00 10 00 00 01 6a 1f c0 a8 2b 22 fb 26 b5 dc 08 00 45 ee 01 00 01 00 49 20 6c 6f 76 65 20 43 6f 6d 70 75 74 65 72 20 4e 65 74 77 6f 72 6b 21
===== Frame Encapsulation =====
Minimum Frame Size Check: Data Length = 52
Enter destination MAC address: 00-1A-2B-3C-4D-5E
Enter source MAC address: F1-23-45-67-89-AB
Enter type field (two bytes): 0800
Frame length: 74
aa aa aa aa aa ab 00 ff 0a fe 0b fd f1 fe 03 fc 05 fa 08 00 45 00 34 00 00 10 00 00 01 6a 1f c0 a8 2b 22 fb 26 b5 dc 08 00 45 ee 01 00 01 00 49 20 6c 6f 76 65 20 43 6f 6d 70 75 74 65 72 20 4e 65 74 77 6f 72 6b 21
===== Frame Decapsulation =====
Destination MAC: 00:ff:0a:fe:0b:fd
Source MAC: f1:fe:03:fc:05:fa
Type: 08:00
Frame data length: 52
45 00 34 00 00 10 00 00 01 6a 1f c0 a8 2b 22 fb 26 b5 dc 08 00 45 ee 01 00 01 00 49 20 6c 6f 76 65 20 43 6f 6d 70 75 74 65 72 20 4e 65 74 77 6f 72 6b 21
===== IP Decapsulation =====
Source IP: 192.168.43.34
Destination IP: 251.38.181.220
ICMP Type: 8
ICMP Code: 0
Data: I love Computer Network!\
Datagram data length: 24
49 20 6c 6f 76 65 20 43 6f 6d 70 75 74 65 72 20 4e 65 74 77 6f 72 6b 21

```

1. 数据初始化

原始数据为字符串“ I love Computer Network!”，长度为 24 字节。通过 IP 数据包和以太网帧的封装处理，数据会经过多层协议头部的附加。

2. IP 数据包封装

原始数据被封装为 IP 数据包，其中包括：

- **IP 头部：**包含了版本号、头部长度的总长度（52 字节，包括 IP 头部、ICMP 头部和数据）、源地址 192.168.43.34 和目标地址 251.38.181.220。
- **ICMP 头部：**ICMP 类型为 8（回显请求），包含校验和 45ee，保证 ICMP 报文完整性。
- 数据部分为原始字符串的字节形式。

最终生成的 IP 数据包长度为 52 字节，通过校验和验证，头部完整性无误。

3. 以太网帧封装

IP 数据包进一步封装为以太网帧，帧结构包括：

- 帧头部：前导符用于同步。

目标和源 MAC 地址由用户输入（分别为 00-1A-2B-3C-4D-5E 和 F1-23-45-67-89-AB）。

类型字段为 0800，表明数据部分为 IP 数据包。

- 数据部分：数据部分为 IP 数据包内容。

封装后的以太网帧总长度为 74 字节（包括帧头和数据）。

4. 解封装流程

以太网帧解封装：

- 提取帧数据中的目标 MAC 地址、源 MAC 地址和类型字段，验证类型字段为 0800，表明包含的是 IP 数据包。

- 数据部分正确提取出封装时的 IP 数据包，长度为 52 字节。

IP 数据包解封装：

- 提取 IP 头部中的源 IP 地址 192.168.43.34 和目标 IP 地址 251.38.181.220。
- 验证 IP 校验和正确，保证头部完整性。
- 提取 ICMP 报文头，验证 ICMP 校验和正确。
- 数据部分解码为原始字符串 I love Computer Network!。

可以看到，我们 IP 封装和帧封装以及 IP 解封装、帧解封装的过程运行结果，我们的原数据为"I love Computer Network!"最后成功被解析。每一部分都分别展示了该部分数据长度、数据报或者帧长度、MAC 地址、头部校验和等信息。图中标红部分数据被成功解析。

5.附录

5.1 CRC.h

```

1.  #pragma once
2.  #ifndef datatype_h_
3.  #define datatype_h_
4.  #include "datatype.h"
5.  #include <string.h>
6.  #endif
7.  const unit_u32 POLY = 0x04c11db7; // 32 bits 生成多项式
8.
9.  unit_u32 CalChecksum(unit_u8* srcdata,int dataLen,unit_u32 checkSum){
10.     // 需要保证 dataLen 为 2 的整数倍
11.     unit_u8 data[1500];
12.     memcpy(data, srcdata, dataLen);
13.     if (dataLen%2){
14.         data[dataLen] = 0x00;
15.         dataLen += 1;
16.     }
17.
18.     unit_u32 num;
19.     int i;
20.     for (i = 0; i <= dataLen-2; i += 2){ // 一次处理 2 字节
21.         num = (data[i] << 8) + data[i + 1]; // 组合成 2 字节
22.         checkSum += num;
23.         checkSum = (checkSum & 0xffff) + (checkSum >> 16);
24.     }
25.     checkSum = (~checkSum) & 0xffff;
26.     return checkSum;
27. }
28.
29. // 位翻转函数
30. unit_u64 Reflect(unit_u64 ref,unit_u8 ch)
31. {
32.     int i;
33.     unit_u64 value = 0;

```

```

34.     for( i = 1; i < ( ch + 1 ); i++ )
35.     {
36.         if( ref & 1 )
37.             value |= 1 << ( ch - i );
38.         ref >>= 1;
39.     }
40.     return value;
41. }
42.
43. // 生成CRC32 普通表 , 第二项是04C11DB7
44. void gen_direct_table(unit_u32 *table)
45. {
46.     unit_u32 gx = 0x04c11db7;
47.     unsigned long i32, j32;
48.     unsigned long nData32;
49.     unsigned long nAccum32;
50.     for ( i32 = 0; i32 < 256; i32++ )
51.     {
52.         nData32 = ( unsigned long )( i32 << 24 );
53.         nAccum32 = 0;
54.         for ( j32 = 0; j32 < 8; j32++ )
55.         {
56.             if ( ( nData32 ^ nAccum32 ) & 0x80000000 )
57.                 nAccum32 = ( nAccum32 << 1 ) ^ gx;
58.             else
59.                 nAccum32 <<= 1;
60.             nData32 <<= 1;
61.         }
62.         table[i32] = nAccum32;
63.     }
64. }
65.
66. // 生成CRC32 翻转表 第二项是77073096
67. void gen_normal_table(unit_u32 *table)
68. {
69.     unit_u32 gx = 0x04c11db7;
70.     unit_u32 temp,crc;
71.     for(int i = 0; i <= 0xFF; i++)
72.     {
73.         temp=Reflect(i, 8);

```

```

74.     table[i]= temp<< 24;
75.     for (int j = 0; j < 8; j++)
76.     {
77.         unsigned long int t1,t2;
78.         unsigned long int flag=table[i]&0x80000000;
79.         t1=(table[i] << 1);
80.         if(flag==0)
81.             t2=0;
82.         else
83.             t2=gx;
84.         table[i] =t1^t2 ;
85.     }
86.     crc=table[i];
87.     table[i] = Reflect(table[i], 32);
88. }
89. }
90.
91. unit_u32 crc32_bit(unit_u8 *ptr, unit_u32 len)
92. {
93.     unit_u32 gx = 0x04c11db7;
94.     unit_u8 i;
95.     unit_u32 crc = 0xffffffff;
96.     while( len-- )
97.     {
98.         for( i = 1; i != 0; i <= 1 )
99.         {
100.             if( ( crc & 0x80000000 ) != 0 )
101.             {
102.                 crc <= 1;
103.                 crc ^= gx;
104.             }
105.             else
106.                 crc <= 1;
107.             if( ( *ptr & i ) != 0 )
108.                 crc ^= gx;
109.         }
110.         ptr++;
111.     }
112.     return ( Reflect(crc,32) ^ 0xffffffff );
113. }
114.

```

```

115. unit_u32 DIRECT_TABLE_CRC(unit_u8 *ptr, int len, unit_u32 * table)
116. {
117.     unit_u32 crc = 0xffffffff;
118.     unit_u8 *p= ptr;
119.     int i;
120.     for ( i = 0; i < len; i++ )
121.         crc = ( crc << 8 ) ^ table[( crc >> 24 ) ^ (unit_u8)Reflect((*p+i),
            8)];
122.     return ~(unit_u32)Reflect(crc, 32) ;
123. }
124.
125. unit_u32 Reverse_Table_CRC(unit_u8 *data, int len, unit_u32 *table)
126. {
127.     unit_u32 crc = 0xffffffff;
128.     unit_u8 *p = data;
129.     int i;
130.     for(i=0; i <len; i++)
131.         crc = table[( crc ^ ( *(p+i)) ) & 0xff] ^ (crc >> 8);
132.     return ~crc ;
133. }

```

5.2 datatype.h

```

1.  typedef      char      unit_8; // 1byte, 8bit
2.  typedef      short     unit_16; // 2bytes, 16bit
3.  typedef      int       unit_32; // 4byte, 32bit
4.  typedef      long long  unit_64; // 8bit, 64bit
5.  typedef unsigned char   unit_u8; // 1byte, 8bit
6.  typedef unsigned short  unit_u16; // 2bytes, 16bit
7.  typedef unsigned int    unit_u32; // 4byte, 32bit
8.  typedef unsigned long long unit_u64; // 8bit, 64bit
9.
10. typedef struct FrameHeader FrameHeader;
11. typedef struct Ipv4Header Ipv4Header;
12. typedef struct ICMPHeader ICMPHeader;
13.
14. const int SizeFrameHeader = 22;
15. const int SizeIpv4Header = 20;

```



```

16. const int SizeICMPHeader = 8;
17.
18. struct FrameHeader {
19.     unit_u64 leader; // 0xffffffffffffffab
20.     unit_u8 desMAC[6]; // 6bytes, 目的地址
21.     unit_u8 srcMAC[6]; // 6bytes, 源地址
22.     unit_u8 type[2]; // 2bytes, 类型
23. };
24.
25. struct Ipv4Header{
26.     unit_u8 version_ihl; // 1byte 4bits 版本, ipv4 = 0x4 和 4bits 首部长度, 最短
        是5
27.     unit_u8 diffserve; // 1byte 区分服务
28.     unit_u16 tot_len; // 2bytes 总长度
29.     unit_u16 id; // 2bytes 标识
30.     unit_u16 frag_off; // 3bits [空 DF MF] + 13bits 片位移 = 16bits
31.     unit_u8 ttl; // 1byte 生存时间
32.     unit_u8 protocol; // 1byte 协议
33.     unit_u16 check; // 2bytes 首部校验核
34.     unit_u32 saddr; // 4bytes 源地址
35.     unit_u32 daddr; // 4bytes 目的地址
36. };
37.
38. struct ICMPHeader{ // 2 word, 8 字节
39.     unit_u8 type; // 1byte
40.     unit_u8 code; // 1byte
41.     unit_u16 check; // 2byte
42.     unit_u16 id; // 2byte
43.     unit_u16 sequence; // 2byte
44. };

```

5.3 FrameToIPdatagram.c

```

1. #pragma comment(lib, "ws2_32.lib")
2. #include <stdlib.h>
3. #include <stdio.h>
4. #include <winsock2.h>
5. #include <windows.h>
6. #include "CRC.h"
7.

```

```

8.  #ifndef datatype_h_
9.  #define datatype_h_
10. #include "datatype.h"
11. #include <string.h>
12. #endif
13.
14. #define SendToNet 0
15. #define MINSIZE 46
16. #define MAXSIZE 1500
17. #define MFS 1526 // MaxFrameSize
18.
19. void display(unit_u8 *data, int data_len, int type);
20. int enpackFrame(unit_u8* data, int data_len, unit_u8* frame);
21. int unpackFrame(unit_u8 *frame, int frame_len, unit_u8 *frame_data);
22. int enpackIP(unit_u8 *data, int data_len, unit_u8 *datagram);
23. int unpackIP(unit_u8 *frame_data, int frame_data_len, unit_u8 *datagram_data)
    ;
24. void getMACAddress(unit_u8 *mac, const char *message);
25. void getTypeField(unit_u8 *type);
26.
27. unit_u32 Reverse_Table[256];
28.
29. int main(){
30.     gen_normal_table(Reverse_Table);
31.     int data_len, datagram_len, frame_len, frame_data_len, datagram_data_len;
32.
33.     unit_u8 data[] = "I love Computer Network!";
34.     unit_u8 datagram[MAXSIZE];
35.     unit_u8 frame[MFS];
36.     unit_u8 frame_data[MAXSIZE];
37.     unit_u8 datagram_data[MAXSIZE];
38.
39.     data_len = strlen(data); // exclude the null terminator
40.     printf("Data length: %d\n", data_len);
41.
42.     // Encapsulate IP
43.     datagram_len = enpackIP(data, data_len, datagram);
44.     printf("Datagram length: %d\n", datagram_len);
45.     display(datagram, datagram_len, 0);
46.
47.     // Encapsulate Frame
48.     frame_len = enpackFrame(datagram, datagram_len, frame);

```

```

49.     printf("Frame length: %d\n", frame_len);
50.     display(frame, frame_len, 0);
51.
52.     // Unpack Frame
53.     frame_data_len = unpackFrame(frame, frame_len, frame_data);
54.     printf("Frame data length: %d\n", frame_data_len);
55.     display(frame_data, frame_data_len, 0);
56.
57.     // Unpack Datagram
58.     datagram_data_len = unpackIP(frame_data, frame_data_len, datagram_data);
59.     printf("Datagram data length: %d\n", datagram_data_len);
60.     display(datagram_data, datagram_data_len, 0);
61.
62.     system("pause");
63.     return 0;
64. }
65.
66. void display(unit_u8* data, int data_len, int type) {
67.     for (int i = 0; i < data_len; i++) {
68.         if (i != 0) {
69.             if (type)
70.                 printf(":");
71.             else
72.                 printf(" ");
73.         }
74.         printf("%02x", data[i]);
75.     }
76.     printf("\n");
77. }
78.
79. void getMACAddress(unit_u8 *mac, const char *message) {
80.     printf("%s", message);
81.     for (int i = 0; i < 6; i++) {
82.         scanf("%02x", &mac[i]);
83.     }
84.     fflush(stdin); // Clear the input buffer
85. }
86.
87. void getTypeField(unit_u8 *type) {
88.     printf("Enter type field (two bytes): ");
89.     scanf("%02x%02x", &type[0], &type[1]);
90.     fflush(stdin);

```

```

91. }
92.
93. int encapsFrame(unit_u8* src_data, int data_len, unit_u8* frame) {
94.     if (data_len == 0) return 0;
95.
96.     printf("===== Frame Encapsulation =====\n");
97.
98.     // Minimum frame check
99.     unit_u8 data[MAXSIZE];
100.    memcpy(data, src_data, data_len);
101.    if (data_len < MINSIZE) {
102.        memset(data + data_len, 0x00, MINSIZE - data_len);
103.        data_len = MINSIZE;
104.    }
105.
106.    printf("Minimum Frame Size Check: Data Length = %d\n", data_len);
107.
108.    FrameHeader fHeader;
109.    fHeader.leader = 0xabaaaaaaaaaaaaaa; // unit_u64 in little-endian format
110.
111.    // Get MAC addresses and type field
112.    getMACAddress(fHeader.desMAC, "Enter destination MAC address: ");
113.    getMACAddress(fHeader.srcMAC, "Enter source MAC address: ");
114.    getTypeField(fHeader.type);
115.
116.    // Encapsulation
117.    memcpy(frame, &fHeader, SizeFrameHeader);
118.    memcpy(frame + SizeFrameHeader, data, data_len);
119.
120.    int frame_len = SizeFrameHeader + data_len;
121.    return frame_len;
122.}
123.
124. int unpackFrame(unit_u8 *frame, int frame_len, unit_u8 *frame_data) {
125.     if (frame_len == 0) return 0;
126.
127.     printf("===== Frame Decapsulation =====\n");
128.
129.     FrameHeader fHeader;
130.     memcpy(&fHeader, frame, SizeFrameHeader);
131.
132.     // Extract frame boundary and data Length

```

```

133.     int p;
134.     for (p = 0; p < frame_len; p++) {
135.         if (frame[p] == 0xab) {
136.             p++;
137.             break;
138.         }
139.     }
140.
141.     int data_len = frame_len - p - 6 - 6 - 2;
142.
143.     memcpy(frame_data, frame + SizeFrameHeader, data_len);
144.
145.     // Output extracted information
146.     printf("Destination MAC: ");
147.     display(fHeader.desMAC, 6, 1);
148.     printf("Source MAC: ");
149.     display(fHeader.srcMAC, 6, 1);
150.     printf("Type: ");
151.     display(fHeader.type, 2, 1);
152.
153.     return data_len;
154. }
155.
156. int encapsIP(unit_u8 *data, int data_len, unit_u8 *datagram) {
157.     if (data_len == 0) return 0;
158.
159.     printf("===== IP Encapsulation =====\n");
160.     static u_int sequenceNum = 0;
161.     sequenceNum += 1;
162.
163.     Ipv4Header ipheader;
164.     ipheader.version_ihl = 0x45; // IPv4, IHL = 5
165.     ipheader.diffserve = 0x00;
166.     ipheader.tot_len = 20 + 8 + data_len; // 20 for IP header, 8 for ICMP header, and data length
167.     ipheader.id = 0x1000;
168.     ipheader.frag_off = 0x0000;
169.     ipheader.ttl = 128;
170.     ipheader.protocol = 0x01; // ICMP
171.     ipheader.check = 0;
172.     ipheader.saddr = inet_addr("192.168.43.34"); // Source IP

```

```

173.     ipheader.daddr = htonl(inet_addr("220.181.38.251")); // Destination IP (
        Baidu)
174.
175.     // Calculate checksum
176.     ipheader.check = CalChecksum((unit_u8 *)&ipheader, SizeIpv4Header, 0);
177.     printf("IP Header Checksum: %x\n", ipheader.check);
178.     ipheader.check = htons((unit_u16)ipheader.check);
179.
180.     ICMPHeader icmpheader;
181.     icmpheader.type = 0x08;
182.     icmpheader.code = 0x0;
183.     icmpheader.check = 0;
184.     icmpheader.id = 0x0001;
185.     icmpheader.sequence = sequenceNum;
186.
187.     unit_u8 checkdata[MAXSIZE];
188.     memcpy(checkdata, (unit_u8 *)&icmpheader, SizeICMPHeader);
189.     memcpy(checkdata + SizeICMPHeader, data, data_len);
190.     icmpheader.check = CalChecksum(checkdata, SizeICMPHeader + data_len, 0);
191.     printf("ICMP Header Checksum: %x\n", icmpheader.check);
192.     icmpheader.check = htons(icmpheader.check);
193.
194.     // Construct datagram
195.     memcpy(datagram, &ipheader, SizeIpv4Header);
196.     memcpy(datagram + SizeIpv4Header, &icmpheader, SizeICMPHeader);
197.     memcpy(datagram + SizeIpv4Header + SizeICMPHeader, data, data_len);
198.
199.     int datagram_len = SizeIpv4Header + SizeICMPHeader + data_len;
200.     return datagram_len;
201. }
202.
203. int unpackIP(unit_u8 *frame_data, int frame_data_len, unit_u8 *datagram_data)
    {
204.     if (frame_data_len == 0) return 0;
205.
206.     printf("===== IP Decapsulation =====\n");
207.
208.     Ipv4Header ipheader;
209.     ICMPHeader icmpheader;
210.     int datagram_data_len = frame_data_len - SizeIpv4Header - SizeICMPHeader;
211.
212.     memcpy(&ipheader, frame_data, SizeIpv4Header);

```

```
213.     memcpy(&icmpheader, frame_data + SizeIPv4Header, SizeICMPHeader);
214.     memcpy(datagram_data, frame_data + SizeIPv4Header + SizeICMPHeader, datagram_data_len);
215.
216.     // Check IP checksum
217.     ipheader.check = CalChecksum((unit_u8 *)&ipheader, SizeIPv4Header, 0);
218.     if (ipheader.check != 0) {
219.         printf("ERROR: IP checksum invalid!\n");
220.         return 0;
221.     }
222.
223.     // Check ICMP checksum
224.     unit_u8 checkdata[MAXSIZE];
225.     memcpy(checkdata, (unit_u8 *)&icmpheader, SizeICMPHeader);
226.     memcpy(checkdata + SizeICMPHeader, datagram_data, datagram_data_len);
227.     icmpheader.check = CalChecksum(checkdata, SizeICMPHeader + datagram_data_len, 0);
228.     if (icmpheader.check != 0) {
229.         printf("ERROR: ICMP checksum invalid!\n");
230.         return 0;
231.     }
232.
233.     // Output results
234.     struct in_addr saddr, daddr;
235.     memcpy(&saddr, &ipheader.saddr, 4);
236.     memcpy(&daddr, &ipheader.daddr, 4);
237.     printf("Source IP: %s\n", inet_ntoa(saddr));
238.     printf("Destination IP: %s\n", inet_ntoa(daddr));
239.
240.     printf("ICMP Type: %d\n", icmpheader.type);
241.     printf("ICMP Code: %d\n", icmpheader.code);
242.
243.     // Output Data
244.     printf("Data: %s\n", datagram_data);
245.
246.     return datagram_data_len;
247. }
```