华东师范大学计算机科学技术系上机实践报告

课程名称：计算机网络	**年级：**2022	**上机实践成绩：**
指导教师：洪道诚	**姓名：**朱宇笑	**创新实践成绩：**
实验名称：用户数据报协议	**学号：**10225001410	**上机实践日期：**2021/12/9
座位编号：F	**组号：**6	**上机实践时间：**2学时

1 实验目的

1. 掌握 UDP 协议的报文格式
2. 掌握 UDP 协议校验和的计算方法
3. 理解 UDP 协议的优缺点
4. 理解协议栈对 UDP 协议的处理方法
5. 理解 UDP 上层接口应满足的条件

2 实验环境

采用网络拓扑结构一

3 实验原理

3.1 进程到进程的通信

在学习 UDP 协议之前，首先应该了解主机到主机的通信和进程到进程的通信，以及这两种通信之间的区别。

IP 协议负责主机到主机的通信。作为一个网络层协议，IP 协议只能把报文交付给目的主机。这是一种不完整的交付，因为这个报文还没有送交到正确的进程。像 UDP 这样的传输层协议负责进程到进程的通信。UDP 协议负责把报文交付到正确的进程。下图描绘了 IP 协议和 UDP 协议的作用范围。

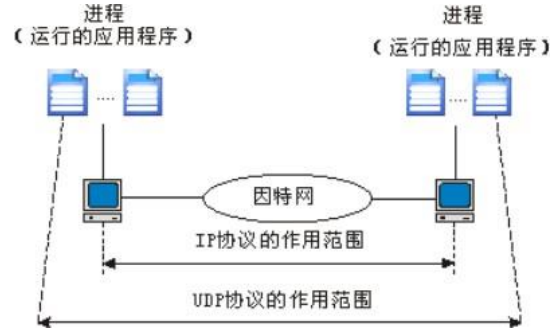


图 1: UDP 与 IP 的区别

3.1.1 端口号

在网络中，主机是用 IP 地址来标识的。而要标识主机中的进程，就需要第二个标识符，

这就是端口号。在 TCP/IP 协议族中，端口号是在 0 ～65535 之间的整数。

在客户/服务器模型中，客户程序使用端口号标识自己，这种端口号叫做短暂端口号，短暂的意思是生存时间比较短。一般把短暂端口取为大于 1023 的数，这样可以保证客户程序工作得比较正常。

服务器进程也必须用一个端口号标识自己。但是这个端口号不能随机选取。如果服务器随机选取端口号，那么客户端在想连接到这个服务器并使用其服务的时候就会因为不知道这个端口号而无法连接。TCP/IP 协议族采用熟知端口号的办法解决这个问题。每一个客户进程都必须知道相应的服务器进程熟知端口号。

UDP 的熟知端口号如下表所示：

端口	协议	说明
7	Echo	把收到的数据报回送到发送端
9	Discard	丢弃收到的任何数据报
11	Users	活跃的用户
13	Daytime	返回日期和时间
17	Quote	返回日期和引用（译者注：可参阅 RFC856）
19	Chargen	返回字符串
53	Nameserver	域名服务
67	Bootps	下载引导程序信息的服务器端口
68	Bootpc	下载引导程序信息的客户端口
69	TFTP	简单文件传送协议
111	RPC	远程过程调用
123	NTP	网络时间协议
161	SNMP	简单网络管理协议
162	SNMP	简单网络管理协议（陷阱）
520	RIP	路由信息协议

图 2: UDP 的熟知端口号

在一个 IP 数据包中，目的 IP 地址和端口号起着不同的寻址作用。目的 IP 地址定义了在世界范围内惟一的一台主机。当主机被选定后，端口号定义了在这台主机上运行的多个进程中的一个。

3.1.2 套接字地址

一个 IP 地址与一个端口号结合起来就叫做一个套接字地址。客户套接字地址惟一地定义了客户进程，而服务器套接字地址惟一地定义了服务器进程。

要使用 UDP 的服务，就需要一对套接字地址：客户套接字地址和服务器套接字地址。客户套接字地址指定了客户端的 IP 地址和客户进程，服务器套接字地址指定了服务器的 IP 地址和服务器进程。

3.2 面向连接的服务与面向无连接的服务

从通信的角度来看，在 OSI 参考模型中，下层能向上层提供两种不同形式的服务：面向连接的服务和面向无连接的服务。

3.2.1 面向连接的服务

所谓连接，就是两个对等实体为进行数据通信而进行的一种结合。面向连接服务在进行数据交换前，先建立连接。当数据传输结束后，应释放这个连接。因此，采用面向连接的服务进行数据传送要经历三个阶段：

1. 建立连接阶段：在有关的服务原语以及协议数据单元中，必须给出源用户和目的用户的完整地址。同时可以协商服务质量和其它一些选项。

2. 数据交换阶段：在这个阶段，每个报文中不必包含完整的源用户和目的用户的完整地址， 而是使用一个连接标识符来代替。由于连接标识符相对于地址信息要短得多，因此使控制信息在报文中所占的比重相对减小，从而可减小系统的额外开销，提高信道的有效利用率。另外，报文的发送和接收都是按固定顺序的，即发送方先发送的报文，在接受方先收到。

3. 释放连接阶段：通过相应的服务原语完成释放操作。

从面向连接服务的三个阶段来看，连接就像一个管道，发送端在其一端依次发送报文，接收者依次在其另一端按同样的顺序接收报文。这种连接又称虚拟电路。它可以避免报文的丢失、重复和乱序。若两个用户经常需要通信，则可以建立永久虚电路。这样可以免除每次通信时建立连接和释放连接这两个阶段。这点与电话网中的专线很相似。

3.2.2 面向无连接的服务

在面向无连接服务的情况下，两个实体之间的通信不必事先建立一个连接。相对于面向连接的服务，面向无连接服务灵活方便且快速。但它不能防止报文的丢失、重复和乱序。由于它的每个报文必须包括完整的源地址的目的地址，因此开销较大。面向无连接服务主要有三种类型：

1. 数据报：它的特点是发完报文就结束，而对方不做任何响应。数据报的服务简单，额外开销少，但可靠性差，它比较适合于数据具有很大的冗余度以及要求有较高的实时性的通信场合。

2. 证实交付：又称可靠的数据报。这种服务对每一个报文产生一个证实给发送方，不过这种证实不是来自对应方用户，而是来自提供服务的层。这种证实只能保证报文已经发给目的站了，而不能保证对应方用户正确地接收到报文。

3. 请求回答：这种类型服务是接收端用户每收到一个报文，即向发送端用户发送一个应答报文。但是双方发送的报文都有可能丢失。如果接收端发现报文有错误，则回送一个表示有错误的报文。

3.3 UDP 协议简介

UDP（用户数据报协议），主要用来支持那些需要在计算机之间传输数据的网络应用。包括网络视频会议系统在内的众多的客户/服务器模式的网络应用都需要使用 UDP 协议。UDP 协议从问世至今已经被使用了很多年，虽然其最初的光彩已经被一些类似的协议所掩盖，但是即使是在今天，UDP 仍然不失为一项非常实用和可行的网络传输层协议。

UDP 协议直接位于 IP 协议的上层。根据 OSI 参考模型，UDP 和 TCP 都属于传输层协

议。UDP 协议不提供端到端的确认和重传功能，它不保证数据包一定能到达目的地，因此是不可靠协议。

UDP 协议有以下特点：

- UDP 是面向事务的协议，它用最少的传输量为应用程序向其它程序发送报文提供了一个途径。
- UDP 是无连接的、不可靠的传输机制。在发送数据报前，UDP 在发送和接收两者之间不建立连接。
- UDP 让应用程序能直接访问网络层的数据报服务，例如分段和重组等网络层所提供的数据报服务。
- UDP 使用 IP 协议作为数据传输机制的底层协议。
- UDP 报头和数据都以与最初传输时相同的形式被传送到最终目的地。
- UDP 不提供确认，也不对数据的到达顺序加以控制。因此 UDP 报文可能会丢失。
- 不实现数据包的传送和重复检测。
- 当数据包在传送过程中发生错误时，UDP 不能报告错误。
- 吞吐量不受拥塞控制算法的调节，只受应用程序生成数据的速率、传输带宽、发送端和接收端主机性能的限制。

3.4 UDP 报文格式

下图显示了 UDP 报文格式。每个 UDP 报文称为一个用户数据报（User Datagram），用户数据报分为两个部分：UDP 首部和 UDP 数据。首部被分为四个 16 位的字段，分别代表源端口号、目的端口号、报文的长度以及 UDP 校验和。

源端口（16 位）	目的端口（16 位）
有效负载长度（16 位）	校验和（16 位）
数据	
……	

图 3: UDP 报文格式

- 源端口：该字段表示发送端的端口号。如果源端口没有使用，那么此字段的值就被指定为0。这是一个可选的字段。不同的应用程序使用不同的端口号，UDP 协议使用端口号为不同的应用程序保留其各自的数据传输通道，从而实现了同一时间段内多个应用程序可以一起使用网络进行数据的发送和接收。
- 目的端口：该字段表示数据包被发往的目的端的端口号。
- 有效负载长度：该字段表示包括 UDP 首部和 UDP 数据在内的整个用户数据报的长度。该字段的最小值是 8。数据报的最大尺寸随操作系统的不同而不同。在两字节字段中，理论上数据报最多可达 65535 字节。然而，一些 UDP 实现将数据报的大小限制到了 8192 字节。
- 校验和：UDP 的校验的校验范围包括伪首部（IP 首部一部分字段）、UDP 首部和 UDP 数据，该字段是可选的。如果该字段值为零就说明不进行校验。

3.5 UDP 封 装

当进程有报文要通过 UDP 发送时，它就把这个报文连同一对套接字地址以及数据的长度传递给 UDP。UDP 收到数据后就加上 UDP 首部。然后 UDP 就把这用户数据报连同套接字地址一起传递给 IP。IP 加上自己的首部，在高层协议类型字段使用值 17，指出该数据是从 UDP 协

议来的。这个 IP 数据报再传递给数据链路层。数据链路层接收到 IP 数据报后，加上自己的首部
（可能还有尾部），再传给物理层。物理层把这些位编码为电信号或光信号，把它发送到远程的主机。如下图所示：

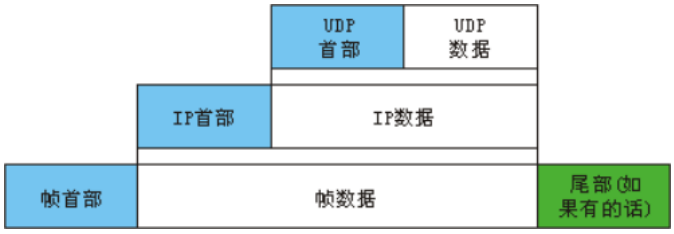


图 4: UDP 封 装

3.6 UDP 校验和

UDP 校验和的计算与 IP 和 ICMP 校验和的计算不同。UDP 校验和校验的范围包括三部分： 伪首部、UDP 首部以及从应用层来的数据。

伪首部是 IP 首部的一部分，其中有些字段要填入 0。用户数据报封装在 IP 数据包中。如下图所示：

图 5: 伪首部添加到 UDP 的数据报上

若校验和不包括伪首部，用户数据报也可能是安全的和正确的。但是，若 IP 首部受到损伤， 则它可能被交付到错误的主机。

伪首部中包含高层协议类型字段是为了确保这个数据包是属于 UDP 而不是属于 TCP（参见实验七）的。使用 UDP 的进程和使用 TCP 的进程可以使用同一个端口号。UDP 的高层协议类型字段是 17。若在传输过程中这个值改变了，在接收端计算校验和时就可检测出来，UDP 就可丢弃这个数据包。这样就不会交付给错误的协议。

3.6.1 在发送端的校验和计算

在发送端按以下步骤计算校验和：

1. 把伪首部填加到 UDP 用户数据报上。
2. 把校验和字段填入零。
3. 按 16 位长度将数据报分段。
4. 若分段总数不是偶数，则增加一个分段的填充（全 0）。填充只是为了计算校验和，计算完毕后就把它丢弃。
5. 把所有 16 位的分段使用反码算术运算相加。
6. 把得到的结果取反码，它是一个 16 位的数，把这个数插入到校验和字段。
7. 把伪首部和填充丢掉。
8. 把 UDP 用户数据报交付给 IP 进行封装。

在伪首部中的各行的顺序对校验和的计算没有任何影响。此外，增加 0 也不影响计算的结果。下图给出了一个计算 UDP 校验和的例子。这里假定用户数据报的长度是 15 字节，因此要添加一个全 0 的字节。

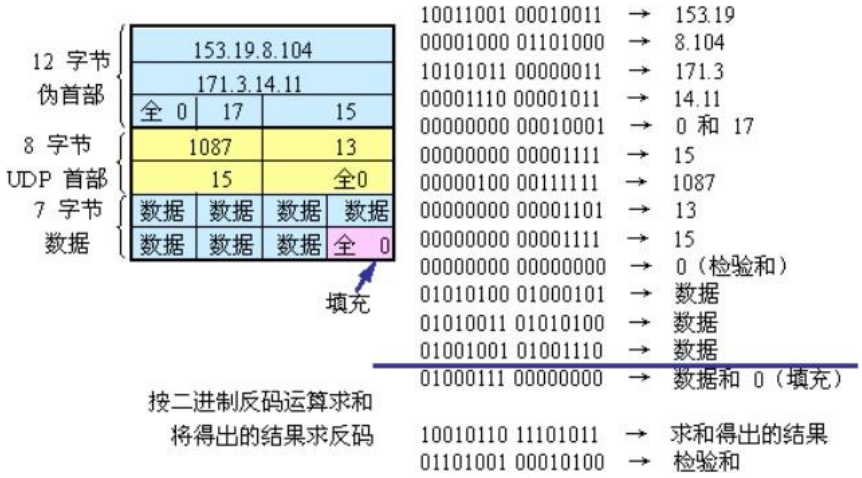


图 6: UDP 校验和的计算过程

3.6.2 在接收端的校验和计算

接收端按以下 6 个步骤计算校验和是否正确：

1. 把伪首部加到 UDP 用户数据报上。
2. 若需要，就增加填充。
3. 把数据报按 16 位长度分段。
4. 把所有 16 位的分段使用反码算术运算相加。
5. 把得到的结果取反码。
6. 若得到的结果是全零，则丢弃首部和填充，并接受这个用户数据报。若结果是非零，就丢弃这个用户数据报。

校验和是可选使用的，若不计算校验和，则校验和字段就填入 0。

3.7 UDP 应 用

下面列出了 UDP 协议的一些用途：

- UDP 适用于这样的进程，它需要简单的请求——响应通信，而较少考虑流量控

制和差错控制。对于需要传送成块数据的进程，如 FTP，则通常不使用 UDP；

- UDP 适用于具有内部流量控制和差错控制机制的进程；
- 对多播和广播来说，UDP 是个比较合适的传输层协议；
- UDP 可用于管理进程，如 SNMP 协议；
- UDP 可用于某些路由选择更新协议，如路由信息协议（RIP 协议，参考实验 17）。

3.8 协议栈实现代码解析

本实验将通过对安装目录 *ExpCNS \ work \ EXPcns_studentnet \ netproto_udp_student \ netproto_udp_student* 下的proto_ udp_student.h、netproto_udp_shudent.c 和安装目录 *ExpCNS\ work\EXPcns_studentnet\netproto_udp_student\netproto_udpif_student* 下的 net_proto_udpif_student.h、netproto_udpif_student.c 四个文件进行编码，完成协议栈中 UDP 协议的实现。

netproto_udp_student.h 和 netproto_udp_shudent.c 文件用于实现 UDP 数据包发送和接收。其中，netproto_udp_student.h 文件中定义了 UDP 协议实现相关数值以及 UDP 的负载内容、负载长度，关键代码如下所示：

```
#define UDF_SOUR_PORT       5893                    /**< 源端口号 */
#define UDr_DEST_PORT       5893                    /**< 目的端口号 */
#define IF_DEST_ADDR        "0.0.0.0"               /**< 目的 IF 地址 */
#define PAYLOAD_DATA        "Hello, World!"         /**< 有 效 负 载 */
#define PAYLOAD_L.EN        sizeof(PAYLOAD_DATA)    /**< 有效负载长度 */
```

宏	值	描述
UDP_SOUR_PORT	5893	定义 UDP 包头中“源端口”字段的值
UDP_DEST_PORT	5893	定义 UDP 包头中“目的端口”字段值
IP_DEST_ADDR	"0.0.0.0"	指定目的 IP 地址，学生需根据要求设置
PAYLOAD_DATA	"Hello,World!"	自定义 UDP 负载内容，学生可修改
PAYLOAD_LEN	sizeof(PAYLOAD_DATA)	自定义 UDP 负载的长度

图 7：定义的宏

在实验的编码过程中，应该使用这些宏对相应的变量进行赋值。netproto_udp_shudent.c 文件是协议栈中 UDP 数据包发送和接收的实现部分，其中定义了 2 个函数。下面介绍这些协议栈的实现部分。

函数 netp_udp_output_student 的功能是构造并发送一个 UDP 数据包，其高层协议为自定义协议类型，负载内容为自定义负载。这个函数的编码工作需要由学生完成。

当有数据到达本机网络接口时，函数 netp_udp_input_student 将被调用，并传递给这个函数原始数据。在该函数中，需要判断一些条件值来确定接收到的数据包为自定义 UDP 数据，如果是自定义 UDP 数据包，则输出负载内容，如果不是，则返回 NETP_PUSH_TO_LWIP 交给协议栈继续处理。

netproto_udpif_student.h 和 netproto_udpif_shudent.c 文件用于实现 UDP 上层

投递的功能，即为高层使用 UDP 协议提供了接口。其中，netproto_udpif_student.h 文件中并没有定义太多内容。netproto_udpif_shudent.c 文件是协议栈中 UDP 上层投递的功能的实现部分，其中定义了一个全局变量 recv_port 和 2 个函数。

全局变量 recv_port 的作用很简单，它记录了发送 UDP 数据报时的源端口号作为接收 UDP

数据报的过滤条件。

函数 netp_send_udp 通过 IP 层接口发送 UDP 数据报，该函数功能需要学生完成。

函数 netp_udp_input_student 处理输入数据包，如果输入的数据报满足过滤条件，则投递给上层协议使用。该函数功能需要学生完成。

在编码过程中可能会设计到一些结构体、宏和函数，下表是对他们进行的介绍：

结构体/宏/函数	声明或定义	描述

struct netp_eth_header	struct netp_eth_header{
	u8_t dest_address[ETH_ADDRESS_LEN];
	u8_t sour_address[ETH_ADDRESS_LEN];
	u16_t type;
	};
		以太网帧头结构

struct netp_ip_header	struct netp_ip_header{
	u8_t headerlen:4;
	u8_t version:4;
	u8_t diff_services;
	u16_t total_length;
	u16_t identification;
	u16_t flags_offset;
	u8_t time_to_live;
	u8_t protocol;
	u16_t header_checksum;
	struct ip_addr source_address;
	struct ip_addr destination_address;
	};
		ipv4 包头结构

struct netp_udp_header	struct netp_udp_header{
	u16_t sour_port;
	u16_t dest_port;
	u16_t length;
	u16_t checksum;
	};
		UDP 报头结构

struct netp_udp_pseudo	struct netp_udp_pseudo{ u32_t sour_addr; u32_t dest_addr; u8_t zero; u8_t protocol; u16_t length;	UDP 伪首部结构

图 8：实验涉及的结构体、宏和函数（一）

结构体/宏/函数	声明或定义	描述
	};	

struct in_addr	struct in_addr{
	u32_t s_addr;
	};
		32 位地址

UDP_HEADER_LEN	#define UDP_HEADER_LEN 8	UDP 包头长度
PAYLOAD_DATA	#define PAYLOAD_DATA "Hello,EXPcns!"	UDP 负载内容
PAYLOAD_LEN	#define PAYLOAD_LEN sizeof(PAYLOAD_DATA)	UDP 负载长度
NETP_UDP_PSEUDO_LEN	#define NETP_UDP_PSEUDO_LEN sizeof(structnetp_udp_pseudo)	UDP 伪首部长度
ETH_HEADER_LEN	#define ETH_HEADER_LEN 14	以太网帧头长度
IP_HEADER_LEN	#define IP_HEADER_LEN 20	IP 包头长度
netp_current_udp_addr	u32_t netp_current_udp_addr();	获取当前正在使用的网络接口的 UDP 地址
htons	u16_t htons(u16_tn);	将 16 位数值由主机字节序转换为网络字节序
inet_addr	u32_t inet_addr(const char *cp);	将 ASCII 编码的 Internet 地址转换成为网络字节序地址
inet_chksum	u16_t inet_chksum(void *dataptr,u16_t len)	计算校验和

图 9：实验涉及的结构体、宏和函数（二）

3.9 各模块推荐流程

3.9.1 UDP 数据包发送流程

编码实现 UDP 数据包发送推荐使用如下流程：

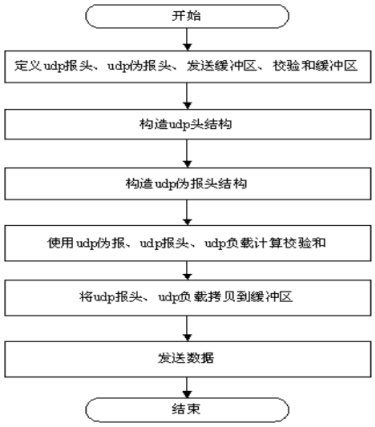


图 10：UDP 数据包发送推荐流程

3.9.2 输入 UDP 数据包处理流程

编码实现处理 UDP 输入数据包推荐使用如下流程：

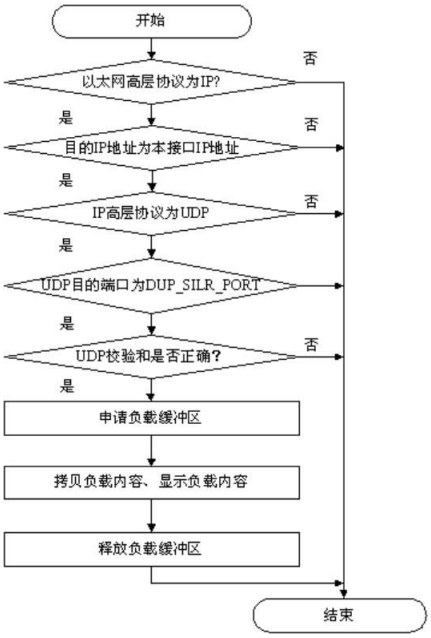


图 11：处理 UDP 输入数据包推荐流程

3.9.3 UDP 发送接口实现流程

编码实现 UDP 发送接口推荐使用如下流程：

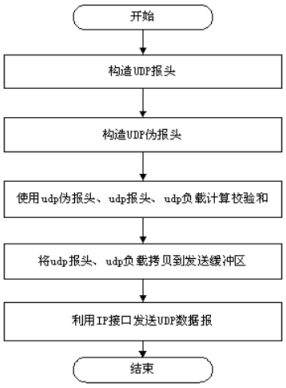


图 12：UDP 发送接口实现推荐流程

3.9.4 UDP 接受接口推荐流程

编码实现 UDP 接收接口推荐使用如下流程：

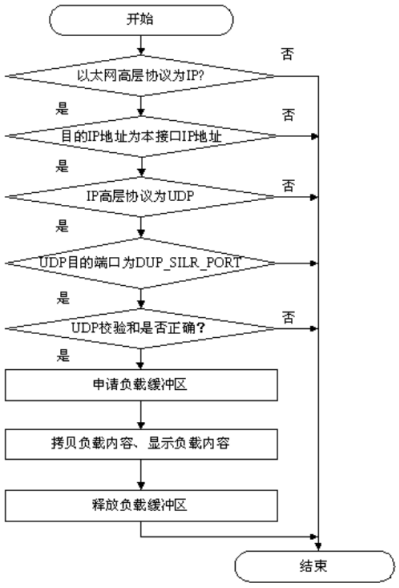


图 13：UDP 接收接口实现推荐流程

4 实验内容

4.1 编辑并发送 UDP 数据报

在实验中每组 A～F 主机的接口 IP 地址分别设置为 172.16.0.n1、172.16.0.n2、172.16.0.n3、172.16.0.n4、172.16.0.n5、172.16.0.n6(其中 n 为组别号，取值范围为 112)，子网掩码设置为 255.255.0.0，默认网关设置为空。

本练习将主机 A 和 B 作为一组，主机 C 和 D 作为一组，主机 E 和 F 作为一组。现仅以主机 A、B 所在组为例，其它组的操作参考主机 A、B 所在组的操作。

1. 主机 A 打开协议编辑器，编辑发送给主机 B 的 UDP 数据报。

MAC 层:

目的 MAC 地址: 接收方 MAC 地址源 MAC 地址: 发送方 MAC 地址

协议类型或数据长度:0800，即 IP 协议

IP 层:

总长度: 包括 IP 层、UDP 层和数据长度高层协议类型:17，即 UDP 协议

首部校验和: 其它所有字段填充完毕后填充此字段源 IP 地址: 发送方 IP 地址

目的 IP 地址: 接收方 IP 地址

UDP 层:

源端口:1030

目的端口: 大于 1024 的端口号

有效负载长度:UDP 层及其上层协议长度 (注:“有效负载长度”这个说法有误，这是软件本身存在的问题，实际上 UDP 协议将此字段定义为整个 UDP“报文长度”!)

其它字段默认，计算校验和。

UDP 在计算校验和时包括哪些内容?

2. 在主机 B 上启动协议分析器捕获数据，并设置过滤条件（提取 UDP 协议)。

3. 主机 A 发送已编辑好的数据报。

4. 主机 B 停止捕获数据，在捕获到的数据中查找主机 A 所发送的数据报。思考问题:

1. 为什么 UDP 协议的“校验和”要包含伪首部?

2. 比较 UDP 和 IP 的不可靠程度?

4.2 UDP 单播通信

本练习将主机 A、B、C、D、E、F 作为一组进行实验。

1. 主机 B、C、D、E、F 上启动实验平台工具栏中的 UDP 工具”，作为服务器端，监听端口设置为 2483，“创建”成功。

2. 主机 C、E 上启动协议分析器开始捕获数据，并设置过滤条件（提取 UDP 协议)。

3. 主机 A 上启动实验平台工具栏中的“UDP 工具”，作为客户端，以主机 C 的 IP 为目的

IP 地址，以 2483 为端口，填写数据并发送。

4. 察看主机 B、C、D、E、F 上的“UDP 工具”接收的信息。

- 哪台主机上的“UDP 工具”能够接收到主机 A 发送的 UDP 报文?

5. 察看主机 C 协议分析器上的 UDP 报文，并回答以下问题:

- UDP 是基于连接的协议吗? 阐述此特性的优缺点。
- UDP 报文交互中含有确认报文吗? 阐述此特性的优缺点。

6. 主机 A 上使用协议编辑器向主机 E 发送 UDP 报文，其中：目的 MAC 地址:E 的 MAC 地址
目的 IP 地址：主机 E 的 IP 地址目的端口:2483 校验和：0
有效负载长度:UDP 层及其上层协议长度
首部校验和：其它所有字段填充完毕后填充此字段总长度：包括 IP 层、UDP 层和数据长度
发送此报文，并回答以下问题：

- 主机 E 上的 UDP 通信程序是否接收到此数据包？UDP 是否可以使用 0 作为校验和进行通信？

7. 主机 B、C、D、E、F 关闭服务端，主机 A 关闭客户端。

4.3 UDP 广播通信

本练习将主机 A、B、C、D、E、F 作为一组进行实验。

1. 主机 B、C、D、E、F 上启动 UDP 工具，作为服务器端，监听端口设为 2483。
2. 主机 B、C、D、E、F 启动协议分析器捕获数据，并设置过滤条件（提取 UDP 协议）。
3. 主机 A 上启动 UDP 工具，作为客户端，以 255.255.255.255 为目的地址，以 2483 为端口，填写数据并发送。
4. 察看主机 B、C、D、E、F 上的“UDP 工具”接收的信息。

- 哪台主机能够接收到主机 A 发送的 UDP 报文？

5. 察看协议分析器上捕获的 UDP 报文，并回答以下问题：

- 主机 A 发送的报文的目的 MAC 地址和目的 IP 地址的含义是什么？

4.4 UDP 数据报发送与接受

本练习将主机 A 和 B 作为一组，主机 C 和 D 作为一组，主机 E 和 F 作为一组。现仅以主机 A、B 所在组为例，其它组的操作参考主机 A、B 所在组的操作。实验开始前，先单击“初始环境”。

在实验中，主机 A 将新接口的 IP 地址设置为 172.16.1.11、主机 B 使用处于连接状态的物理接口，将新接口的 IP 设置为 172.16.1.12、主机 C 将新接口的 IP 地址设置为 172.16.1.13、主机 D 使用处于连接状态的物理接口，将新接口的 IP 地址设置为 172.16.1.14、主机 E 使用处于连接状态的物理接口，将新接口的 IP 地址设置为 172.16.1.15、主机 F 将新接口的 IP 地址设置为 172.16.1.16。所有主机使用子网掩码 255.255.255.0，默认网关设置为 0.0.0.0。

1. 设置目的 IP 地址各主机打开安装目录 ExpCNS\work\EXPcns_student\netproto_udp_student\ netproto_udp_student 下的 netproto_udp_student.h 文件，将 IP_DEST_ADDR
宏所定义的 IP 地址修改为同组同学的新接口 IP 地址。
2. 编码实现发送 UDP 数据包

(a) 各主机打开安装目录ExpCNS\work\EXPcns_student\netproto_udp_student\net proto_udp_student 下的 netproto_udp_student.c 文件，在函数 netp_udp_output_student 内编写实现代码。

(b) 参考实验原理 UDP 数据包发送推荐流程图给出的流程，分析已经存在的代码。已经存在的代码创建了 UDP 头结构、UDP 伪首部结构、源 IP 地址、目的 IP 地址、发送

缓冲区和校验和缓冲区等变量，UDP 校验和的计算过程，将 UDP 头和负载拷贝到发送缓冲区中以及将缓冲区数据发送到网络中的实现。

(c) 构造 UDP 头，校验和字段为 0 构造并填充一个 UDP 数据包头。可以使用 netproto_udp_ student.h 文件中所定义的端口值。总长度为 UDP_HEADER_LEN+PAYLOAD_LEN。

(d) 构造 UDP 伪首部各主机构造并填充 UDP 伪首部。其中，源 IP 地址为本接口 IP 地址， 目的 IP 地址为 IP_DEST_ADDR 所指定的 IP 地址，长度与 UDP 头中的总长度数值相同。

3. 所有主机编码实现 UDP 数据包输入处理功能

(a) 各主机打开安装目录 ExpCNS\work\EXPcns_student\netproto_udp_student 下的 net- proto_udp_student.c 文件，在函数 netp_udp_input _student 内编写实现代码。

(b) 参考实验原理处理 UDP 输入数据包推荐流程图给出的流程，分析已经存在的代码。已经存在的代码定义了以太网帧头、IP 包头、UDP 报头、UDP 伪报头、校验和、负载缓冲区指针和校验和缓冲区指针，以太网高层协议的校验，目的 IP 地址的校验，申请负载缓冲区，打印负载内容等实现。

(c) 提取 UDP 数据包。各主机通过以 IP 头结构 ip_header 中的 protocol 即“高层协议类型”字段值判断该数据帧的高层协议是否为 UDP 协议，可以使用宏 IP_PROTO_UDP。如果不是 UDP 数据包则返回 NETP_PUSH_TO_LWIP 交给协议栈继续执行。

(d) 提取 UDP 目的端口号为 UDP_SOUR_PORT 的数据报。各主机通过 UDP 头结构udp_header 中的 dest_port 即“目的端口号”字段值判断该 UDP 数据报是否是我们感兴趣的UDP 数据报。如果不是则返回 NETP_PUSH_TO_LWIP 交给协议栈继续执行。

(e) 判断 UDP 校验和是否正确。如果 UDP 头的校验和字段值不为 0，则需要验证校验和是否正确。各主机通过 IP 报头和 UDP 报头中的数值验证 UDP 校验和是否正确，如果不正确则丢弃该数据报。

4. 所有主机打开协议分析器，开始捕获数据
5. 所有主机调试并运行程序
6. 各主机停止数据捕获，观察实验现象

- 你收到的负载内容是什么？

7. 参考代码如下：

```
/**
* \brief 编辑并发送一个 UDP 数据包
* 由学生完成这个函数，发送一个 UDP 数据包。主线程将会调用这个函数。
*/
void
netp_udp_output_student()
{
// 定义 udp 报头、udp 伪报头、发送缓冲区、校验和缓冲区
struct netp_udp_header udp_header; struct netp_udp_pseudo udp_pseudo; struct ip_addr
sour_ipaddr; struct ip_addr dest_ipaddr;
u8_t send_buff[UDP_HEADER_LEN+PAYLOAD_LEN];
```

```
    u8_t checksum_buff[NETP_UDP_PSEUDO_LEN+UDP_HEADER_LEN+PAYLOAD_LEN];

    // 构造 udp 头结构
    udp_header.sour_port = htons(UDP_SOUR_PORT); udp_header.dest_port =
    htons(UDP_DEST_PORT); udp_header.length = htons(UDP_HEADER_LEN+PAYLOAD_LEN);
    udp_header.checksum = 0;


    // 构造 udp 伪报头结构
    sour_ipaddr.addr = netp_current_ip_addr();
    dest_ipaddr.addr=inet_addr(IP_DEST_ADDR); udp_pseudo.sour_addr=sour_ipaddr.addr;
    udp_pseudo.dest_addr=dest_ipaddr.addr; udp_pseudo.zero = 0;
    udp_pseudo.protocol = IP_PROTO_UDP; udp_pseudo.length=udp_header.length;

    // 将 udp 伪报头、udp 报头、dup 负载拷贝到校验和缓冲区memcpy(checksum_buff,
    &udp_pseudo, NETP_UDP_PSEUDO_LEN); memcpy(checksum_buff+NETP_UDP_PSEUDO_LEN,
    &udp_header, UDP_HEADER_LEN);
    memcpy(checksum_buff+NETP_UDP_PSEUDO_LEN+UDP_HEADER_LEN,
    PAYLOAD_DATA, PAYLOAD_LEN);

    // 计算校验和
udp_header.checksum = inet_chksum(checksum_buff,
    NETP_UDP_PSEUDO_LEN+UDP_HEADER_LEN+PAYLOAD_LEN);

    // 将 udp 报头、udp 负载拷贝到发送缓冲区memset(send_buff, 0, sizeof(send_buff));
    memcpy(send_buff,&udp_header,UDP_HEADER_LEN);
    memcpy(send_buff+UDP_HEADER_LEN, PAYLOAD_DATA, PAYLOAD_LEN);

    // 发送数据
if (ERR_OK == netp_udp_output(&sour_ipaddr, &dest_ipaddr, send_buff,
    UDP_HEADER_LEN+PAYLOAD_LEN, 0)) {
    printf("\n发送 UDP 数据包 -------------- 成功! \n");
    } else {
    printf("\n发送 UDP 数据包 -------------- 失败! \n");
    }
    }

    /**
* \brief 当有数据帧到达时，将调用这个函数
```

```
 *
* \param packet 指向接收到的数据
* \param packet_len 数据帧的长度
 *
* \return 一个 put_to_lwip 类型的返回值。
* 返回 NETP_PUT_LWIP 表示处理数据后将数据帧交给上层协议栈继续处理
* 返回 NETP_NO_PUT_LIWP 表示不将数据帧交给上层协议栈处理。
 */
enum push_to_lwip

netp_udp_input_student(void *packet, int packet_len)
{
// 定义以太网帧头、IP 包头、UDP 报头、UDP 伪报头、校验和
// 负载缓冲区指针和校验和缓冲区指针struct netp_eth_header eth_header; struct
netp_ip_header ip_header; struct netp_udp_header udp_header; struct netp_udp_pseudo
udp_pseudo; u16_t udp_checksum;
char *payload_buff; u8_t *checksum_buff;

memcpy(&eth_header, packet, ETH_HEADER_LEN);

// 只提取 ip 数据包
if (eth_header.type != htons(MAC_PROTO_IP)){
return NETP_PUSH_TO_LWIP;
}

memcpy(&ip_header, (u8_t*)packet+ETH_HEADER_LEN, IP_HEADER_LEN);
// 过滤掉目的 ip 地址不是本接口的数据包
if (ip_header.destination_address.addr != netp_current_ip_addr()) {
return NETP_PUSH_TO_LWIP;
}

// 只提取 UDP 数据报
if (ip_header.protocol != IP_PROTO_UDP){
return NETP_PUSH_TO_LWIP;
}
memcpy(&udp_header, (u8_t*)packet+ETH_HEADER_LEN+IP_HEADER_LEN, UDP_HEADER_LEN);
// 过滤掉目的端口号不是 UDP_SOUR_PORT 的数据报
if (udp_header.dest_port != htons(UDP_DEST_PORT)){
return NETP_PUSH_TO_LWIP;
}
```

```
    // 判断 UDP 校验和是否正确
    if (0 != udp_header.checksum) {
    checksum_buff = malloc(NETP_UDP_PSEUDO_LEN+udp_header.length); udp_checksum =
    udp_header.checksum;

    udp_header.checksum = 0;
    udp_pseudo.dest_addr = ip_header.destination_address.addr; udp_pseudo.sour_addr =
    ip_header.source_address.addr; udp_pseudo.zero = 0;
    udp_pseudo.protocol = IP_PROTO_UDP; udp_pseudo.length=udp_header.length;

    memcpy(checksum_buff, &udp_pseudo, NETP_UDP_PSEUDO_LEN);
    memcpy(checksum_buff+NETP_UDP_PSEUDO_LEN, &udp_header, UDP_HEADER_LEN);
    memcpy(checksum_buff+NETP_UDP_PSEUDO_LEN+UDP_HEADER_LEN,
    (u8_t*)packet+ETH_HEADER_LEN+IP_HEADER_LEN+UDP_HEADER_LEN,
    ntohs(udp_header.length)-UDP_HEADER_LEN);

if (udp_checksum != inet_chksum(checksum_buff,
    NETP_UDP_PSEUDO_LEN+ntohs(udp_header.length))) {
    return NETP_PUSH_TO_LWIP;
    }

    free(checksum_buff);
    }

    // 申请负载缓冲区
    payload_buff = malloc(udp_header.length-UDP_HEADER_LEN);

    // 拷贝负载内容到负载缓冲区
memcpy(payload_buff, (u8_t*)packet+ETH_HEADER_LEN+IP_HEADER_LEN+UDP_HEADER_LEN,
    udp_header.length-UDP_HEADER_LEN);

    // 打印负载内容
printf("\n接收到目的端口为： d 的 UDP 数据报，负载为： s\n",%
    ntohs(udp_header.dest_port), (char*)payload_buff);

    // 释放负载缓冲区free(payload_buff); return NETP_NO_PUSH_LIWP;
    }
```

4.5 UDP 报文的上层投递的设计与实现

本练习将主机 A 和 B 作为一组，主机 C 和 D 作为一组，主机 E 和 F 作为一组。现仅以主机 A、B 所在组为例，其它组的操作参考主机 A、B 所在组的操作。实验开始前，先单击“初始环境”。

在实验中，主机 A 将新接口的 IP 地址设置为 172.16.1.11、主机 B 使用处于连接状态的物理接口，将新接口的 IP 设置为 172.16.1.12、主机 C 将新接口的 IP 地址设置为 172.16.1.13、主机 D 使用处于连接状态的物理接口，将新接口的 IP 地址设置为 172.16.1.14、主机 E 使用处于连接状态的物理接口，将新接口的 IP 地址设置为 172.16.1.15、主机 F 将新接口的 IP 地址设置为 172.16.1.16。所有主机使用子网掩码 255.255.255.0，默认网关设置为 0.0.0.0。

1. 编码实现发送 UDP 数据报接口

(a) 各主机打开安装目录ExpCNS\work\EXPcns_student\netproto_udp_student\net proto_udpif_student 下的 netproto_udpif_student.c 文件，在函数 netp_send_udp 内编写实现代码。

(b) 参考实验原理 UDP。发送接口实现推荐流程图给出的流程，分析已经存在的代码。已经存在的代码创建了 UDP 头结构、UDP 伪首部结构、源 IP 地址、负载缓冲区指针和校验和缓冲区指针等变量，申请负载缓冲区、校验和缓冲区，UDP 校验和的计算过程，将 UDP 头和负载拷贝到发送缓冲区中以及利用 IP 接口将缓冲区数据发送到网络中的实现。

(c) 构造 UDP 头，校验和字段为 0。构造并填充一个 UDP 数据包头。可以使用安装目 Ex- pCNS\work\EXPcns_student\netproto_udp_student\netproto_udpif_student 下的 netproto_ udp_student.h 文件中所定义的端口值。总长度为 UDP_HEADER_LEN+ buff_len。

(d) 构造 UDP 伪首部各主机构造并填充 UDP 伪首部。其中，源 IP 地址为本接口 IP 地址， 目的 IP 地址为用户所指定的目的 IP 地址，长度与 UDP 头中的总长度数值相同。

2. 编码实现接收 UDP 数据报接口

(a) 各主机打开 netproto_udpif_student.c 文件，在函数 netp_udp_input_student 内编写实现代码。

(b) 参考实验原理 UDP。接收接口实现推荐流程图给出的流程，分析已经存在的代码。已经存在的代码定义了以太网帧头、IP 包头、UDP 报头、UDP 伪报头、校验和、负载缓冲区指针和校验和缓冲区指针，以太网高层协议的校验，目的 IP 地址的校验，申请负载缓冲区，将负载内容投递到上层协议、释放负载缓冲区等实现。

(c) 提取 UDP 数据包。各主机通过以 IP 头结构 ip_header 中的 protocol 即“高层协议类型”字段值判断该数据帧的高层协议是否为 UDP 协议，可以使用宏 IP_PROTO_UDP。如果不是 UDP 数据包则返回 NETP_PUSH_TO_LWIP 交给协议栈继续执行。

(d) 提取 UDP 目的端口号为 recv_port 的数据报各主机通过 UDP 头结构 udp_header 中的dest_port 即“目的端口号”字段值判断该 UDP 数据报是否是我们感兴趣的 UDP 数据报。如果不是则返回 NETP_PUSH_TO_LWIP 交给协议栈继续执行。

(e) 判断 UDP 校验和是否正确。如果 UDP 头的校验和字段值不为 0，则需要验证校验和是否正确。各主机通过 IP 包头和 UDP 报头中的数值验证 UDP 校验和是否正确，如果不正确则丢弃该数据报。

3. 所有主机打开协议分析器，开始捕获数据。

4. 所有主机调试并运行程序。
5. 各主机停止数据捕获，观察实验现象。

- 你收到的负载内容是什么？

6. 参考代码如下：

```
/**
* \brief 对上层的接口，负责利用 IP 接口，发送一个 UDP 数据报
* 由学生完成这个函数，发送一个 UDP 数据包。udpif 将会调用这个函数。
*
* \param dest_ipaddr 目的 IP 地址
* \param dest_port 目的端口
* \param sour port 源端口
* \param buff 负载缓冲区
* \buff_len 负载长度
*
* \return 0 发送成功，-1 发送失败
*/
int
netp_send_udp(struct ip_addr dest_ipaddr, u16_t dest_port, u16_t sour_port,
void *buff, int buff_len)
{
struct netp_udp_header udp_header;                    // UDP 报头struct netp_udp_pseudo
udp_pseudo;                                           // UDP 伪首部struct ip_addr sour_ipaddr;
                                                      // 源 IP 地址
u8_t *send_buff;                                      // 负载缓冲区指针
u8_t*checksum_buff;                                   // 校验和缓冲区指针

// 申请负载缓冲区、校验和缓冲区
send_buff = malloc(UDP_HEADER_LEN+buff_len);
checksum_buff =malloc(NETP_UDP_PSEUDO_LEN+UDP_HEADER_LEN+buff_len);

// 源 IP 地址设置为本接口 IP 地址
sour_ipaddr.addr = netp_current_ip_addr();

// 构造 UDP 报头
udp_header.sour_port = htons(sour_port); udp_header.dest_port = htons(dest_port);
udp_header.length = (u16_t)htons(UDP_HEADER_LEN+buff_len); udp_header.checksum = 0;

// 构造 UDP 伪报头
udp_pseudo.sour_addr=sour_ipaddr.addr; udp_pseudo.dest_addr=dest_ipaddr.addr;
udp_pseudo.zero = 0; udp_pseudo.protocol = IP_PROTO_UDP; udp_pseudo.length =
```

```
  udp_header.length;

  // 将 udp 伪报头、udp 报头、dup 负载拷贝到校验和缓冲区memcpy(checksum_buff,
  &udp_pseudo, NETP_UDP_PSEUDO_LEN); memcpy(checksum_buff+NETP_UDP_PSEUDO_LEN,
  &udp_header, UDP_HEADER_LEN);

  memcpy(checksum_buff+NETP_UDP_PSEUDO_LEN+UDP_HEADER_LEN, buff,buff_len);

  // 计算校验和
  udp_header.checksum = inet_chksum(checksum_buff, NETP_UDP_PSEUDO_LEN+UDP_HEADER_LEN+(u16_t)b

  free(checksum_buff);

  // 将 udp 报头、udp 负载拷贝到发送缓冲区
  memcpy(send_buff, &udp_header, UDP_HEADER_LEN); memcpy(send_buff+UDP_HEADER_LEN, buff,
  buff_len);

  // 利用 IP 接口发送 UDP 数据报
  if (ERR_OK == netp_ip_payload_output(&sour_ipaddr, &dest_ipaddr,
  NETP_TTL, NETP_DS, IP_PROTO_UDP, send_buff, UDP_HEADER_LEN+buff_len, 0)) {recv_port =
  sour_port;
  free(send_buff);
  return 0;
} else { free(send_buff); return -1;
  }
  }

  /**
* \brief 当有数据帧到达时，将调用这个函数
  *
* \param packet 指向接收到的数据
* \param packet_len 数据帧的长度
  *
* \return 一个 put_to_lwip 类型的返回值。
* 返回 NETP_PUT_LWIP 表示处理数据后将数据帧交给上层协议栈继续处理
* 返回 NETP_NO_PUT_LIWP 表示不将数据帧交给上层协议栈处理。
  */
  enum push_to_lwip
  netp_udp_input_student(void *packet, int packet_len)
```

```
{
struct netp_eth_header eth_header; struct netp_ip_header ip_header; struct
netp_udp_header udp_header; struct netp_udp_pseudo udp_pseudo; u16_t udp_checksum;
char *payload_buff;
u8_t *checksum_buff;
memcpy(&eth_header, packet, ETH_HEADER_LEN);

// 只提取 ip 数据包
if (eth_header.type != htons(MAC_PROTO_IP)){
return NETP_PUSH_TO_LWIP;
}

memcpy(&ip_header, (u8_t*)packet+ETH_HEADER_LEN, IP_HEADER_LEN);
// 过滤掉目的 ip 地址不是本接口的数据包
if (ip_header.destination_address.addr != netp_current_ip_addr()) {
return NETP_PUSH_TO_LWIP;
}

// 只提取 UDP 数据报
if (ip_header.protocol != IP_PROTO_UDP){
return NETP_PUSH_TO_LWIP;
}

memcpy(&udp_header,
^^I      (u8_t*)packet+ETH_HEADER_LEN+IP_HEADER_LEN, UDP_HEADER_LEN);

// 过滤掉目的端口号不是 recv_port 的数据报
if (udp_header.dest_port != htons(recv_port)){
return NETP_PUSH_TO_LWIP;
}

// 判断 UDP 校验和是否正确
if (0 != udp_header.checksum) {
checksum_buff = malloc(NETP_UDP_PSEUDO_LEN+udp_header.length); udp_checksum =
udp_header.checksum;
udp_header.checksum = 0;
udp_pseudo.dest_addr = ip_header.destination_address.addr; udp_pseudo.sour_addr =
ip_header.source_address.addr; udp_pseudo.zero = 0;
```

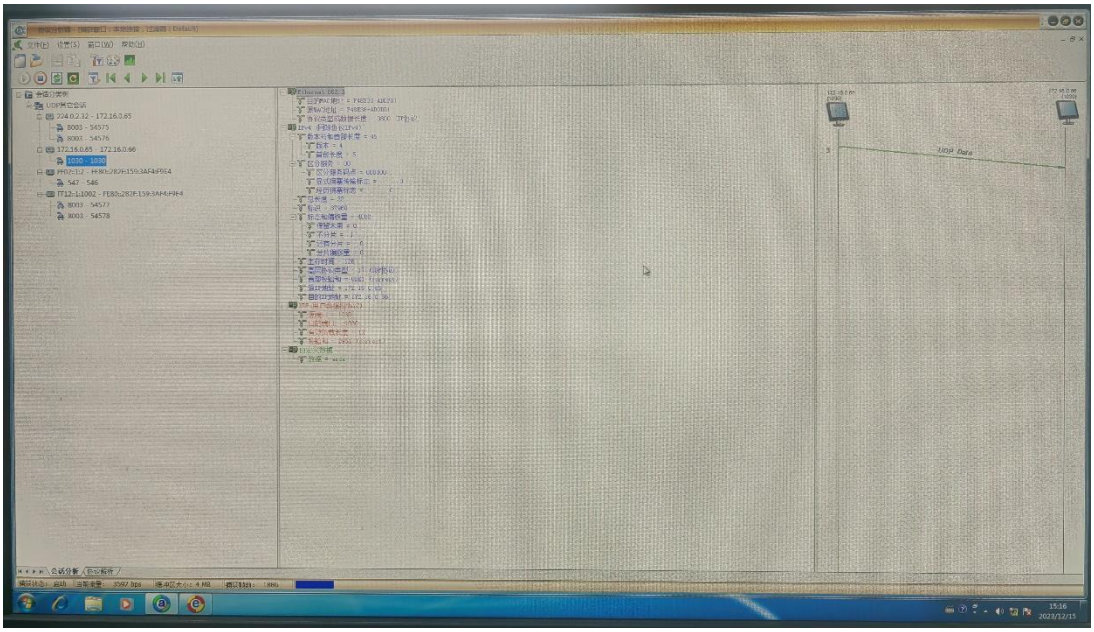
```
    udp_pseudo.protocol = IP_PROTO_UDP; udp_pseudo.length=udp_header.length;

    memcpy(checksum_buff, &udp_pseudo, NETP_UDP_PSEUDO_LEN);
    memcpy(checksum_buff+NETP_UDP_PSEUDO_LEN, &udp_header, UDP_HEADER_LEN);
    memcpy(checksum_buff+NETP_UDP_PSEUDO_LEN+UDP_HEADER_LEN,
    (u8_t*)packet+ETH_HEADER_LEN+IP_HEADER_LEN+UDP_HEADER_LEN,
    ntohs(udp_header.length)-UDP_HEADER_LEN);

  if (udp_checksum != inet_chksum(checksum_buff,
    NETP_UDP_PSEUDO_LEN+ntohs(udp_header.length))) {
    return NETP_PUSH_TO_LWIP;
    }
    free(checksum_buff);
    }

    // 申请负载缓冲区
    payload_buff = malloc(udp_header.length-UDP_HEADER_LEN);

    // 拷贝负载内容到负载缓冲区
  memcpy(payload_buff, (u8_t*)packet+ETH_HEADER_LEN+IP_HEADER_LEN+UDP_HEADER_LEN,
    udp_header.length-UDP_HEADER_LEN);

    // 提交给高层处理
    netp_user_input(&ip_header.source_address, payload_buff,  udp_header.length-UDP_HEADER_LEN);

    // 释放负载缓冲区free(payload_buff); return NETP_NO_PUSH_LIWP;
    }
```

5 实验步骤

5.1 编辑并发送 UDP 数据报

主机 A 编辑报文并发送，在主机 B 上启动协议分析器捕获数据，并设置过滤条件（提取UDP协议）：

主机 B 接收到主机 A 发送的报文：

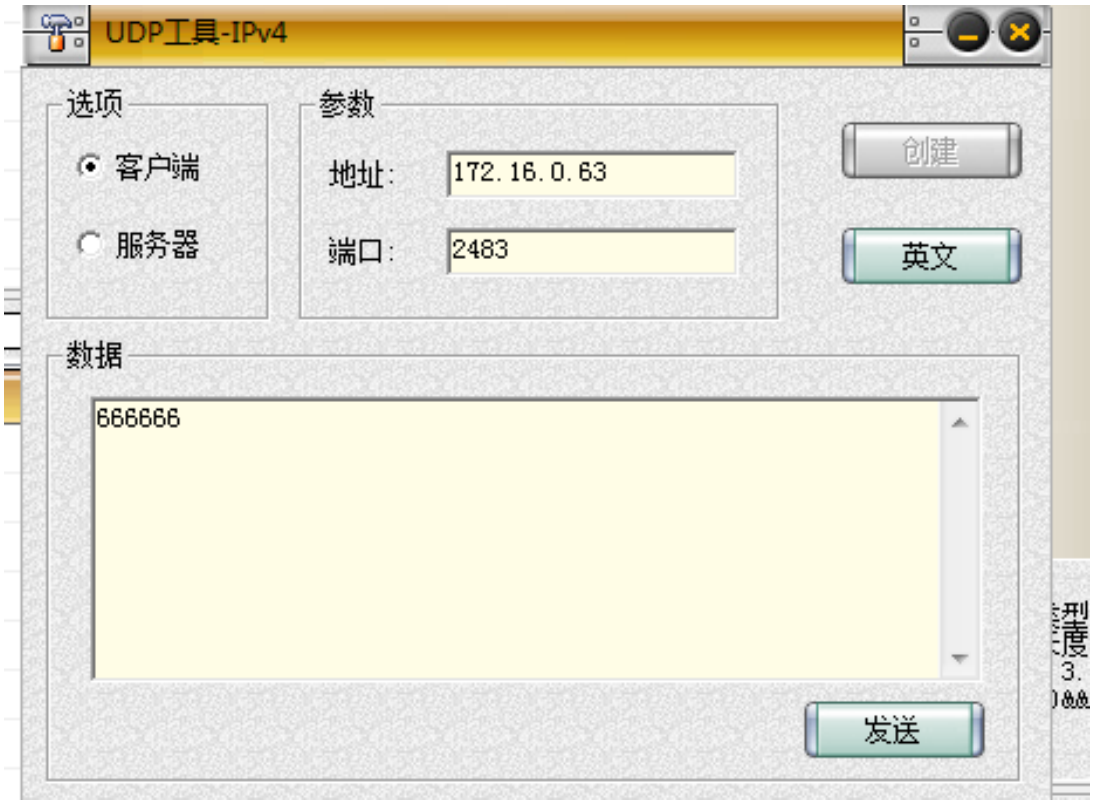


B 接收的数据报

5.2 UDP 单播通信

1. 主机 B、C、D、E、F 上启动实验平台工具栏中的“UDP 工具”，作为服务器端，监听端口设置为 2483，“创建”成功。

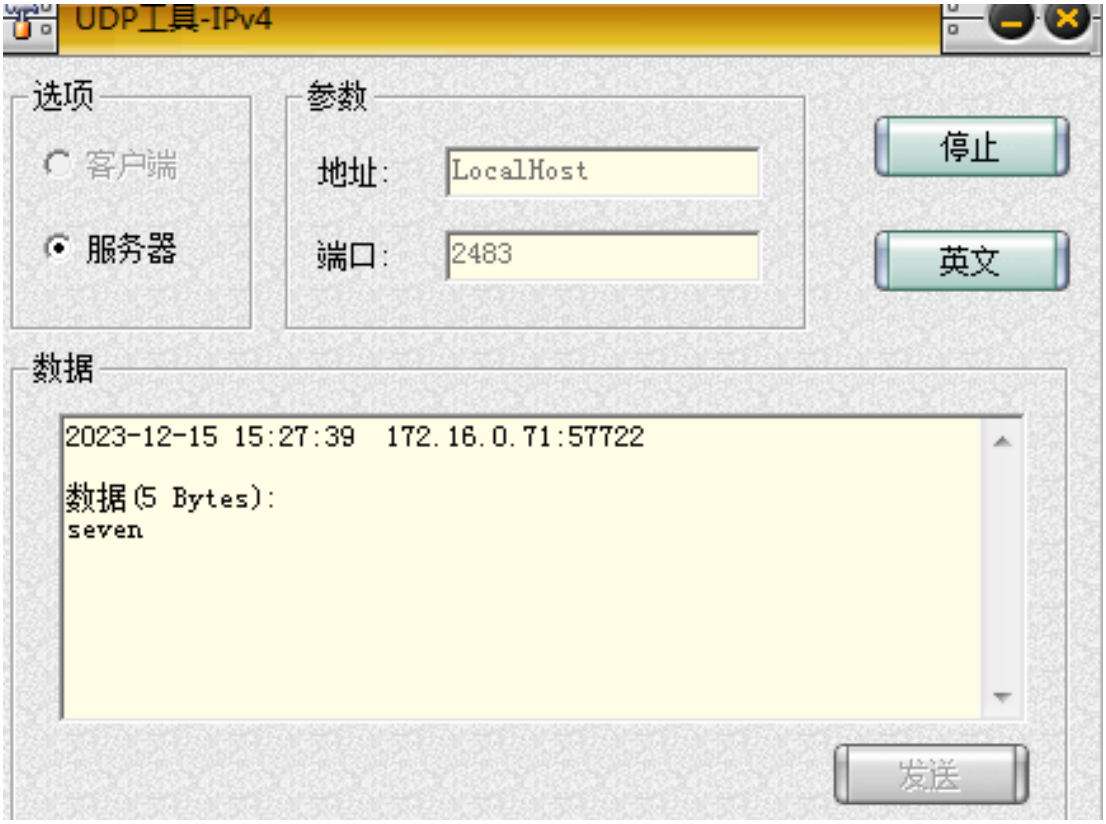
2. 主机 A 上启动实验平台工具栏中的“UDP 工具”，作为客户端，以主机 C 的 IP 为目的IP地址，以 2483 为端口，填写数据并发送。



A 发送的数据

3. 主机 C、E 上启动协议分析器开始捕获数据，并设置过滤条件（提取 UDP 协议）。主机 A 上启动实验平台工具栏中的“UDP 工具”，作为客户端，以主机 C 的 IP 为目的 IP 地址，以 2483 为端口，填写数据并发送。察看主机 B、C、D、E、F 上的“UDP 工具”接收的信息。

4. **察看主机 B、C、D、E、F 上的“UDP 工具”接收的信息。**



回答：观察到只有主机C收到了消息，这是因为该UDP是单播。主机A指定发送到主机C，因此其他主机在收到消息后会丢弃这个帧。特别地，如果协议分析器没有启用过滤，可以看到与丢弃该帧相关的信息。

5. 主机 A 上使用协议编辑器向主机 E 发送 UDP 报文，其中:

目的 MAC 地址:E 的 MAC 地址

目的 IP 地址: 主机 E 的 IP 地址

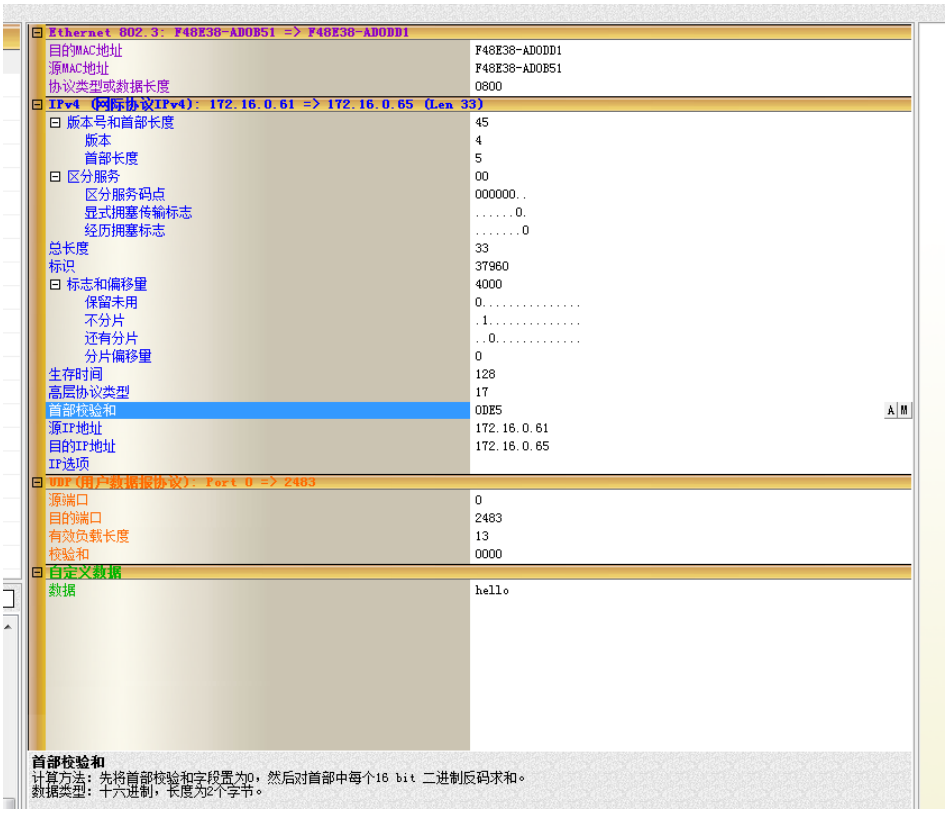
目的端口:2483 校验和: 0

有效负载长度:UDP 层及其上层协议长度

首部校验和: 其它所有字段填充完毕后填充此字段

总长度: 包括 IP 层、UDP 层和数据长度

发送报文：

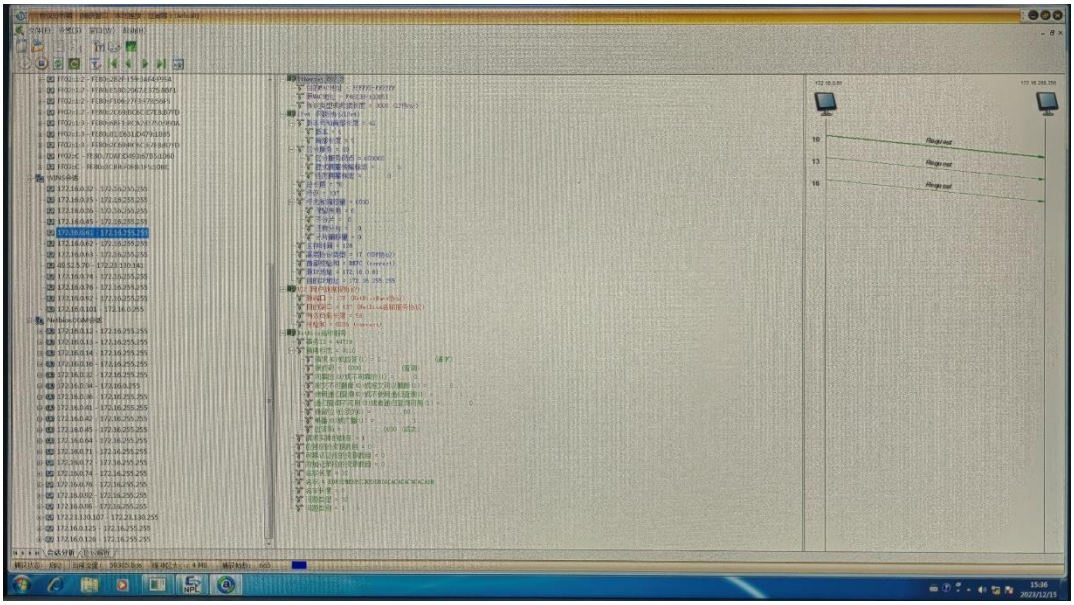


A 发送给 E 的报文

5.3 UDP 广播通信

1. 主机 B、C、D、E、F 上启动 UDP 工具，作为服务器端，监听端口设为 2483。
2. 主机 B、C、D、E、F 启动协议分析器捕获数据，并设置过滤条件（提取 UDP 协议）。
3. 主机 A 上启动 UDP 工具，作为客户端，以 255.255.255.255 为目的地址，以 2483 为端口，填写数据并发送。

主机 A 发送数据



收到的数据

6 实验总结与思考

6.1 编辑并发送 UDP 数据报

1) UDP 在计算校验和时包括哪些内容？

回答：UDP（用户数据报协议）在计算校验和时包括以下内容：

伪首部：UDP计算校验和时将伪首部包括在内。伪首部是由源IP地址、目的IP地址、协议号和UDP长度组成。它提供了足够的信息，使得接收端能够确定这个数据包是属于哪个连接的。

数据部分：UDP校验和还包括数据部分。这是要发送的实际数据，它可能包含多个数据报文。

另外 UDP、TCP 数据报的长度可以为奇数字节，所以在计算校验和时需要最后增加填充字节 0(填充字节只是为了计算校验和，可以不被传送)。

2) 为什么 UDP 协议的校验和要包含伪首部？

回答：UDP（用户数据报协议）在计算校验和时包括以下内容：

UDP协议的校验和包含伪首部的原因是为了确保数据包的完整性和正确性。伪首部包含了源IP地址、目的IP地址、协议号和UDP长度等信息，这些信息对于接收端来说是必要的，以便确定这个数据包是属于哪个连接的。

在UDP传输中，数据可能经过多个路由器和网络接口，每个路由器都会对数据包进行一些修改，例如修改TTL（生存时间）字段、修改数据包

的大小等。如果这些修改导致数据包在到达接收端时与发送端发送的数据包不一致，那么接收端就无法正确地处理这个数据包。

因此，为了确保数据包的完整性和正确性，UDP协议在计算校验和时将伪首部包括在内。这样，即使数据包在传输过程中经过了修改，只要修改的内容不会影响伪首部中的信息，接收端就能够正确地处理这个数据包。

此外，伪首部中的信息还可以帮助接收端进行数据包的排序和重组。由于UDP是无连接的协议，数据包可能会按照不同的路径到达接收端，因此接收端需要按照正确的顺序将数据包重新组合成完整的数据。伪首部中的信息可以帮助接收端确定数据包的顺序和来源，从而正确地重组数据。

3) **比较 UDP 和 IP 的不可靠程度?**

回答：UDP和IP都是不可靠的协议，它们都不保证数据包的可靠性和顺序性。然而，它们的不可靠程度有一些区别。UDP协议的不可靠程度比IP协议低一些，因为UDP协议增加了校验和字段来检测数据包在传输过程中的错误。然而，无论是UDP还是IP协议，它们都不保证数据包的可靠性和顺序性，因此在需要高可靠性的应用中，通常会使用其他协议或技术来确保数据的传输和接收。

6.2 UDP 单播通信

1) **哪台主机上的“UDP 工具”能够接收到主机 A 发送的 UDP 报文?**

回答：主机 C。

2) **UDP 是基于连接的协议吗？阐述此特性的优缺点。**

回答：不是。

优点：相对于面向连接的服务，UDP 传送数据较快速，相对简单，系统开销也少；

缺点：不能防止报文的丢失、重复和乱序。由于它的每个报文必须包括完整的源地址的目的地址，因此开销较大。并且UDP的数据包大小限制在64KB以下，不适合传输大量数据。

3) **UDP 报文交互中含有确认报文吗？阐述此特性的优缺点。**

回答：没有。

优点：无需连接确认，简单，快速灵活；

缺点：发送端将报文发送出去以后就不知道报文的具体情况，不能防

止报文的丢失、重复和乱序，并且缺乏错误控制。

4) **主机 E 上的 UDP 通信程序是否接收到此数据包？UDP 是否可以使用 0 作为校验和进行通信？**

回答：是；可以。

5) **思考 UDP 的差错处理能力**

回答：UDP仅提供有限形式的差错检查，没有纠正差错的能力。

6.3 UDP 广播通信

1) **哪台主机能够接收到主机 A 发送的 UDP 报文？**

回答：所有主机都可以。

2) **主机 A 发送的报文的目的 MAC 地址和目的 IP 地址的含义是什么？**

回答：MAC 地址为FF-FF-FF-FF-FF-FF，表示广播。

IP 地址为255.255.255.255，表示广播。

3) **如果将目的 MAC 地址从广播地址换成某一个主机的 MAC 地址，是否所有的主机还会受到这种报文？**

回答：如果将目的MAC地址从广播地址换成某一个主机的MAC地址，那么只有该主机会收到这种报文，其他主机不会收到。这是因为MAC地址是用于在局域网中唯一标识网络设备的，广播地址是用于向局域网中的所有设备发送广播消息的。

4) **如果将目的 MAC 地址设置成广播地址，目的 IP 设置成某一主机的 IP 地址，结果怎样？**

回答：所有主机都能接收到报文，但当下层服务将报文向上层递交时，其他主机如果发现报文的IP地址与自己的IP地址不匹配，就会将报文丢弃。只有目标主机，即IP地址与报文匹配的主机，会继续将报文递交到应用层。

5) **在可靠性不是最重要的情况下，UDP 可能是一个好的传输协议，试给出这种特定情况的一些示例。**

回答：

1.实时应用：对于需要实时性的应用，如在线游戏、视频会议等，UDP可能是一个更好的选择。由于UDP不提供确认机制，因此它可以更快地传输数据，减少延迟。在这种情况下，即使数据包丢失或乱序到达，也不会对实时性产生太大影响。

2.流媒体传输：对于流媒体传输，如音频或视频流，UDP也是一个不错的选择。由于UDP不保证数据的可靠传输，因此它可以更快速地传输数据，而不需要等待确认。这对于流媒体传输来说是重要的，因为实时性是关键。

3.小数据包传输：对于传输小数据包的应用，如DNS查询、VoIP通话等，UDP也是一个不错的选择。由于UDP的报文头较小，因此它可以更高效地传输小数据包。在这种情况下，即使数据包丢失或乱序到达，也不会对应用产生太大影响。

6) **UDP 协议本身是否能确保数据报的发送和接受顺序？**

回答：不能。

UDP（用户数据报协议）是一种无连接的协议，它在传输层提供了一种面向事务的简单的不可靠信息传送服务。它只提供最基本的错误检测，但不提供任何可靠性保证。因此，UDP协议本身不能确保数据报的发送和接受顺序。

如果需要确保数据的顺序，可以在应用层进行排序操作，或者使用其他机制如TCP等来进行可靠性传输。

6.4 UDP 数据报发送与接收

1) **你收到的负载内容是什么？**

回答：负载内容取决于代码中写的宏定义。

6.5 UDP 报文的上层投递的设计与实现

1) **你收到的负载内容是什么？**

回答：与上一节相同，负载内容即为代码中写的宏定义。