《计算机网络》
实验指导书

李全龙 聂兰顺

哈尔滨工业大学

计算机科学与技术学院

2018 年 4 月 20 日

目录

前言

《计算机网络》课程是计算机科学与技术、软件工程、物联网工程等专业的核心课程之一。随着互联网、移动通信、移动互联网以及物联网等技术的发展，计算机网络技术已成为信息技术的重要基础，是IT从业者必备的专业技术与专业技能。计算机网络技术已在社会、经济、文化等各个领域得到广泛的应用，已经渗透到我们的工作、学习、生活、休闲等活动之中，逐渐成为现代人的“第五基本需求”。在互联网+的时代，对于《计算机网络》课程教学提出了新的、更高的要求。

首先，《计算机网络》需要课程教学内容与国际一流大学接轨，反映最新网络技术与发展趋势。这一点已通过采用最新的国外教材，实时更新教学内容等得以实现。其次，《计算机网络》作为一门实践性较强的课程，在确保优良的课堂教学效果基础上，必须加强实践教学。为此，《计算机网络》课程实验从2015年开始全面更新实验题目，无论是实验内容还是实验难度均大幅提升，已接近或达到国外一流大学相同课程的实验水平。

本指导书是针对更新后的实验题目、目的、内容、要求、步骤等的详细描述，供同学们在实验过程中参考使用。由于首次更新实验内容，因此，其中的不足或错误在所难免，希望同学们在使用本实验指导书及进行实验的过程中，能够帮助我们不断地发现问题，并提出建议，以便我们进一步改进。

实验要求

计算机网络是现代信息社会最重要的基础设施之一。在过去几十年里得到了迅速的发展和应用。《计算机网络》课程实验的目的是为了使学生在课程学习的基础上，通过一系列的实验是学生深入掌握典型的网络协议、Socket编程技术、可靠数据传输、协议分析、路由器工作原理、网络设计与配置等核心实践技术与技能。总之，通过实验，将使学生更加深入了解和掌握《计算机网络》课程的基本原理、典型协议、典型网络以及开发技术等内容。

在《计算机网络》的课程实验过程中，要求学生做到：

（1）每次实验前应预习实验指导书有关内容，认真做好实验准备，要提前熟悉必要的开发环境或实验用工具软件，提前完成系统设计或部分编码工作。

（2）仔细观察实验过程出现的各种现象，认真记录实验数据和实验结果，并对各种实验结果和现象进行必要的说明、解释与分析。

（3）认真撰写实验报告。实验报告应包括实验目的、实验要求、实验过程、实验现象与实验结果以及实验结果/现象分析等。对于需要编程的实验，需要写出程序设计说明，绘制程序框图，给出具有详细注释的源程序清单。

（4）遵守实验室的相关规定，按时到指定实验室进行实验，服从辅导教师指挥与安排，爱护实验设备。

（5）实验课程不迟到、不早退。如有事不能出席，所缺实验须自行补做。

实验的验收将分为两个部分：第一部分是实验操作，包括检查程序运行和即时提问等；第二部分是提交电子版的实验报告。此外，网络实验采用**当堂检查**方式，每个实验都应当在规定的时间内完成并检查通过，过期视为未完成该实验，不计成绩。以避免集中检查方式产生的诸多不良问题，希望同学们抓紧时间，合理安排，认真完成。

实验 1：HTTP 代理服务器的设计与实现

1. 实验目的

熟悉并掌握 Socket 网络编程的过程与技术；深入理解 HTTP 协议，掌握 HTTP 代理服务器的基本工作原理；掌握 HTTP 代理服务器设计与编程实现的基本技能。

2. 实验环境

- 接入 Internet 的实验主机；
- Windows xp 或 Windows 7/8；
- 开发语言：C/C++（或 Java）等。

3. 实验内容

(1) 设计并实现一个基本 HTTP 代理服务器。要求在指定端口（例如 **8080**）接收来自客户的 HTTP 请求并且根据其中的 URL 地址访问该地址所指向的 HTTP 服务器（原服务器），接收 HTTP 服务器的响应报文，并将响应报文转发给对应的客户进行浏览。

(2) 设计并实现一个支持 Cache 功能的 HTTP 代理服务器。要求能缓存原服务器响应的对象，并能够通过修改请求报文（添加 if-modified-since 头行），向原服务器确认缓存对象是否是最新版本。（选作内容，加分项目，可以当堂完成或课下完成）

(3) 扩展 HTTP 代理服务器，支持如下功能：（选作内容，加分项目，可以当堂完成或课下完成）

a) 网站过滤：允许/不允许访问某些网站；

b) 用户过滤：支持/不支持某些用户访问外部网站；

c) 网站引导：将用户对某个网站的访问引导至一个模拟网站（钓鱼）。

4. 实验步骤

（1）浏览器使用代理

为了使浏览器访问网址时通过代理服务器，必须进行相关设置，以 IE 浏览器设置为例：打开浏览器→工具→浏览器选项→连接→局域网设置→代理服务器，具体过程如图 1-1 所示。

图 1-1 浏览器的代理服务器设置

（2）多线程使用

使用函数_beginthreadex 创建子线程，使用函数_endthreadex 结束线程，详情见 CSDN。

5. 实验方式

每位同学独立上机编程实验，实验指导教师现场指导。

6. 参考内容

代理服务器，俗称“翻墙软件”，允许一个网络终端（一般为客户端）通过这个服务与另一个网络终端（一般为服务器）进行非直接的连接。如图 1-2 所示，为普通 Web 应用通信方式与采用代理服务器的通信方式的对比。

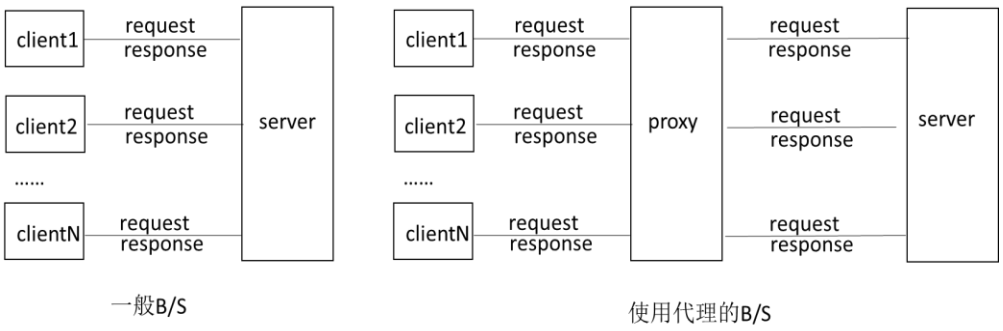

图 1-2 Web 应用通信方式对比

代理服务器在指定端口（例如 **8080**）监听浏览器的访问请求（需要在客户端浏览器进行相应的设置），接收到浏览器对远程网站的浏览请求时，代理服务器开始在代理服务器的缓存中检索 URL 对应的对象（网页、图像等对象），找到对象文件后，提取该对象文件的最新被修改时间；代理服务器程序在客户的请求报文首部插入<If-Modified-Since: 对象文件的最新被修改时间>，并向原 Web 服务器转发修改后的请求报文。如果代理服务器没有该对象的缓存，则会直接向原服务器转发请求报文，并将原服务器返回的响应直接转发给客户端，同时将对象缓存到代理服务

器中。代理服务器程序会根据缓存的时间、大小和提取记录等对缓存进行清理。

本实验需实现一个简单的 HTTP 代理服务器，可以分为两个步骤：（首先请设置浏览器开启本地代理，注意设置代理端口与代理服务器监听端口保持一致）。

a) 单用户代理服务器

单用户的简单代理服务器可以设计为一个非并发的循环服务器。首先，代理服务器创建 HTTP 代理服务的 TCP 主套接字，通过该主套接字监听等待客户端的连接请求。当客户端连接之后，读取客户端的 HTTP 请求报文，通过请求行中的 URL，解析客户期望访问的原服务器 IP 地址；创建访问原（目标）服务器的 TCP 套接字，将 HTTP 请求报文转发给目标服务器，接收目标服务器的响应报文，当收到响应报文之后，将响应报文转发给客户端，最后关闭套接字，等待下一次连接。

b) 多用户代理服务器

多用户的简单代理服务器可以实现为一个多线程并发服务器。首先，代理服务器创建 HTTP 代理服务的 TCP 主套接字，通过该主套接字监听等待客户端的连接请求。当客户端连接之后，创建一个子线程，由子线程执行上述一对一的代理过程，服务结束之后子线程终止。与此同时，主线程继续接受下一个客户的代理服务。

参考代码

```
#include "stdafx.h"
#include <stdio.h>
#include <Windows.h>
#include <process.h>
#include <string.h>

#pragma comment(lib,"Ws2_32.lib")

#define MAXSIZE 65507 //发送数据报文的最大长度
#define HTTP_PORT 80 //http 服务器端口

//Http 重要头部数据
struct HttpHeader{
```

```
        char method[4]; // POST 或者 GET，注意有些为 CONNECT，本实验暂不考虑
        char url[1024];       // 请求的 url
        char host[1024]; //  目标主机
        char cookie[1024 * 10]; //cookie
        HttpHeader(){
            ZeroMemory(this,sizeof(HttpHeader));
        }
    };

    BOOL InitSocket();
    void ParseHttpHead(char *buffer,HttpHeader * httpHeader);
    BOOL ConnectToServer(SOCKET *serverSocket,char *host);
    unsigned int __stdcall ProxyThread(LPVOID lpParameter);

    //代理相关参数
    SOCKET ProxyServer;
    sockaddr_in ProxyServerAddr;
    const int ProxyPort = 10240;

    //由于新的连接都使用新线程进行处理，对线程的频繁的创建和销毁特别浪费资源
    //可以使用线程池技术提高服务器效率
    //const int ProxyThreadMaxNum = 20;
    //HANDLE ProxyThreadHandle[ProxyThreadMaxNum] = {0};
    //DWORD ProxyThreadDW[ProxyThreadMaxNum] = {0};

    struct ProxyParam{
        SOCKET clientSocket;
        SOCKET serverSocket;
    };

    int _tmain(int argc, _TCHAR* argv[])
    {
        printf("代理服务器正在启动\n");
        printf("初始化...\n");
        if(!InitSocket()){
```

```
            printf("socket 初始化失败\n");
            return -1;
        }
        printf("代理服务器正在运行，监听端口 %d\n",ProxyPort);
        SOCKET acceptSocket = INVALID_SOCKET;
        ProxyParam *lpProxyParam;
        HANDLE hThread;
        DWORD dwThreadID;
        //代理服务器不断监听
        while(true){
            acceptSocket = accept(ProxyServer,NULL,NULL);
            lpProxyParam = new ProxyParam;
            if(lpProxyParam == NULL){
                continue;
            }
            lpProxyParam->clientSocket = acceptSocket;
            hThread = (HANDLE)_beginthreadex(NULL, 0,
&ProxyThread,(LPVOID)lpProxyParam, 0, 0);
            CloseHandle(hThread);
            Sleep(200);
        }
        closesocket(ProxyServer);
        WSACleanup();
        return 0;
    }


    //************************************
    // Method:       InitSocket
    // FullName:    InitSocket
    // Access:       public
    // Returns:     BOOL
    // Qualifier:  初始化套接字
    //************************************
    BOOL InitSocket(){

        //加载套接字库（必须）
```

```
        WORD wVersionRequested;
        WSADATA wsaData;
        //套接字加载时错误提示
        int err;
        //版本 2.2
        wVersionRequested = MAKEWORD(2, 2);
        //加载 dll 文件 Scoket 库
        err = WSAStartup(wVersionRequested, &wsaData);
        if(err != 0){
            //找不到 winsock.dll
            printf("加载 winsock 失败，错误代码为: %d\n", WSAGetLastError());
            return FALSE;
        }
        if(LOBYTE(wsaData.wVersion) != 2 || HIBYTE(wsaData.wVersion) !=2)
        {
            printf("不能找到正确的 winsock 版本\n");
            WSACleanup();
            return FALSE;
        }
        ProxyServer= socket(AF_INET, SOCK_STREAM, 0);
        if(INVALID_SOCKET == ProxyServer){
            printf("创建套接字失败，错误代码为：%d\n",WSAGetLastError());
            return FALSE;
        }
        ProxyServerAddr.sin_family = AF_INET;
        ProxyServerAddr.sin_port = htons(ProxyPort);
        ProxyServerAddr.sin_addr.S_un.S_addr = INADDR_ANY;
        if(bind(ProxyServer,(SOCKADDR*)&ProxyServerAddr,sizeof(SOCKADD
R)) == SOCKET_ERROR){
            printf("绑定套接字失败\n");
            return FALSE;
        }
        if(listen(ProxyServer, SOMAXCONN) == SOCKET_ERROR){
            printf("监听端口%d 失败",ProxyPort);
            return FALSE;
        }
        return TRUE;
```

```
    }

    //*************************************
    // Method:       ProxyThread
    // FullName:    ProxyThread
    // Access:       public
    // Returns:     unsigned int __stdcall
    // Qualifier:  线程执行函数
    // Parameter: LPVOID lpParameter
    //*************************************
    unsigned int __stdcall ProxyThread(LPVOID lpParameter){
        char Buffer[MAXSIZE];
        char *CacheBuffer;
        ZeroMemory(Buffer,MAXSIZE);
        SOCKADDR_IN clientAddr;
        int length = sizeof(SOCKADDR_IN);
        int recvSize;
        int ret;
        recvSize = recv(((ProxyParam
*)lpParameter)->clientSocket,Buffer,MAXSIZE,0);
        if(recvSize <= 0){
            goto error;
        }
        HttpHeader* httpHeader = new HttpHeader();
        CacheBuffer = new char[recvSize + 1];
        ZeroMemory(CacheBuffer,recvSize + 1);
        memcpy(CacheBuffer,Buffer,recvSize);
        ParseHttpHead(CacheBuffer,httpHeader);
        delete CacheBuffer;
        if(!ConnectToServer(&((ProxyParam
*)lpParameter)->serverSocket,httpHeader->host)) {
            goto error;
        }
        printf("代理连接主机  %s  成功\n",httpHeader->host);
        //将客户端发送的 HTTP 数据报文直接转发给目标服务器
        ret = send(((ProxyParam *)lpParameter)->serverSocket,Buffer,strlen(Buffer)
+ 1,0);
```

```
        //等待目标服务器返回数据
        recvSize = recv(((ProxyParam
*)lpParameter)->serverSocket,Buffer,MAXSIZE,0);
        if(recvSize <= 0){
            goto error;
        }
        //将目标服务器返回的数据直接转发给客户端
        ret = send(((ProxyParam
*)lpParameter)->clientSocket,Buffer,sizeof(Buffer),0);
        //错误处理
    error:
        printf("关闭套接字\n");
        Sleep(200);
        closesocket(((ProxyParam*)lpParameter)->clientSocket);
        closesocket(((ProxyParam*)lpParameter)->serverSocket);
        delete    lpParameter;
        _endthreadex(0);
        return 0;
    }

    //*************************************
    // Method:       ParseHttpHead
    // FullName:    ParseHttpHead
    // Access:       public
    // Returns:     void
    // Qualifier:  解析 TCP 报文中的 HTTP 头部
    // Parameter: char * buffer
    // Parameter: HttpHeader * httpHeader
    //*************************************
    void ParseHttpHead(char *buffer,HttpHeader * httpHeader){
        char *p;
        char *ptr;
        const char * delim = "\r\n";
        p = strtok_s(buffer,delim,&ptr);//提取第一行
        printf("%s\n",p);
        if(p[0] == 'G'){//GET 方式
            memcpy(httpHeader->method,"GET",3);
```

```
            memcpy(httpHeader->url,&p[4],strlen(p) -13);
        }else if(p[0] == 'P'){//POST 方式
            memcpy(httpHeader->method,"POST",4);
            memcpy(httpHeader->url,&p[5],strlen(p) - 14);
        }
        printf("%s\n",httpHeader->url);
        p = strtok_s(NULL,delim,&ptr);
        while(p){
            switch(p[0]){
            case 'H'://Host
                memcpy(httpHeader->host,&p[6],strlen(p) - 6);
                break;
            case 'C'://Cookie
                if(strlen(p) > 8){
                    char header[8];
                    ZeroMemory(header,sizeof(header));
                    memcpy(header,p,6);
                    if(!strcmp(header,"Cookie")){
                        memcpy(httpHeader->cookie,&p[8],strlen(p) -8);
                    }
                }
                break;
            default:
                break;
            }
            p = strtok_s(NULL,delim,&ptr);
        }
    }

    //*************************************
    // Method:       ConnectToServer
    // FullName:    ConnectToServer
    // Access:       public
    // Returns:     BOOL
    // Qualifier:  根据主机创建目标服务器套接字，并连接
    // Parameter: SOCKET * serverSocket
    // Parameter: char * host
```

```
//***********************************
BOOL ConnectToServer(SOCKET *serverSocket,char *host){
    sockaddr_in serverAddr;
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_port = htons(HTTP_PORT);
    HOSTENT *hostent = gethostbyname(host);
    if(!hostent){
        return FALSE;
    }
    in_addr Inaddr=*( (in_addr*) *hostent->h_addr_list);
    serverAddr.sin_addr.s_addr = inet_addr(inet_ntoa(Inaddr));
    *serverSocket = socket(AF_INET,SOCK_STREAM,0);
    if(*serverSocket == INVALID_SOCKET){
        return FALSE;
    }
    if(connect(*serverSocket,(SOCKADDR *)&serverAddr,sizeof(serverAddr))
== SOCKET_ERROR){
        closesocket(*serverSocket);
        return FALSE;
    }
    return TRUE;
}
```

7. 实验报告

在实验报告中需要总结说明：

（1）Socket 编程的客户端和服务器端主要步骤；

（2）HTTP 代理服务器的基本原理；

（3）HTTP 代理服务器的程序流程图；

（4）实现 HTTP 代理服务器的关键技术及解决方案；

（5）HTTP 代理服务器实验验证过程以及实验结果；

（6）HTTP 代理服务器源代码（带有详细注释）。

实验 2：可靠数据传输协议-停等协议的设计与实现

1. 实验目的

理解可靠数据传输的基本原理；掌握停等协议的工作原理；掌握基于 UDP 设计并实现一个停等协议的过程与技术。

2. 实验环境

- 接入 Internet 的实验主机；
- Windows xp 或 Windows 7/8；
- 开发语言：C/C++（或 Java）等。

3. 实验内容

1) 基于 UDP 设计一个简单的停等协议，实现单向可靠数据传输（服务器到客户的数据传输）。

2) 模拟引入数据包的丢失，验证所设计协议的有效性。

3) 改进所设计的停等协议，支持双向数据传输；（选作内容，加分项目，可以当堂完成或课下完成）

4）基于所设计的停等协议，实现一个 C/S 结构的文件传输应用。（选作内容，加分项目，可以当堂完成或课下完成）

4. 实验方式

每位同学独立上机编程实验，实验指导教师现场指导。

5. 实验要点

1) 基于 UDP 实现的停等协议，可以不进行差错检测，可以利用 UDP 协议差错检测；

2) 为了验证所设计协议是否可以处理数据丢失，可以考虑在数据接收端或发送端引入数据丢失。

3) 在开发停等协议之前，需要先设计协议数据分组格式以及确认分组格式。

4) 计时器实现方法：对于阻塞的 socket 可用 int setsockopt(int socket, int level, int option_name, const void* option_value, size_t option_len)函数设置套接字发送与接收超时时间；对于非阻塞 socket 可以使用累加 sleep 时间的方法判断 socket 接受数据是否超时(当时间累加量超过一定数值时则认为套接字接受数据超时)。

6. 实验报告

在实验报告中要说明所设计停等协议数据分组格式、确认分组格式、各个域作用，协议两端程序流程图，协议典型交互过程，数据分组丢失验证模拟方法，程序实现的主要类（或函数）及其主要作用、UDP 编程的主要特点、实验验证结果，详细注释源程序等。

实验 3：可靠数据传输协议-GBN 协议的设计与实现

1. 实验目的

理解滑动窗口协议的基本原理；掌握 GBN 的工作原理；掌握基于 UDP 设计并实现一个 GBN 协议的过程与技术。

2. 实验环境

- 接入 Internet 的实验主机；
- Windows xp 或 Windows 7/8；
- 开发语言：C/C++（或 Java）等。

3. 实验内容

1) 基于 UDP 设计一个简单的 GBN 协议，实现单向可靠数据传输（服务器到客户的数据传输）。

2) 模拟引入数据包的丢失，验证所设计协议的有效性。

3) 改进所设计的 GBN 协议，支持双向数据传输；（选作内容，加分项目，可以当堂完成或课下完成）

4）将所设计的 GBN 协议改进为 SR 协议。（选作内容，加分项目，可以当堂完成或课下完成）

4. 实验方式

每位同学独立上机编程实验，实验指导教师现场指导。

5. 实验要点

1) 基于 UDP 实现的 GBN 协议，可以不进行差错检测，可以利用 UDP 协议差错检测；

2) 自行设计数据帧的格式，应至少包含序列号 Seq 和数据两部分；

3) 自行定义发送端序列号 Seq 比特数 L 以及发送窗口大小 W，应满足条件 $W+1<=2^L$。

4) 一种简单的服务器端计时器的实现办法：设置套接字为非阻塞方式，则服务器端在 recvfrom 方法上不会阻塞，若正确接收到 ACK 消息，则计时器清零，若从客户端接收数据长度为-1（表示没有接收到任何数据），则计时器+1，对计时器进行判断，若其超过阈值，则判断为超时，进行超时重传。（当然，如果服务器选择阻塞模式，可以用到 select 或 epoll 的阻塞选择函数，详情见 MSDN）

5) 为了模拟 ACK 丢失，一种简单的实现办法：客户端对接收的数据帧进行计数，然后对总数进行模 N 运算，若规定求模运算结果为零则返回 ACK，则每接收 N 个数据帧才返回 1 个 ACK。当 N 取值大于服务器端的超时阀值时，则会出现服务器端超时现象。

6) 当设置服务器端发送窗口的大小为 1 时，GBN 协议就是停-等协议。

6. 参考内容

作为只实现单向数据传输的 GBN 协议，实质上就是实现为一个 C/S 应用。

服务器端：使用 UDP 协议传输数据（比如传输一个文件），等待客户端的请求，接收并处理来自客户端的消息（如数据传输请求），当客户端开始请求数据时进入“伪连接”状态（并不是真正的连接，只是一种类似连接的数据发送的状态），将数据打包成数据报发送，然后等待客户端的 ACK 信息，同时启动计时器。当收到 ACK 时，窗口滑动，正常发送下一个数据报，计时器重新计时；若在计时器超时前没有收到 ACK，则全部重传窗口内的所以已发送的数据报。

客户端：使用 UDP 协议向服务器端请求数据，接收服务器端发送的数据报并返回确认信息 ACK（注意 GBN 为累积确认，即若 ACK=1 和 3，表示数据帧 2 已经正确接收），必须能够模拟 ACK 丢失直至服务器端超

时重传的情况。

(1) 服务器端设计参考

1）命令解析

为了测试客户端与服务器端的通信交互，方便操作，设置了此过程。首先，服务器接收客户端发来的请求数据，

“-time”表示客户端请求获取当前时间，服务器回复当前时间；

“-quit”表示客户端退出，服务器回复“Good bye!”；

“-testgbn”表示客户端请求开始测试 GBN 协议，服务器开始进入 GBN 传输状态；

其他数据，则服务器直接回复原数据。

2）数据传输数据帧格式定义

在以太网中，数据帧的 MTU 为 1500 字节，所以 UDP 数据报的数据部分应小于 1472 字节（除去 IP 头部 20 字节与 UDP 头的 8 字节），为此，定义 UDP 数据报的数据部分格式为：

Seq	Data	0

Seq 为 1 个字节，取值为 0~255，（故序列号最多为 256 个）；

Data≤1024 个字节，为传输的数据；

最后一个字节放入 EOF0，表示结尾。

3）源代码

```
#include "stdafx.h" //创建 VS 项目包含的预编译头文件
#include <stdlib.h>
#include <time.h>
#include <WinSock2.h>
#include <fstream>

#pragma comment(lib,"ws2_32.lib")

#define SERVER_PORT       12340     //端口号
#define SERVER_IP         "0.0.0.0"  //IP 地址
const int BUFFER_LENGTH = 1026;        //缓冲区大小，（以太网中 UDP 的数据帧中包长度应小于 1480 字节）
```

```
    const int SEND_WIND_SIZE = 10;//发送窗口大小为 10，GBN 中应满足 W + 1 <= N（W 为发送窗口大小，N 为序列号个数）
                                  //本例取序列号 0...19 共 20 个
                                  //如果将窗口大小设为 1，则为停-等协议

    const int SEQ_SIZE = 20; //序列号的个数，从 0~19 共计 20 个
                             //由于发送数据第一个字节如果值为 0，则数据会发送失败
                             //因此接收端序列号为 1~20，与发送端一一对应

    BOOL ack[SEQ_SIZE];//收到 ack 情况，对应 0~19 的 ack
    int curSeq;//当前数据包的 seq
    int curAck;//当前等待确认的 ack
    int totalSeq;//收到的包的总数
    int totalPacket;//需要发送的包总数

    //*************************************
    // Method:       getCurTime
    // FullName:    getCurTime
    // Access:       public
    // Returns:     void
    // Qualifier:  获取当前系统时间，结果存入 ptime 中
    // Parameter: char * ptime
    //*************************************
    void getCurTime(char *ptime){
        char buffer[128];
        memset(buffer,0,sizeof(buffer));
        time_t c_time;
        struct tm *p;
        time(&c_time);
        p = localtime(&c_time);
        sprintf_s(buffer,"%d/%d/%d %d:%d:%d",
            p->tm_year + 1900,
            p->tm_mon,
            p->tm_mday,
            p->tm_hour,
            p->tm_min,
```

```
            p->tm_sec);
        strcpy_s(ptime,sizeof(buffer),buffer);
    }

    //*************************************
    // Method:       seqIsAvailable
    // FullName:    seqIsAvailable
    // Access:       public
    // Returns:      bool
    // Qualifier:  当前序列号 curSeq 是否可用
    //*************************************
    bool seqIsAvailable(){
        int step;
        step = curSeq - curAck;
        step = step >= 0 ? step : step + SEQ_SIZE;
        //序列号是否在当前发送窗口之内
        if(step >= SEND_WIND_SIZE){
            return false;
        }
        if(ack[curSeq]){
            return true;
        }
        return false;
    }

    //*************************************
    // Method:       timeoutHandler
    // FullName:    timeoutHandler
    // Access:       public
    // Returns:      void
    // Qualifier:  超时重传处理函数，滑动窗口内的数据帧都要重传
    //*************************************
    void timeoutHandler(){
        printf("Timer out error.\n");
        int index;
        for(int i = 0;i< SEND_WIND_SIZE;++i){
            index = (i + curAck) % SEQ_SIZE;
```

```
                ack[index] = TRUE;
            }
            totalSeq -= SEND_WIND_SIZE;
            curSeq = curAck;
        }

        //*************************************
        // Method:       ackHandler
        // FullName:    ackHandler
        // Access:       public
        // Returns:      void
        // Qualifier: 收到 ack，累积确认，取数据帧的第一个字节
        //由于发送数据时，第一个字节（序列号）为 0（ASCII）时发送失败，因此加一了，此处需要减一还原
        // Parameter: char c
        //*************************************
        void ackHandler(char c){
            unsigned char index = (unsigned char)c - 1; //序列号减一
            printf("Recv a ack of %d\n",index);
            if(curAck <= index){
                for(int i= curAck; i <= index;++i){
                    ack[i] = TRUE;
                }
                curAck = (index + 1) % SEQ_SIZE;
            }else{
                //ack 超过了最大值，回到了 curAck 的左边
                for(int i = curAck;i< SEQ_SIZE;++i){
                    ack[i] = TRUE;
                }
                for(int i = 0; i<= index;++i){
                    ack[i] = TRUE;
                }
                curAck = index + 1;
            }
        }

        //主函数
```

```
int main(int argc, char* argv[])
{
    //加载套接字库（必须）
    WORD wVersionRequested;
    WSADATA wsaData;
    //套接字加载时错误提示
    int err;
    //版本 2.2
    wVersionRequested = MAKEWORD(2, 2);
    //加载 dll 文件 Scoket 库
    err = WSAStartup(wVersionRequested, &wsaData);
    if(err != 0){
        //找不到 winsock.dll
        printf("WSAStartup failed with error: %d\n", err);
        return -1;
    }
    if(LOBYTE(wsaData.wVersion) != 2 || HIBYTE(wsaData.wVersion) !=2)
    {
        printf("Could not find a usable version of Winsock.dll\n");
        WSACleanup();
    }else{
        printf("The Winsock 2.2 dll was found okay\n");
    }
    SOCKET sockServer = socket(AF_INET, SOCK_DGRAM,IPPROTO_UDP);
    //设置套接字为非阻塞模式
    int iMode = 1; //1：非阻塞，0：阻塞
    ioctlsocket(sockServer,FIONBIO, (u_long FAR*) &iMode);//非阻塞设置
    SOCKADDR_IN addrServer;       //服务器地址
    //addrServer.sin_addr.S_un.S_addr = inet_addr(SERVER_IP);
    addrServer.sin_addr.S_un.S_addr = htonl(INADDR_ANY);//两者均可
    addrServer.sin_family = AF_INET;
    addrServer.sin_port = htons(SERVER_PORT);
    err = bind(sockServer,(SOCKADDR*)&addrServer, sizeof(SOCKADDR));
    if(err){
        err = GetLastError();
        printf("Could not bind the port %d for socket.Error code
is %d\n",SERVER_PORT,err);
```

```
            WSACleanup();
            return -1;
        }

        SOCKADDR_IN addrClient;       //客户端地址
        int length = sizeof(SOCKADDR);
        char buffer[BUFFER_LENGTH]; //数据发送接收缓冲区
        ZeroMemory(buffer,sizeof(buffer));
        //将测试数据读入内存
        std::ifstream icin;
        icin.open("../test.txt");
        char data[1024 * 113];
        ZeroMemory(data,sizeof(data));
        icin.read(data,1024 * 113);
        icin.close();
        totalPacket = sizeof(data) / 1024;
        int recvSize ;
        for(int i=0; i < SEQ_SIZE; ++i){
            ack[i] = TRUE;
        }
        while(true){
            //非阻塞接收，若没有收到数据，返回值为-1
            recvSize                                                    =
recvfrom(sockServer,buffer,BUFFER_LENGTH,0,((SOCKADDR*)&addrClient),&length);
            if(recvSize < 0){
                Sleep(200);
                continue;
            }
            printf("recv from client: %s\n",buffer);
            if(strcmp(buffer,"-time") == 0){
                getCurTime(buffer);
            }else if(strcmp(buffer,"-quit") == 0){
                strcpy_s(buffer,strlen("Good bye!") + 1,"Good bye!");
            }else if(strcmp(buffer,"-testgbn") == 0){
                //进入 gbn 测试阶段
                //首先 server（server 处于 0 状态）向 client 发送 205 状态码（server
进入 1 状态）
```

```
                //server 等待 client 回复 200 状态码，如果收到（server 进入 2 状态），则开始传输文件，否则延时等待直至超时\
                //在文件传输阶段，server 发送窗口大小设为
                ZeroMemory(buffer,sizeof(buffer));
                int recvSize;
                int waitCount = 0;
                printf("Begain to test GBN protocol,please don't abort the process\n");
                //加入了一个握手阶段
                //首先服务器向客户端发送一个 205 大小的状态码（我自己定义的）表示服务器准备好了，可以发送数据
                //客户端收到 205 之后回复一个 200 大小的状态码，表示客户端准备好了，可以接收数据了
                //服务器收到 200 状态码之后，就开始使用 GBN 发送数据了
                printf("Shake hands stage\n");
                int stage = 0;
                bool runFlag = true;
                while(runFlag){
                    switch(stage){
                        case 0://发送 205 阶段
                            buffer[0] = 205;
                            sendto(sockServer, buffer, strlen(buffer)+1, 0, (SOCKADDR*)&addrClient, sizeof(SOCKADDR));
                            Sleep(100);
                            stage = 1;
                            break;
                        case 1://等待接收 200 阶段，没有收到则计数器+1，超时则放弃此次“连接”，等待从第一步开始
                            recvSize = recvfrom(sockServer,buffer,BUFFER_LENGTH,0,((SOCKADDR*)&addrClient),&length);
                            if(recvSize < 0){
                                ++waitCount;
                                if(waitCount > 20){
                                    runFlag = false;
                                    printf("Timeout error\n");
                                    break;
                                }
                                Sleep(500);
```

```
                continue;
            }else{
                if((unsigned char)buffer[0] == 200){
                    printf("Begin a file transfer\n");
                    printf("File size is %dB, each packet is 1024B and packet total num is %d\n",sizeof(data),totalPacket);
                    curSeq = 0;
                    curAck = 0;
                    totalSeq = 0;
                    waitCount = 0;
                    stage = 2;
                }
            }
            break;
        case 2://数据传输阶段
            if(seqIsAvailable()){
                //发送给客户端的序列号从 1 开始
                buffer[0] = curSeq + 1;
                ack[curSeq] = FALSE;
                //数据发送的过程中应该判断是否传输完成
                //为简化过程此处并未实现
                memcpy(&buffer[1],data + 1024 * totalSeq,1024);
                printf("send a packet with a seq of %d\n",curSeq);
                sendto(sockServer, buffer, BUFFER_LENGTH, 0, (SOCKADDR*)&addrClient, sizeof(SOCKADDR));
                ++curSeq;
                curSeq %= SEQ_SIZE;
                ++totalSeq;
                Sleep(500);
            }
            //等待 Ack，若没有收到，则返回值为-1，计数器+1
            recvSize = recvfrom(sockServer,buffer,BUFFER_LENGTH,0,((SOCKADDR*)&addrClient),&length);
            if(recvSize < 0){
                waitCount++;
                //20 次等待 ack 则超时重传
                if (waitCount > 20)
```

```
                                    {
                                        timeoutHandler();
                                        waitCount = 0;
                                    }
                                }else{
                                    //收到 ack
                                    ackHandler(buffer[0]);
                                    waitCount = 0;
                                }
                                Sleep(500);
                                break;
                            }
                        }
                    }
                    sendto(sockServer, buffer, strlen(buffer)+1, 0, (SOCKADDR*)&addrClient,
sizeof(SOCKADDR));
                    Sleep(500);
                }
                //关闭套接字，卸载库
                closesocket(sockServer);
                WSACleanup();
                return 0;
            }
```

(2) 客户端设计参考

1) ACK 数据帧定义

ACK	0

由于是从服务器端到客户端的单向数据传输，因此 ACK 数据帧不包含任何数据，只需要将 ACK 发送给服务器端即可。

ACK 字段为一个字节，表示序列号数值；

末尾放入 0，表示数据结束。

2）命令设置

客户端的命令和服务器端的解析命令向对应，获取当前用户输入并发送给服务器并等待服务器返回数据，如输入“-time”得到服务器的当前

时间。

此处重点介绍“-testgbn [X] [Y]”命令，[X],[Y]均为[0,1]的小数，其中：

[X]表示客户端的丢包率，模拟网络中报文丢失；

[Y]表示客户端的 ACK 的丢失率。（使用随机函数完成）。

如果用户不输入，则默认丢失率均为 0.2。

3）源代码

```
// GBN_client.cpp : 定义控制台应用程序的入口点。
//
#include "stdafx.h"
#include <stdlib.h>
#include <WinSock2.h>
#include <time.h>

#pragma comment(lib,"ws2_32.lib")

#define SERVER_PORT       12340 //接收数据的端口号
#define SERVER_IP         "127.0.0.1" // 服务器的 IP 地址

const int BUFFER_LENGTH = 1026;
const int SEQ_SIZE = 20;//接收端序列号个数，为 1~20

/****************************************************************/
/*          -time  从服务器端获取当前时间
            -quit  退出客户端
            -testgbn [X]  测试 GBN 协议实现可靠数据传输
                        [X] [0,1]  模拟数据包丢失的概率
                        [Y] [0,1]  模拟 ACK 丢失的概率
*/
/****************************************************************/
void printTips(){
    printf("*********************************************\n");
    printf("|        -time to get current time          |\n");
    printf("|        -quit to exit client               |\n");
    printf("|        -testgbn [X] [Y] to test the gbn   |\n");
    printf("*********************************************\n");
```

```
}

//************************************
// Method:      lossInLossRatio
// FullName:    lossInLossRatio
// Access:      public
// Returns:     BOOL
// Qualifier: 根据丢失率随机生成一个数字，判断是否丢失,丢失则返回TRUE，否则返回 FALSE
// Parameter: float lossRatio [0,1]
//************************************
BOOL lossInLossRatio(float lossRatio){
    int lossBound = (int) (lossRatio * 100);
    int r = rand() % 101;
    if(r <= lossBound){
        return TRUE;
    }
    return FALSE;
}

int main(int argc, char* argv[])
{
    //加载套接字库（必须）
    WORD wVersionRequested;
    WSADATA wsaData;
    //套接字加载时错误提示
    int err;
    //版本 2.2
    wVersionRequested = MAKEWORD(2, 2);
    //加载 dll 文件 Scoket 库
    err = WSAStartup(wVersionRequested, &wsaData);
    if(err != 0){
        //找不到 winsock.dll
        printf("WSAStartup failed with error: %d\n", err);
        return 1;
    }
    if(LOBYTE(wsaData.wVersion) != 2 || HIBYTE(wsaData.wVersion) !=2)
```

```
        {
            printf("Could not find a usable version of Winsock.dll\n");
            WSACleanup();
        }else{
            printf("The Winsock 2.2 dll was found okay\n");
        }
        SOCKET socketClient = socket(AF_INET, SOCK_DGRAM, 0);
        SOCKADDR_IN addrServer;
        addrServer.sin_addr.S_un.S_addr = inet_addr(SERVER_IP);
        addrServer.sin_family = AF_INET;
        addrServer.sin_port = htons(SERVER_PORT);
        //接收缓冲区
        char buffer[BUFFER_LENGTH];
        ZeroMemory(buffer,sizeof(buffer));
        int len = sizeof(SOCKADDR);
        //为了测试与服务器的连接，可以使用 -time 命令从服务器端获得当前时间
        //使用 -testgbn [X] [Y] 测试 GBN 其中[X]表示数据包丢失概率
        //                                    [Y]表示 ACK 丢包概率
        printTips();
        int ret;
        int interval = 1;//收到数据包之后返回 ack 的间隔，默认为 1 表示每个都返回 ack，0 或者负数均表示所有的都不返回 ack
        char cmd[128];
        float packetLossRatio = 0.2;  //默认包丢失率 0.2
        float ackLossRatio = 0.2;     //默认 ACK 丢失率 0.2
        //用时间作为随机种子，放在循环的最外面
        srand((unsigned)time(NULL));
        while(true){
            gets_s(buffer);
            ret = sscanf(buffer,"%s%f%f",&cmd,&packetLossRatio,&ackLossRatio);
            //开始 GBN 测试，使用 GBN 协议实现 UDP 可靠文件传输
            if(!strcmp(cmd,"-testgbn")){
                printf("%s\n","Begin to test GBN protocol, please don't abort the process");
                printf("The loss ratio of packet is %.2f,the loss ratio of ack
```

```
is %.2f\n",packetLossRatio,ackLossRatio);
                    int waitCount = 0;
                    int stage = 0;
                    BOOL b;
                    unsigned char u_code;//状态码
                    unsigned short seq;//包的序列号
                    unsigned short recvSeq;//接收窗口大小为 1，已确认的序列号
                    unsigned short waitSeq;//等待的序列号
                    sendto(socketClient, "-testgbn", strlen("-testgbn")+1, 0,
(SOCKADDR*)&addrServer, sizeof(SOCKADDR));
                    while (true)
                    {
                    //等待 server 回复设置 UDP 为阻塞模式

    recvfrom(socketClient,buffer,BUFFER_LENGTH,0,(SOCKADDR*)&addrServe
r, &len);
                        switch(stage){
                        case 0://等待握手阶段
                            u_code = (unsigned char)buffer[0];
                            if ((unsigned char)buffer[0] == 205)
                            {
                                printf("Ready for file transmission\n");
                                buffer[0] = 200;
                                buffer[1] = '\0';
                                sendto(socketClient, buffer, 2, 0,
(SOCKADDR*)&addrServer, sizeof(SOCKADDR));
                                stage = 1;
                                recvSeq = 0;
                                waitSeq = 1;
                            }
                            break;
                        case 1://等待接收数据阶段
                            seq = (unsigned short)buffer[0];
                            //随机法模拟包是否丢失
                            b = lossInLossRatio(packetLossRatio);
                            if(b){
                                printf("The packet with a seq of %d loss\n",seq);
```

```
                continue;
            }
            printf("recv a packet with a seq of %d\n",seq);
            //如果是期待的包，正确接收，正常确认即可
            if(!(waitSeq - seq)){
                ++waitSeq;
                if(waitSeq == 21){
                    waitSeq = 1;
                }
                //输出数据
                //printf("%s\n",&buffer[1]);
                buffer[0] = seq;
                recvSeq = seq;
                buffer[1] = '\0';
            }else{
            //如果当前一个包都没有收到，则等待 Seq 为 1 的数据包，不是则不返回 ACK（因为并没有上一个正确的 ACK）
                if(!recvSeq){
                    continue;
                }
                buffer[0] = recvSeq;
                buffer[1] = '\0';
            }
            b = lossInLossRatio(ackLossRatio);
            if(b){
                printf("The ack of %d loss\n",(unsigned char)buffer[0]);
                continue;
            }
            sendto(socketClient, buffer, 2, 0, (SOCKADDR*)&addrServer, sizeof(SOCKADDR));
            printf("send a ack of %d\n",(unsigned char)buffer[0]);
            break;
        }
        Sleep(500);
    }
}
```

```
            sendto(socketClient, buffer, strlen(buffer)+1, 0,
(SOCKADDR*)&addrServer, sizeof(SOCKADDR));
            ret =
recvfrom(socketClient,buffer,BUFFER_LENGTH,0,(SOCKADDR*)&addrServer,
&len);
            printf("%s\n",buffer);
            if(!strcmp(buffer,"Good bye!")){
                break;
            }
            printTips();
        }
        //关闭套接字
        closesocket(socketClient);
        WSACleanup();
        return 0;
    }
```

7. 实验报告

在实验报告中要说明所设计 GBN 协议数据分组格式、确认分组格式、各个域作用，协议两端程序流程图，协议典型交互过程，数据分组丢失验证模拟方法，程序实现的主要类（或函数）及其主要作用、实验验证结果，详细注释源程序等。

实验 4：IPv4 分组收发实验

1. 实验目的

IPv4 协议是互联网的核心协议，它保证了网络节点（包括网络设备和主机）在网络层能够按照标准协议互相通信。IPv4 地址唯一标识了网络节点和网络的连接关系。在我们日常使用的计算机的主机协议栈中，IPv4 协议必不可少，它能够接收网络中传送给本机的分组，同时也能根据上层协议的要求将报文封装为 IPv4 分组发送出去。

本实验通过设计实现主机协议栈中的 IPv4 协议，让学生深入了解网络层协议的基本原理，学习 IPv4 协议基本的分组接收和发送流程。

另外，通过本实验，学生可以初步接触互联网协议栈的结构和计算机网络实验系统，为后面进行更为深入复杂的实验奠定良好的基础。

2. 实验环境

- 接入到 Netriver 网络实验系统服务器的主机
- Windows XP、Windows7/8/10 或苹果 OS
- 开发语言：C 语言

3. 实验要求

根据计算机网络实验系统所提供的上下层接口函数和协议中分组收发的主要流程，独立设计实现一个简单的 IPv4 分组收发模块。要求实现的主要功能包括：

1) IPv4 分组的基本接收处理，能够检测出接收到的 IP 分组是否存在如下错误：校验和错、TTL 错、版本号错、头部长度错、错误目标地址；
2) IPv4 分组的封装发送；

注：不要求实现 IPv4 协议中的选项和分片处理功能

4. 实验内容

1) 实现 IPv4 分组的基本接收处理功能

对于接收到的IPv4分组，检查目的地址是否为本地地址，并检查IPv4分组头部中其它字段的合法性。提交正确的分组给上层协议继续处理，丢弃错误的分组并说明错误类型。

2) 实现 IPv4 分组的封装发送

根据上层协议所提供的参数，封装 IPv4 分组，调用系统提供的发送接口函数将分组发送出去。

5. 参考内容

在主机协议栈中，IPv4 协议主要承担辨别和标识源 IPv4 地址和目的 IPv4 地址的功能，一方面接收处理发送给自己的分组，另一方面根据应用需求填写目的地址并将上层报文封装发送。IPv4 地址可以在网络中唯一标识一台主机，因而在相互通信时填写在 IPv4 分组头部中的 IPv4 地址就起到了标识源主机和目的主机的作用。在后面 IPv4 分组的转发实验中，我们还将深入学习 IPv4 地址在分组转发过程中对选择转发路径的重要作用。

在两个主机端系统通信的环境中，网络的拓扑可以简化为两台主机直接相连，中间的具体连接方式可以抽象为一条简单的链路，如图 4-1 所示。IPv4 分组收发实验就是要在实验系统客户端的开发平台上，实现 IPv4 分组的接收和发送功能。

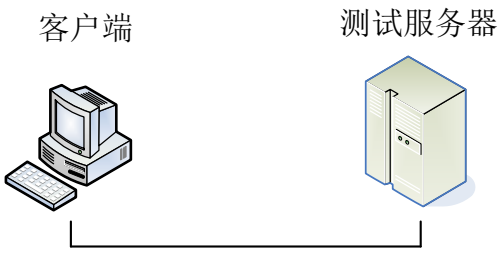

图 4-1 实验环境网络拓扑结构

5.1 NetRiver 网络实验系统简介

整个 NetRiver 网络实验系统由 3 部分构成：实验管理服务器、测试服务器和客户端，三者之间可以互相通信，系统的结构如图 4-2 所示。

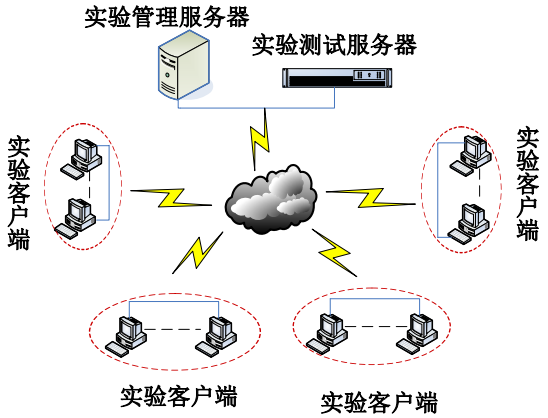

图 4-2 NetRiver 系统网络实验环境

实验管理服务器用于保存所有用户信息，包括用户名、用户密码、实验进度、测试结果等等。实验管理服务器还存储了网络实验的所有测试用例。支持用户用户管理和认证功能，用户登录成功后，服务器会向客户端发送当前测试用例列表，并管理用户提交的实验代码和测试结果。该服务器还能接受到测试服务器的连接请求及向测试服务器发送测试用例内容。

测试服务器会根据实验内容与客户端进行协议通信，完成协议一致性测试。测试服务器从实验管理服务器上获取测试用例内容，测试用例描述了详细的测试过程，测试服务器和客户端按照测试用例完成这一过程。测试服务器向客户端发送协议分组，也接收客户端发来的协议分组，并判断客户端协议行为的正确性。所有的测试结果都是有测试服务器进行判定，再反馈给客户端，并通知实验管理服务器对其进行保存。

客户端是学生实验的操作平台，它不仅可以连接并登录实验管理服务器和测试服务器，还提供了一个可以编辑、编译、调式和执行实验代码的软件开发环境。在测试过程中，客户端软件还能够显示并分析协议测试分组的内容。

5.2 NetRiver 实验系统学生实验流程

整个实验流程图如图 4-3 所示。

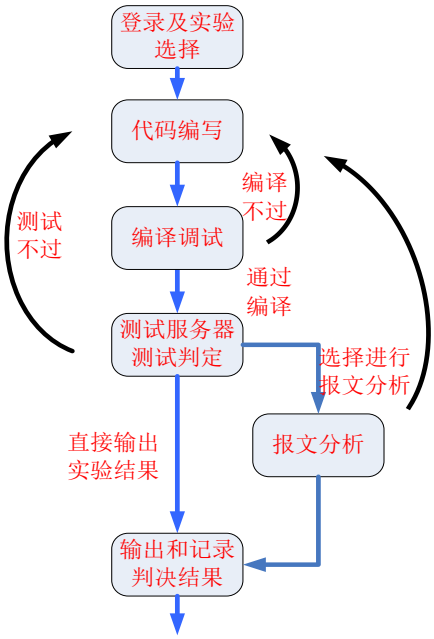

图 4-3 NetRiver 系统实验整体流程

5.2.1 登录和选择实验

安装好客户端软件后，学生需要登录系统并选择改次试验的内容和具体的测试用例，如图 4-4 所示。

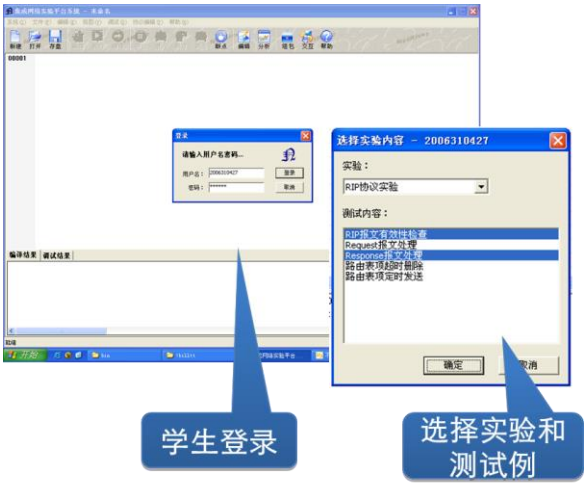

图 4-4 学生登录、选择实验和测试用例

5.2.2 实验代码编辑、编译和运行

如果第一次做某个实验，在“文件”菜单中选择“新建”，系统会自动创建该实验的模板代码并显示在编辑器当中。该模板将给出实验必须的头文件引用、系统函数接口声明以及需要学生完成的函数接口。学生只需在此基础上定义自己需要的数据结构，完成需要实现的函数内容即可。学生可以随时保存或另存代码文件，系统提供了代码的上传和下载服务，可以把代码文件上传到服务器上或者下载。注意，服务器只会为每个用户的每个实验保留一份代码，所以每次上传都会覆盖上次上传的文件。用户在编写完实验代码后可以进行编译。只有代码被正确编译后，用户才能执行相关的测试操作。代码的编辑和编译示意图如图 4-5 所示。

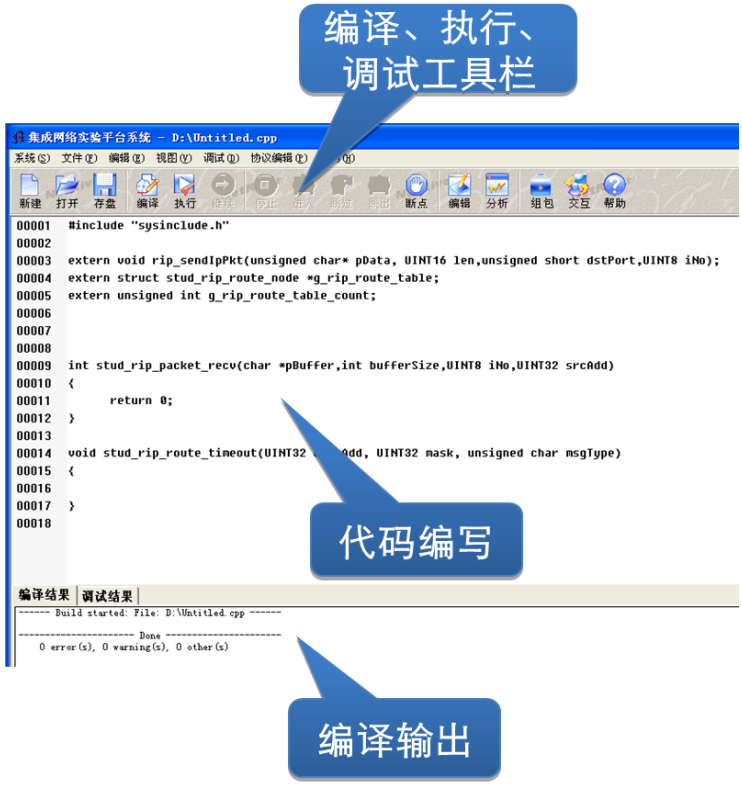

图 4-5 实验代码编辑和编译示意图

代码编译成功后，学生可以运行程序，执行测试用例或者对代码进行调试。NetRiver 提供了一整套代码编写和调试的环境，包括断点设置、单步执行和查看运行时变量等。

在执行或调试程序的时候，学生可以在报文分析界面查看客户端和服务器发出的报文，如图 4-6 所示。

系统会在执行过程当中显示执行结果和测试结果，如图 4-7 所示。

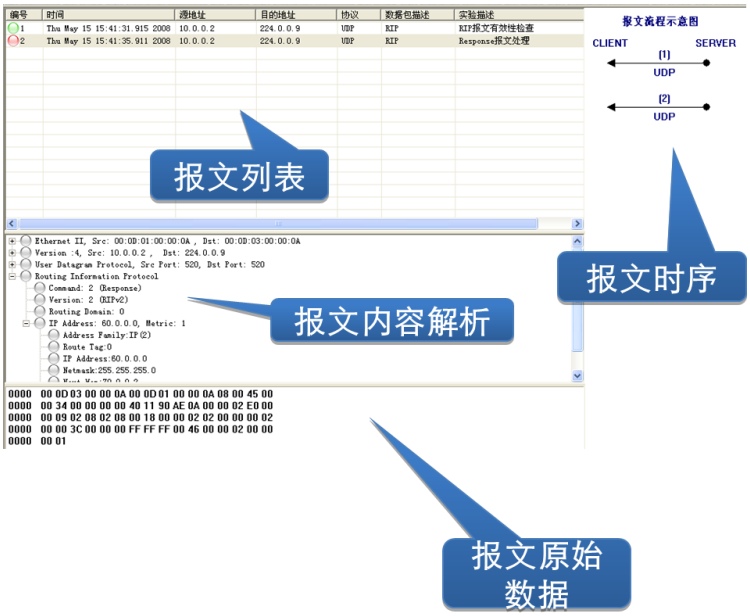

图 4-6 报文分析器界面

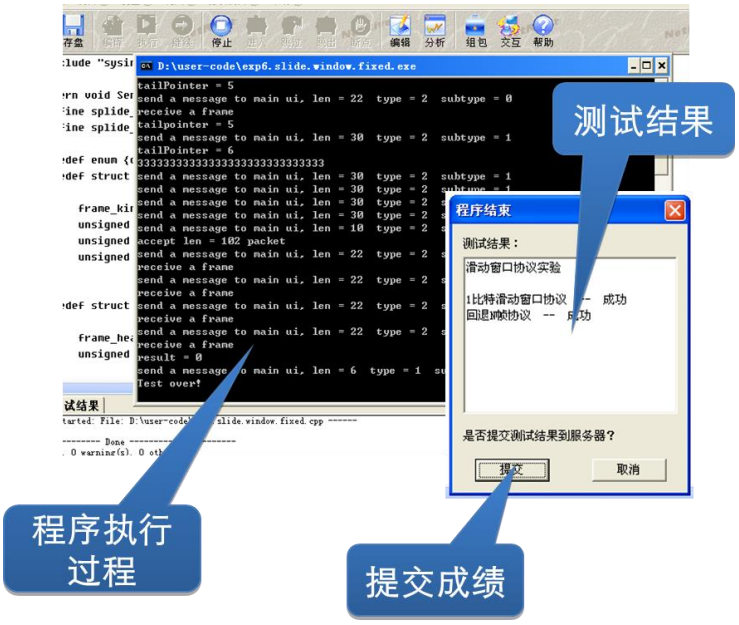

图 4-7 程序执行过程和结果

5.2.3 输出和记录判决结果

在学生完成实验后，可以进入在线实验管理系统查看系统自动给出的判决结果，该结果是由系统的测试服务器与学生程序交互后，按照测试用例自动给出，如图 4-8 所示。

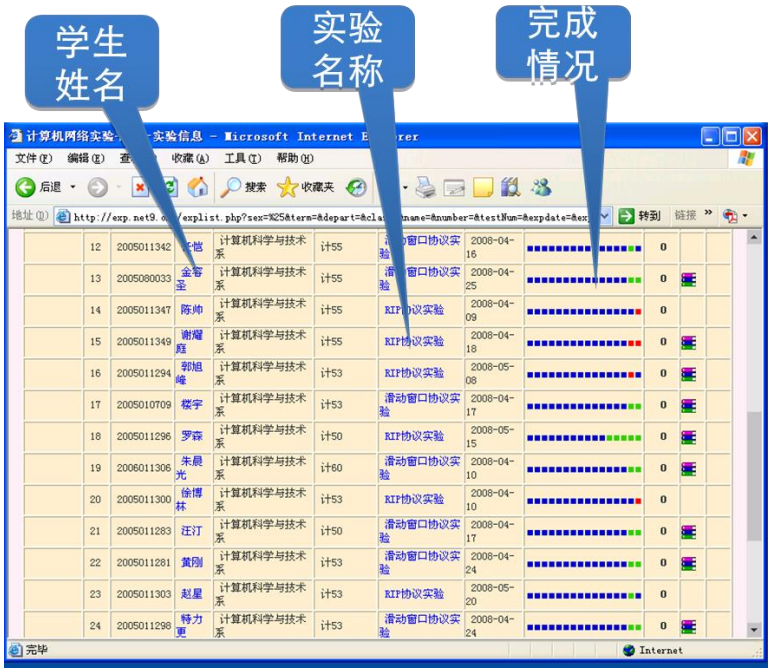

图 4-8 输出和记录判决结果

5.3 处理流程

客户端接收到测试服务器发送来的 IPv4 分组后，调用接收接口函数 stud_ip_recv( )（图 4-9 中接口函数 1）。学生需要在这个函数中实现 IPv4 分组接收处理的功能。接收处理完成后，调用接口函数 ip_SendtoUp( ) 将需要上层协议进一步处理的信息提交给上层协议（图 4-9 中接口函数 2）；或者调用函数 ip_DiscardPkt( )丢弃有错误的分组并报告错误类型（图 4-9 中函数 5）。

在上层协议需要发送分组时，会调用发送接口函数 stud_ip_Upsend( )（图 4-9 中接口函数 3）。学生需要在这个函数中实现 IPv4 分组封装发送的功能。根据所传参数完成 IPv4 分组的封装，之后调用接口函数 ip_SendtoLower( )把分组交给下层完成发送（图 4-9 中接口函数 4）。

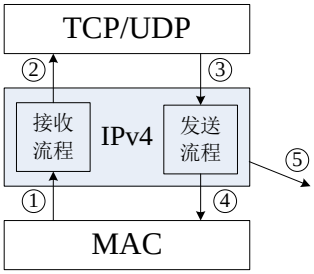

图 4-9 实验接口函数示意图

5.3.1 接收流程

在接口函数 stud_ip_recv( )中，需要完成下列处理步骤（仅供参考）：

① 检查接收到的 IPv4 分组头部的字段，包括版本号（Version）、头部长度（IP Head length）、生存时间（Time to live）以及头校验和（Header checksum）字段。对于出错的分组调用 ip_DiscardPkt( )丢弃，并说明错误类型。

② 检查 IPv4 分组是否应该由本机接收。如果分组的目的地址是本机地址或广播地址，则说明此分组是发送给本机的；否则调用 ip_DiscardPkt( )丢弃，并说明错误类型。

③ 如果 IPV4 分组应该由本机接收，则提取得到上层协议类型，调用 ip_SendtoUp( )接口函数，交给系统进行后续接收处理。

5.3.2 发送流程

在接口函数 stud_ip_Upsend( )中，需要完成下列处理步骤（仅供参考）：

① 根据所传参数（如数据大小），来确定分配的存储空间的大小并申请分组的存储空间。

② 按照 IPv4 协议标准填写 IPv4 分组头部各字段，标识符（Identification）字段可以使用一个随机数来填写。（注意：部分字段内容需要转换成网络字节序）

③ 完成 IPv4 分组的封装后，调用 ip_SendtoLower( )接口函数完成后续的发送处理工作，最终将分组发送到网络中。

5.4 IPv4 分组头部格式

IPv4 分组头部格式如图 4-10 所示。

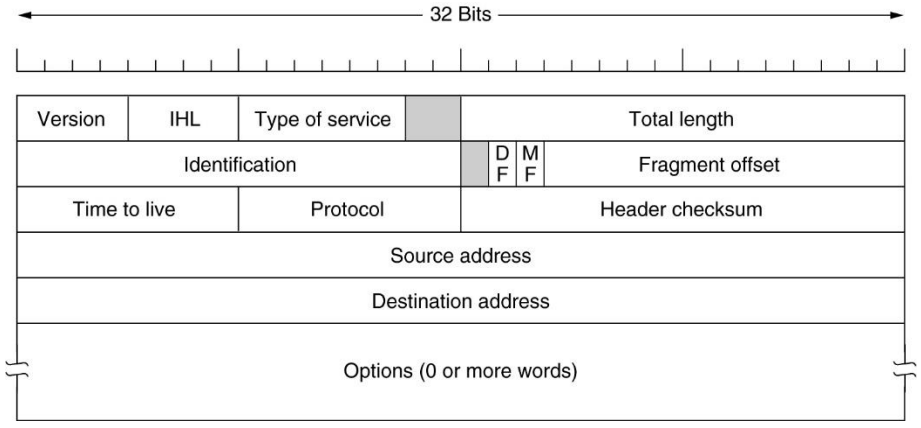

图 4-10 IPv4 分组头部格式

5.5 接口函数说明

5.5.1 需要实现的接口函数

1) 接收接口

int stud_ip_recv(char * pBuffer, unsigned short length)

参数：

pBuffer：指向接收缓冲区的指针，指向 IPv4 分组头部

length：IPv4 分组长度

返回值：

0：成功接收 IP 分组并交给上层处理

1：IP 分组接收失败

2) 发送接口

int stud_ip_Upsend(char* pBuffer, unsigned short len, unsigned int srcAddr, unsigned int dstAddr ,byte protocol, byte ttl)

参数：

pBuffer：指向发送缓冲区的指针，指向 IPv4 上层协议数据头部

len：IPv4 上层协议数据长度
srcAddr：源 IPv4 地址
dstAddr：目的 IPv4 地址
protocol：IPv4 上层协议号
ttl：生存时间（Time To Live）
返回值：
0：成功发送 IP 分组
1：发送 IP 分组失败

5.5.2 系统提供的接口函数

1) 丢弃分组

void ip_DiscardPkt(char * pBuffer ,int type)
参数：
pBuffer：指向被丢弃分组的指针
type：分组被丢弃的原因，可取以下值：

```
STUD_IP_TEST_CHECKSUM_ERROR      //IP 校验和出错
STUD_IP_TEST_TTL_ERROR           //TTL 值出错
STUD_IP_TEST_VERSION_ERROR       //IP 版本号错
STUD_IP_TEST_HEADLEN_ERROR       //头部长度错
STUD_IP_TEST_DESTINATION_ERROR   //目的地址错
```

2) 发送分组

void ip_SendtoLower(char *pBuffer ,int length)
参数：
pBuffer：指向待发送的 IPv4 分组头部的指针
length：待发送的 IPv4 分组长度

3) 上层接收

void ip_SendtoUp(char *pBuffer, int length)
参数：
pBuffer：指向要上交的上层协议报文头部的指针
length：上交报文长度

4) 获取本机 IPv4 地址

unsigned int getIpv4Address( )
参数：无

6. 实验报告

1) 要求给出发送和接收函数的实现程序流程图；
2) 给出自己所新建的数据结构的说明（如果有）；
3) 要求给出版本号（Version）、头部长度（IP Head length）、生存时间（Time to live）以及头校验和（Header checksum）字段的错误检测原理，并根据实验具体情况给出错误的具体数据，例如如果为头部长度错，请给出收到的错误的 IP 分组头部长度字段值为多少。

实验 5：IPv4 分组转发实验

1. 实验目的

通过前面的实验，我们已经深入了解了 IPv4 协议的分组接收和发送处理流程。本实验需要将实验模块的角色定位从通信两端的主机转移到作为中间节点的路由器上，在 IPv4 分组收发处理的基础上，实现分组的路由转发功能。

网络层协议最为关注的是如何将 IPv4 分组从源主机通过网络送达目的主机，这个任务就是由路由器中的 IPv4 协议模块所承担。路由器根据自身所获得的路由信息，将收到的 IPv4 分组转发给正确的下一跳路由器。如此逐跳地对分组进行转发，直至该分组抵达目的主机。IPv4 分组转发是路由器最为重要的功能。

本实验设计模拟实现路由器中的 IPv4 协议，可以在原有 IPv4 分组收发实验的基础上，增加 IPv4 分组的转发功能。对网络的观察视角由主机转移到路由器中，了解路由器是如何为分组选择路由，并逐跳地将分组发送到目的主机。本实验中也会初步接触路由表这一重要的数据结构，认识路由器是如何根据路由表对分组进行转发的。

2. 实验环境

- 接入到 Netriver 网络实验系统服务器的主机
- Windows XP、Windows7/8/10 或苹果 OS
- 开发语言：C 语言

3. 实验要求

在前面 IPv4 分组收发实验的基础上，增加分组转发功能。具体来说，对于每一个到达本机的 IPv4 分组，根据其目的 IPv4 地址决定分组的处理行为，对该分组进行如下的几类操作：

1) 向上层协议上交目的地址为本机地址的分组；
2) 根据路由查找结果，丢弃查不到路由的分组；
3) 根据路由查找结果，向相应接口转发不是本机接收的分组。

4. 实验内容

实验内容主要包括：

1) 设计路由表数据结构。

设计路由表所采用的数据结构。要求能够根据目的 IPv4 地址来确定分组处理行为（转发情况下需获得下一跳的 IPv4 地址）。路由表的数据结构和查找算法会极大的影响路由器的转发性能，有兴趣的同学可以深入思考和探索。

2) IPv4 分组的接收和发送。

对前面实验（IP 实验）中所完成的代码进行修改，在路由器协议栈的 IPv4 模块中能够正确完成分组的接收和发送处理。具体要求不做改变，参见“IP 实验”。

3) IPv4 分组的转发。

对于需要转发的分组进行处理，获得下一跳的 IP 地址，然后调用发送接口函数做进一步处理。

5. 参考内容

分组转发是路由器最重要的功能。分组转发的依据是路由信息，以此将目的地址不同的分组发送到相应的接口上，逐跳转发，并最终到达目的主机。本实验要求按照路由器协议栈的 IPv4 协议功能进行设计实现，接收处理所有收到的分组（而不只是目的地址为本机地址的分组），并根据分组的 IPV4 目的地址结合相关的路由信息，对分组进行转发、接收或丢弃操作。

实验的主要流程和系统接口函数与前面“IP 实验”基本相同。在下层接收接口函数 Stud_fwd_deal( )中（图 5-1 中接口函数 1），实现分组接收处理。主要功能是根据分组中目的 IPv4 地址结合对应的路由信息对分

组进行处理。分组需要上交，则调用接口函数 Fwd_LocalRcv( )（图 5-1 中接口函数 2）；需要丢弃，则调用函数 Fwd_DiscardPkt( )（图 5-1 中函数 5）；需要转发，则进行转发操作。转发操作的实现要点包括，TTL 值减 1，然后重新计算头校验和，最后调用发送接口函数 Fwd_SendtoLower( )（图 5-1 中接口函数 4）将分组发送出去。注意，接口函数 Fwd_SendtoLower( )比前面实验增加了一个参数 pNxtHopAddr，要求在调用时传入下一跳的 IPv4 地址，此地址是通过查找路由表得到的。

另外，本实验增加了一个路由表配置的接口（图 5-1 中函数 6），要求能够根据系统所给信息来设定本机路由表。实验中只需要简单地设置静态路由信息，以作为分组接收和发送处理的判断依据，而路由信息的动态获取和交互，在本实验中不予考虑。

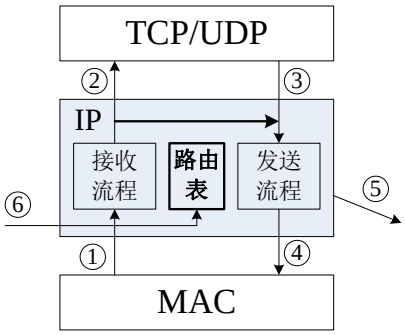

图 5-1 IPv4 分组转发实验接口示意图

与前面 IP 实验不同的是，在本实验中分组接收和发送过程中都需要引入路由表的查找步骤。路由器的主要任务是进行分组转发，它所接收的多数分组都是需要进行转发的，而不像主机协议栈中 IPv4 模块只接收发送给本机的分组；另外，路由器也要接收处理发送给本机的一些分组，如路由协议的分组（RIP 实验中会涉及到）、ICMP 分组等。如何确定对各种分组的处理操作类型，就需要根据分组的 IPV4 目的地址结合路由信息进行判断。

一般而言，路由信息包括地址段、距离、下一跳地址、操作类型等。在接收到 IPv4 分组后，要通过其目的地址匹配地址段来判断是否为本机地址，如果是则本机接收；如果不是，则通过其目的地址段查找路由表信息，从而得到进一步的操作类型，转发情况下还要获得下一跳的 IPv4

地址。发送 IPv4 分组时，也要拿目的地址来查找路由表，得到下一跳的 IPv4 地址，然后调用发送接口函数做进一步处理。在前面实验中，发送流程中没有查找路由表来确定下一跳地址的步骤，这项工作由系统来完成了，在本实验中则作为实验内容要求学生实现。需要进一步说明的是，在转发路径中，本路由器可能是路径上的最后一跳，可以直接转发给目的主机，此时下一跳的地址就是 IPv4 分组的目的地址；而非最后一跳的情况下，下一跳的地址是从对应的路由信息中获取的。因此，在路由表中转发类型要区分最后一跳和非最后一跳的情况。

路由表数据结构的设计是非常重要的，会极大地影响路由表的查找速度，进而影响路由器的分组转发性能。本实验中虽然不会涉及大量分组的处理问题，但良好且高效的数据结构无疑会为后面的实验奠定良好的基础。链表结构是最简单的，但效率比较低；树型结构的查找效率会提高很多，但组织和维护有些复杂，可以作为提高的要求。具体数据结构的设计，可以在实践中进一步深入研究。

5.1 路由表维护

需要完成下列分组接收处理步骤：

1) stud_Route_Init ( )函数中，对路由表进行初始化。
2) stud_route_add ( )函数中，完成路由的增加。

5.2 转发处理流程

在 stud_fwd_deal ( )函数中，需要完成下列分组接收处理步骤：

1) 查找路由表。根据相应路由表项的类型来确定下一步操作，错误分组调用函数 fwd_DiscardPkt ( )进行丢弃，上交分组调用接口函数 fwd_LocalRcv ( )提交给上层协议继续处理，转发分组进行转发处理。注意，转发分组还要从路由表项中获取下一跳的 IPv4 地址。
2) 转发处理流程。对 IPv4 头部中的 TTL 字段减 1，重新计算校验和，然后调用下层接口 fwd_SendtoLower ( )进行发送处理。

5.3 实验接口函数

本小节列出了实验时会用到的各种接口函数，其中有一些函数是需

要学生来完成的，有一些函数则是已经实现好了，供学生在实验时直接使用的。表 3-1 列出了所有的接口函数及其简要说明，更详细的说明在后面描述。

表 3-1 函数接口表

函数名	说明	是否需要学生实现
stud_fwd_deal	系统处理收到的 IP 分组的函数，当接收到一个 IP 分组的时候，实验系统会调用该函数进行处理	是
fwd_SendtoLower	将封装完成的 IP 分组通过链路层发送出去的函数。	否
fwd_LocalRcv	将 IP 分组上交本机上层协议的函数，即当分组需要上交上层函数的时候调用本函数。	否
fwd_DiscardPkt	丢弃 IP 分组的函数。当需要丢弃一个 IP 分组的时候调用。	否
stud_route_add	向路由表添加路由的函数。系统将调用该函数向路由表添加一条 IPv4 路由。	是
stud_Route_Init	路由表初始化函数，系统初始化的时候将调用此函数对路由表进行初始化操作。	是
getIpv4Address	获取本机的 IPv4 地址，用来判断分组地址和本机地址是否相同	否

1）stud_fwd_deal( )

int stud_fwd_deal(char * pBuffer, int length)

参数：

pBuffer：指向接收到的 IPv4 分组头部

length：IPv4 分组的长度

返回值：

0 为成功，1 为失败；

说明：

本函数是 IPv4 协议接收流程的下层接口函数，实验系统从网络中接

收到分组后会调用本函数。调用该函数之前已完成IP报文的合法性检查，因此学生在本函数中应该实现如下功能：

a. 判定是否为本机接收的分组，如果是则调用 fwd_LocalRcv( )；

b. 按照最长匹配查找路由表获取下一跳，查找失败则调用 fwd_DiscardPkt( )；

c.调用 fwd_SendtoLower( )完成报文发送；

d.转发过程中注意 TTL 的处理及校验和的变化。

2）fwd_LocalRcv( )

void fwd_LocalRcv(char *pBuffer, int length)

参数：

pBuffer：指向分组的 IP 头

length：表示分组的长度

说明：

本函数是 IPv4 协议接收流程的上层接口函数，在对 IPv4 的分组完成解析处理之后，如果分组的目的地址是本机的地址，则调用本函数将正确分组提交上层相应协议模块进一步处理。

3）fwd_SendtoLower( )

void fwd_SendtoLower(char *pBuffer, int length, unsigned int nexthop)

参数：

pBuffer：指向所要发送的 IPv4 分组头部

length：分组长度（包括分组头部）

nexthop：转发时下一跳的地址。

说明：

本函数是发送流程的下层接口函数，在 IPv4 协议模块完成发送封装工作后调用该接口函数进行后续发送处理。其中，后续的发送处理过程包括分片处理、IPv4 地址到 MAC 地址的映射（ARP 协议）、封装成 MAC 帧等工作，这部分内容不需要学生完成，由实验系统提供支持。

4）fwd_DiscardPkt( )

void fwd_DiscardPkt(char * pBuffer, int type)

参数：

pBuffer：指向被丢弃的 IPV4 分组头部

type：表示错误类型，包括 TTL 错误和找不到路由两种错误，定义如下：

STUD_FORWARD_TEST_TTLERROR

STUD_FORWARD_TEST_NOROUTE

说明：

本函数是丢弃分组的函数，在接收流程中检查到错误时调用此函数将分组丢弃。

5）stud_route_add( )

void stud_route_add(stud_route_msg *proute)

参数：

proute：指向需要添加路由信息的结构体头部，其数据结构 stud_route_msg 的定义如下：

```
typedef struct stud_route_msg
{
   unsigned int    dest;
   unsigned int    masklen;
   unsigned int    nexthop;
    } stud_route_msg;
```

说明：

本函数为路由表配置接口，系统在配置路由表时需要调用此接口。此函数功能为向路由表中增加一个新的表项，将参数所传递的路由信息添加到路由表中。

6）stud_Route_Init( )

void stud_Route_Init( )

参数：

无。

说明：

本函数将在系统启动的时候被调用，学生可将初始化路由表的代码写在这里。

7）getIpv4Address( )

UINT32 getIpv4Address( )

说明：

本函数用于获取本机的 IPv4 地址，学生调用该函数即可返回本机的 IPv4 地址，可以用来判断 IPV4 分组是否为本机接收。

返回值：

本机 IPv4 地址。

除了以上的函数以外，学生可根据需要自己编写一些实验需要的函数和数据结构，包括路由表的数据结构，对路由表的搜索、初始化等操作函数。

6. 实验报告

1) 要求给出路由表初始化、路由增加、路由转发三个函数的实现流程图；
2) 要求给出所新建数据结构的说明；
3) 请分析在存在大量分组的情况下如何提高转发效率，如果代码中有相关功能实现，请给出具体原理说明。

实验 6：利用 Wireshark 进行协议分析

1、实验目的

熟悉并掌握 Wireshark 的基本操作，了解网络协议实体间进行交互以及报文交换的情况。

2、实验环境

- Windows 9x/NT/2000/XP/2003
- 与因特网连接的计算机网络系统
- Wireshark

3、实验内容

1) 学习 Wireshark 的使用
2) 利用 Wireshark 分析 HTTP 协议
3) 利用 Wireshark 分析 TCP 协议
4) 利用 Wireshark 分析 IP 协议
5) 利用 Wireshark 分析 Ethernet 数据帧

选做内容：

a) 利用 Wireshark 分析 DNS 协议
b) 利用 Wireshark 分析 UDP 协议
c) 利用 Wireshark 分析 ARP 协议

4、实验方式

每位同学上机独立完成实验，并与指导教师讨论。

5、参考内容

要深入理解网络协议，需要仔细观察协议实体之间交换的报文序列。为探究协议操作细节，可使协议实体执行某些动作，观察这些动作及其影响。这些任务可以在仿真环境下或在如因特网这样的真实网络环境中完成。观察在正在运行协议实体间交换报文的基本工具被称为分组嗅探器（packet sniffer）。顾名思义，一个分组嗅探器俘获（嗅探）计算机发送和接收的报文。一般情况下，分组嗅探器将存储和显示出被俘获报文的各协议头部字段的内容。图 6-1 为一个分组嗅探器的结构。

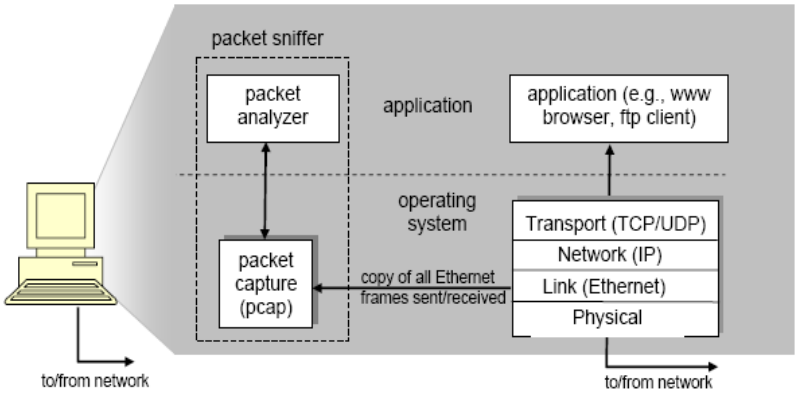

图 6-1 分组嗅探器的结构

图 6-1 右边是计算机上正常运行的协议（在这里是因特网协议）和应用程序（如：web 浏览器和 ftp 客户端）。分组嗅探器（虚线框中的部分）是附加计算机普通软件上的，主要有两部分组成。分组俘获库（packet capture library）接收计算机发送和接收的每一个链路层帧的拷贝。高层协议（如：HTTP、FTP、TCP、UDP、DNS、IP 等）交换的报文都被封装在链路层帧中，并沿着物理媒体（如以太网的电缆）传输。图 1 假设所使用的物理媒体是以太网，上层协议的报文最终封装在以太网帧中。

分组嗅探器的第二个组成部分是分析器。分析器用来显示协议报文所有字段的内容。为此，分析器必须能够理解协议所交换的所有报文的结构。例如：我们要显示图 6-1 中 HTTP 协议所交换的报文的各个字段。分组分析器理解以太网帧格式，能够识别包含在帧中的 IP 数据报。分组分析器也要理解 IP 数据报的格式，并能从 IP 数据报中提取出 TCP 报文段。然后，它需要理解 TCP 报文段，并能够从中提取出 HTTP 消息。最后，它需要理解 HTTP 消息。

Wireshark 是一种可以运行在 Windows，UNIX，Linux 等操作系统上的分组分析器。Wireshark 是免费的，可以从 https://www.wireshark.org/download.html 得到,Wireshark 的 User's Guide 可以从 https://www.wireshark.org/docs/ 获得。运行 Wireshark 程序时，其图形用户界面如图 6-2 所示。最初，各窗口中并无数据显示。在用户选择接口，点击开始抓包按钮之后，Wireshark 的用户界面会变成如图 6-3 所示。

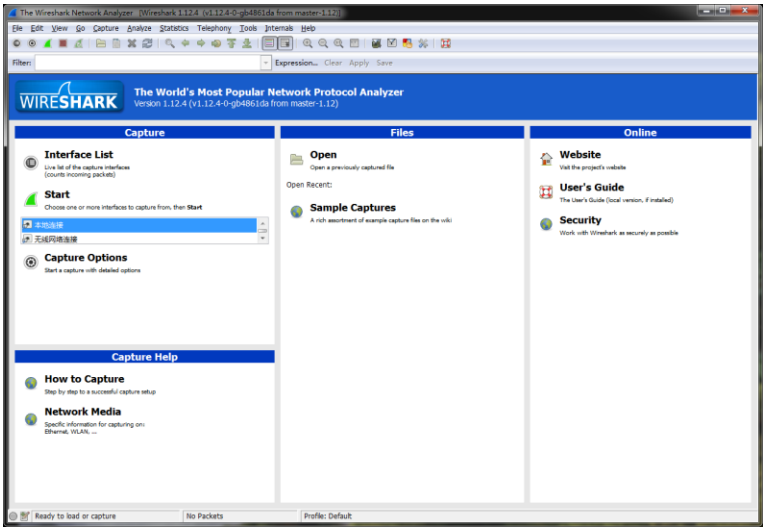

图 6-2 Wireshark 初始用户界面

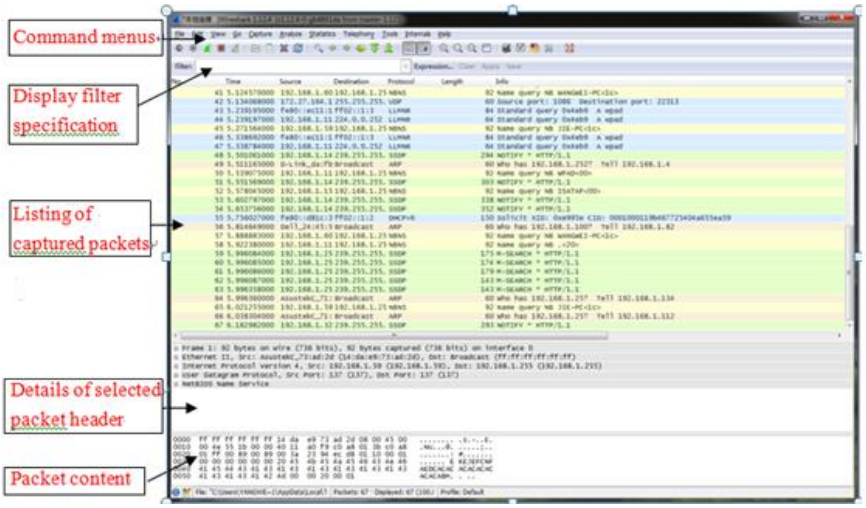

图 6-3 Wireshark 的用户界面

此时 Wireshark 的用户界面主要有 5 部分组成，如图 6-3 所示。

- **命令菜单**（command menus）：命令菜单位于窗口的最顶部，是标准的下拉式菜单。最常用菜单命令有两个：File、Capture。File 菜单允许你保存俘获的分组数据或打开一个已被保存的俘获分组数据文件或退出 Wireshark 程序。Capture 菜单允许你开始俘获分组。
- **俘获分组列表**（listing of captured packets）：按行显示已被俘获的分组内容，其中包括：Wireshark 赋予的分组序号、俘获时间、分组的源地址和目的地址、协议类型、分组中所包含的协议说明信息。单击某一列的列名，可以使分组按指定列进行排序。在该列表中，所显示的协议类型是发送或接收分组的**最高层协议**的类型。
- **分组头部明细**（details of selected packet header）：显示俘获分组列表窗口中被选中分组的头部详细信息。包括：与以太网帧有关的信息，与包含在该分组中的 IP 数据报有关的信息。单击以太网帧或 IP 数据报所在行左边的向右或向下的箭头可以展开或最小化相关信息。另外，如果利用 TCP 或 UDP 承载分组，Wireshark 也会显示 TCP 或 UDP 协议头部信息。最后，分组最高层协议的头部字段也会显示在此窗口中。
- **分组内容窗口**（packet content）：以 ASCII 码和十六进制两种格式显示被俘获帧的完整内容。
- 显示**筛选规则**（display filter specification）：在该字段中，可以填写协议的名称或其他信息，根据此内容可以对分组列表窗口中的分组进行过滤。

（一）Wireshark 的使用

- 启动主机上的 web 浏览器。
- 启动 Wireshark。你会看到如图 6-2 所示的窗口，只是窗口中没有任何分组列表。
- 开始分组俘获：选择“capture”下拉菜单中的“Capture Options”命令，会出现如图 6-3 所示的“Wireshark: Capture Options”窗口，可以设置分组俘获的选项。
- 在实验中，可以使用窗口中显示的默认值。在“Wireshark: Capture

Options”窗口（如图 6-4 所示）的最上面有一个“Interface List”下拉菜单，其中显示计算机所具有的网络接口（即网卡）。当计算机具有多个活动网卡时，需要选择其中一个用来发送或接收分组的网络接口（如某个有线接口）。随后，单击“Start”开始进行分组俘获，所有由选定网卡发送和接收的分组都将被俘获。

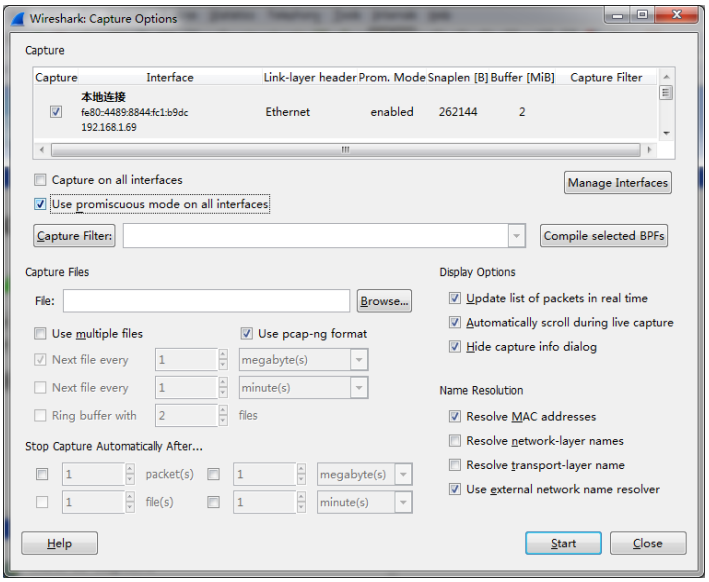

图 6-4 Wireshark 的 Capture Option

- 开始分组俘获后，会出现如图 6-5 所示的窗口。该窗口统计显示各类已俘获数据包。在该窗口的工具栏中有一个“stop”按钮，可以停止分组的俘获。但此时你最好不要停止俘获分组。
- 在运行分组俘获的同时，在浏览器地址栏中输入某网页的 URL，如：http://www.hit.edu.cn。为显示该网页，浏览器需要连接 www.hit.edu.cn 的服务器，并与之交换 HTTP 消息，以下载该网页。包含这些 HTTP 报文的以太网帧将被 Wireshark 俘获。
- 当完整的页面下载完成后，单击 Wireshark 菜单栏中的 stop 按钮，停止分组俘获。Wireshark 主窗口显示已俘获的你的计算机与其他网络实体交换的所有协议报文，其中一部分就是与 www.hit.edu.cn 服务器交换的 HTTP 报文。此时主窗口与图 6-3 相似。
- 在显示筛选规则中输入“http”，单击“回车”，分组列表窗口将只显示 HTTP 协议报文。

➢ 选择分组列表窗口中的第一条 http 报文。它应该是你的计算机发向 www.hit.edu.cn 服务器的 HTTP GET 报文。当你选择该报文后，以太网帧、IP 数据报、TCP 报文段、以及 HTTP 报文首部信息都将显示在分组首部子窗口中。单击分组首部详细信息子窗口中向右和向下箭头，可以最小化帧、以太网、IP、TCP 信息显示量，可以最大化 HTTP 协议相关信息的显示量。

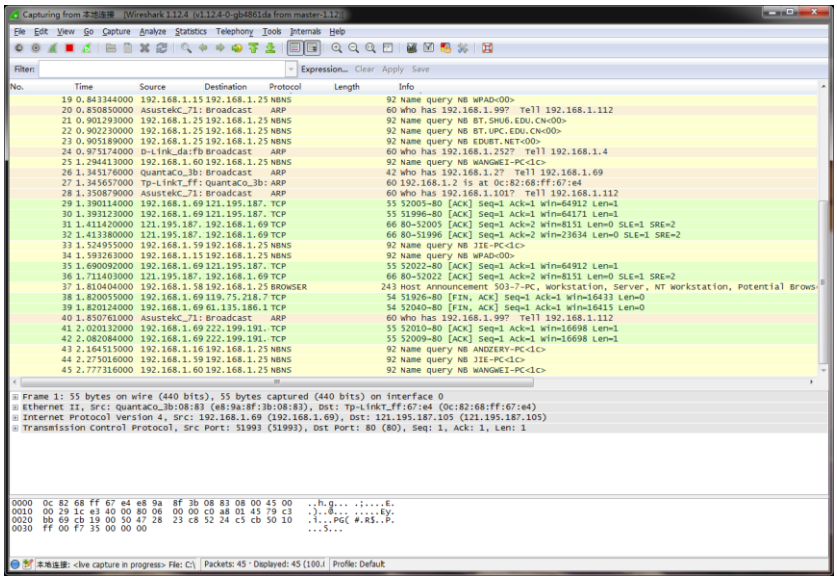

图 6-5 Wireshark 的抓包界面

（二）HTTP 分析

1）HTTP GET/response 交互

✧ 启动 Web browser，然后启动 Wireshark 分组嗅探器。在窗口的显示过滤说明处输入“http”，分组列表子窗口中将只显示所俘获到的 HTTP 报文。

✧ 开始 Wireshark 分组俘获。

✧ 在打开的 Web browser 窗口中输入一下地址：http://hitgs.hit.edu.cn/news

✧ 停止分组俘获。

根据俘获窗口内容，思考以下问题：

✧ 你的浏览器运行的是 HTTP1.0，还是 HTTP1.1？你所访问的服务器所运行 HTTP 协议的版本号是多少？

✧ 你的浏览器向服务器指出它能接收何种语言版本的对象？

✧ 你的计算机的 IP 地址是多少？服务器 http://hitgs.hit.edu.cn/news 的 IP 地址是多少？

✧ 从服务器向你的浏览器返回的状态代码是多少？

2）HTTP 条件 GET/response 交互

✧ 启动浏览器，清空浏览器的缓存（在浏览器中，选择“工具”菜单中的“Internet 选项”命令，在出现的对话框中，选择“删除文件”）。

✧ 启动 Wireshark 分组俘获器。开始 Wireshark 分组俘获。

✧ 在浏览器的地址栏中输入以下 URL: http://hitgs.hit.edu.cn/news,在你的浏览器中重新输入相同的 URL 或单击浏览器中的“刷新”按钮。

✧ 停止 Wireshark 分组俘获，在显示过滤筛选说明处输入“http”,分组列表子窗口中将只显示所俘获到的 HTTP 报文。

根据俘获窗口内容，思考以下问题：

✧ 分析你的浏览器向服务器发出的第一个 HTTP GET 请求的内容，在该请求报文中，是否有一行是：IF-MODIFIED-SINCE？

✧ 分析服务器响应报文的内容，服务器是否明确返回了文件的内容？如何获知？

✧ 分析你的浏览器向服务器发出的较晚的“HTTP GET”请求，在该请求报文中是否有一行是：IF-MODIFIED-SINCE？如果有，在该首部行后面跟着的信息是什么？

✧ 服务器对较晚的 HTTP GET 请求的响应中的 HTTP 状态代码是多少？服务器是否明确返回了文件的内容？请解释。

（三）TCP 分析

注：访问以下网址需要设置代理服务器。如无法访问可与实验 TA 联系，下载 tcp-Wireshark-trace 文件，利用该文件进行 TCP 协议分析。

A. 俘获大量的由本地主机到远程服务器的 TCP 分组

（1）启动浏览器，打开http://gaia.cs.umass.edu/Wireshark-labs/alice.txt网页，得到ALICE'S ADVENTURES IN WONDERLAND文本，将该文件保存到你的主机上。

（2） 打开http://gaia.cs.umass.edu/Wireshark-labs/TCP-Wireshark-file1.html，如图6-6所示，窗口如下图所示。在Browse按钮旁的文本框中输入保存在你的主机上的文件ALICE'S ADVENTURES IN WONDERLAND的全名（含路径），此时不要按“Upload alice.txt file”按钮。

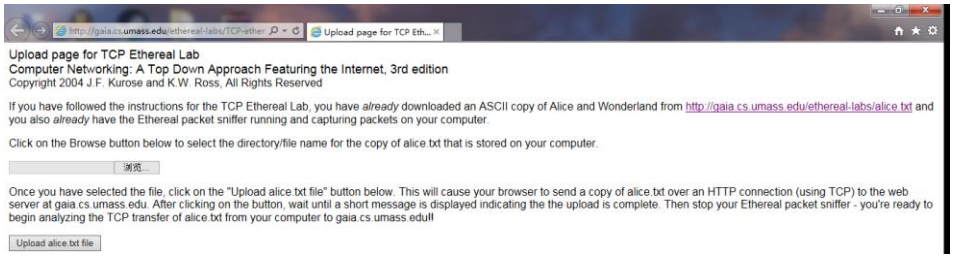

图6-6 Wireshark-labs网页截图

（3） 启动Wireshark，开始分组俘获。

（4） 在浏览器中，单击“Upload alice.txt file”按钮，将文件上传到gaia.cs.umass.edu服务器，一旦文件上传完毕，一个简短的贺词信息将显示在你的浏览器窗口中。

（5） 停止俘获。

B. 浏览追踪信息

在显示筛选规则中输入“tcp”,可以看到在本地主机和服务器之间传输的一系列 tcp 和 http 报文,你应该能看到包含 SYN 报文的三次握手。也可以看到有主机向服务器发送的一个 HTTP POST 报文和一系列的“http continuation”报文。

根据操作思考以下问题：

- 向 gaia.cs.umass.edu 服务器传送文件的客户端主机的 IP 地址和 TCP 端口号是多少？
- Gaia.cs.umass.edu 服务器的 IP 地址是多少？对这一连接，它用来发送和接收 TCP 报文的端口号是多少？

C. TCP 基础

根据操作思考以下问题：

- 客户服务器之间用于初始化 TCP 连接的 TCP SYN 报文段的序号（sequence number）是多少？在该报文段中，是用什么来标示该

报文段是 SYN 报文段的？

- 服务器向客户端发送的 SYNACK 报文段序号是多少？该报文段中，Acknowledgement 字段的值是多少？Gaia.cs.umass.edu 服务器是如何决定此值的？在该报文段中，是用什么来标示该报文段是 SYNACK 报文段的？
- **你能从捕获的数据包中分析出 tcp 三次握手过程吗？**
- 包含 HTTP POST 命令的 TCP 报文段的序号是多少？
- 如果将包含 HTTP POST 命令的 TCP 报文段看作是 TCP 连接上的第一个报文段，那么该 TCP 连接上的第六个报文段的序号是多少？是何时发送的？该报文段所对应的 ACK 是何时接收的？
- 前六个 TCP 报文段的长度各是多少？
- 在整个跟踪过程中，接收端公示的最小的可用缓存空间是多少？限制发送端的传输以后，接收端的缓存是否仍然不够用？
- 在跟踪文件中是否有重传的报文段？进行判断的依据是什么？
- TCP 连接的 throughput (bytes transferred per unit time)是多少？请写出你的计算过程。

（四）IP 分析

通过分析执行 traceroute 程序发送和接收到的 IP 数据包，我们将研究 IP 数据包的各个字段，并详细研究 IP 分片。

A. 通过执行 traceroute 执行捕获数据包

为了产生一系列 IP 数据报，我们利用 traceroute 程序发送具有不同大小的数据包给目的主机 X。回顾之前 ICMP 实验中使用的 traceroute 程序，源主机发送的第一个数据包的 TTL 设位 1，第二个为 2，第三个为 3，等等。每当路由器收到一个包，都会将其 TTL 值减 1。这样，当第 n 个数据包到达了第 n 个路由器时，第 n 个路由器发现该数据包的 TTL 已经过期了。根据 IP 协议的规则，路由器将该数据包丢弃并将一个 ICMP 警告消息送回源主机。

在 Windows 自带的 tracert 命令不允许用户改变由 tracert 命令发送的 ICMP echo 请求消息（ping 消息）的大小。一个更优秀的 traceroute 程序是 pingplotter，下载并安装 pingplotter。ICMP echo 请求消息的大小可以

通过下面方法在 pingplotter 中进行设置。Edit->Options->Packet，然后填写 Packet Size(in bytes，default=56)域。实验步骤：

（1） 启动 Wireshark 并开始数据包捕获

（2） 启动 pingplotter 并“Address to Trace Window”域中输入目的地址。

在“# of times to Trace”域中输入“3”，这样就不过采集过多的数据。Edit->Options->Packet，将 Packet Size(in bytes,default=56)域设为 56，这样将发送一系列大小为 56 字节的包。然后按下“Trace”按钮。得到的 pingplotter 窗口如图 6-7 所示。

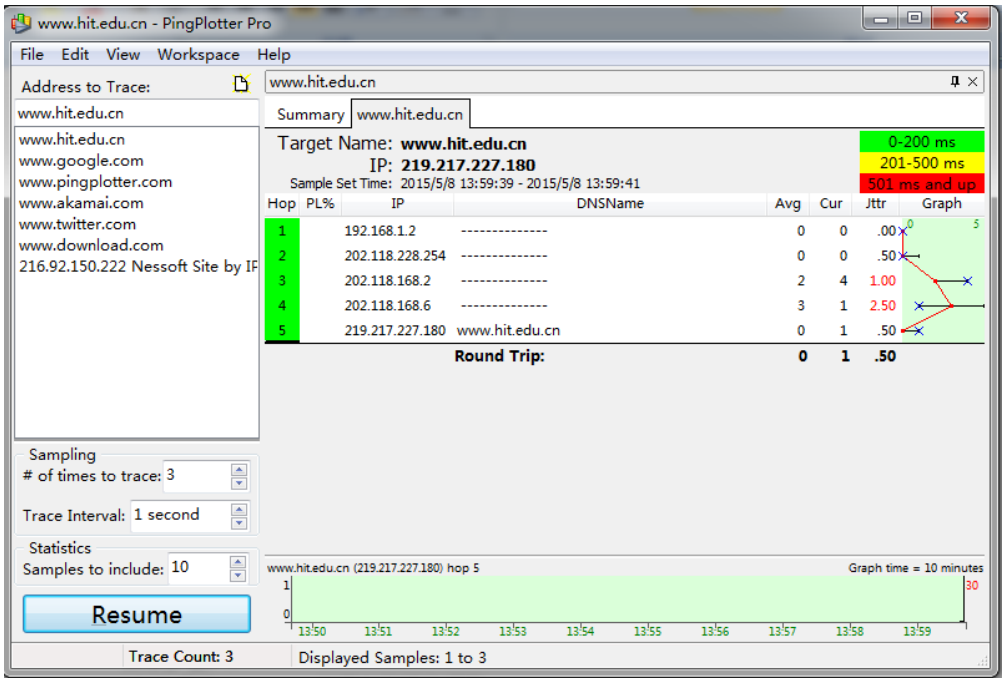

图 6-7 pingplotter 窗口

（1） Edit->Options->Packet，然后将 Packet Size(in bytes,default=56)域改为 2000，这样将发送一系列大小为 2000 字节的包。然后按下“Resume”按钮。

（2） 最后，将 Packet Size(in bytes,default=56)域改为 3500，发送一系列大小为 3500 字节的包。然后按下“Resume”按钮。

（3） 停止 Wireshark 的分组捕获。

注：如无法访问可与实验 TA 联系，下载已有的 ip-Wireshark-trace

文件，利用该文件进行 IP 协议分析

B. 对捕获的数据包进行分析

（1）在你的捕获窗口中，应该能看到由你的主机发出的一系列ICMP Echo Request包和中间路由器返回的一系列ICMP TTL-exceeded消息。选择第一个你的主机发出的ICMP Echo Request消息，在packet details窗口展开数据包的Internet Protocol部分，如图6-8所示。

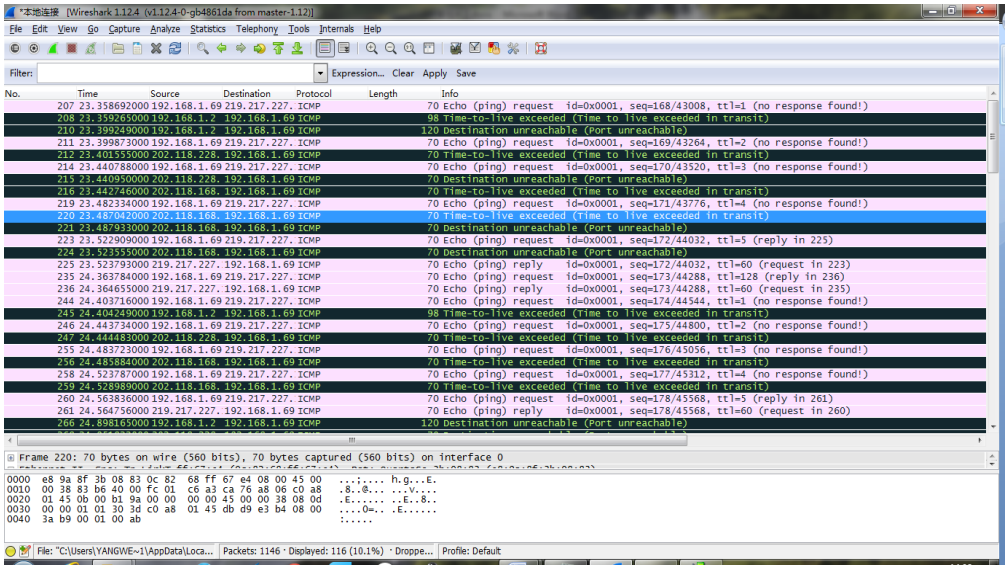

图6-8 Wrieshark窗口

思考下列问题：

- 你主机的IP地址是什么？
- 在IP数据包头中，上层协议（upper layer）字段的值是什么？
- IP头有多少字节？该IP数据包的净载为多少字节？并解释你是怎样确定
- 该IP数据包的净载大小的？
- 该IP数据包分片了吗？解释你是如何确定该P数据包是否进行了分片

（2）单击Source列按钮，这样将对捕获的数据包按源IP地址排序。选择第一个你的主机发出的ICMP Echo Request消息，在packet details窗口展开数据包的Internet Protocol部分。在“listing of captured packets”窗口，你会看到许多后续的ICMP消息（或许还有你主机上运行的其他协议的数

据包）

思考下列问题：

- 你主机发出的一系列ICMP消息中IP数据报中哪些字段总是发生改变？
- 哪些字段必须保持常量？哪些字段必须改变？为什么？
- 描述你看到的IP数据包Identification字段值的形式。

（3）找到由最近的路由器（第一跳）返回给你主机的ICMP Time-to-live exceeded消息。

思考下列问题：

- Identification字段和TTL字段的值是什么？
- 最近的路由器（第一跳）返回给你主机的ICMP Time-to-live exceeded消息中这些值是否保持不变？为什么？

（4）单击Time列按钮，这样将对捕获的数据包按时间排序。找到在将包大小改为2000字节后你的主机发送的第一个ICMP Echo Request消息。

思考下列问题：

- 该消息是否被分解成不止一个IP数据报？
- 观察第一个IP分片，IP头部的哪些信息表明数据包被进行了分片？IP头部的哪些信息表明数据包是第一个而不是最后一个分片？该分片的长度是多少

C. 找到在将包大小改为3500字节后你的主机发送的第一个ICMP Echo Request消息。

思考下列问题：

- 原始数据包被分成了多少片？
- 这些分片中IP数据报头部哪些字段发生了变化？

（五）抓取 ARP 数据包

（1）利用 MS-DOS 命令：arp 或 c:\windows\system32\arp 查看主机上 ARP 缓存的内容。

（2）在命令行模式下输入：ping 192.168.1.82（或其他 IP 地址）

（3）启动 Wireshark，开始分组俘获。抓取的数据包大致如下图 6-9

所示。

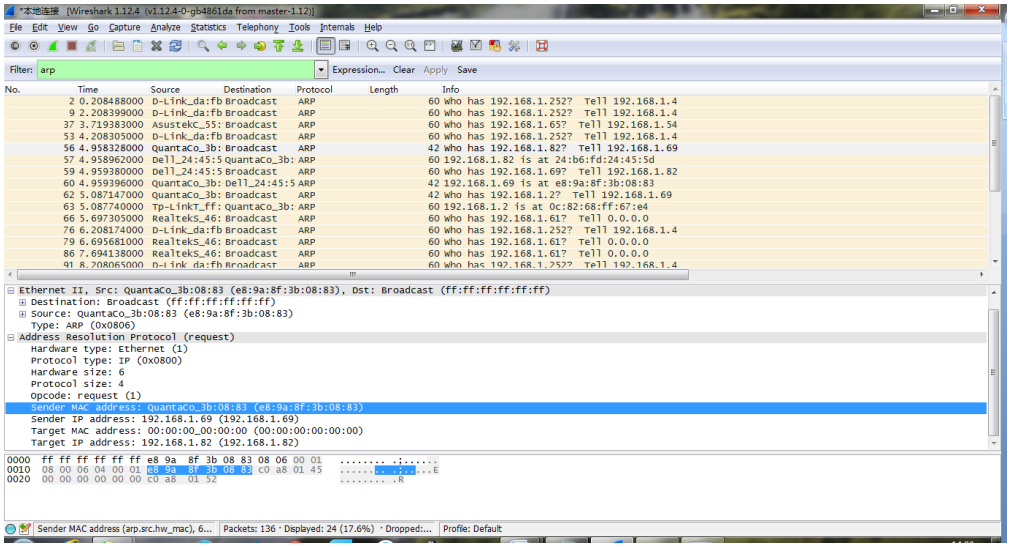

图 6-9 ARP 广播包

从 Wireshark 的第一栏中，我们看到这是个 ARP 解析的广播包，如上图。由于这个版本的 Wireshark 使用的是 Ethernet II 来解码的，我们先看看 Ethernet II 的封装格式。如图 6-10 所示。

图 6-10 Ethernet II 的封装格式

从 Ethernet II 知道了是 ARP 解析以后，我们来看看 Wireshark 是如何判断是 ARP 请求呢还是应答的。

以太网的 ARP 请求和应答的分组格式，如图 6-11 所示。

图 6-11 ARP 请求和应答的分组格式

从上图中我们了解到判断一个 ARP 分组是 ARP 请求还是应答的字

段是“OP”，当其值为 0×0001 时是请求，为 0×0002 时是应答。如下两图：

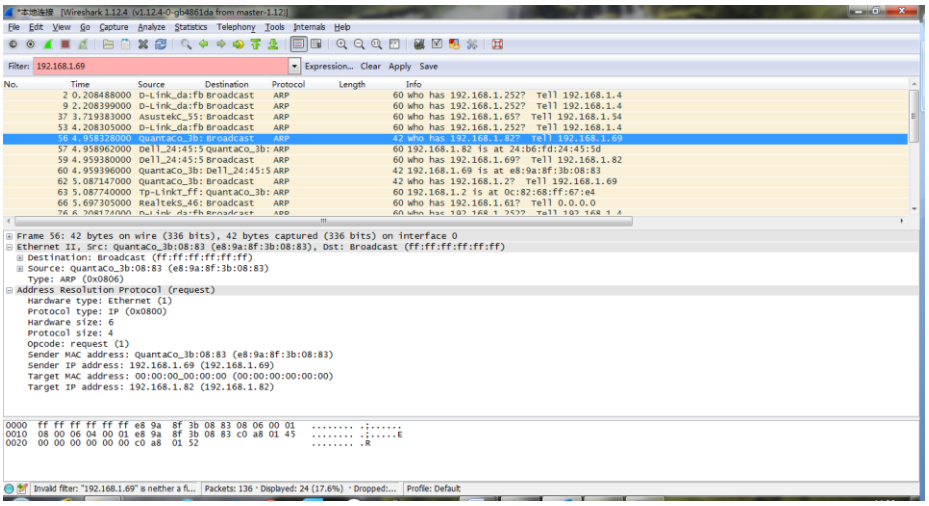

图 6-12 ARP 请求包格式

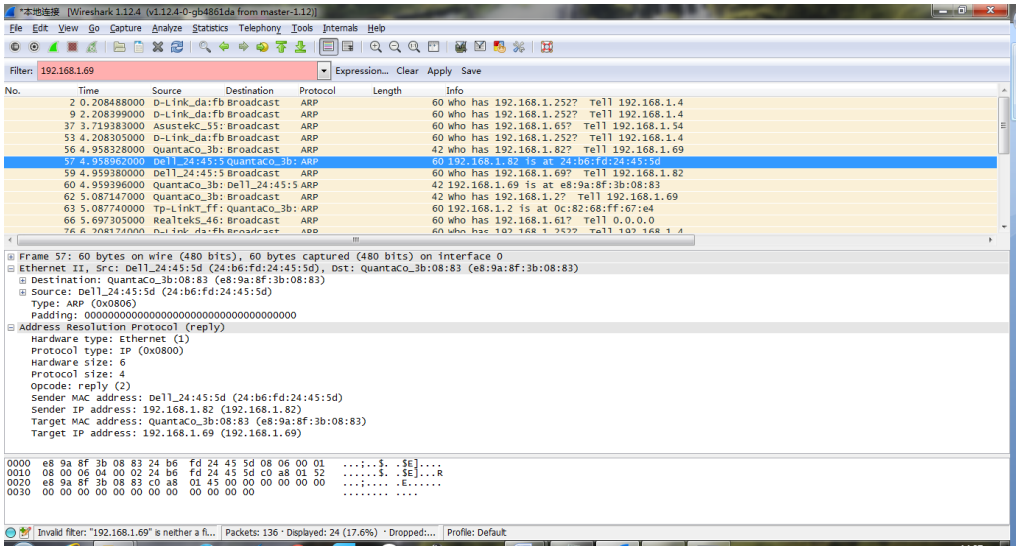

图 6-13 ARP 应答包格式

思考下面问题：

（1）利用 MS-DOS 命令：arp 或 c:\windows\system32\arp 查看主机上 ARP 缓存的内容。说明 ARP 缓存中每一列的含义是什么?

（2）清除主机上 ARP 缓存的内容,抓取 ping 命令时的数据包。分析数据包,回答下面的问题：

➢ ARP数据包的格式是怎样的？由几部分构成，各个部分所占的字

节数是多少？

- 如何判断一个ARP数据是请求包还是应答包？
- 为什么ARP查询要在广播帧中传送，而ARP响应要在一个有着明确目的局域网地址的帧中传送？

（六）抓取 UDP 数据包

（1）启动 Wireshark，开始分组捕获；

（2）发送 QQ 消息给你的好友；

（3）停止 Wireshark 组捕获；

（4）在显示筛选规则中输入“udp”并展开数据包的细节，如图 6-14 所示。

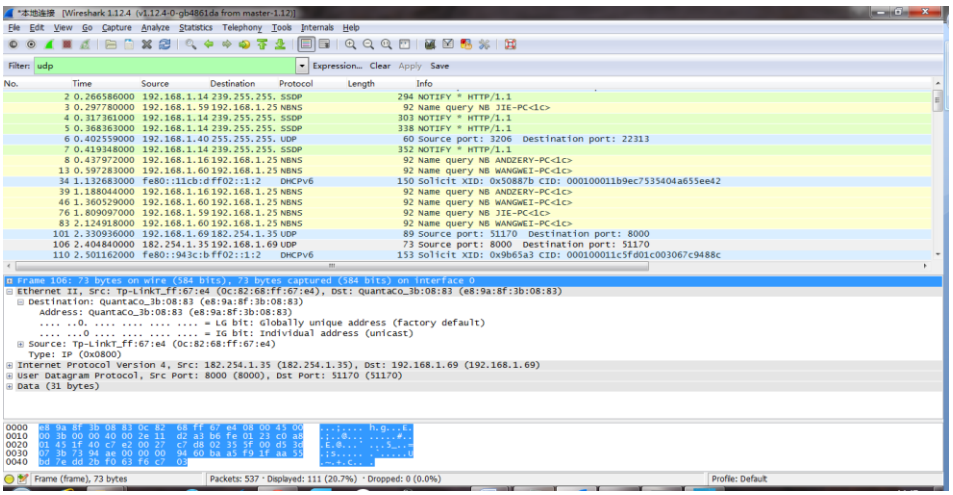

图 6-14 UDP 抓包图

分析 QQ 通讯中捕获到的 UDP 数据包。根据操作思考以下问题：

- 消息是基于UDP的还是TCP的？
- 你的主机ip地址是什么？目的主机ip地址是什么？
- 你的主机发送QQ消息的端口号和QQ服务器的端口号分别是多少？
- 数据报的格式是什么样的？都包含哪些字段，分别占多少字节？
- 为什么你发送一个ICQ数据包后，服务器又返回给你的主机一个ICQ数据包？这UDP的不可靠数据传输有什么联系？对比前面的TCP协议分析，你能看出UDP是无连接的吗？

（七）利用 WireShark 进行 DNS 协议分析

（1）打开浏览器键入:www.baidu.com

（2）打开 Wireshark,启动抓包.

（3）在控制台回车执行完毕后停止抓包.Wireshark 捕获的 DNS 报文如图 6-15 所示。

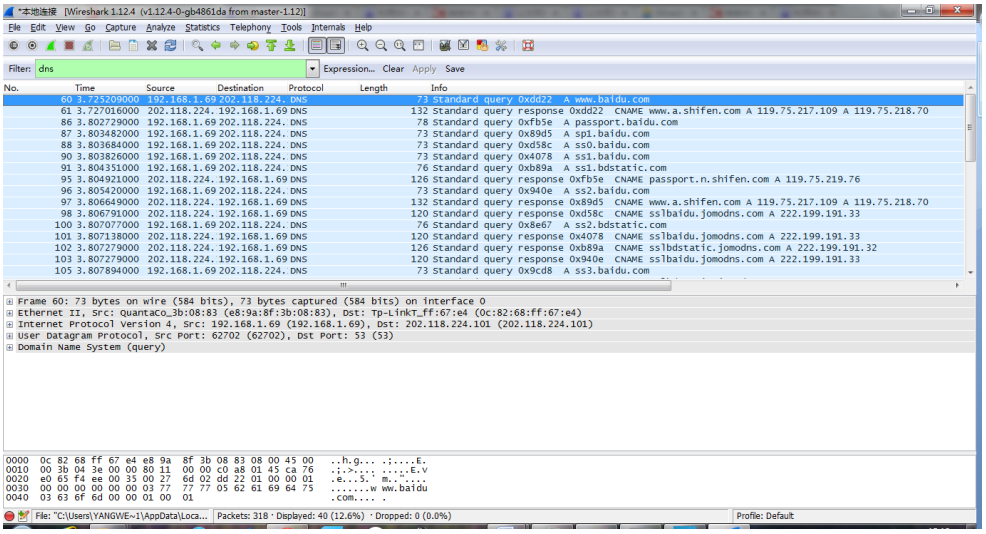

图 6-15 DNS 报文

6. 实验报告

要求撰写实验报告对利用 Wireshark 分析 HTTP、TCP、IP、以太网帧、ARP、DNS 等的抓包分析实验过程、发现的问题、得到的结果、对协议的认识等内容进行总结（可结合每个实验后面的思考题进行分析、总结）。