北京郵電大學

Beijing University of Posts and Telecommunications

《计算机网络》课程设计

——DNS中继服务器的实现

实验报告

成员：

目录

一. 需求分析

DNS是Domain Name System域名系统的缩写，为其他因特网应用提供支持其采用Client-Server模式，在传输层主要使用UDP，特点是层次化的命名空间：主机的域名是分级命名的和采用分布式数据库存储和管理域名。

我们要设计的DNS中继服务器有**DNS服务器、不良网站拦截和DNS中继三种功能**。具体实现方式为：首先读入“域名-IP地址”对照表，当客户端查询域名对应的IP地址时，用域名检索该对照表，①检索结果为有效IP地址，则向客户返回这个地址（即DNS服务器功能）②检索结果为IP地址0.0.0.0，则向客户端返回“域名不存在”的报错消息（即不良网站拦截功能）③表中未检到该域名，则向实际的本地DNS服务器发出查询，并将结果返给客户端（即DNS中继功能）。

在这三种基础功能实现的基础上，还需要考虑到两种特殊情况：①多个计算机上的客户端同时查询，因而需要进行**消息ID的转换的情况**，②由于UDP的不可靠性，考虑请求外部DNS服务器，却不能得到应答或者收到迟到应答的情形，即**超时的处理**。

消息ID的转换通过自定义的数据结构IDTranslate的一个数组实现：

```
//ID转换表结构
typedef struct IDChange
{
    unsigned short oldID;                    //原有ID
    BOOL done;                               //标记是否完成解析
    SOCKADDR_IN client;                      //请求者套接字地址
    int expireTime;                          //超时时间
} IDTransform;
```

超时的处理：考虑超时的设置方式时，因为超时设置主要是考虑到请求外部DNS服务器时，由于UDP的不可靠性，不能得到应答或者收到迟到应答的情形。也就是说，只有在域名不在cache和域名解析表里，需要同外部DNS通信时要考虑超时问题。联想到ID转换表也是只在与外部DNS通信时调用，所以考虑把超时问题与ID转换问题相结合，所以在设置ID转化表结构时加入了超时时间，如果长时间未接收到返回ID，直接把ID抛弃，也就实现了超时请求的忽略。

分析完整体功能后，经过对DNS运行流程的仔细研究，我们认为程序还要实现：**文件更新功能**和**cache高速缓存器功能**。（实现方式见二、程序实现要点）

二. 程序实现要点

①一些支持功能

项目	除A类型外支持的其他类型查询	Socket数目	端口号	非阻塞	并发
结果	无	2	53	是	否

②DNS文件内容：
IP地址＋空格＋域名

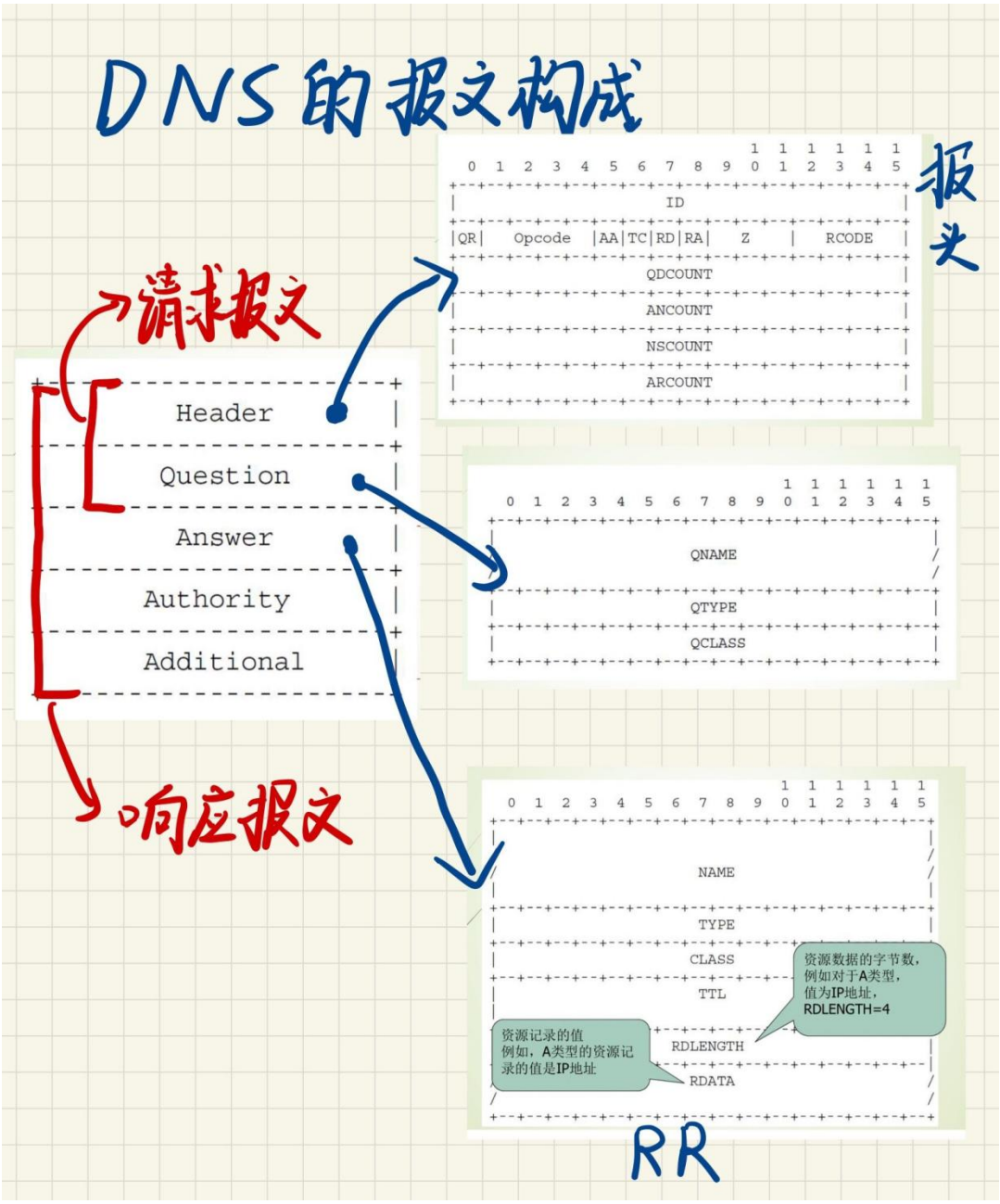
③文件如何更新：

在每当接收到本地文件和原cache中没有储存的域名和IP，把其存入cache后就当即写入域名解析表，即本地文件在程序中的储存方式；接着用addToFile函数将据写入文件，实现文件的更新功能。

```
void addToFile(char* url, char* ip)
{
    FILE* file;
    if ((file = fopen(filePath, "a")) == NULL)//文件不存在
    {
        printf("文件打开错误！");
    }
    fputs("\n", file);
    fputs(ip, file);
    fputs(" ", file);
    fputs(url, file);
    fclose(file);
    printf("已成功把新IP写入文件。\n");
}
```

④DNS消息格式：

DNS报文的构成分析如下，响应报文为在请求报文后添ANSWER和其他部分构成。



在程序中，并没有设置专门的struct存储报文结构，而是采用字节截断的方式控制更改报文不同的内容，并以详细注释的方式增加可读性。如下图构造响应报文的例子：

```
//FLAG部分
unsigned short a;
a = htons(0x8180);//wireshark
memcpy(&bufSend[2], &a, sizeof(unsigned short));

//ANCOUNT answer section部分RR个数
a = htons(0x0001);
memcpy(&bufSend[6], &a, sizeof(unsigned short));

//Answer部分
unsigned short tmp;
unsigned long tmp1;
char answer[16];
int len = 0;
//Name
tmp = htons(0xc00c); // 开头两个11代表指向域名的指针
memcpy(answer, &tmp, sizeof(unsigned short));
len += sizeof(unsigned short);
//Type
tmp = htons(0x0001); //typeA
memcpy(answer + len, &tmp, sizeof(unsigned short));
len += sizeof(unsigned short);
```

⑤如何实现消息ID的映射：
程序维护着一个ID转换表数组，数组定长，通过registerNewID函数更新

```
//更新ID转换表中ID
unsigned short registerNewID(unsigned short ID, SOCKADDR_IN temp)
{
    int i = 0;
    //int flag = 0;
    for (i = 0; i != MAX_ID_TRANS_TABLE_SIZE; ++i)
    {
        if (checkIDExpired(&IDTransTable[i]) == 1 || IDTransTable[i].done == TRUE)//如果是需要更新的条目
        {
            IDTransTable[i].oldID = ID;
            IDTransTable[i].client = temp;
            IDTransTable[i].done = FLASE;
            setIDExpire(&IDTransTable[i], ID_EXPIRE_TIME);
            IDcount++;
            if (debugLevel >= D)
            {
                printf("ID%d注册成功。\n现有ID数目：%d\n", i + 1, IDcount);
            }
            //flag = 1;
            break;
        }
    }
    if (i == MAX_ID_TRANS_TABLE_SIZE)
        return 0;
    return (unsigned short)i + 1;
}
```

⑥异常处理功能及实现方式

程序在多处实现了差错控制，尤其是在文件打开和socket编程过程中，基本上每一次打开文件或发送接收socket信息都会对函数的返回值加以判断和处理。此外，在用户输入时也同样加入了输入IP是否合法的判断，进行差错控制。

举例如下：

Bind Socket时：

```
if (bind(local_sock, (struct sockaddr*)&local_name, sizeof(local_name)) < 0)
{
    if (debugLevel >= 1) printf("Bind socket port failed.\n");
    exit(1);
}
printf("Bind socket port successfully.\n");
```

判断输入IP是否合法时：

```
//判断输入IP地址合法
int ifLegalIP(char* ip)
{
    int a, b, c, d;
    if (4 == sscanf(ip, "%d.%d.%d.%d", &a, &b, &c, &d))
    {
        if (0 <= a && a <= 255 && 0 <= b && b <= 255 && 0 <= c && c <= 255 && 0 <= d && d <= 255)
        {
            return 1;
        }
        else  return 0;
    }
    else  return 0;

}
```

打开文件时：

```
FILE* file;
if ((file = fopen(filePath, "a")) == NULL)//文件不存在
{
    printf("文件打开错误！");
}
```

⑦其他增强功能：Cache缓存

第一版代码出炉后，我们开始思考如何完善查找效率以及增加扩展功能。虽然我们没有找出将查找效率从O（n）提高的方法，但是我们从扩展功能的缓存查询里得到了思路，认为如果增加高速缓存器cache，**使得程序避免多次繁琐查找文件内容**，同样能提高查找效率，因此我们引入了cache功能。

在设计cache部分时，我们最开始只是有一个朴素的按照读取概率维持固定大小数组的思路，查找资料后发现与LRU算法不谋而合，于是采用了Least Recently Used最近最少使用算法完成了Cache的维护。

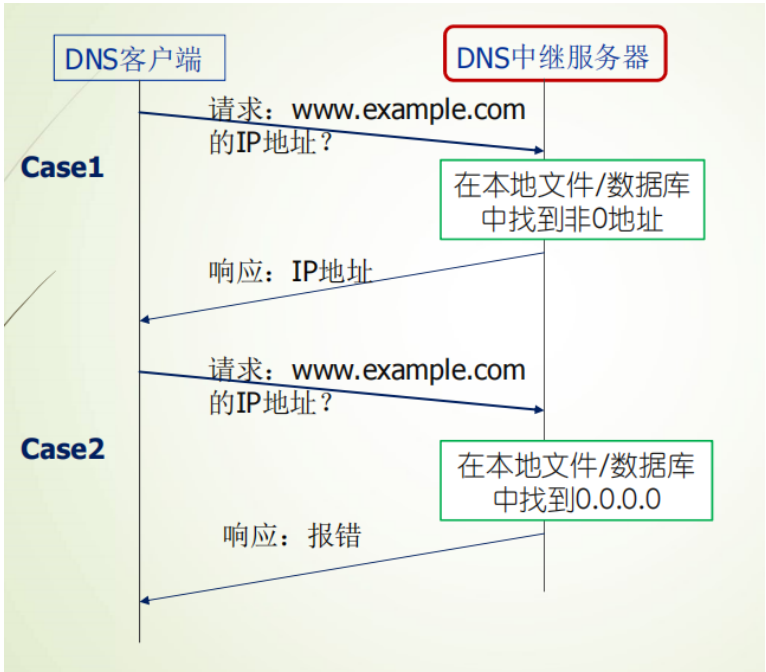
Cache表的维护：

```
void addToCache(char* url, char* ip) //利用LRU算法，使最近使用的域名排在最前面，提升效率
{
    int i, j;
    int place = -1;
    for (i = 0; i < cacheCount; i++)
    {
        if (strcmp(url, cache[i].domain) == 0) { place = i; break; }
    }
    if (place > -1)//如果在cache内
    {
        for (j = i - 1; j >= 0; j--)
        {
            cache[j + 1] = cache[j];
        }
    }
    else//不在
    {
        for (i = cacheCount - 2; i >= 0; i--)
        {
            cache[i + 1] = cache[i];
        }
        if (cacheCount < MAX_CACHE_SIZE) cacheCount++;
    }
    strcpy(cache[0].domain, url);
    strcpy(cache[0].IP, ip);
}
```

每当获得一个IP-URL对应关系，都将它添加到cache中：

```
//打印资源记录里的IP地址
sprintf(ip, "%d.%d.%d.%d", ip1, ip2, ip3, ip4);
if (debugLevel >= D) printf("IP address : %d.%d.%d.%d\n", ip1, ip2, ip3, ip4);
//加到cache中
addToCache(url, ip);
addToTable(url, ip);
addToFile(url, ip);
```

三.　　系统的功能设计



①DNS服务器功能

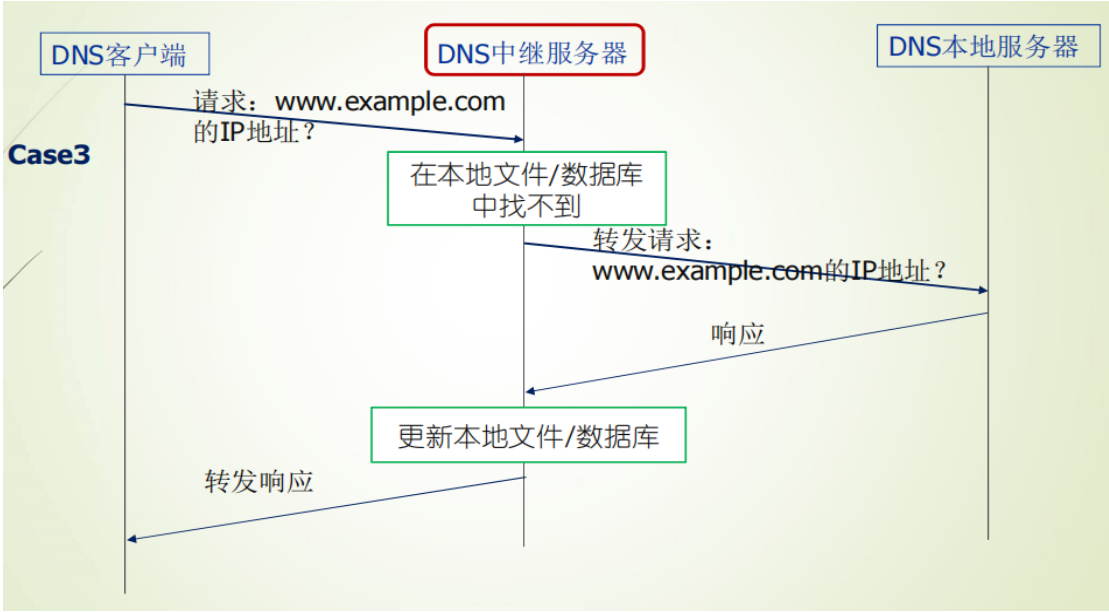
首先从收到的请求报文中解析域名url，接着在本地资源内部，即文件或数据库读取到程序内部的对应域名解析表中，或在程序中维护的cache中进行查找，如果可以成功找到对应的IP地址，则使用解析出的域名url和查找到的IP地址按照RFC1035协议

构造出相应报文，发回给客户端。

②不良网站拦截功能

在进行查询时，如果查询出的IP地址为“0.0.0.0”，即内部资源记录显示该域名为非法网站，此时对请求报文进行相应操作，也就是将RCODE更改为“3”：域名记录不存在，接着把报文发回，以实现不良网站的拦截功能。

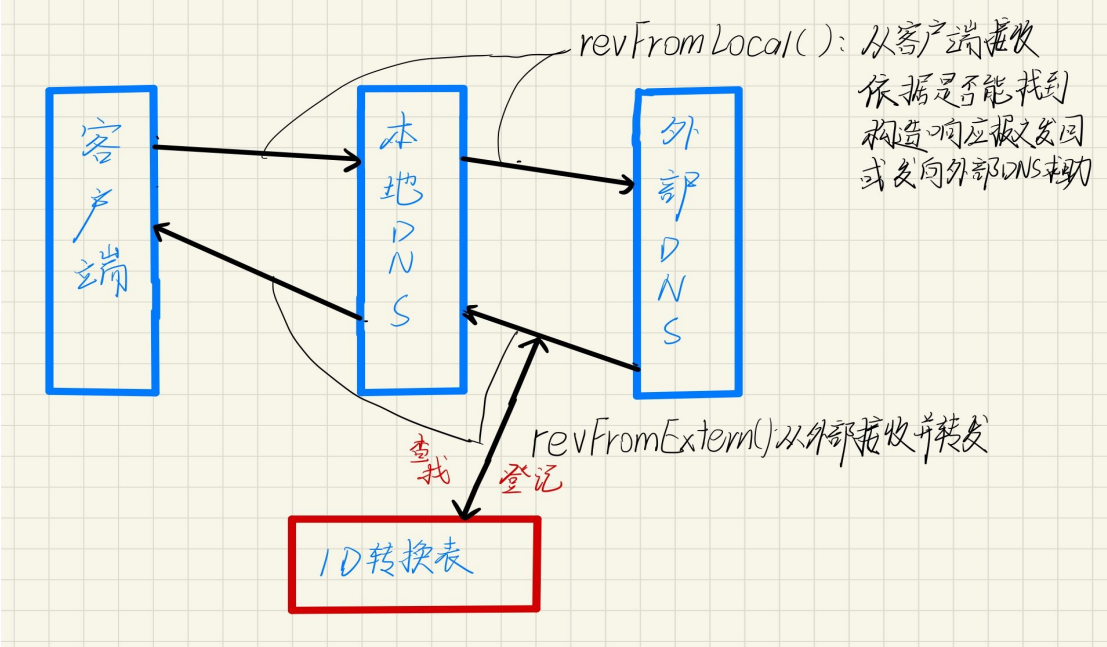
③DNS中继功能



当在本地资源内部查询解析出的url失败时，转向外部url进行询问。此时由于多客户端的存在需要对用户请求报文的id进行更改，然后将其转发给外部DNS，即DNS的中继功能。

四. 模块划分

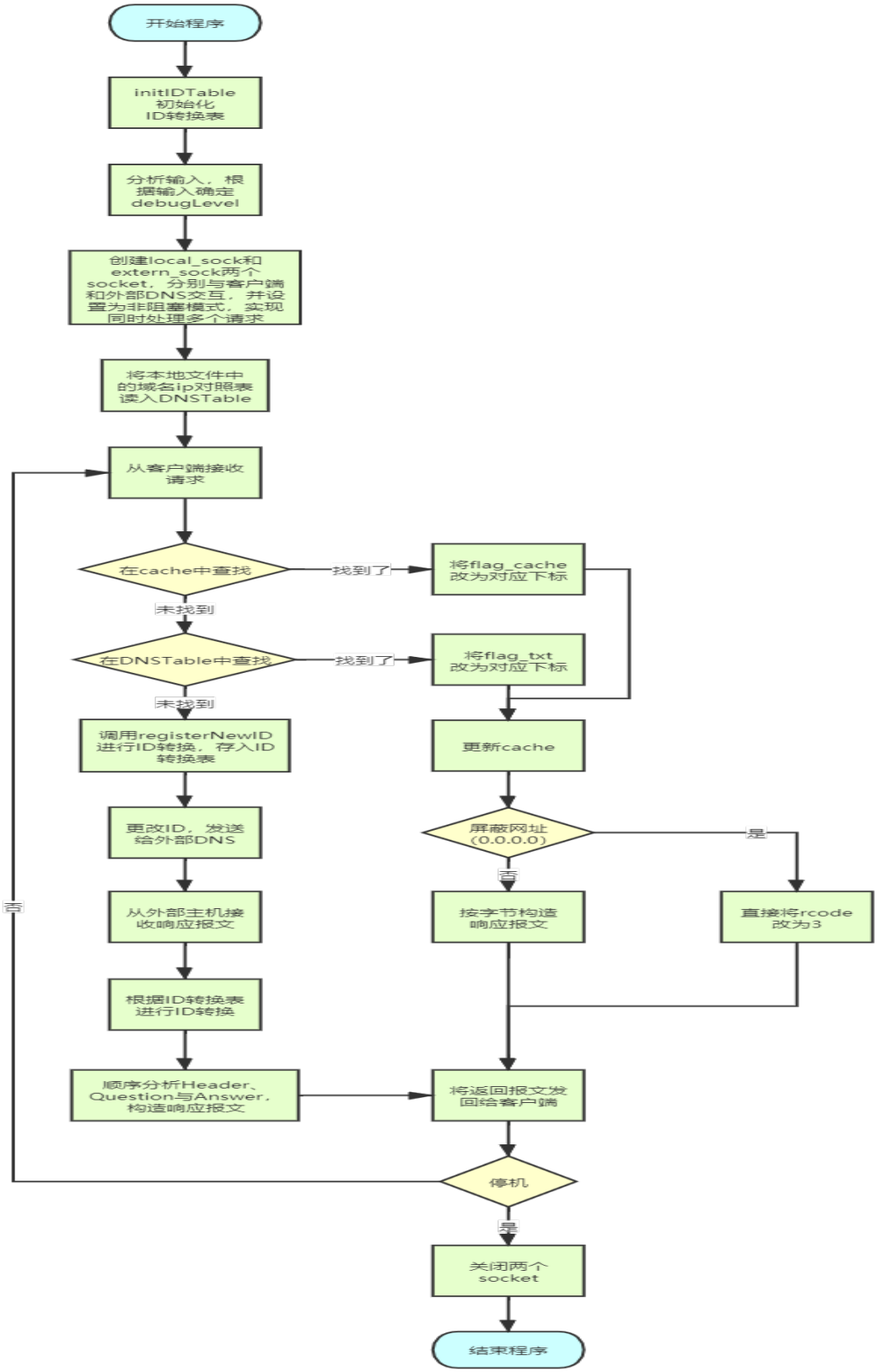
在构思整体处理方式时，最初的想法是常规的接收报文→分析报文类型→进行查找或解析操作，而这一切必然要放到一个main函数里完成，造成主函数的臃肿。在对思路进行抽象，尤其是在仔细研究了socket编程的接口问题，理清了收发信息的方式后，认为站在本地DNS的角度看待运行过程，会发现只有“从客户端接收→分析判断→选择性发送给外部DNS” 和 “从外部DNS接收→解析→发送给客户端”这两种进程，因此构思了“revFromLocal”和“revFromExtern”两个主要函数。



确立了整体夸奖后，我们又经过分析需求和实际，划分出了三个模块：用户交互层、数据管理层和应用服务层。

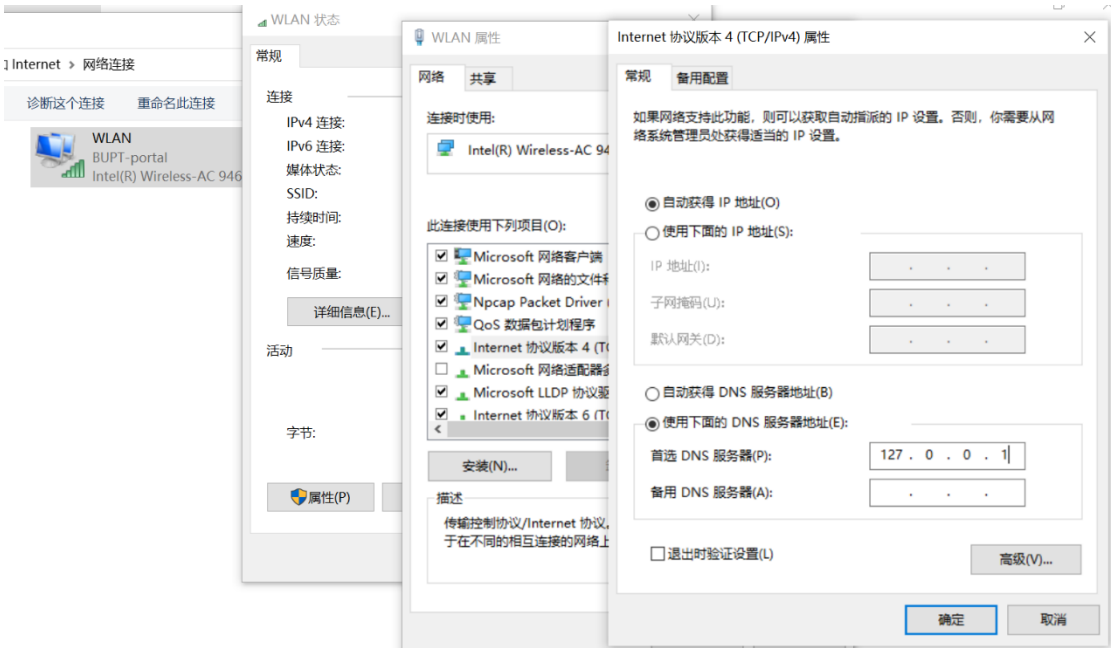
①用户交互层接收cliet发送的DNS请求报文，以及根据用户需要的调试等级，将所需打印在控制台中；
③数据管理层将本地数据读入内存，并使用了Cache配合动态调整内存中存储数据，整个层次为应用服务层提供服务。
②应用服务层客户端查询域名对应的IP地址时，用域名检索域名解析表，进行DNS服务器、不良网站拦截和DNS中继三大功能。

四、软件流程图



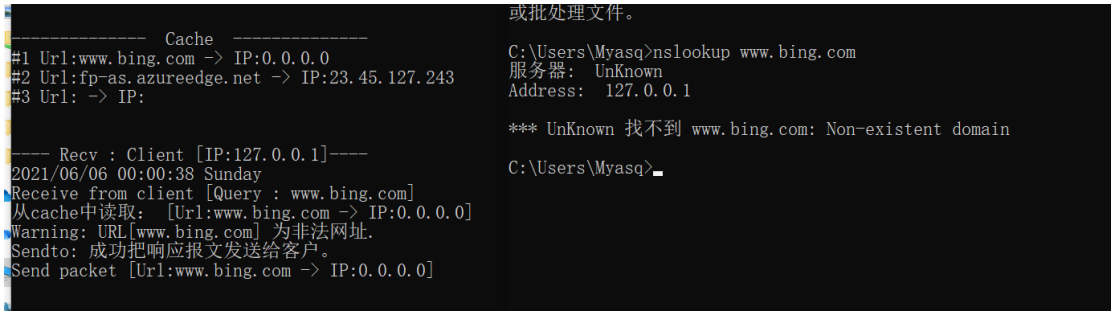
五、测试用例以及运行结果

①设置DNS服务器地址为127.0.0.1



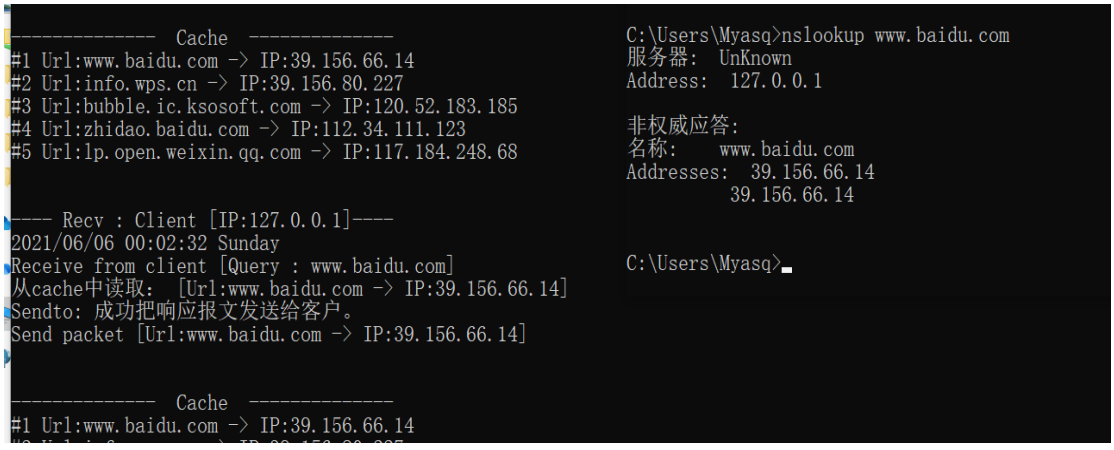
②首先查看dnsrelay.txt，即本地文件中存储信息

②用 nslookup 查询DNS 数据库/文件中的 IP 地址为0.0.0.0 的域名：



③用 nslookup 查询 DNS 数据库/文件中没有的域名：

④用nslooku查询上面查询过的域名：



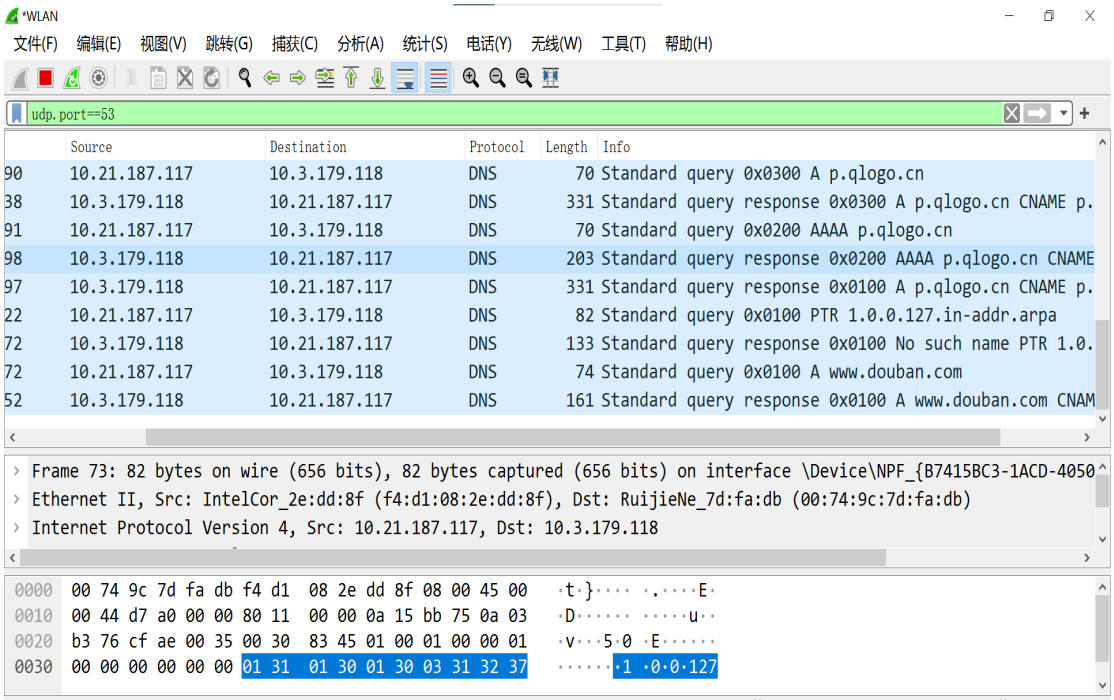
⑤查看文件更新情况：

dnsrelay.txt - 记事本

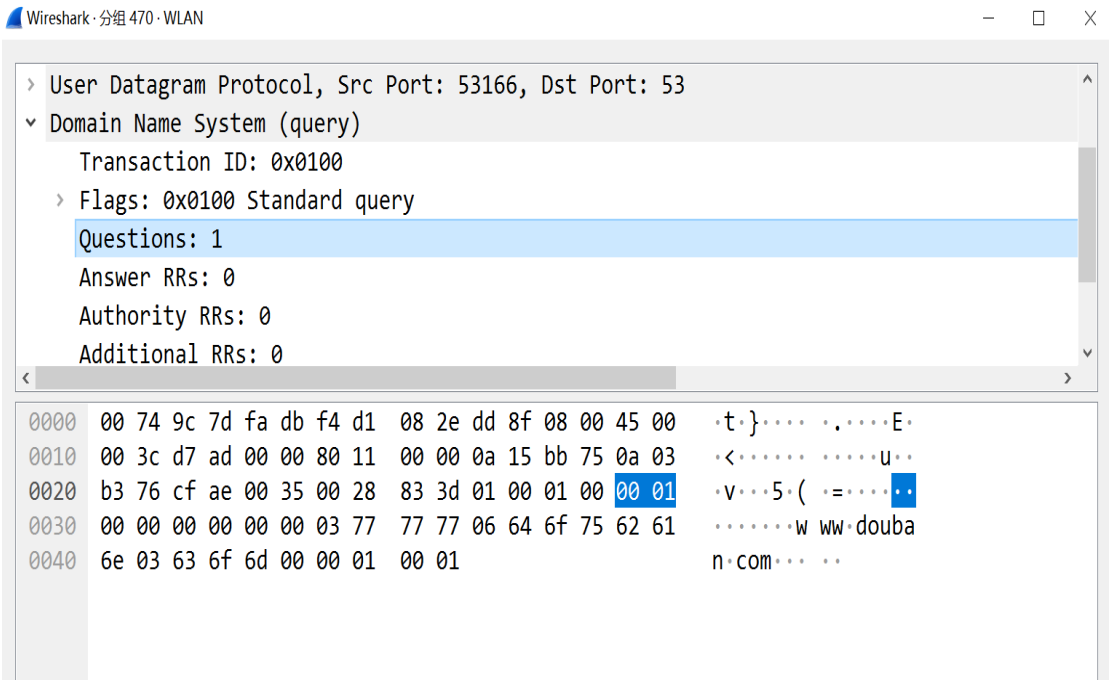
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
0.0.0.0 www.blogger.com
0.0.0.0 www.4399.com
0.0.0.0 www.bing.com
13.107.18.254 k-ring.msedge.net
13.107.246.49 fp-afd-nocache.azureedge.net
23.45.127.243 fp-as.azureedge.net
39.156.66.14 www.baidu.com
117.184.248.68 lp.open.weixin.qq.com
112.34.111.123 zhidao.baidu.com
120.52.183.185 bubble.ic.ksosoft.com
39.156.80.227 info.wps.cn
```

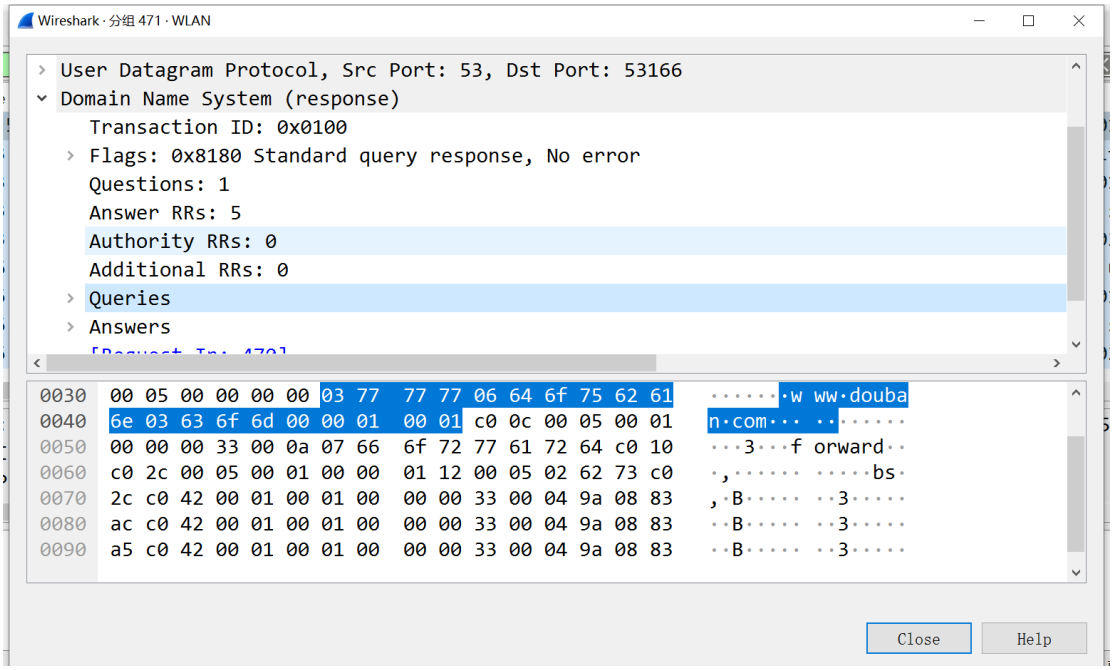
⑥设置wireshark端口，查看抓包情况



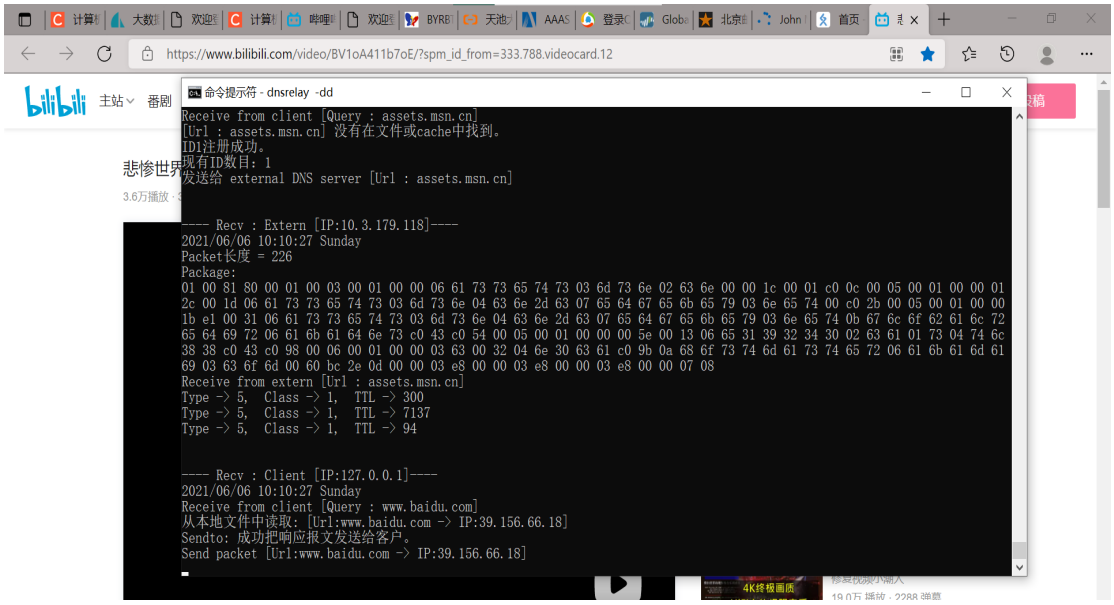
⑦查看请求报文的具体信息：（以www.douban.com为例）



⑧查看其相对应的响应报文：（以www.douban.com为例）



⑨在电脑上快速点击多个链接，测试多客户端并发的实现情况：



至此，在本机上的dnsrelay测试运行完成，使用两台电脑进行交互运行的dnsrelay的结果已在验收时展示。

六. 调试中遇到并解决的问题

①文件打开失败

最开始进行调试时，文件在VS内调试显示可以打开，用cmd运行却一直失败，分析后发现：在VS内使用调试功能运行的，打开的文件路径是存放程序的文件夹内的“dnsrelay.txt”；而用cmd运行时，由于生成的.exe文件存放在项目中的debug文件夹内，所以只能在debug目录下打开本地文件。将文件放置到debug目录下后成功打开。

②超时timeout

程序第一遍运行时，出现的错误是所以查询都timeout无法出现结果，第一遍改正发现了一个报文构造错误，即应该使用4字节的地方误用了2字节的unsigned short；在改正后，沮丧的发现还是无法正确运行。此时想到用wireshark抓包，发现是发送异常。最后发现是dnsServerIP设置错误，还是由于概念不清，把ipconfig/all查询出的DHCP服务器误认为成DNS，造成了发送失败。更正后成功解决了timeout问题。

③“0.0.0.0”问题

最开始时，查找到非法IP只会把answer数设置为0，但运行后发现程序只会正常把IP输出为“0.0.0.0”，并不会出现示例中提供的找不到的情况。在查找资料后，我们发现：需要把RCODE设置为3，即表明此域名不存在才可以出现正确的提示。设置后果然运行成功。

④两台电脑交互问题

由于验收需求两台电脑，所以在进行验收测试时由于校园网和IP的问题经

常出现辅助电脑和主电脑无法交互，或是一方没有相应问题。经过多次的实验和抓包观察，我们终于总结出了一定的规律，即发现了更改WIFI连接后本机IPv4地址会发生相应的变化；同时连接BUPT-mobile仍会无法接收信息，只有BUPT-portal可以成功交互；必须先在有网条件下连接WIFI等等。很多次的实验以及规律总结最终保证了验收时的稳定运行。

七. 小组成员分工及承担任务比例

八.

八、心得体会

本次计网课设成功使得小组成员都加深了对DNS中继器的理解，还学习了socket编程的有关内容，更是通过对wireshark的反复使用一步步加深了对计算机间信息传输的理解。通过小组合作，我们更是体会到了小组合作的意义，三个人在思想碰撞中感受到了合作的重要性。

附：源代码

Head.h:

```
#pragma once
#define LOCAL_ADDRESS "127.0.0.1"      //本地DNS服务器地址
#define MAX_BUF_SIZE 1024         //最大缓存长度
#define DNS_PORT 53               //端口号
#define MAX_ID_TRANS_TABLE_SIZE 16 // 转换表大小
#define ID_EXPIRE_TIME 10         //超时时间
#define MAX_CACHE_SIZE 5          //cache大小
#define DNS_HEAD_SIZE 12          //header大小
#define AMOUNT 300
#define TRUE  1
#define FLASE 0
#define D 1                    //调试等级D
#define DD 2                   //调试等级DD（详细）

#include <WinSock2.h>
#include<stdio.h>
#include<time.h>
#pragma comment(lib, "ws2_32.lib")
#pragma warning(disable:4996)//把strcpy_s的错误提示消除
```

```
//ID转换表结构
typedef struct IDChange
{
    unsigned short oldID;                   //原有ID
    BOOL done;                              //标记是否完成解析
    SOCKADDR_IN client;                     //请求者套接字地址
    int expireTime;                 //超时时间
} IDTransform;

//DNS域名解析表结构
typedef struct translate
{
    char IP[16];                        //IP地址
    char domain[65];                        //域名
} Translate;

int IDcount = 0;                        //ID转换表中的条目个数
int cacheCount = 0;
int DNSNum = 0;                 //DNS域名解析表中数目
int debugLevel = 0;                     //调试等级
char dnsServerIP[16] = "10.3.179.118";
char filePath[AMOUNT]="dnsrelay.txt";


Translate DNSTable[AMOUNT];                 //DNS域名解析表
Translate cache[MAX_CACHE_SIZE];        //cache表
IDTransform IDTransTable[MAX_ID_TRANS_TABLE_SIZE];          //ID转
换表


WSADATA wsaData;
SOCKET local_sock, extern_sock;

struct sockaddr_in local_name, extern_name;//IPv4 网络协议地址
struct sockaddr_in client, external;
int length_client = sizeof(client);


//主要函数
void initIDTable();//初始化ID转换表
void dealPara(int argc, char* argv[]);//分析输入参数
int readLocalData();//读本地文件
void receiveFromExtern();//从extern接收报文，解析后发送
void receiveFromLocal();//从client接收报文
```

```
//辅助函数
int ifLegalIP(char* ip);          //判断输入IP合法
void addToCache(char* url, char* ip);//添加到cache
void outputCache();               //输出cache
void addToTable(char* url, char* ip);//添加到DNS域名解析表
void addToFile(char* url, char* ip);//添加到文件内
void setIDExpire(IDTransform* record, int ttl);//设置超时时间
int checkIDExpired(IDTransform* record);//检查超时
unsigned short registerNewID(unsigned short ID, SOCKADDR_IN temp);//在
ID转换表中添加ID
void getUrl(char* buf, char* dest);//转换域名格式
void outputPacket(char* buf, int length);//输出报文
```

Main.h

```
#include"head.h"

int main(int argc, char* argv[])
{
    dealPara(argc, argv);//分析输入（调试等级）
    initIDTable();//初始化ID转换表

    WSAStartup(MAKEWORD(2, 2), &wsaData);//使用2.2版本的Socket

    local_sock = socket(AF_INET, SOCK_DGRAM, 0);
    extern_sock = socket(AF_INET, SOCK_DGRAM, 0);

    //使用非阻塞性socket
    int non_block = 1;
    ioctlsocket(extern_sock, FIONBIO, (u_long FAR*) & non_block);
    ioctlsocket(local_sock, FIONBIO, (u_long FAR*) & non_block);
    if (local_sock < 0)
    {
        if (debugLevel >= D) printf("socket创建失败.\n");
        exit(1);
    }
    printf("socket创建成功.\n");
    //设置local socket
    local_name.sin_family = AF_INET; //协议簇设为TCP/IP
    local_name.sin_addr.s_addr = INADDR_ANY; //IP地址设为any
    local_name.sin_port = htons(DNS_PORT);  //把端口设为53
```

```
    //设置extern socket
    extern_name.sin_family = AF_INET;
    extern_name.sin_addr.s_addr = inet_addr(dnsServerIP);
    extern_name.sin_port = htons(DNS_PORT);

    int reuse = 1;
    setsockopt(local_sock, SOL_SOCKET, SO_REUSEADDR, (const
char*)&reuse, sizeof(reuse));
    //SO_REUSEADDER：允许重用本地地址和端口

    if (bind(local_sock, (struct sockaddr*)&local_name, sizeof(local_name)) <
0)
    {
        if (debugLevel >= 1) printf("Bind socket port failed.\n");
        exit(1);
    }
    printf("Bind socket port successfully.\n");

    DNSNum=readLocalData();

    while (TRUE)
    {
        receiveFromLocal();
        receiveFromExtern();
    }

    return 0;
}

//初始化ID转换表
void initIDTable()
{
    int i;
    IDcount = 0;
    for (i = 0; i < MAX_ID_TRANS_TABLE_SIZE; i++)
    {
        IDTransTable[i].oldID = 0;
        IDTransTable[i].done = TRUE;
        IDTransTable[i].expireTime = 0;
        memset(&(IDTransTable[i].client), 0, sizeof(SOCKADDR_IN));
    }
}

void dealPara(int argc, char* argv[])
{
    switch (argc)
    {
    case 1: debugLevel = 0; break;
    case 2:
```

```
            if (strcmp(argv[1], "-d") == 0) debugLevel = D;
            if (strcmp(argv[1], "-dd") == 0) debugLevel = DD;
            break;
        case 3:
            if (strcmp(argv[1], "-d") == 0)
                debugLevel = D;
            else if (strcmp(argv[1], "-dd") == 0)
                debugLevel = DD;
            if (ifLegalIP(argv[2]))//判断输入IP是否合法
            {
                strcpy(dnsServerIP, argv[2]);
                printf("DNS server 已设置为 : %s\n", argv[2]);
            }
            else
            {
                printf("Warning:%s不是合法的ip地址\n", argv[2]);
            }
            break;

        case 4:
            if (strcmp(argv[1], "-d") == 0)
                debugLevel = D;
            else if (strcmp(argv[1], "-dd") == 0)
                debugLevel = DD;
            if (ifLegalIP(argv[2]))//判断输入IP是否合法
            {
                strcpy(dnsServerIP, argv[2]);
                printf("DNS server 已设置为 : %s\n", argv[2]);
            }
            else
            {
                printf("Warning:%s不是合法的ip地址\n", argv[2]);
            }
            strcpy(filePath, argv[3]);//使用用户输入的配置文件路径
            break;
        default:
            printf("输入格式错误\n");
        }
    }

    void outputCache()
    {
        printf("\n\n----------  Cache  ----------\n");
        int i = 0;
        for (i = 0; i < cacheCount; i++)
        {
            printf("#%d Url:%s -> IP:%s\n", i + 1, cache[i].domain, cache[i].IP);
        }
```

```
}

//判断输入IP地址合法
int ifLegalIP(char* ip)
{
    int a, b, c, d;
    if (4 == sscanf(ip, "%d.%d.%d.%d", &a, &b, &c, &d))
    {
        if (0 <= a && a <= 255 && 0 <= b && b <= 255 && 0 <= c && c <= 255
&& 0 <= d && d <= 255)
        {
            return 1;
        }
        else  return 0;
    }
    else  return 0;

}

//从dnsrelay.txt中读取DNS域名解析表
int readLocalData()
{
    int i = 0;
    FILE* file;
    char ip[16], url[65];
    if ((file = fopen(filePath, "r")) == NULL)//文件不存在
    {
        printf("文件打开错误！");
        return -1;
    }
    while (fscanf(file, "%s %s", ip, url) != EOF && i < AMOUNT)//当文件没有读
完并且解析表没有满时
    {
        if (debugLevel == DD)
            printf("从文件 “%s”中读入：[Url:% s, IP : % s]\n",filePath, url, ip);
        strcpy(DNSTable[i].IP, ip);
        strcpy(DNSTable[i].domain, url);
        i++;
    }
    if (i == AMOUNT - 1) printf("域名解析表已满！");
    fclose(file);
    printf("文件已成功读入。\n");
    return i - 1;//返回域名解析表中的条目个数
}

//设立超时时间
void setIDExpire(IDTransform* record, int ttl)
```

```
{
    record->expireTime = time(NULL) + ttl;
}

//检查超时
int checkIDExpired(IDTransform* record)
{
    return record->expireTime > 0 && time(NULL) > record->expireTime;
}


void addToCache(char* url, char* ip) //利用LRU算法，使最近使用的域名排在最前面，提升效率
{
    int i, j;
    int place = -1;
    for (i = 0; i < cacheCount; i++)
    {
        if (strcmp(url, cache[i].domain) == 0) { place = i; break; }
    }
    if (place > -1)//如果在cache内
    {
        for (j = i - 1; j >= 0; j--)
        {
            cache[j + 1] = cache[j];
        }
    }
    else//不在
    {
        for (i = cacheCount - 2; i >= 0; i--)
        {
            cache[i + 1] = cache[i];
        }
        if (cacheCount < MAX_CACHE_SIZE) cacheCount++;
    }
    strcpy(cache[0].domain, url);
    strcpy(cache[0].IP, ip);
}

//把新的对应关系添加到DNS域名解析表中
void addToTable(char* url, char* ip)
{
    strcpy(DNSTable[DNSNum].domain, url);
    strcpy(DNSTable[DNSNum].IP, ip);
    DNSNum++;
}
//添加到文件中
void addToFile(char* url, char* ip)
```

```
{
    FILE* file;
    if ((file = fopen(filePath, "a")) == NULL)//文件不存在
    {
        printf("文件打开错误！");
    }
    fputs("\n", file);
    fputs(ip, file);
    fputs(" ", file);
    fputs(url, file);
    fclose(file);
    printf("已成功把新IP写入文件。\n");
}

//更新ID转换表中ID
unsigned short registerNewID(unsigned short ID, SOCKADDR_IN temp)
{
    int i = 0;
    //int flag = 0;
    for (i = 0; i != MAX_ID_TRANS_TABLE_SIZE; ++i)
    {
        if (checkIDExpired(&IDTransTable[i]) == 1 || IDTransTable[i].done ==
TRUE)//如果是需要更新的条目
        {
            IDTransTable[i].oldID = ID;
            IDTransTable[i].client = temp;
            IDTransTable[i].done = FLASE;
            setIDExpire(&IDTransTable[i], ID_EXPIRE_TIME);
            IDcount++;
            if (debugLevel >= D)
            {
                printf("ID%d注册成功。\n现有ID数目：%d\n", i + 1, IDcount);
            }
            //flag = 1;
            break;
        }
    }
    if (i == MAX_ID_TRANS_TABLE_SIZE)
        return 0;
    return (unsigned short)i + 1;
}

//组装DNS请求报文中的域名
void getUrl(char* buf, char* dest)
{
    int i = 0, j = 0, k = 0;
    int len = strlen(buf);
    while (i < len)
```

```
    {
        if (buf[i] > 0 && buf[i] <= 63)//如果是数字
        {
            for (j = buf[i], i++; j > 0; j--, i++, k++)
                dest[k] = buf[i];
        }
        if (buf[i] != 0)
        {
            dest[k] = '.';
            k++;
        }
    }
    dest[k] = '\0';
}

//输出整个包
void outputPacket(char* buf, int length)
{
    unsigned char unit;
    printf("Packet长度 = %d\n", length);
    printf("Package:\n");
    for (int i = 0; i < length; i++)
    {
        unit = (unsigned char)buf[i];
        printf("%02x ", unit);
    }
    printf("\n");
}

//从外部主机接收数据
void receiveFromExtern()
{
    int i;

    //得到buf
    char buf[MAX_BUF_SIZE], url[65];
    memset(buf, 0, MAX_BUF_SIZE);
    int length = -1;
    length = recvfrom(extern_sock, buf, sizeof(buf), 0, (struct
sockaddr*)&external, &length_client);

    if (length > -1)
    {
        if (debugLevel >= D)
        {
            printf("\n\n---- Recv : Extern [IP:%s]----\n",
inet_ntoa(external.sin_addr));
            time_t t = time(NULL);
```

```
            char temp[64];
            strftime(temp, sizeof(temp), "%Y/%m/%d %X %A", localtime(&t));
            printf("%s\n", temp);

            if (debugLevel == DD)
                outputPacket(buf, length);
        }

        //首先进行ID转换
        unsigned short* pID = (unsigned short*)malloc(sizeof(unsigned
short));
        memcpy(pID, buf, sizeof(unsigned short));
        int idIndex = (*pID) - 1;//得到ID编号index
        free(pID);
        memcpy(buf, &IDTransTable[idIndex].oldID, sizeof(unsigned short));//
buf的前16位赋给ID
        IDcount--;
        IDTransTable[idIndex].done = TRUE;
        client = IDTransTable[idIndex].client;
        //if (debugLevel >= D) printf("#ID Count : %d\n", IDcount);

        //开始分析Header
        int queryN = ntohs(*((unsigned short*)(buf + 4)));//question section的
问题个数
        int responseN = ntohs(*((unsigned short*)(buf + 6)));//answer section
的RR个数

        //开始分析Question
        char* p = buf + 12;
        char ip[16];
        int ip1, ip2, ip3, ip4;
        for (i = 0; i < queryN; i++)//得到quesion中存储的url
        {
            getUrl(p, url);
            //跳过操作
            while (*p > 0) p += (*p) + 1;
            p += 5;
        }
        if (responseN > 0 && debugLevel >= D) printf("Receive from extern
[Url : %s]\n", url);


        //开始分析Answer
        for (int i = 0; i < responseN; ++i)
        {
            if ((unsigned char)*p == 0xc0) //name field是指针，从wireshark读
出来的
```

```
                p += 2;
            else
            {
                while (*p > 0)
                    p += (*p) + 1;
                ++p;
            }
            unsigned short type = ntohs(*(unsigned short*)p);
            p += 2;
            unsigned short class = ntohs(*(unsigned short*)p);
            p += 2;
            unsigned short high = ntohs(*(unsigned short*)p);
            p += 2;
            unsigned short low = ntohs(*(unsigned short*)p);
            p += 2;
            int ttl = (((int)high) << 16) | low;
            int datalen = ntohs(*(unsigned short*)p);
            p += 2;

            if (debugLevel == DD) printf("Type -> %d,  Class -> %d,  TTL ->
%d\n", type, class, ttl);

            if (type == 1) //如果是IPv4类型的地址
            {
                ip1 = (unsigned char)*p++;
                ip2 = (unsigned char)*p++;
                ip3 = (unsigned char)*p++;
                ip4 = (unsigned char)*p++;
                //打印资源记录里的IP地址
                sprintf(ip, "%d.%d.%d.%d", ip1, ip2, ip3, ip4);
                if (debugLevel >= D) printf("IP address : %d.%d.%d.%d\n",
ip1, ip2, ip3, ip4);
                //加到cache中
                addToCache(url, ip);
                addToTable(url, ip);
                addToFile(url, ip);

                break;
            }
            else p += datalen; //不是typeA就忽略
        }

        //发送给客户端

        length = sendto(local_sock, buf, length, 0, (SOCKADDR*)&client,
sizeof(client));

    }
```

```
}

//从客户端接收
void receiveFromLocal()
{
    int i, j;
    int flag_txt = -1;
    int flag_cache = -1;
    char buf[MAX_BUF_SIZE], url[65];
    memset(buf, 0, MAX_BUF_SIZE);
    //从客户端接收到buf
    int length = -1;
    length = recvfrom(local_sock, buf, sizeof buf, 0, (struct sockaddr*)&client,
&length_client);
    if (length > 0)
    {
        char urlTmp[65]; //允许url最大长度为65
        memcpy(urlTmp, &(buf[DNS_HEAD_SIZE]), sizeof(urlTmp));

        getUrl(urlTmp, url); //得到url
        if (debugLevel >= D)
        {
            printf("\n\n---- Recv : Client [IP:%s]----\n",
inet_ntoa(client.sin_addr));
            time_t t = time(NULL);
            char temp[64];
            strftime(temp, sizeof(temp), "%Y/%m/%d %X %A", localtime(&t));
            printf("%s\n", temp);
            printf("Receive from client [Query : %s]\n", url);
        }

        for (i = 0; i < MAX_CACHE_SIZE; i++)//先在cache中查找
        {
            if (strcmp(url, cache[i].domain) == 0) flag_cache = i;
        }

        if (flag_cache == -1)//当在cache内没有找到时再到域名解析表中查找
        {
            for (i = 0; i < DNSNum; i++)
            {
                if (strcmp(url, DNSTable[i].domain) == 0)
                {
                    flag_txt = i;//与DNSTable中第i条相等
                    break;
                }
            }
        }
```

```
        char ip[16];
        //当文件和cache中都找不到时
        if (flag_txt == -1 && flag_cache == -1)
        {
            printf("[Url : %s] 没有在文件或cache中找到。\n", url);

            //存到ID转换表中
            unsigned short* pID = (unsigned short*)malloc(sizeof(unsigned
short));
            memcpy(pID, buf, sizeof(unsigned short));
            unsigned short nID = registerNewID(*pID, client);

            if (nID == 0)
            {
                if (debugLevel >= D)printf("ID转换表已满，添加失败！\n");
            }
            else
            {
                memcpy(buf, &nID, sizeof(unsigned short));//更改buf中的id

                //发送给外部DNS
                length = sendto(extern_sock, buf, length, 0, (struct
sockaddr*)&extern_name, sizeof(extern_name));
                if (debugLevel >= D) printf("发送给 external DNS server [Url :
%s]\n", url);
            }
            free(pID);
        }
        else //在文件或cache中找到了
        {
            //在文件中找到时
            if (flag_txt != -1 && flag_cache == -1)
            {
                strcpy(ip, DNSTable[flag_txt].IP);
                addToCache(url, ip);
                if (debugLevel >= D) printf("从本地文件中读取: [Url:%s -> IP:
%s]\n", url, ip);
            }

            //在cache中找到时
            if (flag_txt == -1 && flag_cache != -1)
            {
                strcpy(ip, cache[flag_cache].IP);
                addToCache(url, ip);
                if (debugLevel >= D) printf("从cache中读取： [Url:%s -> IP:
%s]\n", url, ip);
            }
```

```
                //开始构造响应报文
                char bufSend[MAX_BUF_SIZE];
                //添加请求报文部分
                memcpy(bufSend, buf, length);

                //如果是屏蔽网址，把rcode改为3直接发回
                if (strcmp(ip, "0.0.0.0") == 0)
                {
                    unsigned short a = htons(0x8183);
                    memcpy(&bufSend[2], &a, sizeof(unsigned short));
                    if (debugLevel > D)  printf("Warning: URL[%s] 为非法网址.\n",
url);
                    sendto(local_sock, bufSend, length, 0, (SOCKADDR*)&client,
sizeof(client));
                }

                //如果是非屏蔽地址

                //FLAG部分
                unsigned short a;
                a = htons(0x8180);//wireshark
                memcpy(&bufSend[2], &a, sizeof(unsigned short));

                //ANCOUNT answer section部分RR个数
                a = htons(0x0001);
                memcpy(&bufSend[6], &a, sizeof(unsigned short));

                //Answer部分
                unsigned short tmp;
                unsigned long tmpl;
                char answer[16];
                int len = 0;
                //Name
                tmp = htons(0xc00c); // 开头两个11代表指向域名的指针
                memcpy(answer, &tmp, sizeof(unsigned short));
                len += sizeof(unsigned short);
                //Type
                tmp = htons(0x0001); //typeA
                memcpy(answer + len, &tmp, sizeof(unsigned short));
                len += sizeof(unsigned short);
                //Class
                tmp = htons(0x0001);//固定为1，表示IN
                memcpy(answer + len, &tmp, sizeof(unsigned short));
                len += sizeof(unsigned short);
                //TTL
                tmpl = htonl(0x64);
```

```
                memcpy(answer + len, &tmpl, sizeof(unsigned long));
                len += sizeof(unsigned long);
                //RDLength
                tmp = htons(0x0004);
                memcpy(answer + len, &tmp, sizeof(unsigned short));
                len += sizeof(unsigned short);
                //RData
                tmpl = (unsigned long)inet_addr(ip);
                memcpy(answer + len, &tmpl, sizeof(unsigned long));
                len += sizeof(unsigned long);
                //组装
                len += length;
                memcpy(bufSend + length, answer, sizeof(answer));//把answer附
加到要发送的buf后

                //发送
                length = sendto(local_sock, bufSend, len, 0,
(SOCKADDR*)&client, sizeof(client));
                //输出调试信息
                if (length == -1 && debugLevel > D)
                    printf("Sendto:发送给客户失败。 错误代码 = %d\n",
WSAGetLastError());
                else if (length == 0)
                {
                    if (debugLevel > D) printf("Sendto: 连接失败。\n");
                }
                else if (debugLevel > D)
                    printf("Sendto: 成功把响应报文发送给客户。\n");
                char* p;
                p = bufSend + length - 4;
                if (debugLevel >= D)
                    printf("Send packet [Url:%s -> IP:%u.%u.%u.%u]\n", url,
(unsigned char)*p, (unsigned char)*(p + 1), (unsigned char)*(p + 2), (unsigned
char)*(p + 3));

                if (flag_cache != -1 && debugLevel >= D) outputCache();
            }
        }

}
```