

北京邮电大学课程设计报告

课程设计名称	计算机网络课程设计		学 院	计算机	指导教师	高占春
班 级	班内序号	学 号	学生姓名		成绩	
2020211310	32	2020211616	付容天			
2020211310	20	2020211596	王子晗			
2020211310	21	2020211598	向俊达			
课程设计内容	我们设计了一个 DNS 中继服务器，可以指定外部 DNS 服务器的 IP 地址以及特定的“域名-IP”对照表，利用哈希映射、双向链表、LRU 等算法实现缓存信息的初始化、管理与更新。当客户端查询指定域名的 IP 地址时，利用哈希映射高效检索缓存信息，有三种情况：当检索地址为 0.0.0.0 时实现拦截功能；当不为 0.0.0.0 时则向本地返回相应 IP 地址；当缓存未命中时向外部服务器中继 DNS 查询请求并接收响应报文，利用消息 ID 的转换拦截期望报文，并更新缓存文件以及向本地发送响应报文。我们也设计了报文构造与报文解析模块，用于构造响应报文和提取查询信息。我们也设计了缓存信息更新模块，利用 LRU 算法维护缓存信息，并利用 enter_time 和 TTL 及时删除超时记录。					
	其中，付容天主要完成了主函数编写、操作逻辑设计、消息 ID 转换、协议分析、报文构造、全部调试工作、实验报告撰写；王子晗主要完成了缓存文件读取、哈希链表编写、缓存文件更新、“域名-IP”查找功能的编写；向俊达主要完成了 socket 连接建立、与本地通信函数编写、与外部 DNS 服务器通信函数编写。					
学生课程设计报告(附页)						
课程设计成绩评定	评语：					
	成绩：					
	指导教师签名：					
	年 月 日					

注：评语要体现每个学生的工作情况，可以加页。



《计算机网络课程设计》课程报告

“DNS 中继服务器”的实现

付容天 学号 2020211616

王子晗 学号 2020211596

向俊达 学号 2020211598

班级 2020211310

计算机科学与技术系

计算机学院（国家示范性软件学院）

2022 年 7 月 1 日

目录

1	实验任务与小组分工	2
2	协议分析	3
2.1	Header 部分	3
2.2	Question 部分	4
2.3	资源记录部分	5
3	系统架构与实现细节	7
3.1	系统架构概述	7
3.2	数据结构定义与全局变量说明	8
3.3	消息通信功能	11
3.4	读取“域名-IP”记录文件	12
3.5	缓存维护与更新	12
3.6	消息 ID 的映射与转换	12
3.7	各函数实现细节	13
3.8	综合：各部分调用关系	15
4	功能展示	16
4.1	启动阶段	16
4.2	屏蔽功能	17
4.3	一般查询功能	18
4.4	中继功能	19
4.5	缓存更新功能	20
4.6	多客户端同时工作	21
5	调试过程中出现的问题与解决方法	23
6	总结	24

引言

DNS(Domain Name System), 域名系统, 是互联网的一项服务。它作为将域名和 IP 地址相互映射的一个分布式数据库, 能够使人更方便地访问互联网。DNS 使用 TCP 和 UDP 端口 53。当前, 对于每一级域名长度的限制是 63 个字符, 域名总长度则不能超过 253 个字符。(来自Wikipedia)

在本次实验中, 本小组设计并实现了一个简单、高效的基于哈希映射、双向链表以及 LRU 更新算法的 DNS 中继服务器。我们设计的 DNS 中继服务器可以在 53 端口监听本地发送的 DNS 查询请求, 检测缓存中是否存在, 并在必要的时候转发到远程服务器, 实现 DNS 代理功能。同时, 我们的 DNS 中继服务器也能够指定域名记录文件, 若命中了预先给定文件中的域名记录, 则直接构造出符合协议格式规定的帧并返回到本地, 通过这一手段, 我们可以实现对 DNS 污染的反制、去除广告、防御恶意网址等功能。

1 实验任务与小组分工

本次实验要求设计一个 DNS 服务器程序, 读入“IP 地址-域名”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, 有三种可能检索结果:

- 检索结果: ip 地址 0.0.0.0, 则向客户端返回“域名不存在”的报错消息(不良网站拦截功能)
- 检索结果: 普通 IP 地址, 则向客户端返回该地址(服务器功能)
- 表中未检到该域名, 则向因特网 DNS 服务器发出查询, 并将结果返给客户端(中继功能)

并且, 考虑到多个计算机上的客户端会同时查询, 需要进行消息 ID 的转换。

我们选择在 Windows 操作系统上进行我们的实验, 编程语言选择 C 语言。

我们小组分工介绍如下:

- **付容天**: 主函数编写、操作逻辑设计、消息 ID 转换、协议分析、报文构造、全部调试工作、实验报告撰写
- **王子晗**: 缓存文件读取、哈希链表编写、缓存文件更新、“域名-IP”查找功能的编写
- **向俊达**: socket 连接建立、与本地通信函数编写、与外部 DNS 服务器通信函数编写

2 协议分析

根据 RFC1035，DNS 定义了两类报文，一种为查询报文；另一种是对查询报文的响应，称为响应报文。无论是查询报文还是响应报文，都有 12 个字节的头和查询问题。总体来说，报文由下面的字段构成：

Header	
Question	the question for the name server
Answer	RRs answering the question
Authority	RRs pointing toward an authority
Additional	RRs holding additional information

图 1: DNS 报文构成

2.1 Header 部分

头部分 Header 的细节如下图所示：

											1	1	1	1	1	1
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	
+--+																

图 2: 头部 Header 的详细定义

Header 部分中各字段的含义如下：

- ID: 由客户程序设置并由服务器返回结果，客户程序通过它来确定响应与查询是否匹配，我们的 DNS 服务器中设计了消息 ID 的中转功能，可能会改变该字段
- QR: 0 表示查询报，1 表示响应报
- Opcode: 通常值为 0（标准查询），其他值为 1（反向查询）和 2（服务器状态请求）
- AA: 权威答案（Authoritative Answer）
- TC: 截断的（Truncated），当应答的总长度超 512 字节时，只返回前 512 个字节
- RD: 期望递归（Recursion Desired），查询报中设置，响应报中返回。告诉名字服务器处理递归查询。如果该位为 0，且被请求的名字服务器没有一个权威回答，就返回一个能解答该查询的其他名字服务器列表，这称为迭代查询
- RA: 递归可用（Recursion Available），如果名字服务器支持递归查询，则在响应中该比特置为 1
- Z: 保留字段，必须为零
- RCODE: 响应码（Response Coded），仅用于响应报。值为 0 表示没有差错，值为 3 表示名字差错。从权威名字服务器返回，表示在查询中指定域名不存在
- QDCOUNT: 问题字段数
- ANCOUNT: 回答记录字段数
- NSCOUNT: 权威记录字段数
- ARCOUNT: 附加记录字段数

2.2 Question 部分

Question 部分的细节如下图所示：

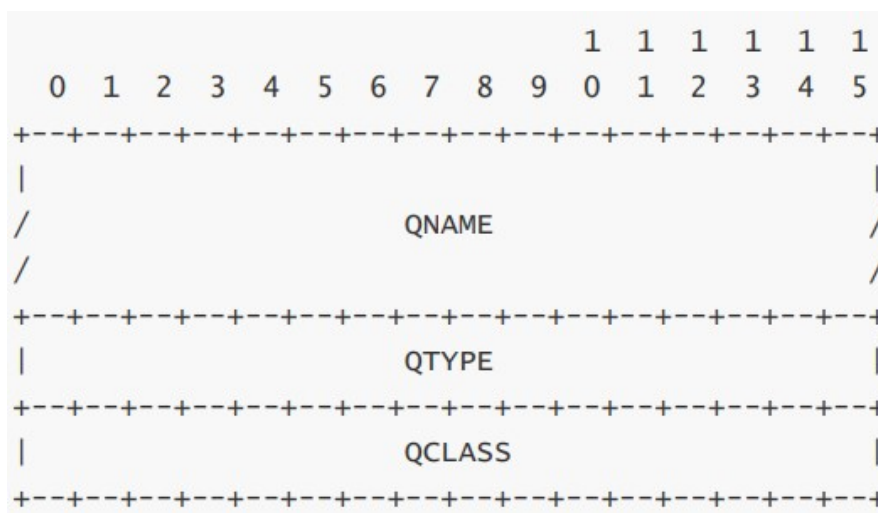


图 3: Question 部分的详细定义

Question 部分中各字段的含义如下：

- QNAME：域名，例如 www.bupt.edu.cn
- QTYPE：请求类型，例如 A(1)、MX(15)、CNAME(5)、PTR(12)
- QCLASS：请求类别，例如 IN(1)

2.3 资源记录部分

资源记录部分包括 Answer、Authority、Additional 三个部分，统称为 RR 部分，它们的格式是类似的，均为：

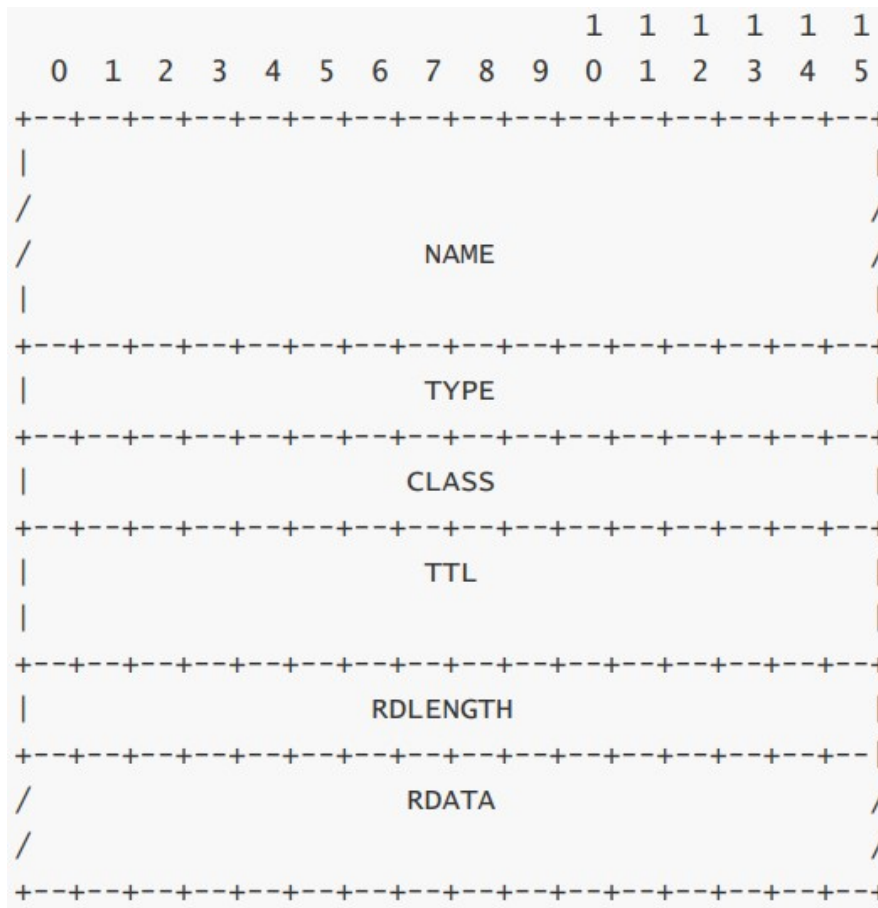


图 4: 资源记录部分的详细定义

RR 部分中各字段的含义如下:

- NAME: 名字
- TYPE: RR 的类型码, 例如 A(1)、MX(15)、CNAME(5)、PTR(12)
- CLASS: 通常为 IN(1), 指 Internet 数据
- TTL: 客户程序保留该资源记录的秒数, 稳定的资源记录生存时间值可以为 2 天, 它确定了客户端 DNS 的 cache 可以缓存该记录多长时间
- RDLENGTH: 资源数据长度, 说明资源数据的字节数, 对类型 1 (TYPE A 记录) 资源数据是 4 字节的 IP 地址
- RDATA: 资源数据

特别注意, 在实际应用中, NAME 字段具有一种“压缩”技术, 即: 使用偏移指针代替重复的字符串, 从而减少信息的长度。该指针用 8 比特表示, 并且最开始的两个比特为 1, 如下图所示:



图 5: 偏移指针定义

在我们的程序中，资源记录中的 NAME 字段为重复字段，故 NAME 字段用 0xc00c 表示，表示 NAME 字段的内容为报文第 12 字节开始的内容，即 Question 部分中的 QNAME 字段。

3 系统架构与实现细节

3.1 系统架构概述

总体上，我们的系统可以分成三个模块：缓存功能模块、通信消息模块和功能逻辑模块。

其中，**缓存功能模块**实现的功能为：

- 读入指定加载域名记录文件，并通过哈希映射按相应的数据结构进行存储
- 基于 LRU 算法更新缓存文件
- 给定域名查找其对应的 IP 地址，并按要求返回相应内容

通信消息模块实现的功能为：

- 从本地截获 DNS 查询请求
- 向外部 DNS 服务器发送 DNS 查询请求
- 从外部 DNS 服务器接收 DNS 查询结果
- 按照协议格式规定构造发送到本地的数据包
- 向本地发送 DNS 查询结果
- 消息 ID 的转换

功能逻辑模块实现的功能为：

- 查询结果的判断与功能分支的跳转
- 综合调用各种功能模块，实现系统功能

从功能点来看，我们实现的功能点包括：

- 对本地 DNS 请求报文的抓包分析，获取并翻译请求报文中的域名地址
- 程序中的包的处理过程为非阻塞单线程，实现简单，并可以快速处理报文
- 在 cache 缓存中查找该域名地址对应的 IP，若查到，则将此 IP 地址作为回答生成回答报文发给客户端，若未查到，则自动将此查询报文发送给外部 DNS 服务器
- 从外部 DNS 服务器接收回答，并对此报文进行分析，将第一个 Type A 类型的标准回答缓存到 cache 中，并设置其生存时间 TTL，更新本地文件，然后再将次回答报文转发给客户端。
- 通过对报文中标示符 ID 的转换，实现多客户端并行工作
- 预读在 cache 文件中的“域名-ip”参照表，并将这些对照表项的生存时间 TTL 设置为 LONG_MAX
- cache 与 ID 表中的文件存储覆盖遵循 LRU 算法，即最近最少使用算法
- 超时处理：当 TTL 超时之后，缓存会删除对应的资源记录

3.2 数据结构定义与全局变量说明

在我们的系统中，涉及到的数据结构为缓存的数据结构。我们综合采用了哈希映射和双向链表来存储缓存信息，具体定义如下：

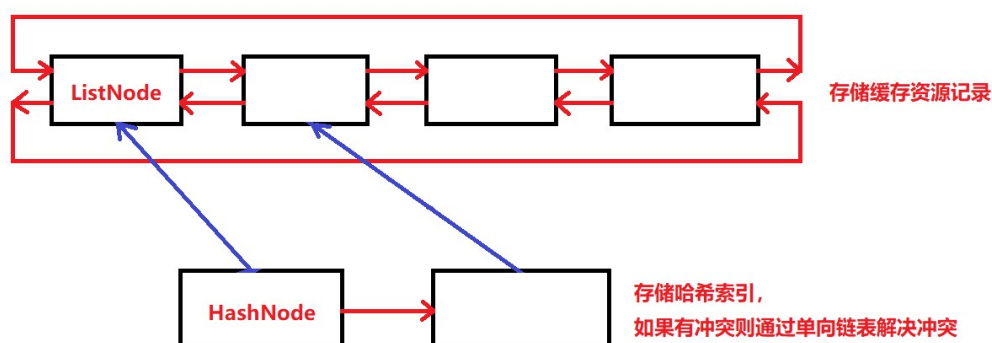


图 6: 数据结构逻辑图

我们将所有的资源记录以双向链表的形式存储在 ListNode 节点中，并且，一个哈希索引是一个单向列表，链表中的每个元素分别指向所有映射之后的哈希值一样的域名记录。在具体实现的时候，我们声明一系列哈希索引，将这些哈希索引存放在一个数组中，这样，任意域名经过哈希映射之后，找到数组中的一个元素（HashNode），并根据这个元素进一步找到对应的资源记录（ListNode），由此实现缓存管理、缓存查询、缓存更新等功能。

需要说明的是，我们将所有详细资源记录通过 ListNode 存储在一个双向链表中，是为了管理方便，具体来讲有以下目的：

- 在查询是否有资源记录超时的时候，我们仅需要线性扫描一遍双向链表、删除超时记录即可
- 在新增资源记录的时候，仅需要在双向链表的末尾新建一个 ListNode 节点即可
- 在利用 LRU 算法更新缓存的时候，我们仅需要改动 ListNode 的前后指针即可实现缓存节点的移动，保持其哈希映射的 HashNode 节点不受任何影响

其中，HashNode 和 ListNode 的详细数据结构定义如下：

```
// we use hash method to manage the cache after reading from the cache file
typedef struct HashNode{
    char name[256];
    char ipv4[64];
    char ipv6[64];
    ListNode *Ladr;
    struct HashNode *nextnode;
}HashNode;
```

图 7: HashNode 的数据结构定义

其中，HashNode 的数据结构定义包括以下内容：

- name：记录了域名，最长为 256 个 char 字符
- ipv4：记录了资源的 IPv4 地址，如果没有则为空
- ipv6：记录了资源的 IPv6 地址，如果没有则为空
- Ladr：这是指向 ListNode 的指针，ListNode 中存储了更为详细的资源记录
- nextnode：这是指向下一个 HashNode 的指针

```
// in our dnsrelay, we use doubly linked list to store the cache read from the cache file
typedef struct ListNode{
    int len;
    struct ListNode *front;
    struct ListNode *next;
    char name[256];
    char ipv4[64];
    char ipv6[64];
    int original; // 1 only when data is not changed after reading from the cache file
    long TTL;
    long enter_time; // when this ip is added into our cache
}ListNode;
```

图 8: ListNode 的数据结构定义

其中，ListNode 的数据结构定义包括以下内容：

- len: 表示由 ListNode 节点组成的双向链表的长度
- front: 指向双向链表的前一个节点
- next: 指向双向链表的后一个节点
- name: 记录了域名, 最长为 256 个 char 字符
- ipv4: 记录了资源的 IPv4 地址, 如果没有则为空
- ipv6: 记录了资源的 IPv6 地址, 如果没有则为空
- original: 如果为 1 则说明是从指定缓存文件 txt 中读取的资源记录, 这种资源记录由于承担了“屏蔽”功能, 因此不会被判断超时并删除
- TTL: 超时时间
- enter_time: 该资源记录进入缓存文件的时间, 结合 TTL 可以判断该记录是否超时, 进而判断是否应该删除该节点

这种基于哈希映射和双向链表的数据结构可以进行高效查找和快速更新, 性能表现非常好。

然后, 我们在此给出程序中所有宏定义和全部变量, 如下图所示:

```

/*****
 *      macro definition      *
 *****/
#define BUF_SIZE 1024        // maxsize of the buffer
#define DNS_PORT 53          // port between local and our dnsrelay
#define MAX_ID_SIZE 256      // maxsize of the ID conversion table
#define DNS_HEAD_SIZE 12     // the length of DNS header
#define MAX_SPACE_CACHE 256  // maxsize of the cache
#define MAX_TRANS_ID 256     // maxsize of the trans_id

```

图 9: 系统中的宏定义

```

/*****
 *      global variable      *
 *****/
struct sockaddr_in client, server;
struct sockaddr_in local_name, server_name;
WSADATA wsaData;
int length_client = sizeof(client);
int length_server = sizeof(server);
SOCKET local_sock, server_sock;
char DNS_Server_IP[32] = "10.3.9.44"; // ip of the external server
char DNS_Local_IP[32] = "127.0.0.1"; // ip of the local(default)
char Data_File_Name[100] = "dnsrelay.txt"; // cache file containing the website and its ip
int debug_level = 0; // 0,1,2, the higher, the more detailed our debug info is
int trans_id = 0; // trans_id ranges from 1 to MAX_ID_SIZE
int trans_ids[MAX_TRANS_ID] = {0}; // trans_ids[i]=1 when i is used as trans_id

```

图 10: 系统中的全局变量

3.3 消息通信功能

我们将消息通信功能封装为下图所示的四个函数：

```
// receive and send the frame
int recv_from_local(char *buffer_from_local, int buffer_max_len);           // receive the nslookup query from local
int send_to_local(char *buffer_to_local, int buffer_max_len, int buffer_lenth); // send the frame containing the ip back to local
int send_to_server(char *buffer_to_server, int buffer_max_len, int buffer_lenth); // send the nslookup to the external server
int recv_from_server(char *buffer_to_server, int buffer_max_len, int buffer_lenth); // receive the frame containing the ip
```

图 11: 有关消息通信的四个函数

以 `recv_from_local` 为例，实现细节如下图所示：

```
/*
 *   recv_from_local
 *
 */
int recv_from_local(char *buffer_from_local, int buffer_max_len){
    int lenth = -1;
    char buffer[buffer_max_len];
    memset(buffer, 0, buffer_max_len);
    lenth = recvfrom(local_sock, buffer, sizeof(buffer), 0, (struct sockaddr *)&client, &length_client);
    if(lenth > 0){
        printf("\nGet from local successfully!\n");
        for (int i=0; i<lenth; i++){
            buffer_from_local[i] = buffer[i];
        }
        buffer_from_local[lenth] = '\0';
    }
    return lenth;
}
```

图 12: `recv_from_local` 函数

并且，我们在系统中采用了 `socket` 编程接口进行 TCP 通信，监听 53 号端口，如下所示：

```
// build the socket for client and the socket for server
local_sock = socket(AF_INET, SOCK_DGRAM, 0);
server_sock = socket(AF_INET, SOCK_DGRAM, 0);
int lenth = -1;
int non_block = 1;
ioctlsocket(server_sock, FIONBIO, (u_long FAR*) & non_block); // non-blocking mode
ioctlsocket(local_sock, FIONBIO, (u_long FAR*) & non_block);

local_name.sin_family = AF_INET;
local_name.sin_addr.s_addr = inet_addr(DNS_Local_IP);
local_name.sin_port = htons(DNS_PORT);
server_name.sin_family = AF_INET;
server_name.sin_addr.s_addr = inet_addr(DNS_Server_IP);
server_name.sin_port = htons(DNS_PORT);
```

图 13: 绑定并监听 53 号端口

3.4 读取“域名-IP”记录文件

我们设计的系统支持读取不同的“域名-IP”记录文件，该功能通过如下图定义的 `init_cache` 函数实现：

```
// load the cache info from the cache file
void init_cache(char *txt_name, HashList *hash, ListNode *head);
```

图 14: 读取指定资源文件函数定义

该函数在实现细节上没有过多的难度，唯一需要注意的地方是：从 txt 文件中读取的字符串不能直接存入缓存中，原因是直接读取的内容是“字符串 0”，而存于缓存之中的应该为“(无符号) 数字 0”，这一点需要特别注意。另外，由于“(无符号) 数字 0”会被认为是“空”，所以在代码中部分地方引入“*”作为辅助符号出现，帮助避免歧义。

在读取教师提供的 `dnsrelay.txt` 文件时，将其中的资源记录的 TTL 均设为 `LONG_MAX`，使之长期有效。

3.5 缓存维护与更新

该功能的主要目的是防止缓存超出 TTL，主要思路是在每次查询结束之后都扫一遍资源记录，观察是否有记录超出了 TTL，删除这些超出 TTL 的记录。是否超出 TTL 是通过 `ListNode` 中的 `TTL` 和 `enter_time` 两个变量，结合系统当前时间进行判断。

缓存更新在每次查询结束进行，使用 LRU 算法，将最近最常使用的记录移到最后面，保持 TTL 不变，但改变 `enter_time`。在此功能中，我们也设计了输出功能，用户可以通过查看 `Output.txt` 文件的内容观察资源记录在查询过程中的变化。

3.6 消息 ID 的映射与转换

当缓存文件中没有查询到相应的记录时候，我们的 DNS 中继服务器需要向外部 DNS 服务器中转查询报文，由于本地可能有多个进程查询 IP 地址，所以我们设计了消息 ID 的转换。

转换的思路是利用整型变量 `trans_id` 和整型数组 `trans_ids[MAX_TRANS_ID]` 来标记某个 `trans_id` 是否被使用，并且在接收到外部报文的时候提取出其中的 Transaction ID，判断 `trans_ids[Transaction ID]` 是否为 1，如果为 1 则说明当前报文是符合要求的，应该被截获并转换 ID 并发送回本地；如果不为 1，则不符合要求。

```

// build a new ip query frame and send it to the external server
printf("\nIPv4 or IPv6 are not in our cache");
char new_frame_to_server[256] = {0};
trans_id++; trans_id %= MAX_TRANS_ID;
int trans_id_temp_2 = trans_id; // get a new transaction id for new_frame_to_server
trans_ids[trans_id_temp_2] = 1;
for(int i=0; i<buffer_lenth; i++){
    if(i == 0) new_frame_to_server[i] = trans_id_temp_2 / 256;
    else if(i == 1) new_frame_to_server[i] = trans_id_temp_2 % 256;
    else new_frame_to_server[i] = buffer_from_local[i];
}
int new_frame_to_server_len = buffer_lenth;
send_to_server(new_frame_to_server, 256, new_frame_to_server_len);

```

图 15: 分配消息 ID 并发送到外部

```

// receive the desired frame from the external server
char new_frame_from_server[256] = {0};
char new_frame_from_server_len = 0;
int trans_id_temp_3 = 0;
while(1){
    new_frame_from_server_len = rcv_from_server(new_frame_from_server, 256, new_frame_to_server_len);
    int temp_id = get_transaction_id(new_frame_from_server);
    temp_id %= MAX_TRANS_ID;
    if(trans_ids[temp_id] == 1){
        trans_ids[temp_id] = 0;
        trans_id_temp_3 = temp_id;
        break;
    }
    memset(new_frame_from_server, 0, 256);
    new_frame_from_server_len = 0;
}

```

图 16: 从外部接收报文并检测消息 ID 是否符合要求

```

// change the id of the frame received from the external server and send it to the local
for(int i=0; i<new_frame_from_server_len; i++){
    if(i == 0) new_frame_to_local[i] = trans_id_temp_3 / 256;
    else if(i == 1) new_frame_to_local[i] = trans_id_temp_3 % 256;
    else new_frame_to_local[i] = new_frame_from_server[i];
}

```

图 17: 报文消息 ID 转换

3.7 各函数实现细节

我们首先给出系统中所有函数的定义及必要的注释，如下图所示：

```

/*****
 *      function definition
 *****/
// output the program info
void display_program_info(ListNode *head);

// load the cache info from the cache file
void init_cache(char *txt_name, HashList *hash, ListNode *head);

// receive and send the frame
int recv_from_local(char *buffer_from_local, int buffer_max_len); // receive the nslookup query from local
int send_to_local(char *buffer_to_local, int buffer_max_len, int buffer_len); // send the frame containing the ip back to local
int send_to_server(char *buffer_to_server, int buffer_max_len, int buffer_len); // send the nslookup to the external server
int recv_from_server(char *buffer_to_server, int buffer_max_len, int buffer_len); // receive the frame containing the ip

// analyze the frame
int get_question_or_response_from_buffer(char *buffer); // this buffer is a question(return 0) or a response(return 1)
int get_website_name_from_buffer(char *buffer, char *website_name); // get website name from the buffer, ipv4->return 1, ipv6->return 2
int get_ip_from_buffer(char *buffer, char *ip); // get ip from a frame(from the external server), ipv4->return 1, ipv6->return 2
int get_transaction_id(char *buffer); // get Transaction ID and return it
int get_ip_and_TTL(char *buffer, int buffer_len, char *ipv4_ip, char *ipv6_ip, long *TTL); // return 1 when getting ipv4 and TTL, return 0 when ipv6

// some important functions
void look_for_ip(HashList *hash, ListNode *head, char *name, int *whether_find, int *RCODE, char *ipv4_ip, char *ipv6_ip, long *TTL_temp);
void build_frame_to_local(char *new_frame, int transaction_id, char *website_name, int RCODE, int ipv4_or_ipv6, char *ipv4_ip, char *ipv6_ip);
void update_cache(HashList *hash, ListNode *head, char *name, char *ipv4_ip, char *ipv6_ip, long TTL);

```

图 18: 系统中的函数定义

现在来简要介绍各函数：

- **display_program_info** 函数：输出程序信息，包括作者信息、程序介绍、DNS 服务器地址、缓存文件名、缓存信息等，支持不同调试等级输出不同内容
- **init_cache** 函数：从指定的缓存文件中读取所有的资源记录，计算哈希值，并提供哈希映射存入相应的链表位置
- **recv_from_local** 函数：从本地截取 DNS 查询报文，通过 socket 编程接口提供的 TCP 通信机制来实现，返回值是报文的长度
- **send_to_local** 函数：向本地发送 DNS 查询报文，通过 socket 编程接口提供的 TCP 通信机制来实现，返回值是报文的长度
- **send_to_server** 函数：向外部服务器发送 DNS 查询报文，通过 socket 编程接口提供的 TCP 通信机制来实现，返回值是报文的长度
- **recv_from_server** 函数：从外部服务器接收 DNS 查询报文，通过 socket 编程接口提供的 TCP 通信机制来实现，返回值是报文的长度
- **get_question_or_response_from_buffer** 函数：根据协议信息，从报文中解析出当前报文的查询报文还是响应报文，如果是查询报文则返回 0，如果是响应报文则返回 1
- **get_website_name_from_buffer** 函数：根据协议定义，从查询报文或响应报文中解析出当前报文的域名，如果想要查询 IPv4 地址则返回 1，如果想要查询 IPv6 地址则返回 2
- **get_ip_from_buffer** 函数：根据协议定义，从响应报文中解析出当前报文的 IP 结果，如果是 IPv4 地址则返回 1，如果是 IPv6 地址则返回 2
- **get_transaction_id** 函数：根据协议定义，从报文中解析出当前报文的消息 ID，返回值是消息 ID

- `get_ip_and_TTL` 函数：根据协议定义，从报文中解析出当前报文中的 IPv4 地址或者 IPv6 地址，并存入相应的位置，返回值为 1 说明 IPv4 查询成功，返回值为 0 说明 IPv6 查询成功
- `look_for_ip` 函数：给定欲查询的域名，通过计算域名的哈希值，结合哈希映射快速查找指定域名的 IP 地址是否在缓存中，如果存在则存入相应位置。该函数也同时从缓存文件中得到了 TTL 值。该函数需要合理设置 RCODE 值，用于程序判断下一步操作。该函数在查询过程中会主动判断资源记录是否超时，如果超时则标记为未命中，由程序进一步中转到外部 DNS 服务器进行查询
- `build_frame_to_local` 函数：当指定域名的 IP 地址存在于缓存文件中时，调用该函数根据协议定义构造出符合要求的 DNS 数据包并发回到本地
- `update_cache` 函数：该函数在每次查询结束之后自动调用，检查当前是否有资源记录超出了 TTL，判断方法是利用 `ListNode` 中的 `enter_time` 和 `TTL` 进行作差判断，如果超时则删除该节点

3.8 综合：各部分调用关系

我们设计的整个 DNS 中继服务器可以用下面的流程图来概括：

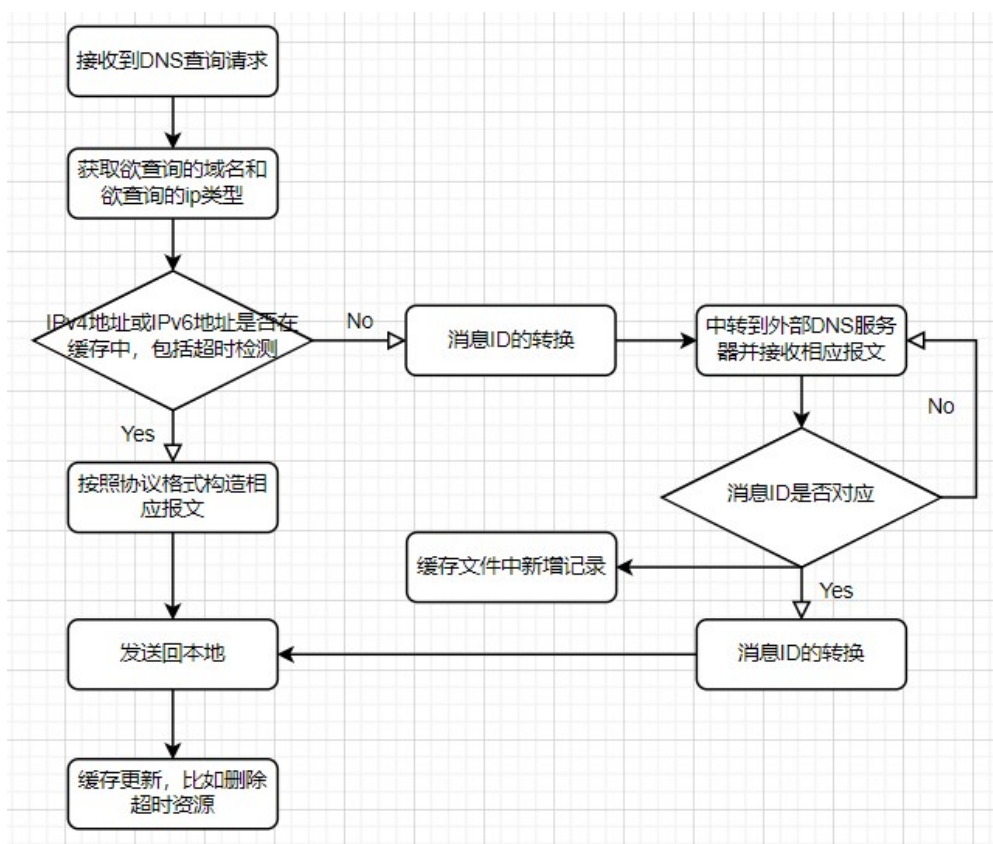
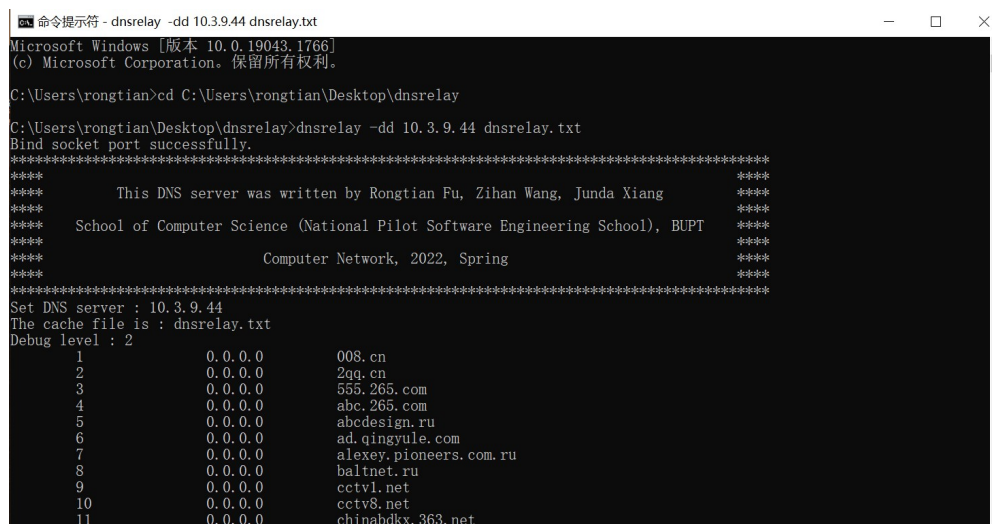


图 19: DNS 中继服务器运行逻辑

4 功能展示

4.1 启动阶段

我们首先连接校园网服务器并指定外部 DNS 服务器的 IP 地址为 10.3.9.44, 并指定缓存记录文件为 dnsrelay.txt, 即输入命令 `dnsrelay -dd 10.3.9.44 dnsrelay.txt` 启动 DNS 中继服务器, 可以看到下面的界面:



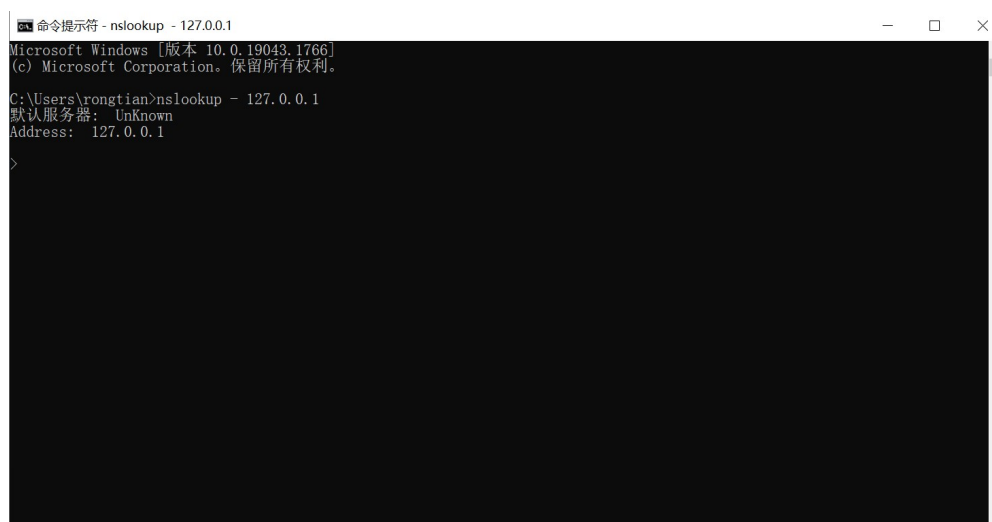
```
命令提示符 - dnsrelay -dd 10.3.9.44 dnsrelay.txt
Microsoft Windows [版本 10.0.19043.1766]
(c) Microsoft Corporation. 保留所有权利。

C:\Users\rongtian>cd C:\Users\rongtian\Desktop\dnsrelay
C:\Users\rongtian\Desktop\dnsrelay>dnsrelay -dd 10.3.9.44 dnsrelay.txt
Bind socket port successfully.
*****
****          This DNS server was written by Rongtian Fu, Zihan Wang, Junda Xiang          ****
****          School of Computer Science (National Pilot Software Engineering School), BUPT          ****
****          Computer Network, 2022, Spring          ****
****          *****          ****
Set DNS server : 10.3.9.44
The cache file is : dnsrelay.txt
Debug level : 2
 1          0.0.0.0          008.cn
 2          0.0.0.0          2qq.cn
 3          0.0.0.0          555.265.com
 4          0.0.0.0          abc.265.com
 5          0.0.0.0          abcdesign.ru
 6          0.0.0.0          ad.qingyule.com
 7          0.0.0.0          alexey.pioneers.com.ru
 8          0.0.0.0          baltnet.ru
 9          0.0.0.0          cctvl.net
10          0.0.0.0          cctv8.net
11          0.0.0.0          chinabdkx.363.net
```

图 20: DNS 中继服务器启动界面

可以看到, 外部 DNS 服务器被设为 10.3.9.44, 调试等级为 2, 资源记录文件为 dnsrelay.txt 并且成功读入并输出。

然后我们启动 cmd 命令行窗口, 输入 `nslookup - 127.0.0.1` 绑定到指定端口上, 可以看到:



```
命令提示符 - nslookup - 127.0.0.1
Microsoft Windows [版本 10.0.19043.1766]
(c) Microsoft Corporation. 保留所有权利。

C:\Users\rongtian>nslookup - 127.0.0.1
默认服务器: UnKnown
Address: 127.0.0.1
>
```

图 21: nslookup 查询界面

```
命令提示符 - dnsrelay -dd 10.3.9.44 dnsrelay.txt
192      0.0.0.0      www.zhengdian.com
193      0.0.0.0      www.zknew.com
194      0.0.0.0      xajh.15888.net
195      0.0.0.0      xyxy68.8u8.net
196      0.0.0.0      youlove.3322.net
197      0.0.0.0      zbszx.vicp.net
198      0.0.0.0      zelnet.ru
199      0.0.0.0      test0
200      11.111.11.111      test1
201      22.22.-34.-34      test2
202      -54.108.33.89      sina
203      61.-121.-75.-81      sohu
204      123.127.-122.10      bupt
205      0.0.0.0      ad4.sina.com.cn
206      0.0.0.0      www.163daohang.com.cn

Get from local successfully!
00 01 01 00 00 01 00 00 00 00 00 00 01 31 01 30 01 30 03 31 32 37 07 69 6e 2d 61 64 64 72
04 61 72 70 61 00 00 0c 00 01
You want to find: 1.0.0.127.in-addr.arpa

Finding IPv4...
IPv4 or IPv6 are not in our cache
Send to server successfully!

Get from server successfully!
Send to local successfully!
```

图 22: DNS 中继服务器界面

4.2 屏蔽功能

我们可以在查询界面输入 008.cn 进行查询，由于 008.cn 的 IP 地址在缓存文件 txt 中被设为 0.0.0.0，所以该域名应当屏蔽，得到查询结果如下：

```
命令提示符 - nslookup - 127.0.0.1
Microsoft Windows [版本 10.0.19043.1766]
(c) Microsoft Corporation. 保留所有权利。

C:\Users\rongtian>nslookup - 127.0.0.1
默认服务器: UnKnown
Address: 127.0.0.1

> 008.cn
服务器: UnKnown
Address: 127.0.0.1

*** UnKnown 找不到 008.cn: Non-existent domain
```

图 23: nslookup 查询界面

```
命令提示符 - dnsrelay -dd 10.3.9.44 dnsrelay.txt
00 03 81 83 00 01 00 01 00 00 00 00 03 30 30 38 02 63 6e 00 00 1c 00 01 c0 0c 00 1c 00 01
00 00 01 01 00 10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Send to local successfully!

Get from local successfully!

00 04 01 00 00 01 00 00 00 00 00 00 03 30 30 38 02 63 6e 00 00 01 00 01
You want to find: 008.cn

This domain should be blocked!

Finding IPv4...
IPv4 is in the cache!
00 04 81 83 00 01 00 01 00 00 00 00 03 30 30 38 02 63 6e 00 00 01 00 01 c0 0c 00 01 00 01
00 00 01 01 00 04 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Send to local successfully!

Get from local successfully!

00 05 01 00 00 01 00 00 00 00 00 00 03 30 30 38 02 63 6e 00 00 1c 00 01
You want to find: 008.cn

This domain should be blocked!

Finding IPv6...
IPv6 is in the cache!
00 05 81 83 00 01 00 01 00 00 00 00 03 30 30 38 02 63 6e 00 00 1c 00 01 c0 0c 00 1c 00 01
00 00 01 01 00 10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Send to local successfully!
```

图 24: DNS 中继服务器界面

4.3 一般查询功能

当缓存文件中有想要查询的资源记录、即查询命中的时候，DNS 中继服务器不会向外部服务器进行中转，而是直接按照协议格式构造出相应的数据包并发回本地，比如，当我们在查询界面输入 test1 的时候，可以看到：

```
命令提示符 - nslookup - 127.0.0.1
Microsoft Windows [版本 10.0.19043.1766]
(c) Microsoft Corporation. 保留所有权利。

C:\Users\rongtian>nslookup - 127.0.0.1
默认服务器: UnKnown
Address: 127.0.0.1

> 008.cn
服务器: UnKnown
Address: 127.0.0.1

*** UnKnown 找不到 008.cn: Non-existent domain

> test1
服务器: UnKnown
Address: 127.0.0.1

非权威应答:
名称: test1
Addresses: ::
          11.111.11.111

> -
```

图 25: nslookup 查询界面

```
命令提示符 - dnsrelay -dd 10.3.9.44 dnsrelay.txt
This domain should be blocked!
Finding IPv6...
IPv6 is in the cache!
00 05 81 83 00 01 00 01 00 00 00 00 03 30 30 38 02 63 6e 00 00 1c 00 01 c0 0c 00 1c 00 01
00 00 01 01 00 10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Send to local successfully!
Get from local successfully!
00 06 01 00 00 01 00 00 00 00 00 00 05 74 65 73 74 31 00 00 01 00 01
You want to find: test1
Finding IPv4...
IPv4 is in the cache!
00 06 81 80 00 01 00 01 00 00 00 00 05 74 65 73 74 31 00 00 01 00 01 c0 0c 00 01 00 01 00
00 01 01 00 04 0b 6f 0b 6f 00
Send to local successfully!
Get from local successfully!
00 07 01 00 00 01 00 00 00 00 00 00 05 74 65 73 74 31 00 00 1c 00 01
You want to find: test1
Finding IPv6...
IPv6 is in the cache!
00 07 81 80 00 01 00 01 00 00 00 00 05 74 65 73 74 31 00 00 1c 00 01 c0 0c 00 1c 00 01 00
00 01 01 00 10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Send to local successfully!
```

图 26: DNS 中继服务器界面

4.4 中继功能

当本地缓存未命中时，我们将查询请求中转到外部服务器进行查询，并接收外部 DNS 服务器发来的报文，如果消息 ID 符合要求，则一方面更新缓存、另一方面进行消息 ID 的转换并将报文发回本地。例如，当我们在查询界面输入 `www.bupt.edu.cn` 的时候，可以看到：

```
命令提示符 - nslookup - 127.0.0.1
Address: 127.0.0.1
> 008.cn
服务器: UnKnown
Address: 127.0.0.1

*** UnKnown 找不到 008.cn: Non-existent domain
> test1
服务器: UnKnown
Address: 127.0.0.1

非权威应答:
名称: test1
Addresses: ::
11.111.11.111

> www.bupt.edu.cn
服务器: UnKnown
Address: 127.0.0.1

DNS request timed out.
timeout was 2 seconds.
DNS request timed out.
timeout was 2 seconds.
非权威应答:
名称: www.bupt.edu.cn
Addresses: 2001:da8:215:4038::161
10.3.9.161
>
```

图 27: nslookup 查询界面

```
命令提示符 - dnsrelay -dd 10.3.9.44 dnsrelay.txt

Get from server successfully!
Send to local successfully!
Get from local successfully!
00 0a 01 00 00 01 00 00 00 00 00 00 03 77 77 77 04 62 75 70 74 03 65 64 75 02 63 6e 00 00
01 00 01
You want to find: www.bupt.edu.cn
Finding IPv4...
IPv4 is in the cache!
00 0a 81 80 00 01 00 01 00 00 00 00 03 77 77 77 04 62 75 70 74 03 65 64 75 02 63 6e 00 00
01 00 01 c0 0c 00 01 00 00 01 01 00 04 0a 03 09 a1 00
Send to local successfully!
Get from local successfully!
00 0b 01 00 00 01 00 00 00 00 00 00 03 77 77 77 04 62 75 70 74 03 65 64 75 02 63 6e 00 00
1c 00 01
You want to find: www.bupt.edu.cn
Finding IPv6...
IPv6 is in the cache!
00 0b 81 80 00 01 00 01 00 00 00 00 03 77 77 77 04 62 75 70 74 03 65 64 75 02 63 6e 00 00
1c 00 01 c0 0c 00 1c 00 01 00 00 01 01 00 10 20 01 0d a8 02 15 40 38 00 00 00 00 00 01
61 00
Send to local successfully!
```

图 28: DNS 中继服务器界面

4.5 缓存更新功能

缓存更新功能主要有两方面的作用：

- 存储外部 DNS 服务器返回的资源记录：当外部 DNS 服务器返回一个查询请求的时候，检查消息 ID 是否符合要求，如果是，一方面进行消息 ID 的转换并发回本地、另一方面按照协议格式对报文进行解析，提取出有用信息并存储在缓存（哈希链表）中
- 按照 LRU 算法更新资源文件：每当资源命中的时候，该条资源就被移动到最后面，当缓存空间满的时候，就删除最前面的资源记录（即最长时间未使用的资源记录）

显然，当我们再次查询 `www.bupt.edu.cn` 的时候，将不会向外部服务器进行中转，而是直接从缓存文件中读取相应资源记录，按照协议格式构造出数据包并发回本地。

现在，我们可以查看 `Output.txt` 文本文档，可以发现最近查询的几个域名被移动到了最末尾，实现了基于 LRU 的缓存记录更新：

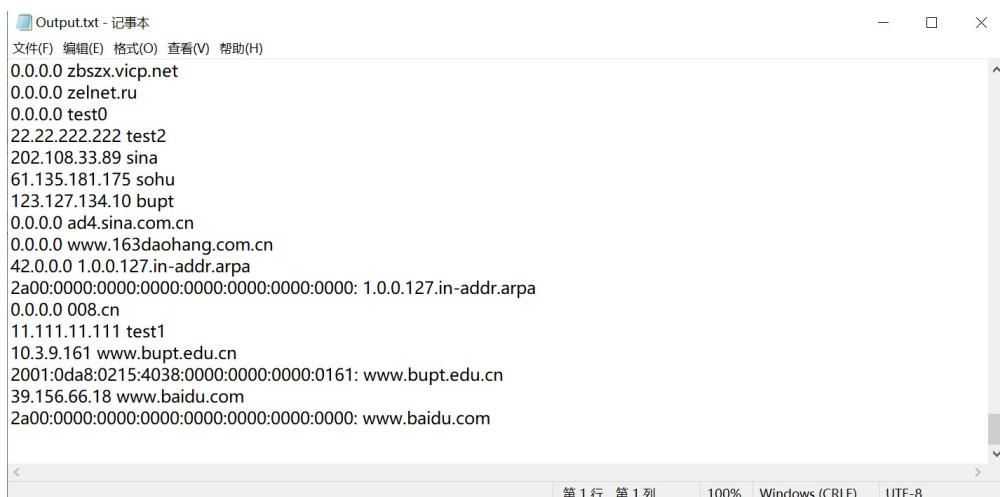


图 29: 基于 LRU 的缓存文件更新

4.6 多客户端同时工作

我们设计的系统支持多客户端同时查询域名 IP, 在查询界面 1 中查询 www.bupt.edu.cn 之后, 如果在查询界面 2 中再查询 www.bupt.edu.cn, DNS 中继服务器中将会命中缓存, 直接构造出协议定义的数据包并发挥到本地 (不会发到外部 DNS 服务器), 如下图所示:

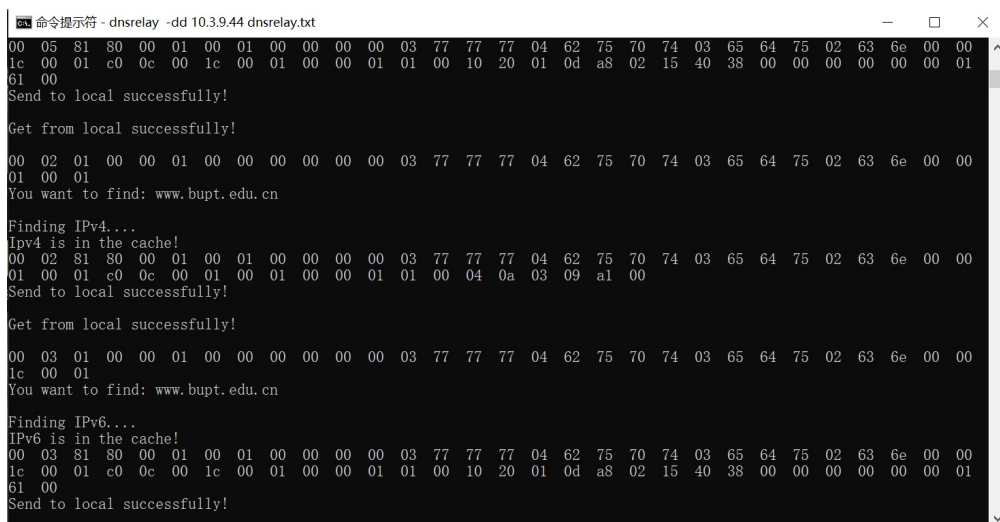


图 30: DNS 中继服务器界面

```
命令提示符 - nslookup - 127.0.0.1
Microsoft Windows [版本 10.0.19043.1766]
(c) Microsoft Corporation. 保留所有权利。

C:\Users\rongtian>nslookup - 127.0.0.1
默认服务器: UnKnown
Address: 127.0.0.1

> www.bupt.edu.cn
服务器: UnKnown
Address: 127.0.0.1

DNS request timed out.
  timeout was 2 seconds.
DNS request timed out.
  timeout was 2 seconds.
非权威应答:
名称: www.bupt.edu.cn
Addresses: 2001:da8:215:4038::161
          10.3.9.161

> -
```

图 31: nslookup 查询界面 1

```
命令提示符 - nslookup - 127.0.0.1
Microsoft Windows [版本 10.0.19043.1766]
(c) Microsoft Corporation. 保留所有权利。

C:\Users\rongtian>nslookup - 127.0.0.1
DNS request timed out.
  timeout was 2 seconds.
默认服务器: UnKnown
Address: 127.0.0.1

> www.bupt.edu.cn
服务器: UnKnown
Address: 127.0.0.1

非权威应答:
名称: www.bupt.edu.cn
Addresses: 2001:da8:215:4038::161
          10.3.9.161

> -
```

图 32: nslookup 查询界面 2

5 调试过程中出现的问题与解决方法

问题一：在利用 Wireshark 进行抓包分析的时候，发现报文的 NAME 字段总是出现“c00c”类型，查阅资料发现，这是一种“压缩”机制，具体内容在实验报告正文中已有详细说明。

问题二：一开始在查询的时候，发现查询域名的最后面总是出现“.tendaw-ifi.com”，导致在缓存中查询相应域名总是失败。经过反复排查相关信息，我选择手机开启 4G 网络并共享热点给笔记本电脑进行连接，解决了这个问题。

问题三：由于我们使用 char 类型的数组存储报文，所以一个自然的问题就是 char 类型变量本身具有“字符”的含义，故在输出的时候不能选择“%c”的转义字符，经过查阅相关资料得知应该选择“%hhx”的转义字符进行输出；并且，char 变量如果为 0 则会在判空操作中被判为空，这可能会引发矛盾（部分 IP 地址为 0.0.0.0），因此在程序的必要地方我采用了“符号*”作为“整数 0”的临时替代。

问题四：从外部 DNS 服务器返回的响应报文中常常会带有“CNAME”类型的资源记录，这相当于一个别名，我们在缓存文件中新增资源记录的时候并不存储别名 CNAME，因此这里需要进行转换。

问题五：在进行缓存初始化操作中，由于未对 ListNode 和 HashNode 结构体中的 name、ipv4 和 ipv6 以及 ipv4_temp 和 ipv6_temp 数组进行初始化和重置，而导致在对 dnsrelay.txt 文件的读写过程中出现如下图所示的乱码。为了解决这一问题，在每次使用上述数组之前都使用 memset 对其进行强制重置，以避免乱码的出现。

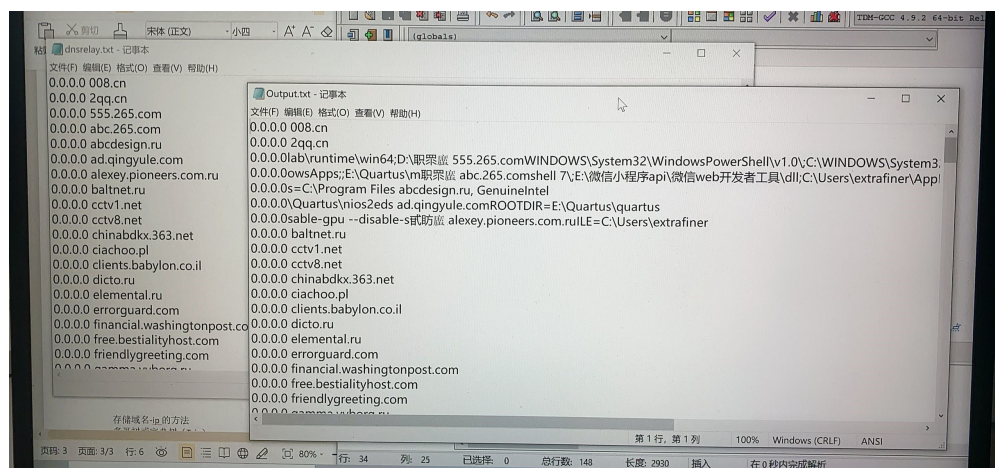


图 33: 未初始化时出现的乱码

问题六：有关缓存命中的问题：在设计之初为了提升缓存命中的效率，采用了 HashMap+ 双向链表的方式进行命中判断，但由于结构体的声明顺序，双向链表中的节点无法指向 HashMap 中的节点，在缓存空间不足或者 TTL 超时需要删除节点时无法删除 HashMap 中相应节点，所以在 HashMap 命中之后需要判断双向链表中是否由于缓存空间或者 TTL 超时而不存在相应节点而导致的命中误判。

6 总结

通过设计并实现 DNS 中继服务器，我们使用 UDP 通信机制并利用 Socket 编程接口 API 实现了 DNS 中继服务器的软件功能。在阅读 RFC1034 和 RFC1035 协议的过程中，较为熟练的掌握了相应协议的包结构和功能实现原理以及实际应用等相关知识，并且提升了对实际英文的协议的阅读与分析水平，以及应用实际协议的能力；在对通信协议设计的过程中，通过使用 Wireshark 抓包软件，熟练掌握了数据包的捕获、测试和分析等技能，并通过对包结构的分析对协议进行高效、系统性设计；在实现过程中，通过资料学习和沟通交流，掌握了 SOCKET 编程的相关知识要点，可以熟练使用 Socket API 完成主机和服务器之间的通信，实现所需功能，同时在实践探索和解决实践产出问题的过程中较好地培养了我们自主学习的能力，可以主动学习计算机网络领域相关的技术和知识要点，增强了终身学习的意识和能力。

同时通过计算机网络课程设计，利用编程实践增进了我们对计算机网络课程中习得的理论知识的理解，将概念应用于实际，提高计算机网络与通信系统的设计和开发能力和实际环境中的系统设计及开发能力，使我们更加系统化、精准化理解了通信理论。在具体程序开发的过程中，增加了对软件工程开发的理解，可以进行系统性设计，对软件工程的步骤有了深入实践并总结了相关经验。

通过线上会议、分工实践、讨论学习等方式，培养了我们的团队意识以及合作开发的能力，提升了沟通能力和协作能力；并在最后的实验报告撰写和答辩验收环节，增进了实验报告撰写能力和实践成功展示及语言表达能力，为我们未来的学习和工作打下坚实基础。