

# 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	蒋砚军老师
班 级	班内序号	学 号		学生姓名	成绩	
2020211305		2020211376		马天成		
2020211305		2020211383		王宸		
2020211305		2020211409		罗帅		
课 程 设 计 内 容	<p>一、课程设计教学目的：把计算机网络中所学的 DNS 的理论知识与实际编程联系起来，加强学生对协议的了解、思考能力和编程能力，培养团队合作和沟通能力。</p> <p>二、基本内容：设计一个 DNS 服务器程序，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，三种检索结果：          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，而不是返回 IP 地址为 0.0.0.0（不良网站拦截功能）。          检索结果为普通 IP 地址，则向客户返回这个地址（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）。</p> <p>三、实验方法：明确流程进行顺序、程序模块划分和函数接口，合作编写程序、测试及调试程序，完成课程设计；</p> <p>四、团队分工：          马天成：参与初期设计、编写本地映射表和 chche 的构造、检索、更新功能、改进程序的结构主体，进行程序调试。          罗帅：参与初期设计、编写报文的解析读取信息和响应报文的重构功能、完善程序的结构主体，参与程序调试。          王宸：参与初期设计、编写程序主体框架和 socket 通信配置初始化，完善程序的结构主体，参与程序调试。</p>					
学生课程设 计报告 (附页)						
课程设计 成绩评定	<p>评语：</p> <p>成绩：</p> <p style="text-align: right;">指导教师签名：</p> <p style="text-align: right;">年    月    日</p>					

注：评语要体现每个学生的工作情况，可以加页。

一：实验目的.....	3
二：实验环境.....	3
三：实验内容详解.....	3
3.1 功能设计.....	3
3.2 全局模块.....	4
3.3 header.....	5
3.4 package.....	5
3.4.1 报文格式说明.....	5
3.4.2 报文处理函数.....	6
3.5 cache.....	9
3.5.1 数据结构.....	9
3.5.2 Cache 函数.....	9
3.6 ID.....	11
3.6.1 数据结构.....	11
3.6.2 ID 函数.....	11
3.7 dnsTire.....	13
3.7.1 数据结构.....	13
3.7.2 dnsTire 函数.....	13
3.8 主函数模块.....	15
3.8.1 准备阶段.....	15
3.8.2 工作阶段.....	16
四：测试与调整.....	18
4.1 Cache 机制调整.....	18
4.2 Cache 容量参数调整.....	18
4.3 ID 表的机制调整.....	18
4.4 ID 表的容量调整.....	18
4.5 各种宏定义参数的设置.....	18
4.6 package 处理问题.....	19
4.7 外部发送和 Ipv6 引出的屏蔽机制升级.....	19
4.8 遇到的小问题.....	19
五：正式测试结果及分析.....	19
5.1 本机测试.....	19
5.2 外部主机测试.....	20
5.3 屏蔽外部 DNS 服务器测试核心功能.....	21
5.3.1 屏蔽功能.....	21
5.3.2 本地查询功能.....	22
5.3.3 外部查询功能&突发式查询测试.....	22
5.4 跨平台机制.....	23
5.5 字典接口.....	24
5.6 LRU 缓冲池.....	24
六：实验总结.....	24
6.1 实验经历.....	24
6.2 实验感想.....	24

# 一：实验目的

设计一个 DNS 服务器程序，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，三种检索结果：

- 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（不良网站拦截功能）；
- 检索结果为普通 IP 地址，则向客户返回这个地址（服务器功能）；
- 表中未检测到该域名，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）；

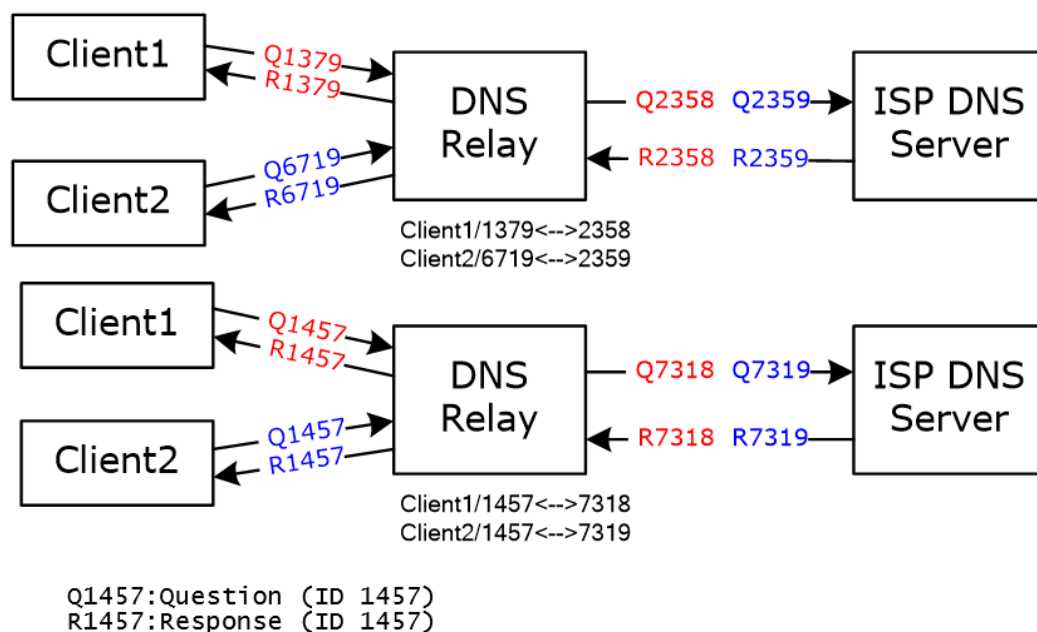
此外，虑多个计算机上的客户端会同时查询，需要进行消息 ID 的转换。

# 二：实验环境

1. 操作环境：windows
2. 编译软件：vs2020
3. 程序语言：c 语言

# 三：实验内容详解

## 3.1 功能设计



采用模块化设计。主要分为以下几个模块：

- DNS.c：主函数模块，进行 socket 的绑定；收发流程和各类模块功能的调用
- global：（分为.h&.c）全局模块，进行一些通用的宏定义；定义时间计数线程函数
- header.h：头部报文结构体。（参考 PPT）
- package：（分为.h&.c）包处理模块，有取出 Url，ip 以及构造报文等
- cache：（分为.h&.c）cache 模块，各类查找更新机制
- ID：（分为.h&.c）ID 转换表，核心转换机制
- dnsTire：（分为.h&.c）字典树，使用域名针对本地进行快速查询

主函数模块调用其他模块进行功能实现。

## 3.2 全局模块

```
7 extern int level; //调试等级
8
9
10 #define ASCII_SIZE 128
11 #define URL_MAX_SIZE 100
12 #define IP_MAX_SIZE 16
13 #define LEN 512
14
15
16 #define TIME_MOD 1000
17
18 // seconds count
19 extern int timeCircle;
20 // run time thread
21 void* timePass();
```

global.h

- 全局变量  
level：调试等级  
timeCircle：时间循环，一秒一条，0-999
- 宏定义  
ASCII\_SIZE：ascii 码的规模，在字典树作为子树数量使用  
URL\_MAX\_SIZE：Url（域名）的最大长度  
IP\_MAX\_SIZE：IP（IP 地址）的最大长度  
LEN：package 的最大长度  
TIME\_MODE：时间的轮回限制，如同因特网的 12000 使用
- 时间函数

```
5 // global variables for main
6 int level; //调试等级
7
8 // for cache
9 int timeCircle;
10
11 void* timePass()
12 {
13     timeCircle = 0;
14     while (1) {
15         Sleep(1000);
16         timeCircle = (timeCircle + 1) % TIME_MOD;
17         //printf("%d secs.\n", timeCircle);
18     }
19 }
```

global.c

timeCircle 计数：歇一秒跳一下。数值为 0-999

## 3.3 header

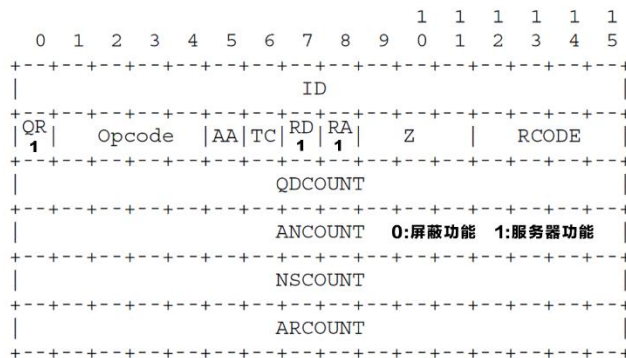
```
4 #define u16 unsigned short
5
6
7 typedef struct {
8     unsigned id : 16; /* query identification number */
9     unsigned rd : 1; /* recursion desired */
10    unsigned tc : 1; /* truncated message */
11    unsigned aa : 1; /* authoritative answer */
12    unsigned opcode : 4; /* purpose of message */
13    unsigned qr : 1; /* response flag */
14    unsigned rcode : 4; /* response code */
15    unsigned cd : 1; /* checking disabled by resolver */
16    unsigned ad : 1; /* authentic data from named */
17    unsigned z : 1; /* unused bits, must be ZERO */
18    unsigned ra : 1; /* recursion available */
19    u16 qdcount; /* number of question entries */
20    u16 ancount; /* number of answer entries */
21    u16 nscount; /* number of authority entries */
22    u16 arcount; /* number of resource entries */
23 } HEADER;
```

header.h

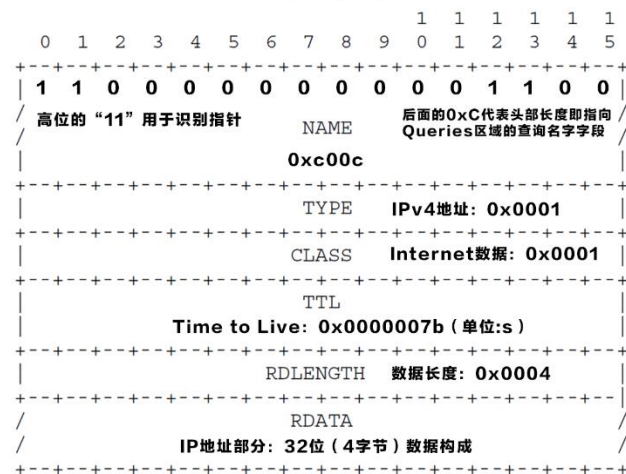
## 3.4 package

### 3.4.1 报文格式说明

#### 首部字段



#### 响应字段





### ● 标识

即 ID 字段，用于区分不同的报文。

这个 ID 对于客户端是唯一的，但对于中继器和服务器是不唯一的。

### ● 标志

对于不同的报文类型，对应不同的数值，如请求报文为 0001x;

(。。。。补充一下其他的段，尽量写全)

## 3.4.2 报文处理函数

```

5 int UrlInDns(char* DnsInfo, int DnsLength, char* UrlInDns);
6 int IpInDns(char* DnsInfo, int DnsLength, char* ipInDns);
7 int CreateResponse(char* DnsInfo, int DnsLength, char* FindIp, char* DnsResponse);
8 int CreateRequest(char* Domain, char* DnsInfo, unsigned short ID);
9
10 void NumToCharIp(char* NumStart, char* ipInDns);
11 int GetLengthOfNds(char* DnsInfo);

```

package.h

### 1. 在 dns 报文中取出 URL

```

4 //参数说明: DnsInfo 原始报文, DnsLength 报文长度, UrlInDns 请求的域名
5 //返回说明: IPV4返回1, 否则返回0
6 int UrlInDns(char* DnsInfo, int DnsLength, char* UrlInDns)
7 {
8     int i = 12, j, k = 0;
9     while (DnsInfo[i] != 0)
10     {
11         for (j = 0; j < DnsInfo[i]; j++)
12         {
13             UrlInDns[k] = DnsInfo[i + j + 1];
14             k++;
15         }
16         UrlInDns[k] = '.';
17         k++;
18         i = i + j + 1;
19     }
20     UrlInDns[k-1] = 0;
21     for (j = 0; UrlInDns[j + 3] != 0; j++)
22     {
23         if (UrlInDns[j] == 'D' && UrlInDns[j + 1] == 'H' && UrlInDns[j + 2] == 'C' && UrlInDns[j + 3] == 'P')
24         {
25             UrlInDns[j - 1] = 0;
26             UrlInDns[j + 4] = 0;
27         }
28     }
29     return ((DnsInfo[i + 2] - 1) && (DnsInfo[i + 1]));
30 }

```

**int UrlInDns(char \* DnsInfo , int DnsLength , char \*UrlInDns);**

1. 参数说明: DnsInfo :原始报文 , DnsLength : 原始报文长度 , UrlInDns:域名
2. 返回说明: ipv4 返回 1, 否则返回 0
3. 函数说明: 根据传入的参数, 将报文中的域名提取出来, 放在 UrlInDns 中

## 2. 在 dns 报文中取出 IP

```
33 //参数说明: DnsInfo为报文 , DnsLength为长度 , ipInDns 为存放IP的地方
34 //返回说明: 返回1.
35 int IpInDns(char *DnsInfo , int DnsLength , char *ipInDns)
36 {
37     int i = 12;
38     int j , offset=0;
39     int qnum=0 , anum=0;
40     int Qnum = DnsInfo[5];
41     int Anum = DnsInfo[7];
42     for(qnum;qnum<Qnum;qnum++)
43     {
44         while(DnsInfo[i]!=0)
45         {
46             for(j=0;j<DnsInfo[i];j++)
47             {
48             }
49             i = i + j + 1;
50         }
51     }
52     i = i + 4;
53     for(anum ; anum<Anum-1;anum++)
54     {
55         offset = offset + 12 + DnsInfo[i+offset+12];
56     }
57     i = i + offset + 12;
58     NumToCharIp(&DnsInfo[i+1] , ipInDns);
59 }
60 }
```

**int IpInDns(char \*DnsInfo , int DnsLength ,char \*ipInDns)**

1. 参数说明: DnsInfo:原始 DNS 报文 , DnsLength:报文长度 , ipInDns:存放报文中的 IP
2. 返回说明: 返回构造的 ip 的长度
3. 函数说明: 根据原始报文, 将该报文中的 ip 地址提取出来, 以此填充 cache

## 3. 创建回应报文

```
63 //参数说明: DnsInfo 原始报文 , DnsLength 报文长度 , FindIp 找到的IP , DnsResponse 回复报文
64 //返回说明: 返回回复报文的长度
65 int CreateResponse(char *DnsInfo , int DnsLength , char *FindIp , char *DnsResponse)
66 {
67     int LengthOfDns = GetLengthOfDns(DnsInfo);
68     memcpy(DnsResponse, DnsInfo, LengthOfDns);
69     unsigned short a = htons(0x8180);
70     if(DnsLength==LengthOfDns)
71     {
72         memcpy(&DnsResponse[2], &a, sizeof(unsigned short)); //更改标志位
73     }
74     else
75     {
76         DnsLength = LengthOfDns;
77         IpInDns(DnsInfo , DnsLength , FindIp);
78     }
79     if (strcmp(FindIp, (char *) "0.0.0.0") == 0)
80     {
81         a = htons(0x0000); //如果找到的IP为0.0.0.0, 则屏蔽, 回答数为0
82     }
83     else
84     {
85         a = htons(0x0001); //找到ip, 因为是在本地的找到, 回答数为1
86     }
87     memcpy(&DnsResponse[6], &a, sizeof(unsigned short));
88     int curLen = 0;
89     char answer[16];
90     unsigned short Name = htons(0xc00c);
91     memcpy(answer, &Name, sizeof(unsigned short));
92     curLen += sizeof(unsigned short);
93
94     unsigned short TypeA = htons(0x0001);
95     memcpy(answer + curLen, &TypeA, sizeof(unsigned short));
96     curLen += sizeof(unsigned short);
97
98     unsigned short ClassA = htons(0x0001);
99     memcpy(answer + curLen, &ClassA, sizeof(unsigned short));
100    curLen += sizeof(unsigned short);
101
102    unsigned long timeLive = htonl(0x7b);
103    memcpy(answer + curLen, &timeLive, sizeof(unsigned long));
104    curLen += sizeof(unsigned long);
105
106    unsigned short IPLen = htons(0x0004);
107    memcpy(answer + curLen, &IPLen, sizeof(unsigned short));
108    curLen += sizeof(unsigned short);
109
110    unsigned long IP = (unsigned long)inet_addr(FindIp);
111    memcpy(answer + curLen, &IP, sizeof(unsigned long));
112    curLen += sizeof(unsigned long);
113    curLen += DnsLength;
114    memcpy(DnsResponse + DnsLength, answer, sizeof(answer));
115
116    return curLen;
117 }
```

**int CreatResponse(char \*DnsInfo ,int DnsLength ,char \* FindIp , char \* DnsResponse)**

#### 1. 参数说明

DnsInfo :原始报文， DnsLength : 原始报文长度， FindIp:响应报文的IP , DnsResponse: 响应报文

#### 2. 返回说明: 返回构造的响应报文的长度

3. 函数说明: 根据传入的原始报文, 和找到的 IP, 构造出对应的响应报文, 并且在函数内部实现屏蔽功能

### 4. 创建请求报文

```
121 //返回说明: 返回长度
122 int CreateRequest(char * Domain, char * DnsInfo, unsigned short ID)
123 {
124     int i, j, k;
125     unsigned short id = ID;
126     memcpy(DnsInfo, &id, sizeof(unsigned short));
127     DnsInfo[2] = 1;
128     DnsInfo[3] = 0;
129     DnsInfo[4] = 0;
130     DnsInfo[5] = 1;
131     DnsInfo[6] = 0;
132     DnsInfo[7] = 0;
133     DnsInfo[8] = 0;
134     DnsInfo[9] = 0;
135     DnsInfo[10] = 0;
136     DnsInfo[11] = 0;
137     k = 12;
138     for(i=0; Domain[i] != 0; i++)
139     {
140         j = 1;
141         while(Domain[i] != '.' && Domain[i] != 0)
142         {
143             DnsInfo[k+j] = Domain[i];
144             i++;
145             j++;
146         }
147         DnsInfo[k] = j-1;
148         //printf("k:%d\n", k);
149         k = k+j;
150     }
151     //printf("k:%d\n", k);
152     DnsInfo[k] = 0;
153     DnsInfo[k+1] = 0;
154     DnsInfo[k+2] = 1;
155     DnsInfo[k+3] = 0;
156     DnsInfo[k+4] = 1;
157     return k+4+1;
158 }
```

**int CreateRequest(char \*Domain , char \*DnsInfo , unsigned short ID)**

1. 参数说明: Domain:构造请求报文的域名， ID: 该请求包的ID， DnsInfo: 存放构造的请求报文

2. 返回说明: 返回构造的请求报文的长度

3. 函数说明: 根据传入的域名和ID, 构造出询问该域名的报文, 并且ID为ID转化表中的ID

### 5. long 型数字转化为 IP

```
161 //将long形式的IP转化成字符串IP
162 void NumToCharIp(char * NumStart, char * ipInDns)
163 {
164     int i, k=0;
165     unsigned char xxhh;
166     for(i=0; i<4; i++)
167     {
168         xxhh = NumStart[i];
169         if(xxhh/100>0)
170         {
171             ipInDns[k] = xxhh/100+'0';
172             k++;
173         }
174         if(xxhh/10>0)
175         {
176             ipInDns[k] = (xxhh%100)/10+'0';
177             k++;
178         }
179         ipInDns[k] = xxhh%10+'0';
180         k++;
181         ipInDns[k] = '.';
182         k++;
183     }
184     ipInDns[k-1] = 0;
185 }
```

### 6. 获得 dns 报文长度

```
188 //得到长度
189 int GetLengthOfDns(char *DnsInfo)
190 {
191     int length = 12;
192     while (DnsInfo[length] != 0)
193     {
194         length++;
195     }
196     length = length + 5;
197     return length;
198 }
```



## 3.5 cache

### 3.5.1 数据结构

```
7  /* cache unit */
8  typedef struct record
9  {
10     // information for this record
11     char* url;
12     char* ip;
13     int time;
14
15     struct record* next;
16 } Record, * RecordPtr;
17
18 /* cache */
19 typedef struct cache
20 {
21     RecordPtr records;
22     int capacity;
23     int size;
24 } Cache, * CachePtr;
25
26 // initialize cache
27 CachePtr createCache();
28 // find IP in cache, if not exist, find in dnsTire
29 char* find_WithRefresh(char* url, CachePtr dnsCache);
30 void addRecord(CachePtr dnsCache, char* url, char* ip);
31 void printCache(CachePtr dnsCache);
32 void freeCache(CachePtr dnsCache);
```

- Record

每一个 Cache 记录，有 url，ip 和 time（进入的时间，timeCircle 赋值）三个属性。因为链表结构，所以有指向下一个的指针。

- Cache

使用链表数据结构，因为有 LRU 机制，需要经常性的替换和增删。链表新纪录存储在头部，越往下时间越靠近更新的机制。便于更新。Records 数组，有头部空链表。size 控制当前记录数量，capacity 控制容量。

- 宏定义

```
9  #define CAPACITY 20
10 #define REFRESHSIZE 10
11 #define OVERTIME 50
```

**CAPACITY**：容量限制

**REFRESHSIZE**：一般在添加时先判断在头部此规模中是否在 Cache 已有。有则更新至最顶端，无需添加。测试后选中此值。

**OVERTIME**：Cache 的更新时间。超过这个时间记录视为超时。需要在扫描过后进行删除。

### 3.5.2 Cache 函数

#### 1. 创建 cache

```
18 CachePtr createCache()
19 {
20     CachePtr dnsCache = (CachePtr)malloc(sizeof(Cache));
21     dnsCache->capacity = CAPACITY;
22     dnsCache->size = 0;
23
24     // create and initialize Record Head
25     dnsCache->records = (RecordPtr)malloc(sizeof(Record));
26     dnsCache->records->url = NULL;
27     dnsCache->records->ip = NULL;
28     dnsCache->records->next = NULL;
29
30     return dnsCache;
31 }
```

主要是建立 cache 里的 record 链表头部，并初始化容量和个数。返回值为申请的 cache 指针。

## 2. 根据 URL 查找对应的 IP 记录，并进行刷新

```
29 char* find_WithRefresh(char* url, CachePtr dnsCache)
30 {
31     char* ans = NULL;
32     /*
33      * Find IP for this domain in cache:
34      * if record exists, refresh its time and put it on the top
35      * Meanwhile, delete those which
36      * 1. time more than TIMEMOD/2
37      * 2. the one out of capacity in the tail
38      */
39     RecordPtr preNode = dnsCache->records;
40     RecordPtr curNode;
41     while (preNode->next != NULL) {
42         // find ip
43         if (strcmp(preNode->next->url, url) == 0) {
44             ans = preNode->next->ip;
45             if (level >= 1)
46                 printf("Cache find:\turl=%s\tip=%s\tcur=%d time=%d, put it on the top.\n", url, ans, timeCircle, preNode->next->time);
47
48             // refresh it, and put it on the top
49             RecordPtr curNode = preNode->next;
50             curNode->time = timeCircle;
51             preNode->next = curNode->next;
52             curNode->next = dnsCache->records->next;
53             dnsCache->records->next = curNode;
54
55             break;
56         }
57         preNode = preNode->next;
58     }
59     return ans;
60 }
61 }
```

- 返回结果

如果找到，则返回对应的指针。（该指针已指向一片空间）

如果没有找到，则返回 NULL。

- 算法设计

从头部链表开始找，找到则返回，找不到就不返回；

如果找到，那么刷新这条记录，放到 cache 最顶端。

## 3. 在 cache 中添加 URL 与 IP

```
63 void addRecord(CachePtr dnsCache, char* url, char* ip)
64 {
65     // find it on the top REFRESHSIZE, put it on the top
66     if (find_WithRefresh(url, dnsCache) == NULL)
67     {
68         // don't find it
69         RecordPtr curNode = (RecordPtr)malloc(sizeof(Record));
70         curNode->url = (char*)malloc(sizeof(char) * URLMAXSIZE);
71         memcpy(curNode->url, url, strlen(url) + 1);
72         curNode->ip = (char*)malloc(sizeof(char) * IPMAXSIZE);
73         memcpy(curNode->ip, ip, strlen(ip) + 1);
74         curNode->time = timeCircle;
75         curNode->next = dnsCache->records->next;
76         dnsCache->records->next = curNode;
77
78         dnsCache->size++;
79         printf("Cache added URL=%s, IP=%s\n", url, ip);
80
81         // delete time out of OVERTIME or out of capacity
82         RecordPtr preNode = dnsCache->records;
83         for (int count = 1; preNode->next != NULL; count++) {
84             if (((timeCircle - preNode->next->time + TIMEMOD) % TIMEMOD) > OVERTIME || count >= CAPACITY) break;
85             preNode = preNode->next;
86         }
87         while (preNode->next != NULL) {
88             dnsCache->size--;
89             if (level >= 1)
90                 printf("Cache delete:\turl=%s\tip=%s\tcur=%d time=%d\n", preNode->next->url, preNode->next->ip, timeCircle, preNode->next->time);
91             curNode = preNode->next;
92             preNode->next = curNode->next;
93             free(curNode->url);
94             free(curNode->ip);
95             free(curNode);
96         }
97     }
98
99     printCache(dnsCache);
100 }
```

- 对于每一次 cache 插入，加入先行寻找刷新机制，根据找到了记录就放至顶端。

- 每一次没有找到并且需要插入的时候。再插入一个记录后，把超出容量或者超出生存时间限制的记录全部删除。

## 3.6 ID

### 3.6.1 数据结构

```
8 #pragma comment(lib, "wsock32.lib")
9
10 // single ID record
11 typedef struct
12 {
13     char* url; // domain
14     int urlLength; // package length
15     unsigned short Question_id; // 客户端发给DNS服务器的ID
16     SOCKADDR_IN client_addr; // 请求者的客户端套接字
17     int time; // 进入的时间点
18     BOOL finished; // 标记该请求是否已经完成
19 } record_ID, *record_IDptr;
20
21 // ID table
22 typedef struct
23 {
24     record_IDptr records; // array of record_ID
25     int index;
26 } table_ID, *table_IDptr;
27
28
29 void initialize_table_ID(table_IDptr ID_table);
30 void findOutOfTime(table_IDptr ID_table, int my_socket, SOCKADDR_IN server_addr);
31 record_IDptr search_id(table_IDptr ID_table, unsigned short ID);
32 unsigned short save_id(table_IDptr ID_table, char* buf, int length, unsigned short ID, SOCKADDR_IN client_addr);
```

ID.h

记录结构体含有六个字段：

- url：域名指针
- urlLength：域名长度
- Question\_id：客户端发给 DNS 服务器的 ID
- SOCKADDR\_IN client\_addr：请求者的客户端套接字
- int time：进入的时间点（0-999）
- BOOL finished：状态标识，False 标识该记录正在使用不可用，True 标识可用

ID 表结构体含有两个字段：

- record\_IDptr records：记录数组指针
- int index：当前查找空间所在的起始下标

### 3.6.2 ID 函数

#### 1. 创建 ID 表

```
8 // initialize
9 void initialize_table_ID(table_IDptr ID_table)
10 {
11     ID_table->records = (record_IDptr)malloc(sizeof(record_ID) * ID_SIZE);
12     ID_table->index = 0;
13     for (int i = 0; i < ID_SIZE; i++)
14     {
15         ID_table->records[i].url = (char*)malloc(sizeof(char) * URLMAXSIZE);
16         ID_table->records[i].urlLength = 0;
17         ID_table->records[i].Question_id = 0;
18         ID_table->records[i].finished = TRUE;
19         ID_table->records[i].time = -1;
20         memset(&ID_table->records[i].client_addr, 0, sizeof(SOCKADDR_IN));
21     }
22 }
```

定量化数组空间申请，然后初始化 index。

#### 2. 超时重传机制

```

24 // resend out of time
25 void findOutOfTime(table_IDptr ID_table, int my_socket, SOCKADDR_IN server_addr)
26 {
27     for (int i = 0; i < ID_SIZE; i++) {
28         // out of time
29         if (ID_table->records[i].urlLength != 0 && ((timeCircle - ID_table->records[i].time + TIMEMOD) % TIMEMOD) > TTL/2) {
30             ID_table->records[i].time = timeCircle;
31             // resend
32             //sendto(my_socket, ID_table->records[i].buf, ID_table->records[i].length, 0, (struct sockaddr*)&server_addr, sizeof(server_addr));
33             //printf("resend: %s\n", ID_table->records[i].buf);
34         }
35     }
36 }

```

假如记录时间超过 TTL 一半，则触发重传。

根据 PPT 中的要求，该功能不予使用。

### 3. 回应客户端报文时需要查找原的 ID 记录

```

38 // 根据下标寻找
39 record_IDptr search_id(table_IDptr ID_table, unsigned short ID)
40 {
41     return &(ID_table->records[ID - 1]);
42 }

```

这里直接根据下标返回，速度极快。

### 4. 创建服务端报文时需要查找空间记录 ID

```

44 // 根据下标生成转发id
45 unsigned short save_id(table_IDptr ID_table, char* buf, int length, unsigned short ID, SOCKADDR_IN client_addr)
46 {
47     if (level >= 2)
48         printf("\nFind space in ID table.\nIn record: ");
49     unsigned short transID = 0;
50
51     int i = ID_table->index;
52     do {
53         // refresh state(overTime)
54         if (((timeCircle - ID_table->records[i].time + TIMEMOD) % TIMEMOD) > TTL)
55             ID_table->records[i].finished = TRUE;
56         if (level >= 2)
57             printf("%d-%d ", i + 1, ID_table->records[i].finished);
58
59         // has found space for record, conver buf with replacing ID
60         if (ID_table->records[i].finished == TRUE) {
61             transID = (unsigned short)(i + 1); // new_id, 对应存储在的空间下标+1
62             memcpy(buf, &transID, sizeof(unsigned short)); // 构造转发包的id, 对应下标
63
64             // 然后存入ID表
65             if (UrlInDns(buf, length, ID_table->records[i].url))
66                 ID_table->records[i].urlLength = strlen(ID_table->records[i].url);
67             else
68                 ID_table->records[i].urlLength = 0;
69             ID_table->records[i].Question_id = ID;
70             ID_table->records[i].client_addr = client_addr;
71             ID_table->records[i].finished = FALSE;
72             ID_table->records[i].time = timeCircle; // 0-999
73
74             i = (i + 1 + ID_SIZE) % ID_SIZE;
75             break;
76         }
77         i = (i + 1 + ID_SIZE) % ID_SIZE;
78     } while (i != ID_table->index);
79     ID_table->index = i; // refresh index
80     return transID;
81 }
82 }

```

#### ● 需求

- 客户端发来的包，Cache 和 Tire 字典树里未找到对应记录，转发给服务端进行 DNS 查询。
- 针对多用户环境，不同地址发送来的 ID 可能相同，需要有独特的 ID 冲突避免机制。
- 要保证性能：保证替换的效率；保证查找效率；保证空间利用率。
- **存储 ID&发送地址记录唯一标识**：针对多用户环境，不同地址发送来的 ID 可能相同，那么我们需要进行一个 ID 和发送地址的唯一标识来对应一个新的 ID。这里的新 ID 定义为找到的 ID 表空间的下标+1。
- **package 的 ID 替换机制**：ID 不进行大小端的转换，完全当作字节处理，使得给服务端的发送 ID 和接受 ID 字节一定相同，在本程序的识别也是相同的。目的是减小开销。
- **ID 表状态更新机制**：根据 TTL 的设定值进行查找状态更新。

#### ● 具体机制

ID 表状态更新机制：在每次查找到该记录空间时，更新状态，避免过多的刷新浪费性能。

ID 表寻找空间机制：初始化一个 **index**，每次查找后让 index 为已经找到下标数值+1 的新值，使得每次寻找都是可能性最大化。

## 3.7 dnsTire

### 3.7.1 数据结构

功能要求：根据本地文件读入信息，全字符匹配 URL 找出 IP。

思路：因为本地数据量巨大，而且是全字符匹配类型，所以选择字典树进行存储。

```
1  #ifndef DNSTIRE_H
2  #define DNSTIRE_H
3
4  // struct for domain to IP
5  typedef struct tire
6  {
7      // IP impormation
8      char* IPAddress;
9
10     // ptr array for the behind element which is ascii, thus contains 128 ptrs
11     struct tire** nextEleArray;
12 } Tire, * TirePtr;
13
14 TirePtr buildTire(FILE* dnsFile);
15 void freeTrieNode(TirePtr node);
16 char* findIP(char* domain, TirePtr dnsRoot);
17
18 #endif
```

dnsTire.h

- 通过 128（宏定义为 ASCII\_SIZE）叉树来存字典树。
- ascii 码直接转换为方向下标往下找，无需对应表。
- 通过牺牲空间来提高查询速度。域名多长找多少次。

### 3.7.2 dnsTire 函数

#### 1. 建立字典树

```
9  /* build Tire */
10 TirePtr buildTire(FILE* dnsFile)
11 {
12     /* create head */
13     TirePtr dnsRoot = (TirePtr)malloc(sizeof(Tire));
14     dnsRoot->IPAddress = NULL;
15     dnsRoot->nextEleArray = (TirePtr*)malloc(sizeof(TirePtr) * ASCII_SIZE);
16     memset(dnsRoot->nextEleArray, 0, sizeof(TirePtr) * ASCII_SIZE);
17
18     // build for each domain
19     TirePtr curNode;
20     char* url = (char*)malloc(sizeof(char) * URLMAXSIZE);
21     char* ip = (char*)malloc(sizeof(char) * IPMAXSIZE);
22     char* select = (char*)malloc(sizeof(char) * 2);
23     while (fscanf(dnsFile, "%s %s", ip, url) > 0)
24     {
25         printf("[ URL : %s, IP : %s ]\n", url, ip);
26
27         // for this domain, find from root and build paths which not exist
28         curNode = dnsRoot;
29         int len = (int)strlen(url);
30         for (int i = 0; i < len; i++)
31         {
32             // if path not exist, build it
33             if (curNode->nextEleArray[url[i]] == NULL) {
34                 curNode->nextEleArray[url[i]] = (TirePtr)malloc(sizeof(Tire));
35                 curNode->nextEleArray[url[i]]->IPAddress = NULL;
36                 curNode->nextEleArray[url[i]]->nextEleArray = (TirePtr*)malloc(sizeof(TirePtr) * ASCII_SIZE);
37                 memset(curNode->nextEleArray[url[i]]->nextEleArray, 0, sizeof(TirePtr) * ASCII_SIZE);
38             }
39             curNode = curNode->nextEleArray[url[i]];
40         }
41     }
```

```

42 // this node is for this domain
43 if (curNode->IPAddress == NULL) {
44     curNode->IPAddress = (char*)malloc(sizeof(char) * (strlen(ip) + 1));
45     memcpy(curNode->IPAddress, ip, strlen(ip) + 1);
46 }
47 // if IP exists and not equal to current one, select for replacing
48 else if (strcmp(curNode->IPAddress, ip) != 0) {
49     // has conflicts, select for replace
50     printf("\t%s: former IP: %s, current IP: %s.\n", url, curNode->IPAddress, ip);
51     printf("\t\tIP for this domain has been built, do you want to replace the former one? [y/n] ");
52     gets(select);
53     if (select[0] == 'y' || select[0] == 'Y') {
54         free(curNode->IPAddress);
55         curNode->IPAddress = (char*)malloc(sizeof(char) * (strlen(ip) + 1));
56         memcpy(curNode->IPAddress, ip, strlen(ip) + 1);
57         printf("\n\treplace successfully.\n");
58     }
59     else printf("\n\talready skip it.\n");
60 }
61 free(url);
62 free(ip);
63 free(select);
64 return dnsRoot;
65 }
66 }
67 }

```

## ● 函数流程

1. 通过打开的文件进行读取，每一条记录一个循环
2. 从根节点开始逐步读入域名，下标代表字母。如果没有子树那么申请一下
3. 在域名读完的节点中存入 IP 信息

## ● 遇到的问题

1. 一个域名可能对应多个 IP

解决方法：假如 IP 一样，直接跳过；不一样，根据用户输入查看是否替换。

2. 空间处理：没有的前缀节点就不申请空间

## ● 优点

1. 在数据量庞大的时候查找速度极快，查找复杂度只与查找数据长度有关。
2. 空间处理使用 malloc，减小栈的压力，提高性能。

## 2. 释放字典树

```

69 /* free Tire */
70 void freeTrieNode(TirePtr node)
71 {
72     if (node->IPAddress != NULL)
73         free(node->IPAddress);
74     for (int i = 0; i < ASCII_SIZE; i++) {
75         if (node->nextEleArray[i] != NULL)
76             freeTrieNode(node->nextEleArray[i]);
77     }
78     free(node->nextEleArray);
79     free(node);
80 }

```

释放空间，进行内存回收管理。

## 3. 根据 URL 查找对应 IP

```

82 /* domain -> IP */
83 char* findIP(char* domain, TirePtr dnsRoot)
84 {
85     if (dnsRoot == NULL)
86         return NULL;
87
88     TirePtr curNode = dnsRoot;
89     int len = (int)strlen(domain);
90     for (int i = 0; i < len; i++) {
91         // can't find path
92         if (curNode->nextEleArray[domain[i]] == NULL)
93             return NULL;
94         curNode = curNode->nextEleArray[domain[i]];
95     }
96     // has found it
97     return curNode->IPAddress;
98 }

```

非常简单的搜索机制。根据域名逐个搜索子树即可。

找到则返回 IP 指针；没找到则返回 NULL。

### 3.8.1 准备阶段

### 3.8.1 准备阶段

## ● 函数

- 全局变量

初始化各个变量。

初始化套接字。

工作时需要的局部变量。

## 3.8.2 工作阶段

对于每一次处理包，都是先以最少的步骤发送出去后再处理其他事情

鉴于代码过长，这里只讲伪代码和流程图：

```
while(1) {
    收到一个包，存入 buf
    if (服务端来的) {
        if (是屏蔽域名) 不予发送    // IP 对于 Ipv4/6 的屏蔽机制
    }
    else {
        在 ID 转换表里搜索对应的记录，转换 ID 后发送给对应客户端。// Ipv4/6 都有
        if (Ipv4) 将该记录加入 Cache
    }
}
// 从客户端来的
else {
    if (Ipv4) {
        cache 里找
        if (没找到) {
            dnsTire (字典树) 里找
            if(找到了) 添加该记录到 cache
        }
        if (cache 或者 tire 找到了)
            构造报文发给客户端    // 假如是屏蔽字段则会在构造报文时设置屏蔽报文
        else {
            使用 ID 表存放记录，构造报文发给服务端
            if(ID 表没找到空间存放) 扔掉
        }
    }
    // Ipv6
    else {
        if (是屏蔽域名)continue;
        使用 ID 表存放记录，构造报文发给服务端
    }
}
}
```

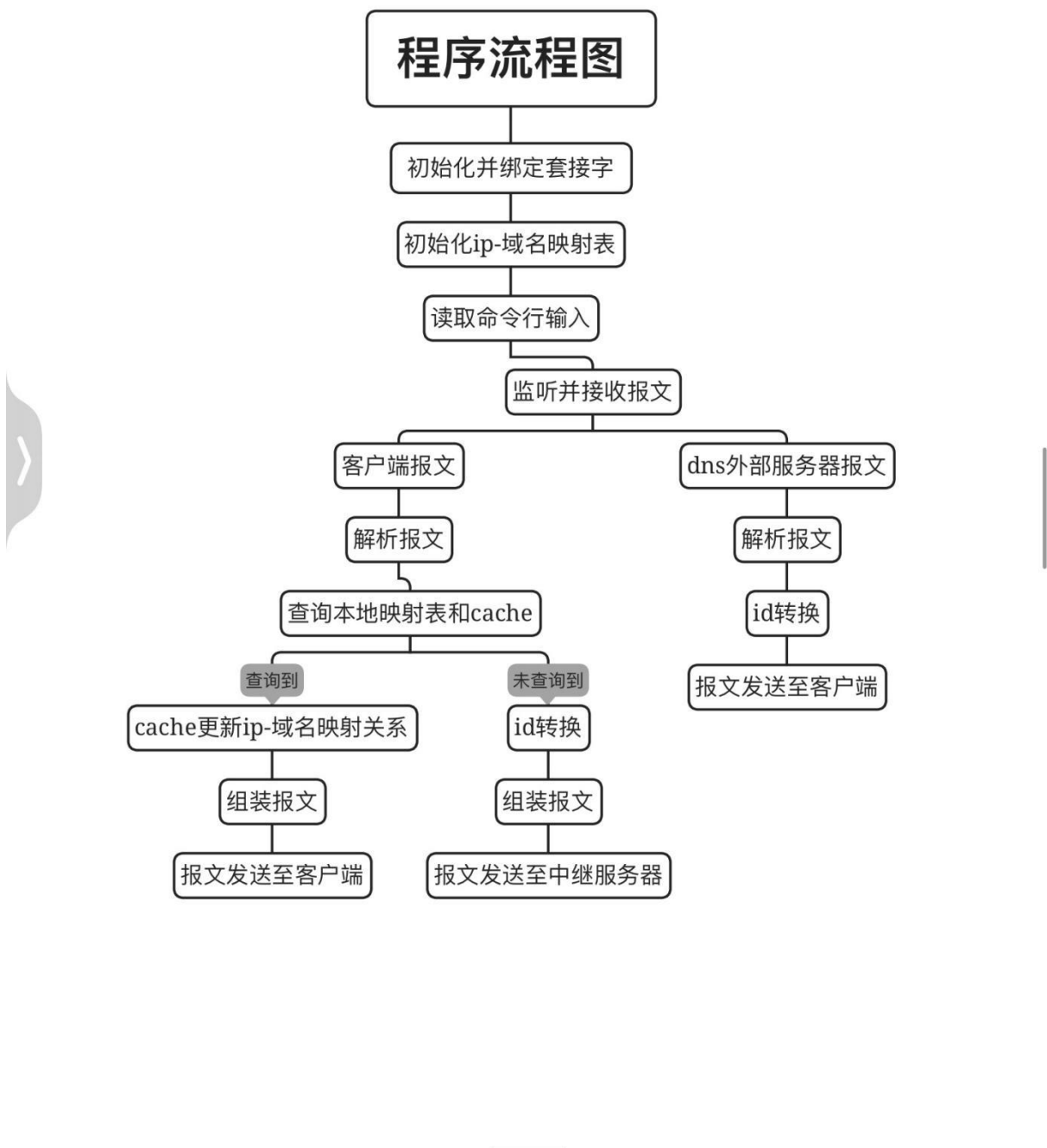
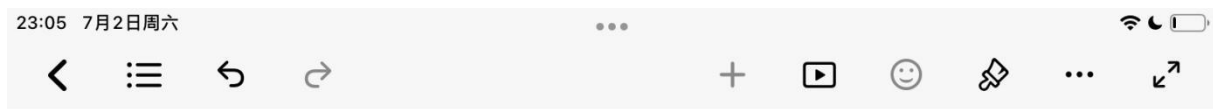
### ● 流程设计

- (1) 从客户端获取报文，解析报文得到域名，在本地 DNS 和 cache 中查找报文中域名所对应的 ip 地址，若找到，构造并返回响应报文；
- (2) 从客户端获取报文，解析报文得到域名，在本地 DNS 和 cache 中查找报文中域名所对应的 ip 地址，未找到，存储信息至 id 转化表，修改报文 id，转发请求报文至外部 DNS 服务器；
- (3) 从外部服务器获取报文，解析报文得到 id，读取 id 转化表获得转发 id，修改报文 id 转发至客户端；



## ● 流程图

23:05 7月2日周六



## 四：测试与调整

### 4.1 Cache 机制调整

1. cache 用普通单链表完全可以完成工作。所以并不采用所谓双向链表。
2. 在寻找时将找到的记录进行刷新，添加在记录最高处。
3. 在添加记录时，先在 cache 中寻找，找到则不用添加，查找时已经自动刷新位置。
4. 将删除超时和超出容量的步骤加在加入新记录时刻，避免过于频繁的删除，保证容量利用率。

### 4.2 Cache 容量参数调整

在经过狂开浏览器新页面的测试环境下，最终生存时间设置为 50，容量设置为 20 此时，在一般使用浏览时也 cache 保持在 95%的空间利用率。

### 4.3 ID 表的机制调整

1. 模拟空间特性：（不涉及增删）使用数组方式，固定容量。
2. 搜索机制优化：设定 index，固定下一次搜索的下标，使得开始搜索的位置一定是上次的下一个，使得搜索空间成功的概率最大化。
3. 使用空间下标+1 作为转换后 ID 的唯一标识，在取出 ID 记录时通过下标取出，无需匹配。
4. TTL 查询得知为 10，所以设置 TTL 为 10。在这个时间后不可能再次出现回应包

### 4.4 ID 表的容量调整

在经过狂开浏览器新页面的测试环境下，最终 TTL 设置为 10，容量设置为 72。基本第一次即可找到可使用的 ID 转换空间。

### 4.5 各种宏定义参数的设置

```
16 | #define TIMEMOD 1000
```

对于轮回时间的处理：设为 1000，基本不会出现 1000s 之内不上网的可能。

```
11 | #define URLMAXSIZE 100
```

```
12 | #define IPMAXSIZE 16
```

```
13 | #define LEN 512
```

查询得知上述几个宏定义如上。

## 4.6 package 处理问题

package 主要处理和报文相关的内容，如读取报文中的域名，ip 地址。以及对报文内部信息的修改，如构造响应报文，请求报文等。

## 4.7 外部发送和 Ipv6 引出的屏蔽机制升级

测试 [www.bilibili.com](http://www.bilibili.com) 得知：Ipv4 和 Ipv6 中的包可能都含有很多地址，所以必须两者都要实现屏蔽。此外，外部攻击也有可能造成域名无法屏蔽，所以接收 server 端时也要进行屏蔽处理。一开始仅针对 Ipv6 进行了外部屏蔽。没想到 Ipv4 外部攻击也需要进行屏蔽操作。

## 4.8 遇到的小问题

cache 一开始设置的过小（10），导致替换删除极为频繁。  
ID 表设置过小（50），经常丢包。  
socket 使用不当，不注重报文中存在\0 字符，使得一开始出错。

# 五：正式测试结果及分析

## 5.1 本机测试



```
命令提示符 - DNS - d
[ URL : h253.cn, IP : 192.168.0.253 ]
[ URL : h254.cn, IP : 192.168.0.254 ]
[ URL : h255.cn, IP : 192.168.0.255 ]
Tire build ended.
Start initialize socket
Socket build successfully!
Bind socket port successfully.

----- Received from Client.
IPv6 package has URL: 1.0.0.127.in-addr.arpa
IPv6 package. Should straightly send to server.
In ID_table, find 1th record to use. Transfer it as this ID.

----- Received from Server.
Find URL=1.0.0.127.in-addr.arpa in ID table, transfer ID and send to client.

----- Received from Client.
IPv4 package has URL: www.bilibili.com
1. Find in Cache
2. Find in Tire
Don't find in cache or tire for url=www.bilibili.com, should send to server.
In ID_table, find 2th record to use. Transfer it as this ID.

----- Received from Server.
Find URL=www.bilibili.com in ID table, transfer ID and send to client.
Cache added URL=www.bilibili.com, IP=112.13.92.195

-----Cache content:
1: www.bilibili.com
112.13.92.195 time=3

----- Received from Client.
IPv6 package has URL: www.bilibili.com
IPv6 package. Should straightly send to server.
In ID_table, find 3th record to use. Transfer it as this ID.

----- Received from Server.
Find URL=www.bilibili.com in ID table, transfer ID and send to client.

*** Unknown 找不到 www.bilibili.com: No response from server
C:\Users\Spike>nslookup www.bilibili.com 127.0.0.1
服务器: localhost
Address: 127.0.0.1

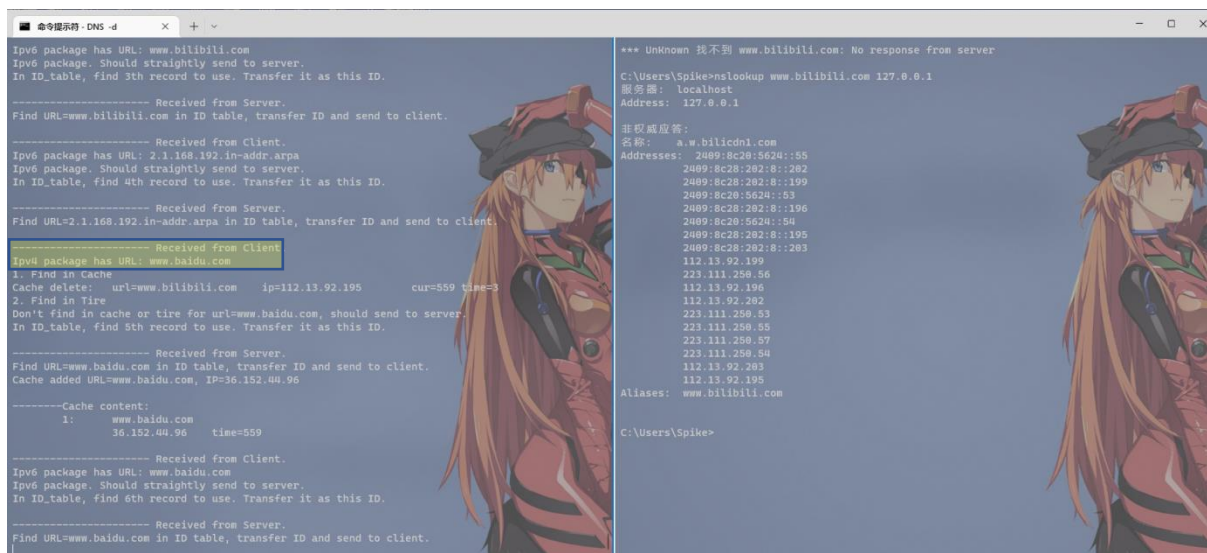
非权威应答:
名称: a.w.bilibidn1.com
Addresses: 2409:8c28:5624::55
2409:8c28:262:8::282
2409:8c28:262:8::199
2409:8c28:5624::53
2409:8c28:262:8::196
2409:8c28:5624::54
2409:8c28:262:8::195
2409:8c28:262:8::283
112.13.92.199
223.111.250.56
112.13.92.196
112.13.92.262
223.111.250.53
223.111.250.55
223.111.250.57
223.111.250.54
112.13.92.283
112.13.92.195
Aliases: www.bilibili.com

C:\Users\Spike>
```

1. 主机发送 dnsQ 包到本地 dns 服务器
2. 先收到 Ipv4 的报文，在检测本地没有查找结果后，使用 2 号 ID 空间直接转发给服务器申请
3. 收到 Ipv4 的 R 报文，将记录加入 cache，并 ID 转换转发给客户端

4. 先收到 Ipv6 的报文，取出 URL 检测本地没有屏蔽为 0.0.0.0 后，使用 1 号 ID 空间转发给服务器
5. 收到 Ipv6 的 R 报文，我们对 Ipv6 并没有报文分析，无法将其加入 cache，直接转发给客户端

## 5.2 外部主机测试



```
Ipv6 package has URL: www.bilibili.com
Ipv6 package. Should straightly send to server.
In ID_table, find 3th record to use. Transfer it as this ID.

----- Received from Server.
Find URL=www.bilibili.com in ID table, transfer ID and send to client.

----- Received from Client.
Ipv6 package has URL: 2.1.168.192.in-addr.arpa
Ipv6 package. Should straightly send to server.
In ID_table, find 4th record to use. Transfer it as this ID.

----- Received from Server.
Find URL=2.1.168.192.in-addr.arpa in ID table, transfer ID and send to client.

----- Received from Client.
Ipv6 package has URL: www.baidu.com
1. Find in Cache
Cache delete: url=www.bilibili.com ip=112.13.92.195 cur=559 time=3
2. Find in Tire
Don't find in cache or tire for url=www.baidu.com, should send to server.
In ID_table, find 5th record to use. Transfer it as this ID.

----- Received from Server.
Find URL=www.baidu.com in ID table, transfer ID and send to client.
Cache added URL=www.baidu.com, IP=36.152.44.96

----- Cache content:
1: www.baidu.com
36.152.44.96 time=559

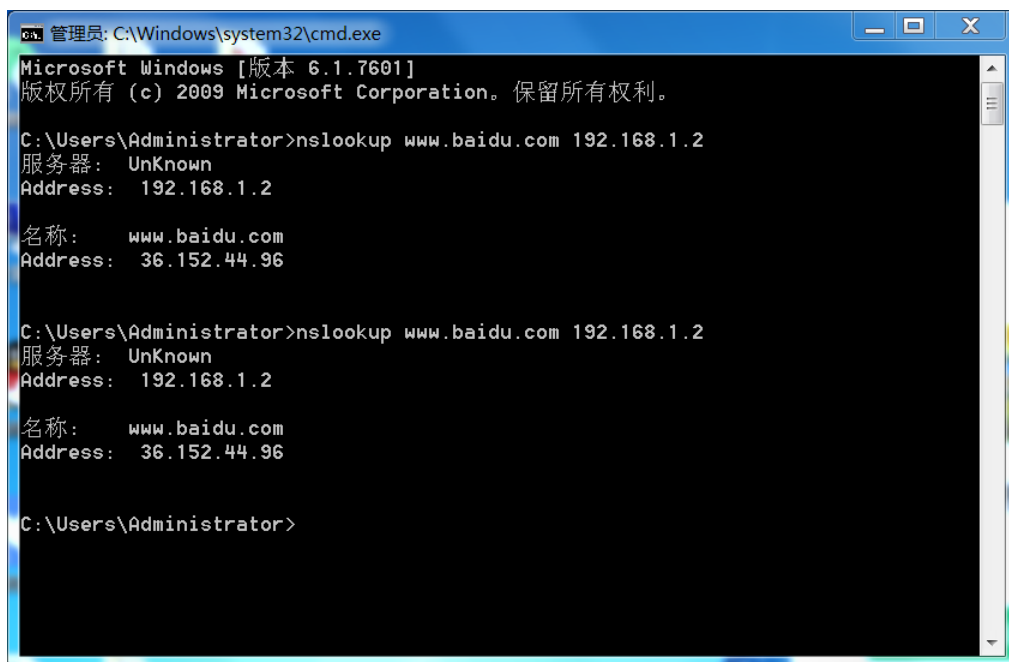
----- Received from Client.
Ipv6 package has URL: www.baidu.com
Ipv6 package. Should straightly send to server.
In ID_table, find 6th record to use. Transfer it as this ID.

----- Received from Server.
Find URL=www.baidu.com in ID table, transfer ID and send to client.
```

```
*** Unknown 找不到 www.bilibili.com: No response from server
C:\Users\Spike>nslookup www.bilibili.com 127.0.0.1
服务器: localhost
Address: 127.0.0.1

非权威应答:
名称: a.w.bilibidn1.com
Addresses: 2409:8c28:5624::55
2409:8c28:202:8::202
2409:8c28:202:8::199
2409:8c28:5624::53
2409:8c28:202:8::196
2409:8c28:5624::54
2409:8c28:202:8::195
2409:8c28:202:8::203
112.13.92.199
223.111.258.56
112.13.92.196
112.13.92.202
223.111.258.53
223.111.258.55
223.111.258.57
223.111.258.54
112.13.92.203
112.13.92.195
Aliases: www.bilibili.com

C:\Users\Spike>
```



```
管理员: C:\Windows\system32\cmd.exe
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation. 保留所有权利。

C:\Users\Administrator>nslookup www.baidu.com 192.168.1.2
服务器: UnKnown
Address: 192.168.1.2

名称: www.baidu.com
Address: 36.152.44.96

C:\Users\Administrator>nslookup www.baidu.com 192.168.1.2
服务器: UnKnown
Address: 192.168.1.2

名称: www.baidu.com
Address: 36.152.44.96

C:\Users\Administrator>
```

win7 系统

局域网内访问（前一个是上一次测试，不用管）  
这里可以看到我们成功的转发了 package。

## 5.3 屏蔽外部 DNS 服务器测试核心功能

首先将本机电脑的 DNS 服务器改为 127.0.0.1

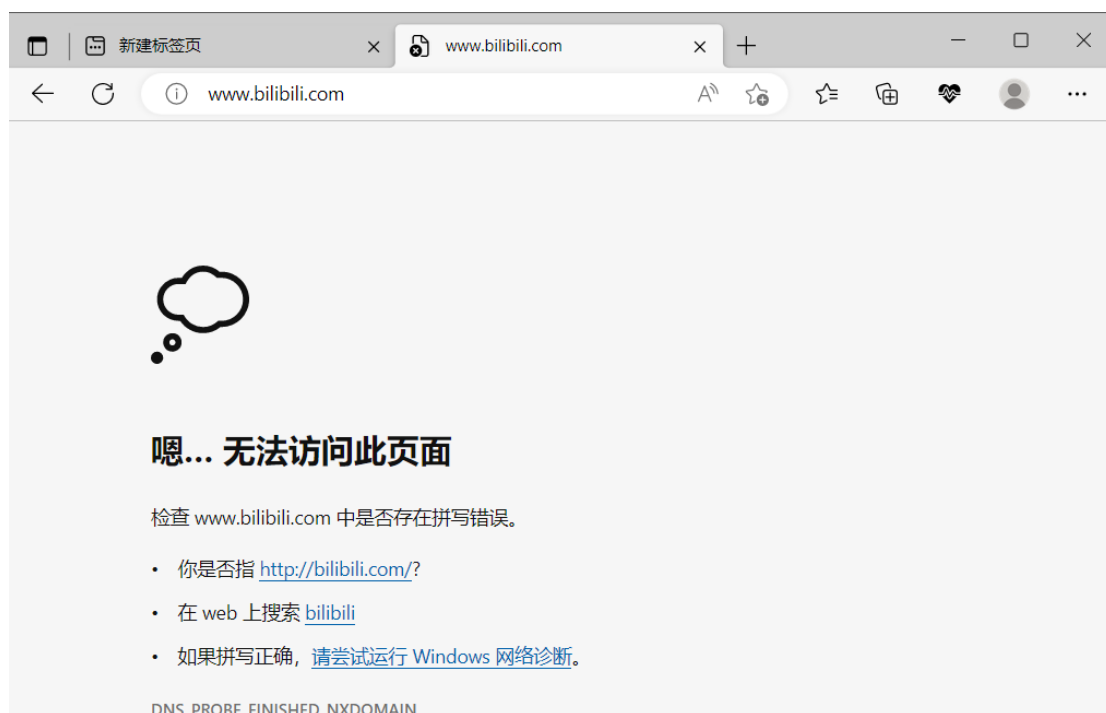


### 5.3.1 屏蔽功能

首先将 www.bilibili.com 加入屏蔽：（在文件里添加记录）

0.0.0.0 www.bilibili.com

然后打开浏览器，输入网址 www.bilibilib.com，可以看到无法刷新出网页



使用 nslookup 工具，可以看到如下，顺利完成屏蔽功能

```

Microsoft Windows [版本 10.0.19044.1766]
(c) Microsoft Corporation。保留所有权利。

C:\Users\丁奎标\Desktop\DNS1>nslookup
默认服务器: UnKnown
Address: 127.0.0.1

> www.bilibili.com
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.
*** 没有 www.bilibili.com 可以使用的 internal type for both IPv4 and IPv6 Addresses (A+AAAA) 记录

```

### ● 解读

bilibili 网站会发送 dns 的 Ipv6 和 Ipv4 包。我们都会进行屏蔽。

在第一个本机测试中可知，Ipv4 和 Ipv6 中的包都含有很多地址，所以必须两者都要屏蔽。

## 5.3.2 本地查询功能

在本地文件中加入该条记录(其中 IP 地址为百度 IP 地址):

**183.232.231.172 www.xxhh.com**

在 nslookup 中进行查询如下:

```

C:\Users\丁奎标\Desktop\DNS1>nslookup
默认服务器: UnKnown
Address: 127.0.0.1

> www.xxhh.com
服务器: UnKnown
Address: 127.0.0.1

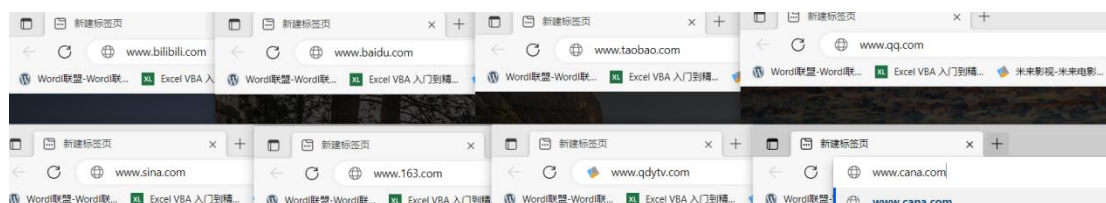
非权威应答:
名称: www.xxhh.com.DHCP HOST
Address: 183.232.231.172

```

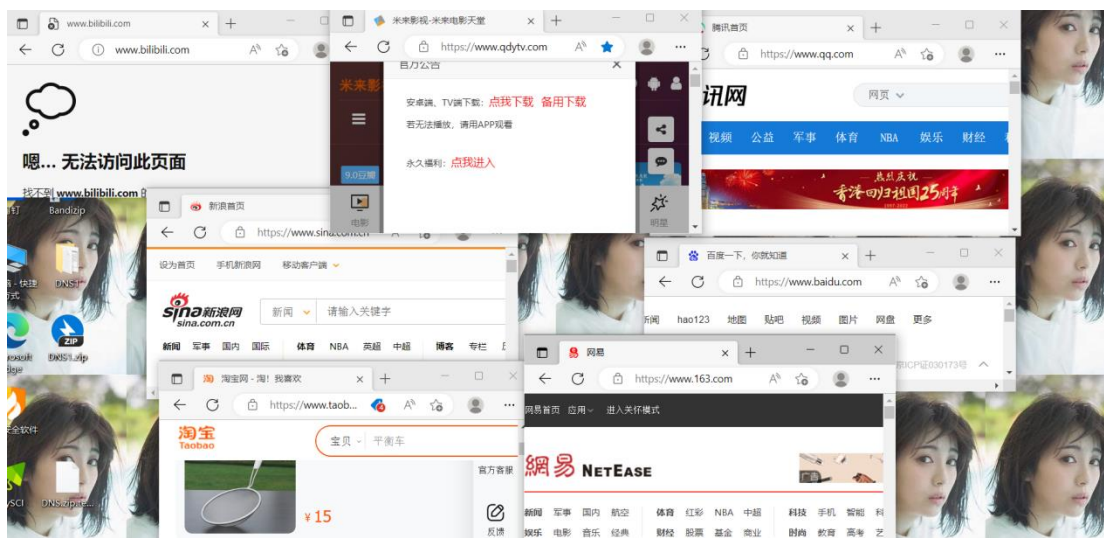
顺利返回对应 IP 地址。

## 5.3.3 外部查询功能&突发式查询测试

首先在浏览器中输入如下网址:



然后很快的依次点击回车，如下:



各个界面几乎同时刷新。

## 5.4 跨平台机制

```

5  #ifdef _WIN32
6      #include<winsock2.h>
7      #pragma comment(lib,"wssock32.lib")
8  #endif
9
10 #ifdef __linux__
11     #include <netinet/in.h>
12     #include <sys/socket.h>
13     #include <sys/types.h>
14 #endif

```

```

spike@localhost:~/桌面/dnsrelay_improve
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
[spike@localhost dnsrelay_improve]$ ls
cache.c  cache.h  DNS.c  dnsTire.c  dnsTire.h  global.c  global.h  header.h  ID.c  ID.h  package.c  package.h
[spike@localhost dnsrelay_improve]$ gcc ID.c -c
In file included from ID.c:1:0:
ID.h:23:2: 错误: 未知的类型名 'SOCKADDR_IN'
    SOCKADDR_IN client_addr;
    ^
ID.h:25:2: 错误: 未知的类型名 'BOOL'
    BOOL finished;
    ^
ID.h:37:57: 错误: 未知的类型名 'SOCKADDR_IN'
    void findOutOfTime(table_IDptr ID_table, int my_socket, SOCKADDR_IN server_addr);
                                                         ^
ID.h:39:88: 错误: 未知的类型名 'SOCKADDR_IN'
    unsigned short save_id(table_IDptr ID_table, char* buf, int length, unsigned short ID, SOCKADDR_IN client_addr);
                                                                    ^
In file included from ID.c:2:0:
global.h:4:20: 致命错误: process.h: 没有那个文件或目录
#include<process.h>
                   ^
编译中断。
[spike@localhost dnsrelay_improve]$ gcc package.c -c
package.c: 在函数 'createResponse' 中:
package.c:68:2: 警告: 隐式声明与内建函数 'memcpy' 不兼容 [默认启用]
    memcpy(DnsResponse, DnsInfo, LengthOfDns);
    ^
package.c: 在函数 'createRequest' 中:
package.c:126:2: 警告: 隐式声明与内建函数 'memcpy' 不兼容 [默认启用]
    memcpy(DnsInfo, &id, sizeof(unsigned short));
    ^
[spike@localhost dnsrelay_improve]$ s

```

最终我们在 linux 平台尝试了宏定义条件编译，发现很多变量类型并不通用。



## 5.5 字典接口

本地查询我们采用了**字典树**进行查找，长度即为查找次数，牺牲空间换取时间。

## 5.6 LRU 缓冲池

我们对于 **Cache** 实现了 LRU 缓冲机制；  
且对于宏参数定义进行了调试调整，使得最大程度上利用空间（频繁使用时在 95%的利用率）

此外，在 **ID** 转换表中，我们也采用了类似的机制，是的每一次查找 ID 转换空间时都寻找的第一个一定是最大概率可用的空间。

# 六：实验总结

## 6.1 实验经历

本次实验我们花费了整整一周时间完成。

1. 一天时间将模块搭建好；
2. 两天时间进行模块编写和整合；
3. 三天时间针对各种样例和环境进行测试，优化代码
4. 最后花费一天时间撰写实验报告。

## 6.2 实验感想

在本次课程设计中，虽然是线上合作，小组成员还是通过腾讯会议等方式频繁进行沟通交流讨论。从课设初期的讨论程序模块的划分和明确各自的任务，再到代码编写过程中实现的细节以及遇到的问题，在沟通中一起理解一起进步。

无论是 socket 的应用还是报文的结构、含义，小组成员在本次实验之前对他们的了解都仅限于认识层次，所以在程序编写时进行的不断理解和结构梳理上花了较多的时间。在课设代码的编写过程中，理论知识在程序编写时的应用加深了小组成员对 DNS 服务器工作机制的理解。小组成员在程序初步完成后，为了提升程序响应的速度，决定再次对部分模块进行重新编写，包括了本地映射表以及 cache 的查询等。

小组成员在此次课设中合作编写 DNS 中继器，对于服务机制和过程的认识更加清晰，也磨练了团队合作的意识和能力。希望在以后的学习中，能够多多参与团队合作和讨论，有更好的状态学习专业知识。