实验报告

PB19030861 王湘峰

一、实验要求

1. 编写应用程序，从配置文件中读取"域名-IP地址"列表，并按以下要求处理DNS请求：

- **Intercept** : 如果待查询的域名在列表中且对应IP地址为 0.0.0.0 ，则向客户端发送 0.0.0.0
- **Local resolve**：如果待查询的域名在列表中且对应IP地址有意义，则将对应的IP地址返回给客户端
- **Relay**：如果带查询域名不在列表中，则将查询报文转交给外部服务器并中继响应报文

2.对于每个DNS报文，输出查询的报文域名，并显示处理方式（Intercept/Resolve/Relay）和处理用时

二、实验原理

DNS报文结构

DNS报文分为请求报文和响应报文，其格式如下图：

事务ID（Transaction ID）	标志（Flags）
问题计数（Questions）	回答资源记录数（Answer RRs）
权威名称服务器计数（Authority RRs）	附加资源记录数（Additional RRs）
查询问题区域（Queries）	
回答问题区域（Answers）	
权威名称服务器区域（Authoritative nameservers）	
附加信息区域（Additional records）	

其中，事务 ID、标志、问题计数、回答资源记录数、权威名称服务器计数、附加资源记录数这 6 个字段是DNS的报文首部(header)，共 12 个字节。

整个 DNS 格式主要分为 3 部分内容，即首部、问题部分和资源记录部分。

首部

如图所示：

事务ID（Transaction ID）	标志（Flags）
问题计数（Questions）	回答资源记录数（Answer RRs）
权威名称服务器计数（Authority RRs）	附加资源记录数（Additional RRs）

该部分中每个字段含义如下：

- 事务 ID：DNS 报文的 ID 标识。对于请求报文和其对应的应答报文，该字段的值是相同的。通过它可以区分 DNS 应答报文是对哪个请求进行响应的。

- 标志：DNS 报文中的标志字段。
- 问题计数：DNS 查询请求的数目。
- 回答资源记录数：DNS 响应的数目。
- 权威名称服务器计数：权威名称服务器的数目。
- 附加资源记录数：额外的记录数目（权威名称服务器对应 IP 地址的数目）。

其中Flag字段又分为若干个字段：

QR	Opcode	AA	TC	RD	RA	Z	rcode

Flag字段中每个字段的含义如下：

- QR（Response）：查询请求/响应的标志信息。查询请求时，值为 0；响应时，值为 1。
- Opcode：操作码。其中，0 表示标准查询；1 表示反向查询；2 表示服务器状态请求。
- AA（Authoritative）：授权应答，该字段在响应报文中有效。值为 1 时，表示名称服务器是权威服务器；值为 0 时，表示不是权威服务器。
- TC（Truncated）：表示是否被截断。值为 1 时，表示响应已超过 512 字节并已被截断，只返回前 512 个字节。
- RD（Recursion Desired）：期望递归。该字段能在一个查询中设置，并在响应中返回。该标志告诉名称服务器必须处理这个查询，这种方式被称为一个递归查询。如果该位为 0，且被请求的名称服务器没有一个授权回答，它将返回一个能解答该查询的其他名称服务器列表。这种方式被称为迭代查询。
- RA（Recursion Available）：可用递归。该字段只出现在响应报文中。当值为 1 时，表示服务器支持递归查询。
- Z：保留字段，在所有的请求和应答报文中，它的值必须为 0。
- rcode（Reply code）：返回码字段，表示响应的差错状态。当值为 0 时，表示没有错误；当值为 1 时，表示报文格式错误（Format error），服务器不能理解请求的报文；当值为 2 时，表示域名服务器失败（Server failure），因为服务器的原因导致没办法处理这个请求；当值为 3 时，表示名字错误（Name Error），只有对授权域名解析服务器有意义，指出解析的域名不存在；当值为 4 时，表示查询类型不支持（Not Implemented），即域名服务器不支持查询类型；当值为 5 时，表示拒绝（Refused），一般是服务器由于设置的策略拒绝给出应答，如服务器不希望对某些请求者给出应答。

问题部分

问题部分指的是报文格式中查询问题区域（Queries）部分。该部分是用来显示 DNS 查询请求的问题，通常只有一个问题。该部分包含正在进行的查询信息，包含查询名（被查询主机名字）、查询类型、查询类。

问题部分格式如下图：

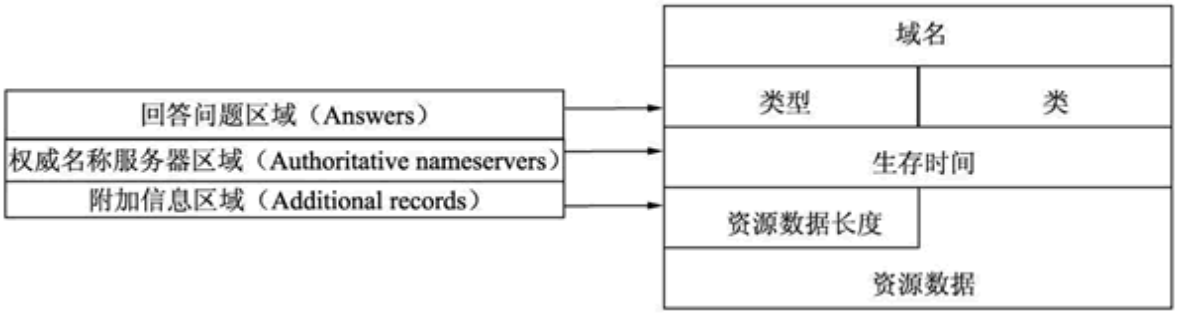
<table>
<tr><td colspan="2">查询名</td></tr>
<tr><td>查询类型</td><td>查询类</td></tr>
</table>

该部分中每个字段含义如下：

- 查询名：一般为要查询的域名，有时也会是 IP 地址，用于反向查询。
- 查询类型：DNS 查询请求的资源类型。通常查询类型为 A 类型，表示由域名获取对应的 IPv4地址。
- 查询类：地址类型，通常为互联网地址(IN)，值为 1。

资源记录部分

资源记录部分是指 DNS 报文格式中的最后三个字段，包括回答问题区域字段、权威名称服务器区域字段、附加信息区域字段。这三个字段均采用一种称为资源记录的格式，格式如下：



资源记录格式中每个字段含义如下：

- 域名：DNS 请求的域名。
- 类型：资源记录的类型，与问题部分中的查询类型值是一样的。
- 类：地址类型，与问题部分中的查询类值是一样的。
- 生存时间：以秒为单位，表示资源记录的生命周期，一般用于当地址解析程序取出资源记录后决定保存及使用缓存数据的时间。它同时也可以表明该资源记录的稳定程度，稳定的信息会被分配一个很大的值。
- 资源数据长度：资源数据的长度。
- 资源数据：表示按查询段要求返回的相关资源记录的数据。

三、代码展示与解释

环境配置与导入的包

```
# Python编译器版本为3.8
import struct
import socket
from time import time
import threading
```

其中struct用来解码二进制报文，time用于计算请求的处理时间，threading可以让请求并行的处理。

数据结构设计

本次实验创建了两个类：**DNS_Relay** 和 **DNS_Package**.

其中 **DNS_Relay** 类用于多线程处理请求（决定是否 本地解析/转发/拦截），**DNS_Package** 用于解析报文内容，并赋值给类中的成员，外部函数可以直接访问其成员。

在读取配置文件时，将域名-ip地址映射保存为字典 {域名：ip}

在记录中继报文时，将报文信息保存为字典 {事务ID：(源端口，域名，事务开始时间)}，其中记录开始时间是为了计算处理的时间。

函数与算法设计

DNS_Relay 中的函数

__init__：初始化：指定外部服务器和读入配置文件

```
class DNS_Relay:
    def __init__(self, config='config'):
        self.config = config
        self.map = {}  # dict<name:str, ip:str>
        self.read_config()
        self.nameserver = ('223.5.5.5', 53)
        self.relay = {}  # dict{id: int, tuple<addr,name, start_time>}
```

read_config：读取配置文件。以空格为分割，记录到字典中。

```
def read_config(self):
    with open(self.config, 'r') as f:
        for line in f:
            if line.strip() != '':
                ip, name = line.split(' ')
                self.map[name.strip()] = ip.strip()
```

run：处理报文的主函数，将报文多线程地处理。

```
def run(self):
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    s.bind(('0.0.0.0', 53))
    while True:
        try:
            data, addr = s.recvfrom(512)
            thread = threading.Thread(target=self.handle, args=(s, data, addr))
            thread.start()
            thread.join(timeout=1)
            except Exception:
                pass
```

handle：用于决定代理的行为（转发/拦截/本地解析），对每一个报文用DNS_Package 类解析字段。

```
def handle(self, s, data, addr):
    start = time()
    dns_package = DNS_Package(data)
    ID = dns_package.id
    if dns_package.is_query:
        name = dns_package.name
        # print(dns_package.type)
        if name in self.map and dns_package.type == 1:
            ip = self.map[name]
            response = dns_package.generate_answer(ip)
            s.sendto(response, addr)
            print(name, end='\t')
            if ip == '0.0.0.0':
                print(' INTERCEPT {}s'.format(time() - start))
            else:
                print(' RESOLVED {}s'.format(time() - start))
        else:
            s.sendto(data, self.nameserver)
```

```
            self.relay[ID] = (addr, name, start)
    else:
        if ID in self.relay:
            target_addr, name, start_time = self.relay[ID]
            s.sendto(data, target_addr)
            print(name, ' RELAY {}s'.format(time() - start_time))
            del self.relay[ID]
```

DNS_Package 中的函数

__init__：初始化类，提取首部和查询部分（如果需要的话）的信息。其中每个成员的含义如下：（加粗部分是因程序需要而自定义的成员）

- name　域名
- id　事务的ID
- flags　首部的flag字段
- type　查询类型(IPv4或IPv6)
- classify　查询类（问题部分中的类）
- quests　问题计数
- answers 回答资源计数
- author　权威名称服务器计数
- addition 附加资源记录数
- **position** 指针，指向域名的最后一位
- **is_query** 标记，表示该报文是否为IPv4查询报文
- **query**　如果是查询报文，则提取其问题部分。

```
class DNS_Package:
    def __init__(self, data):
        self.type = b''
        self.classify = b''
        self.querybytes = b''
        self.position = 0

        self.name = self.get_name(data)
        self.id, self.flags = struct.unpack('>HH', data[0:4])
        self.quests, self.answers, self.author, self.addition =
struct.unpack('>HHHH', data[4:12])
        self.is_query = not (self.flags & 0x8000)
        if self.is_query:
            self.query = self.query_part(data[12:])
```

get_name：返回值为域名，根据域名的规则来解析（data[i] 即为每个域名段的字符长度）

```
def get_name(self, data):
    name = ''
    i = 12
    while data[i] != 0:
        for j in range(1, data[i] + 1):
            name += chr(data[i + j])
        name += '.'
        i += data[i] + 1
    self.position = i - 12
    return name[:-1]
```

query_part：提取问题部分的type、class字段，并返回完整的问题部分。

```
def query_part(self, data):
    p = self.position
    self.querybytes = data[:p + 1]
    self.type, self.classify = struct.unpack('>HH', data[p + 1:p + 5])
    return self.querybytes + struct.pack('>HH', self.type, self.classify)
```

generate_answer：针对包含在配置文件中的域名，由程序自动生成响应报文。

函数按照 DNS 报文的结构构造相应报文，从头到尾分别是

- 首部
- 问题部分
- 回答部分
- 权威名称服务器部分
- 附加资源部分

其中对于拦截报文，置其flag字段为 0x8583 ， rcode = 3；对于非拦截报文，flag = 0x8180.

```
def generate_answer(self, ip):
    # Header
    response = struct.pack('>H', self.id)
    if ip == '0.0.0.0':
        response += struct.pack('>H', 0x8583)  # Flags: 0x8583 Intercept query
    else:
        response += struct.pack('>H', 0x8180)  # Flags: 0x8180 Standard query ,
No error
    self.answers = 1
    response += struct.pack('>HHHH', self.quests, self.answers, self.author,
self.addition)
    # Query
    response += self.query
    # Answer Author and Addition
    response += b'\xC0\x0C'  # point to name
    response += b'\x00\x01'  # TYPE:A
    response += b'\x00\x01'  # Class: IN (0x0001)
    response += b'\x00\x00\x00\xC8'  # Time to live: 200
    response += b'\x00\x04'  # Data length: 4
    # ip data
    s = ip.split(".")
    response += struct.pack('BBBB', int(s[0]), int(s[1]), int(s[2]), int(s[3]))
    return response
```

四、程序输入输出展示

nslookup

分别查询配置文件中的域名、被重定向的域名以及外部的域名：

```
C:\Users\It's Wonderful>nslookup www.test1.com
服务器:  localhost
Address:  127.0.0.1

非权威应答:
名称:    www.test1.com
Address:  127.0.0.1
```

```
C:\Users\It's Wonderful>nslookup www.bilibili.com
服务器:  localhost
Address:  127.0.0.1

非权威应答:
名称:    s.w.bilicdn1.com
Addresses:  119.3.65.116
          120.92.163.246
          120.92.147.239
          119.3.74.154
Aliases:  www.bilibili.com
```

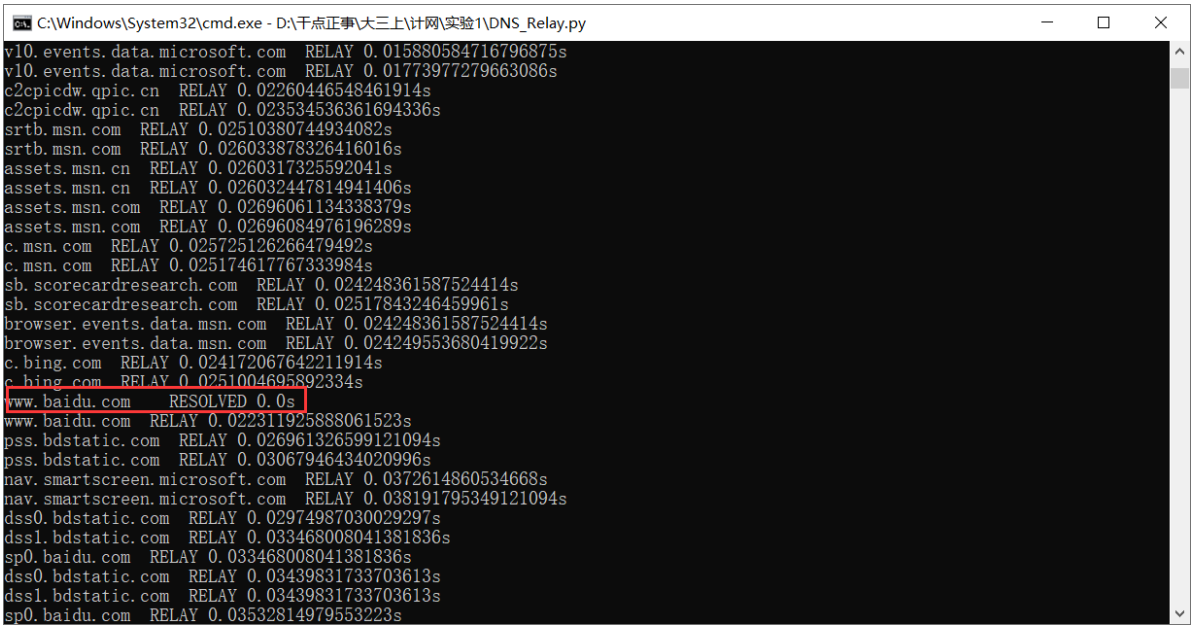
可以看到nslookup运行正常。

浏览器打开网页

www.baidu.com

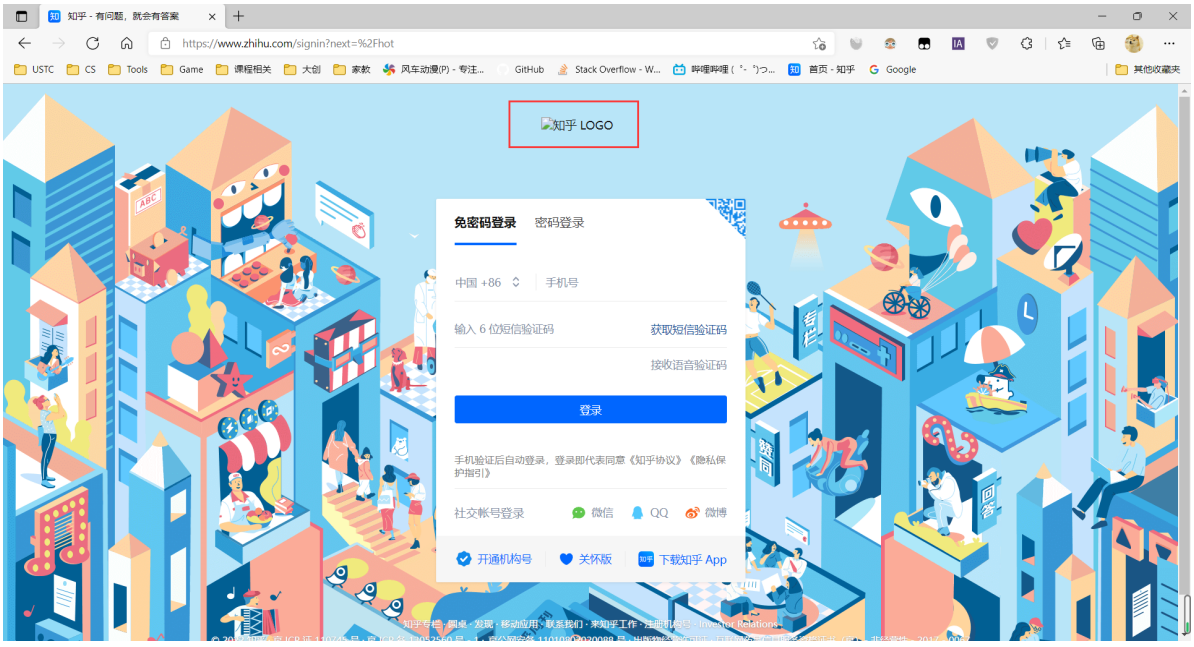
百度的域名已经被储存在配置文件中，浏览器成功打开网站。

可以看到baidu.com被本地解析了。（第二个baidu.com是IPv6请求，由于实验仅要求处理IPv4请求，故被移交给了外部服务器处理）

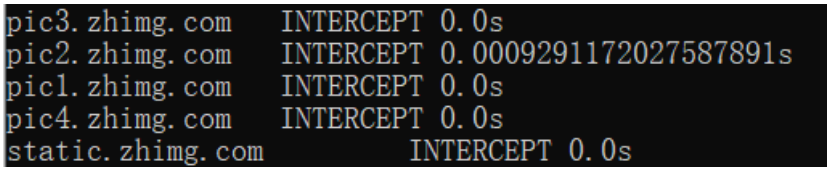


www.zhihu.com

知乎的四个静态图片都加入了拦截名单，可以看到 logo图片加载不出来了。



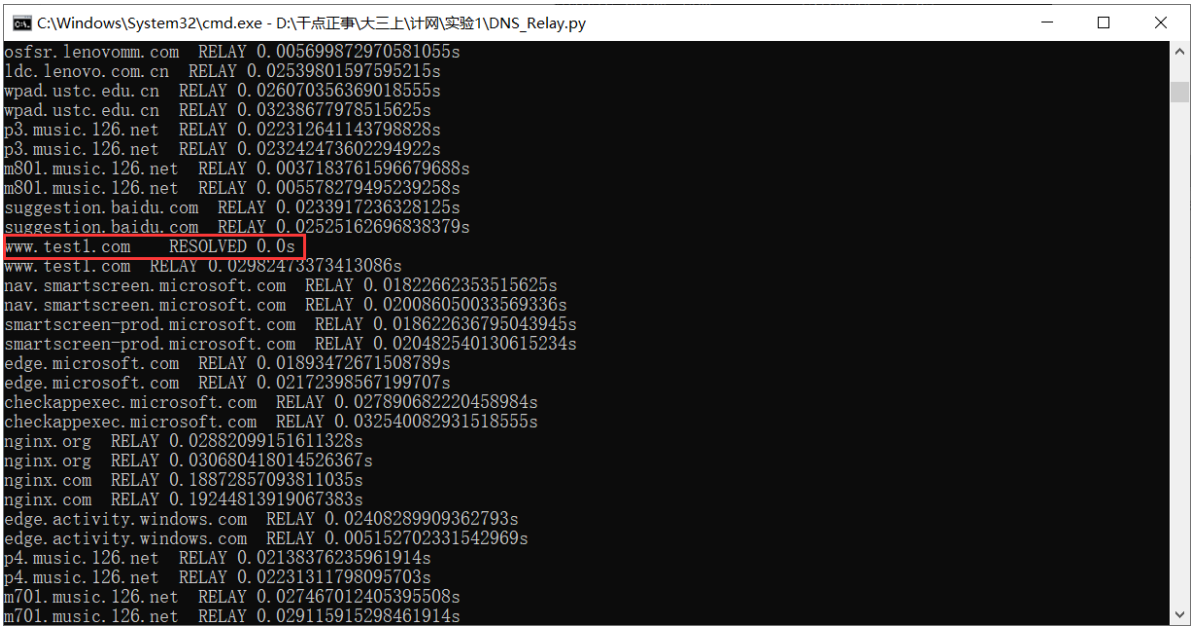
四个图片的域名显示被拦截。



www.test1.com

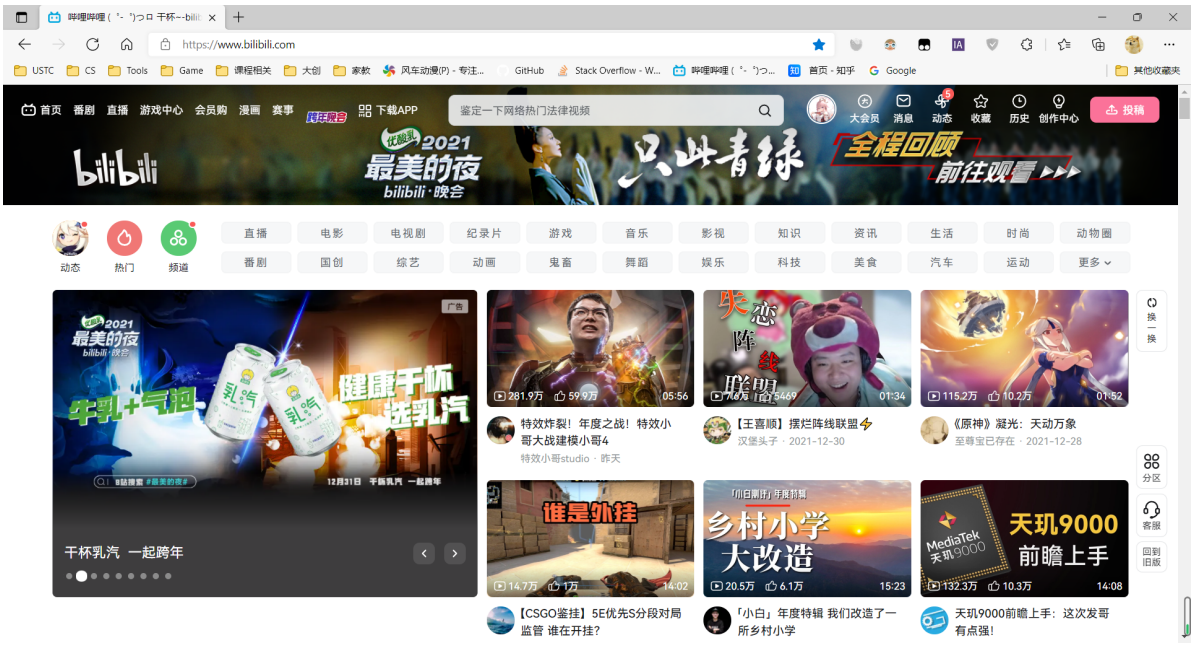
test1.com是个不存在的网站，但是在配置文件中被改写为了本地127.0.0.1. 由于提前部署了nginx作为本地服务器，打开后可以看到进入了nginx的欢迎页面。

下图为程序的output：

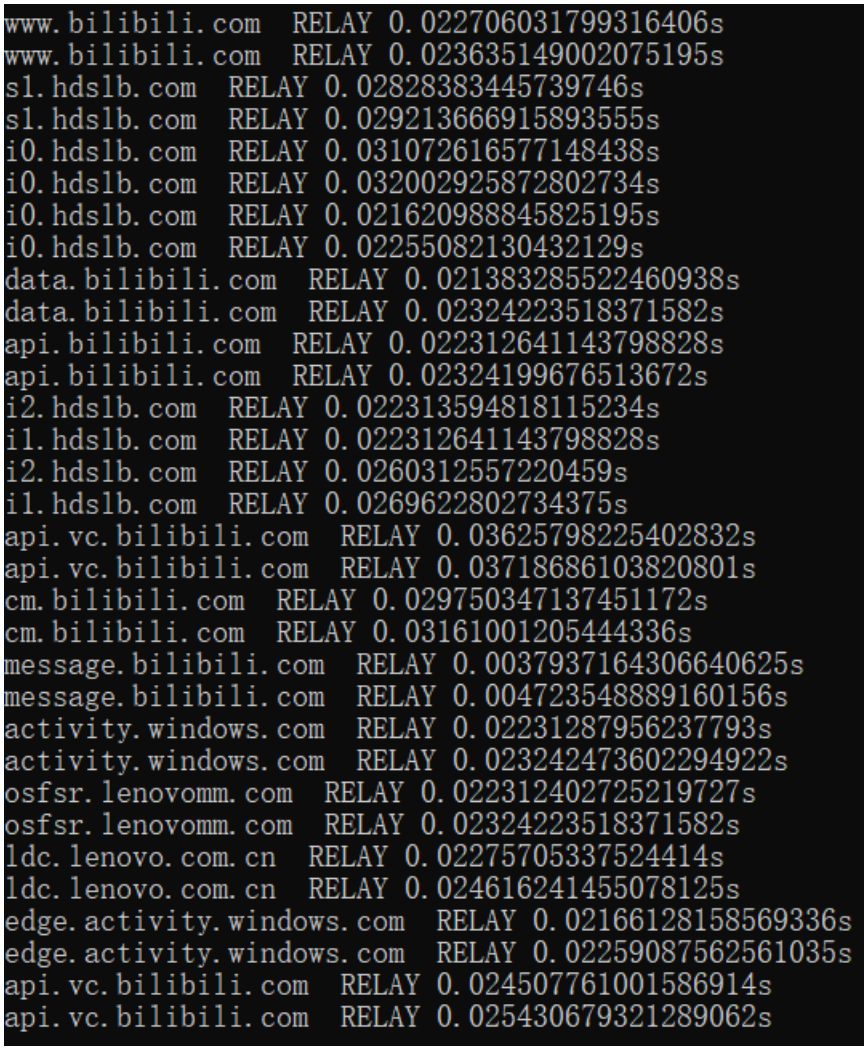


www.bilibili.com

B站并没有出现在配置文件中，网页所有的请求都转交给外部服务器处理。



所有网页元素都被正确请求。



五、实验总结

在本次实验的过程中，我详细学习了DNS报文的结构，阅读了RFC1035文档，丰富了课内知识；在编写代码的过程中学习了threading、struct等库的用法，接触了类(class)的概念，锻炼了动手能力；在代码完成后进行了充分的测试（例如运行着代码打CSGO、看视频等），通过程序的输出意识到平时看似简单的上网冲浪背后其实隐藏着复杂的信息流动，对计算机网络有了更加深刻的认识。

代码存在的不足之处在于未能正确处理IPv6请求，在以后的学习中会尝试继续了解相关的知识。