史文翰　2014211218　shiwenhan@bupt.edu.cn

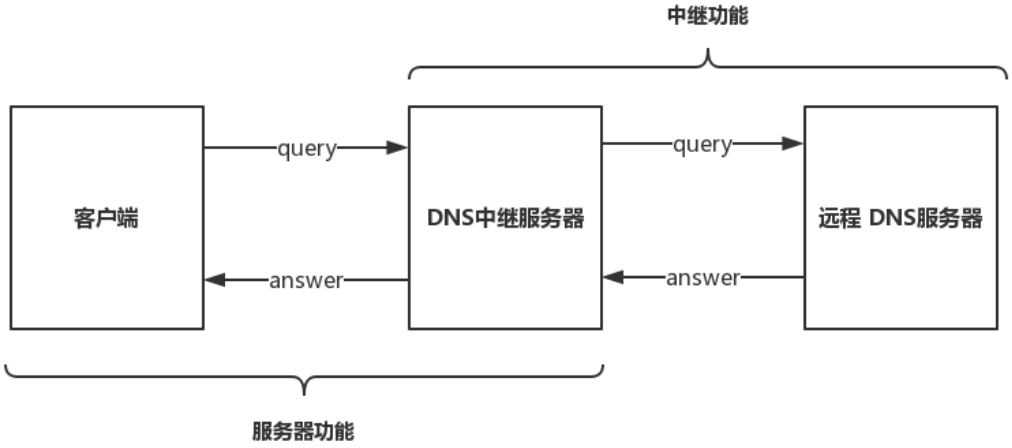
计算机网络实验报告——DNS 中继服务器

一、 功能设计

DNS 中继服务器运行在本地主机上，附有一个“域名-IP 地址”对照表。服务器接收 DNS 查询请求，有如下可能的情况和结果：

- 在对照表中检索结果为 0.0.0.0，向客户端返回“域名不存在”的报错信息
- 在对照表中检索到普通 IP 地址，则向客户端返回这个地址，完成一次 DNS answer
- 如果未检索到该域名，则向程序指定的远程 DNS 服务器发送 DNS query，并将结果（如果有）返回给客户端，完成一次 DNS answer

即该中继服务器的输入输出可表示如下：



二、 模块划分

本次实验采用 Python 编码，并采用类似脚本的面向过程思想来编写程序，系统共分为如下模块：

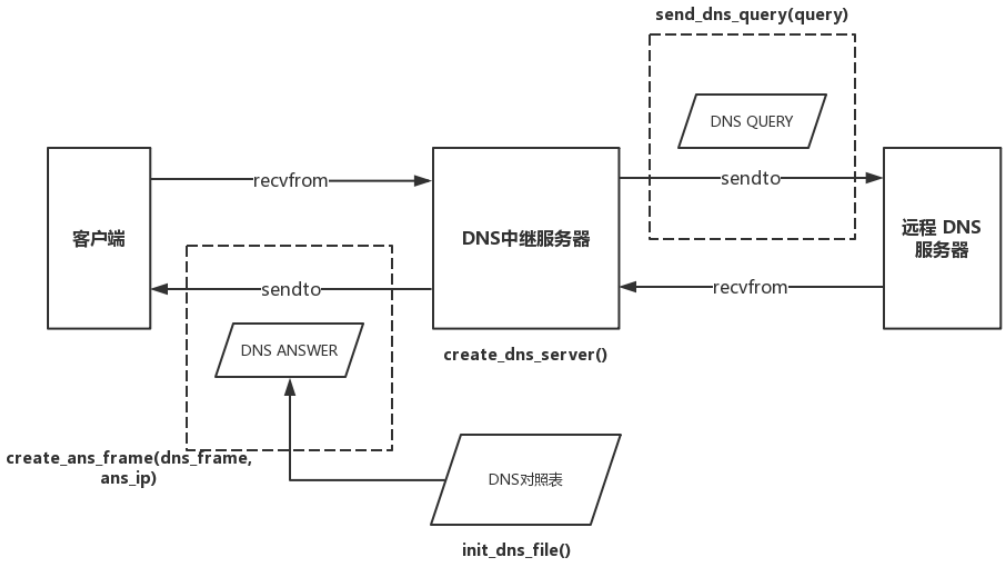
- **init_dns_file()**
  负责对照表的读入，以便后续过程能正常使用对照表。
- **create_dns_server()**
  是本程序的主体，该过程负责构建 DNS 服务器，包括进行 socket 绑定，设置非阻塞模式，并对 DNS 帧进行判断和收发。
- **create_ans_frame(dns_frame, ans_ip,filter)**
  是该程序面向局域网内 DNS 客户端的接口，负责构造并发送一条 DNS answer。报文的构造严格按照 DNS 协议中对 DNS 帧的定义，并将应答内容仅限于“只有一个非权威答案”。filter 表示这个回送帧是否要包含答案，按照需求定义，当回送一个黑名单域名时，并不需要显式地回送答案，只需要回送“域名不存在”帧即可。
- **send_dns_query(server, frame, addr)**

是该程序面向 DNS 远程服务器的发送接口，也是中继功能的一部分。当在对照表中找不到对应表项时，DNS 中继服务器向远程服务器以 socket 的形式发送 UDP 报文，其中包含了客户端发出的 DNS query。在发送之前，源地址（客户端的地址）应该被维护在中继服务器上，以便完成后续的回送任务。

- **handle_dns_ans()**

  用于处理 query_buf，以此来实现双线程。query_buf 负责收集那些将要发送给远端服务器的请求，在该函数中一并发送，只要其队列不为空。该过程仍然有接收过程，它从远端服务器接收 DNS answer，并将它置入 ans_buf。很显然，这个 answer 不能直接被发送，因为 nslookup 要求它收到的 answer 一定要来自 53 号端口（经查阅资料，这与 nslookup 内部实现有关），回送 answer 的任务要由“服务器”角色来最终完成。

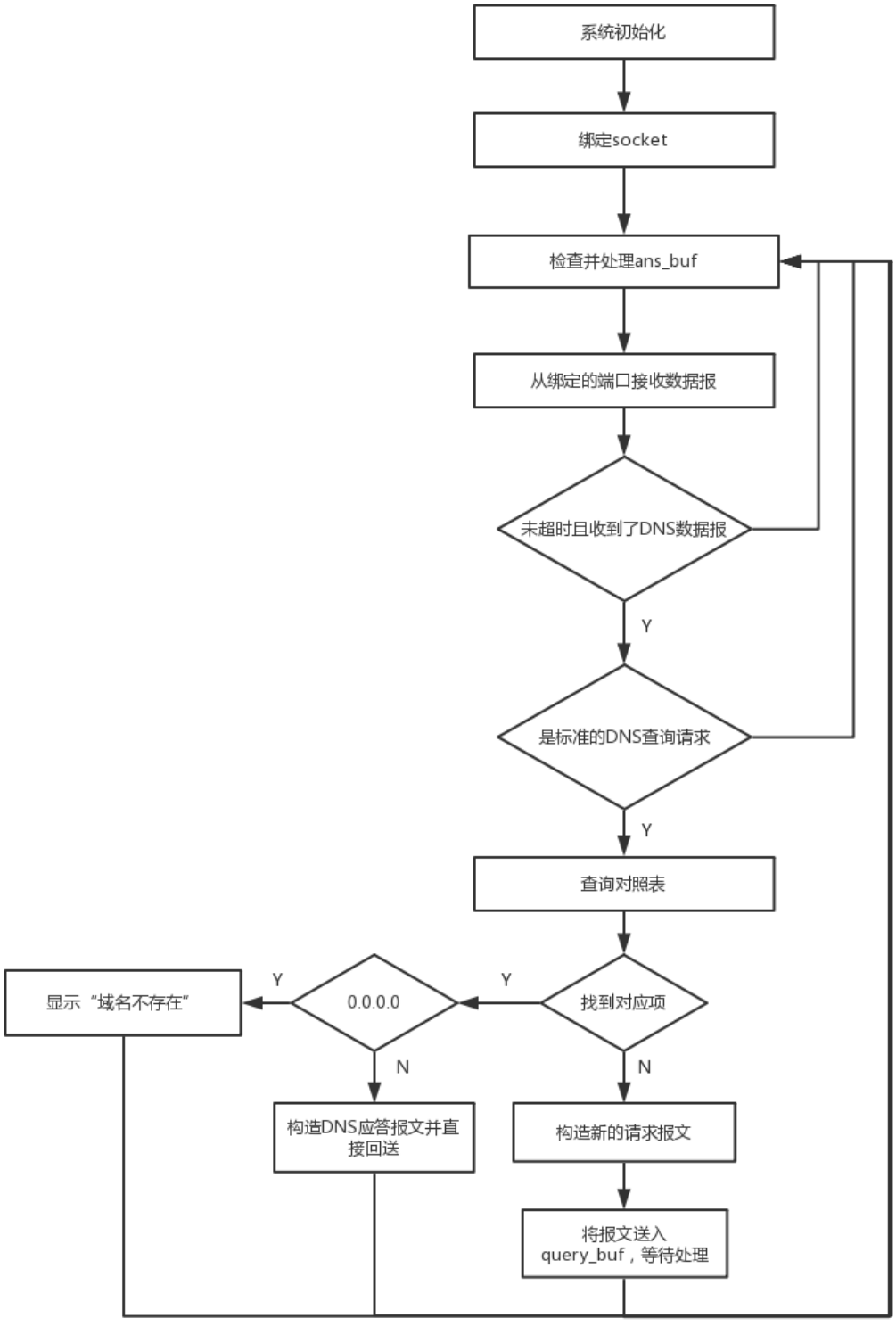
可用图表示模块之间的关系与交互：



可见本次设计的主要内容均为 DNS 中继服务器的 send 方法，其一是向远程 DNS 服务器发送 DNS 请求，其二是向客户端回送 DNS 应答。

DNS 中继服务器扮演了双重角色，他对于客户端来说，自己是 DNS 服务器，需要一个 ans_buf 来控制所有 answer 的回传。同时自身相对于远程 DNS 服务器来说，又是一个客户端，它利用自身的高位端口向远程服务器发送 query，因此还需要维护一个 query_buf。这两个 buffer 是双线程控制架构的基础。

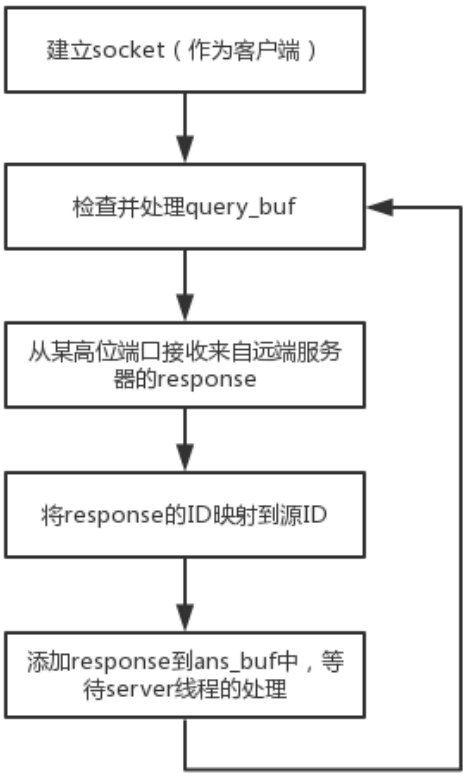
三、 软件流程图



上图是对本实验核心线程的顺序描述图。

首先看 create_dns_server，该过程首先进行初始化，包括传入需要绑定的地址，端口号，对照表等。随后，利用 UDP 模式绑定 socket 至 localhost，端口号为 53。在监听此 socket 的情况下，系统维护一个发送队列 query_buf，这是为了实现非阻塞发送而设计的，即服务器在没有收到回应的情况下，仍然可以处理当前需要发送的报文。在收到了实际需要的报文后，

服务器先查找对照表，如果对照表中有匹配选项，则判断匹配项的 IP 是否为 0.0.0.0，按照功能要求，如果是，则返回一个提示“该域名不存在”；如果存在普通 IP，则需要构造 DNS 应答报文，并直接回送。如果没有匹配项，服务器将该 DNS 请求中继到远程 DNS 服务器上，即加入到 query_buf 中。



上图对应于线程 handle_dns_ans，此线程负责处理报文的回送，即 handle_dns_ans，它维护一个 ans_buf 队列，在收到来自于远程服务器的应答后，这个应答不能被立即发送，而是缓存在 ans_buf 中等待处理。期间涉及到 ID 转换的问题。

四、　测试用例以及运行结果

测试本程序的过程将分为单机的和多机的。所谓单机，就是 DNS 中继服务器自己向自己请求 DNS，确保自身的域名解析工作是正常的；在多机测试中，我利用了寝室的局域网，将自己的主机作为 DNS 中继服务器，将学校的 DNS 作为远程 DNS 服务器，测试客户端（室友的电脑）是否能进行名字解析工作。

无论是单机还是多机，测试从四个方面进行：

- 利用 nslookup 命令解析较为有名的域名
- 根据对照表，利用 nslookup 命令解析普通 IP
- 根据对照表，利用 nslookup 命令解析 0.0.0.0
- 打开浏览器，能正常访问国内外的网站

1、单机测试

首先在服务器端对自身的 DNS 请求和应答做测试。

将首选 DNS 配置为 127.0.0.1，且利用 ipconfig /flushdns 命令清除 DNS 缓存。
利用 ipconfig 查找远程 DNS 服务器，为 10.3.9.5，利用如下命令开始程序（默认使用 dnsrelay.txt，且与脚本在同一级目录下，也可用-f 设置绝对路径）：

```
$ python dns_server.py –d 10.3.9.5
```

```
G:\PycharmProject\dns_server>python dns_server.py -d 10.3.9.5
[('-d', '10.3.9.5')]
Loading data files success.
Items:  905
Creating local DNS server...(UDP)
Remote DNS server: [10.3.9.5]
Binding socket [127.0.0.1:53]...
```

利用 nslookup 命令，首先查找一个较为普通的域名。

```
C:\Users\Administrator>nslookup www.baidu.com
DNS request timed out.
    timeout was 2 seconds.
服务器:  UnKnown
Address:  127.0.0.1

非权威应答:
DNS request timed out.
    timeout was 2 seconds.
名称:    www.a.shifen.com
Addresses:  220.181.111.188
          220.181.112.244
Aliases:  www.baidu.com
```

再查找一个可在对照表中找到的域名，但非 0.0.0.0。

```
C:\Users\Administrator>nslookup test1
DNS request timed out.
    timeout was 2 seconds.
服务器:  UnKnown
Address:  127.0.0.1

非权威应答:
DNS request timed out.
    timeout was 2 seconds.
名称:    test1
Address:  11.111.11.111
```

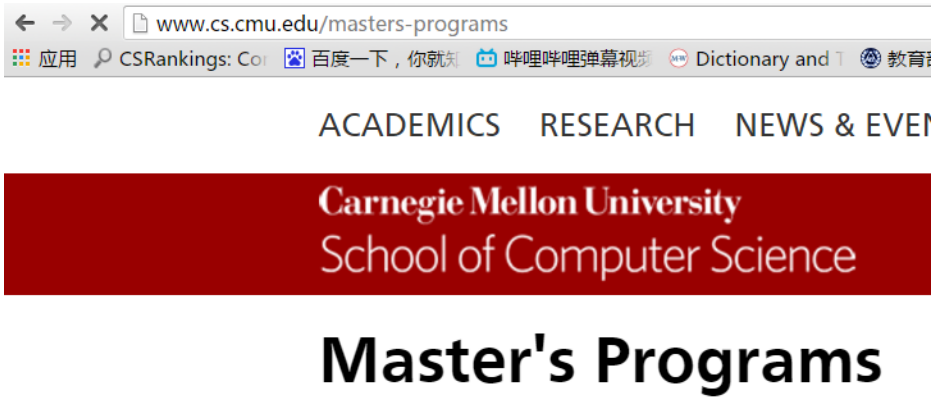
```
0.0.0.0 test0
11.111.11.111 test1
22.22.222.222 test2
```

与对照表中的结果相符。
随后，再查找一个对照表中为 0.0.0.0 的项。

```
C:\Users\Administrator>nslookup test0
DNS request timed out.
    timeout was 2 seconds.
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.
*** 请求 UnKnown 超时
```

```
[DNS QUERY FROM] ('127.0.0.1', 51349)
[Q_NAME] test0
[NOTE] Domain does not exist.
```

可见 nslookup 查找超时，且程序提示“域名不存在”。
再测试上网等环境，是否能进行正常的域名解析。



也没有任何问题。
至此，单机测试通过。

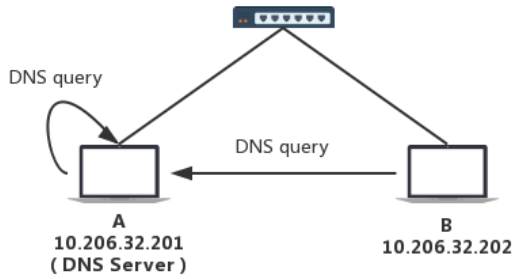
2、 多机测试

选宿舍的另外一台电脑做主机，依次通过 nslookup 命令和浏览器测试上述的各种情况。

```
以太网适配器 以太网:

   连接特定的 DNS 后缀 . . . . . . . : bupt.edu.cn
   描述. . . . . . . . . . . . . . . : Realtek PCIe GBE Family Controller
   物理地址. . . . . . . . . . . . . : C4-54-44-73-7C-BB
   DHCP 已启用 . . . . . . . . . . . : 是
   自动配置已启用. . . . . . . . . . : 是
   IPv6 地址 . . . . . . . . . . . . : 2001:da8:215:d606:b84e:1ff7:388a:e382(首选)
   临时 IPv6 地址. . . . . . . . . . : 2001:da8:215:d606:fd32:2b93:a4e7:3cf7(首选)
   本地链接 IPv6 地址. . . . . . . . : fe80::b84e:1ff7:388a:e382%19(首选)
   IPv4 地址 . . . . . . . . . . . . : 10.206.32.197(首选)
   子网掩码 . . . . . . . . . . . . : 255.255.255.224
   获得租约的时间  . . . . . . . . . : 2017年5月14日 19:42:12
   租约过期的时间  . . . . . . . . . : 2017年5月14日 20:42:07
   默认网关. . . . . . . . . . . . . : fe80::223:89ff:fe57:f1%19
                                       10.206.32.193
   DHCP 服务器 . . . . . . . . . . . : 10.3.9.32
   DHCPv6 IAID . . . . . . . . . . . : 79975492
   DHCPv6 客户端 DUID  . . . . . . . : 00-01-00-01-1F-E0-2F-AA-C4-54-44-73-7C-BB
   DNS 服务器  . . . . . . . . . . . : 10.206.32.201
   TCPIP 上的 NetBIOS  . . . . . . . : 已启用
```

将它的首选 DNS 指向我主机（10.206.32.201）

网络拓扑图如下：

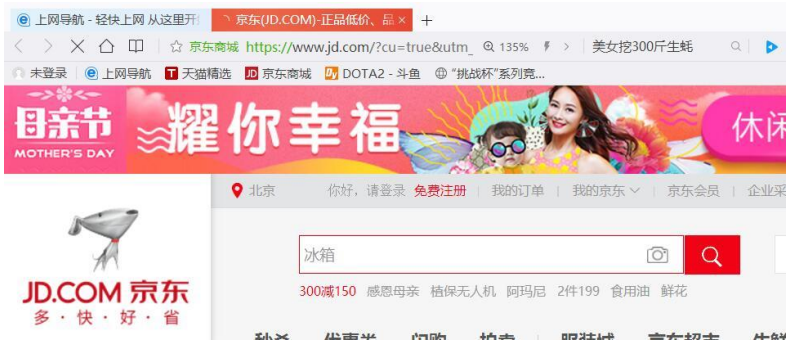


进行 nslookup 和浏览器测试如下：

```
C:\Users\张万里>nslookup test0
DNS request timed out.
    timeout was 2 seconds.
服务器:  UnKnown
Address:  10.206.32.201

DNS request timed out.
    timeout was 2 seconds.
DNS request timed out.
    timeout was 2 seconds.
*** 请求 UnKnown 超时
```

```
C:\Users\张万里>nslookup www.x.com.cn
DNS request timed out.
    timeout was 2 seconds.
服务器:  UnKnown
Address:  10.206.32.201

DNS request timed out.
    timeout was 2 seconds.
DNS request timed out.
    timeout was 2 seconds.
非权威应答:
DNS request timed out.
    timeout was 2 seconds.
名称:    www.x.com.cn
Address:  69.43.161.167
```

```
C:\Users\张万里>nslookup www.qq.com
DNS request timed out.
    timeout was 2 seconds.
服务器:  UnKnown
Address:  10.206.32.201

DNS request timed out.
    timeout was 2 seconds.
DNS request timed out.
    timeout was 2 seconds.
非权威应答:
DNS request timed out.
    timeout was 2 seconds.
名称:    www.qq.com
Address:  123.151.148.111
```



可以看出 nslookup 正常工作，浏览器也可以正常工作。

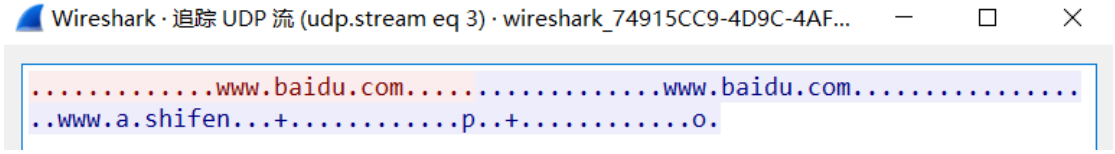
五、 问题与解决

1、 DNS 报文格式具体如何？怎样解析他们？在 Python 中怎样处理类似的数据格式？

这是我开始编程遇到的第一个问题。DNS 格式看起来十分复杂，但利用 wireshark 进行抓包解析，它能提供类似于 Python 字典一样清晰的格式，对 DNS 报文格式一目了然。更重要的是，去繁就简，本程序应该只关注部分 DNS 报文，比如类型为 A 的标准请求，只有一个非权威答案的应答报文等。

利用 nslookup 命令先构造一些标准的请求和应答报文，如对 www.baidu.com：

一个最标准的 DNS 请求-应答 UDP 流可以轻易被捕捉



红色部分是客户端发出的 query，蓝色部分是服务器返回的 answer。

No.	Time	Source	Destination	Protocol	Length	Info
6	0.842813	10.128.240.40	10.3.9.6	DNS	73	Standard query 0xc2bd A www.baidu.com
33	0.932018	10.3.9.6	10.128.240.40	DNS	132	Standard query response 0xc2bd A www.baidu.com

只解析这一组报文，都能对 DNS 的最简单的请求-应答格式了解个十有八九。

Python 提供了 struct 包来处理类似于 C 语言的结构，利用 pack，unpack 方法可以很容易构造/解析一个字节流。

2、 在 Python 中进行 socket 绑定，为了使 DNS 指向自己，是应该显示绑定('127.0.0.1', 53）还是（'',53）？两者有何区别？

这是本次实验遇到的最大的问题，以至于写了篇思考与老师进行了交流。值得注意的是，在 Python 的 socket 方法中，**不显示指出 127.0.0.1** 与 C 语言中的 INADDR_ANY 宏具有相同的含义，后者表示绑定 localhost 地址和网络真实地址（可能经过了 NAT）。如果显示指出 127.0.0.1，那么在多机测试中，客户端向服务器发送的 DNS 请求不能被此 socket 监听到。

关于此问题的更多思考和细节请见《对于 DNS 实验的一些问题和思考.pdf》。

3、 如何在 Python 中将 UDP socket 变为非阻塞？

在一开始编程时，我才用的是阻塞的方法，这明显不符合实验的要求。因为 socket 通讯被阻塞在 recvfrom 过程下，只要没收到数据报，系统无法继续运行。

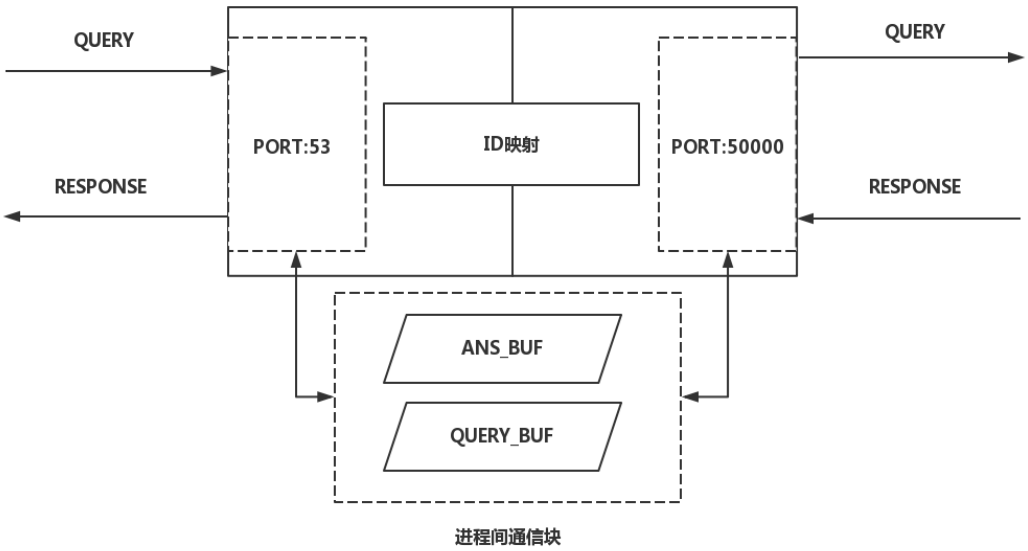
在 Python 语境下，socket 提供了 settimeout 的内置方法，来决定接收超时时间。显然地，阻塞模式的 socket 相当于显示指出 settimeout（∞）。

在本次试验中，默认设置为 settimeout（1），即每次接收报文会阻塞 1 秒，否则被视为超时并跳过接收过程。

4、 为什么要设计双线程？

其一，这是为了功能的需要，处理多机并发，并且合理解决“在未得到应答之前可以继续询问”的问题。

其二，也是更重要的，从系统的设计角度来看，本次 DNS 中继服务器同时扮演了两个角色：服务器和客户端，可以用下图表示。



不难看出，由于服务器扮演了双重角色，那么在处理收发 DNS 帧时，自然应该将功能解耦，并通过一定的通讯手段将二者联系起来。因此我设计了双线程模式，使程序可以更加清晰，各重构性强。具体来说，在使用中继功能时，相对于远程 DNS 服务器来说，本地 DNS 服务器是一个“客户端”；而在使用黑名单和白名单功能时，相对于局域网内其他的客户端，本地 DNS 服务器是一个真正的“服务器”。选择高地址端口中继 DNS 请求，是为了在远端 DNS 服务器看来，这个帧就像是某个实际存在的客户端发来的一

样，而不是被中继的信息。

其实，在此之前我采用的是单线程假并发，也取得了不错的效果。但如果客户端数量上升，网络情况极不稳定等情况发生时，就不是那么靠谱了。

六、 心得体会

本次实验的心得体会主要体现在如下三个方面。

其一，编程语言方面。我选择的是 Python 而非 C，虽然我知道 C 语言更适合写类似网络协议这种的底层程序，速度更快也更轻便，但考虑到 C 语言较为复杂的语法，缺乏丰富的库的支持，以及对于一些需求需要较高的学习成本，因此还是直接选择拿 Python 编写。在网络编程方面，Python 的 socket 包对各个平台支持性良好，且提供了丰富的接口函数，与 C 语言十分类似，上手简单。更重要的是，Python 一些简单的语法特性和库（如 struct）的支持，使得编写脚本程序十分顺畅敏捷。整个程序从编写到维护共约 10 小时，且总代码量不超过 200 行。

其二，是对于网络协议本身。在学习《计算机网络》时，总对书中的各种各样的网络协议不得其解，认为其晦涩难懂，格式杂乱，各个字段纷繁复杂。更不要说 RFC 的各种文档，英语不好，耐心也不够。但从上一个网络编程实验开始，积极使用 wireshark 抓包解析工具，并熟练掌握了抓包相关的功能，在本次实验中它的帮助为我节约了大量的时间。它对于流行的协议大多都提供了良好的界面支持，良好的 UI 和很容易理解的信息表达方式。

其三，在调试方面，应该多查阅相关资料，不仅仅关于协议本身，更多的是你所用的语言对于你要完成的任务最多可以支持到哪里。最好不要造轮子，有一些解析工具、现成的标准函数和一些接口已经足够使用来漂亮地构造或调试程序。尤其是对于网络编程，多收发几个相关的包，拿到 wireshark 一一解析其字段，哪里有问题立即就会清晰可见。